UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Eduardo dos Santos

FORMALIZAÇÃO E VERIFICAÇÃO DE UM PROTOCOLO DE AUTENTICAÇÃO MULTIFATOR

Florianópolis
2012

Eduardo dos Santos

FORMALIZAÇÃO E VERIFICAÇÃO DE UM PROTOCOLO DE AUTENTICAÇÃO MULTIFATOR

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do grau de Mestre em Ciência da Computação.

Prof. Ricardo Felipe Custódio, Dr.
Orientador

Florianópolis
2012

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

dos Santos, Eduardo
Formalização e Verificação de um Protocolo de Autenticação Multifator [dissertação] / Eduardo dos Santos ; orientador, Ricardo Felipe Custódio - Florianópolis, SC, 2012.
139 p. ; 21cm

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Autenticação Multifator. 3. Protocolo de Segurança. 4. Verificação Formal. 5. Prova Automática de Teoremas. I. Custódio, Ricardo Felipe. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Eduardo dos Santos

FORMALIZAÇÃO E VERIFICAÇÃO DE UM PROTOCOLO DE AUTENTICAÇÃO MULTIFATOR

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, área de concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Florianópolis, 22 de Outubro de 2012

Prof. Ronaldo dos Santos Mello, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Ricardo Felipe Custódio, Dr.
Orientador
Universidade Federal de Santa Catarina

Profa. Noemi de La Rocque Rodriguez, Dra.
Pontíficia Universidade Católica do Rio de Janeiro

Prof. Julio César López Hernández, Dr.
Universidade Estadual de Campinas

Prof. Olinto José Varela Furtado, Dr.
Universidade Federal de Santa Catarina

Prof. Jean Everson Martina, Dr.
University of Hertfordshire, Reino Unido

AGRADECIMENTOS

Esta Dissertação de Mestrado não teria se tornado realidade sem a colaboração de diversas pessoas. Peço desculpas adiantadas caso me esqueça de alguém.

Inicialmente, agradeço aos meus pais, Paulo Roberto dos Santos e Rosângela dos Santos, pelas condições dadas ao longo de toda a vida para que eu pudesse chegar até aqui. O investimento em educação proporcionado desde a infância para que eu pudesse me tornar uma pessoa instruída não foi, com certeza, em vão.

Em seguida, direciono meu reconhecimento aos meus orientadores, professor Ricardo Felipe Custódio e Jean Everson Martina, os quais foram fundamentais para a realização deste trabalho. Sem a constante troca de ideias e correções, este trabalho não teria o mesmo grau de qualidade técnica. Além do crescimento como pesquisador e aumento do gosto pela ciência, ambos proporcionaram meu aprimoramento como pessoa e cidadão do mundo durante o curso de Mestrado.

Não por menores também são meus cumprimentos a todos os atuais colegas do LabSEC, não apenas pelas sugestões feitas ao trabalho, mas também pelos momentos de descontração. Em particular, ao Bruno Imhof, pela oportunidade de ser o coorientador de seu Trabalho de Conclusão de Curso. Também agradeço aos bolsistas de Iniciação Científica, Lucas Vinícius da Rosa e Leandro Machado, pela implementação da biblioteca de integração do protocolo proposto. Os antigos membros do laboratório, nas pessoas de Juliano Romani, Marcelo Carlomagno Carlos e Túlio Cícero Salvaro de Souza, também são dignos de nota, pelos orientações e troca de ideias desde minha primeira passagem pelo LabSEC, em 2005.

No âmbito da Universidade, não posso me esquecer de agradecer ao Ambiente de Acessibilidade Informacional (AAI) da Biblioteca Universitária. A adaptação de material bibliográfico realizada foi imprescindível para meu bom acesso a algumas obras. Agradeço também ao professor Arthur Ronald de Vallauris Buchsbaum pelas inúmeras horas dispendidas no ensino da Lógica.

Gostaria também de deixar registrado meu muito obrigado à minha fonoaudióloga, Andréa Hackradt Silva, e professora de Francês, Pauline Bianca Erbs. Ambas me ajudaram de maneira indireta, para que eu não desistisse ou perdesse o rumo.

Por fim, agradeço à CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pela concessão de bolsa de estudos, permitindo que eu me dedicasse integralmente às atividades de pesquisa. Nesse sentido, também agradeço à empresa Softplan pelo apoio financeiro em parte do meu Mestrado.

Eduardo dos Santos, Agosto de 2012

"Although the world is full of suffering,
it is also full of the overcoming of it."

"Embora o mundo esteja cheio de sofrimento, ele
também está cheio da superação do sofrimento."

Helen Adams Keller (1880–1968)
A primeira pessoa surdocega a obter um
diploma universitário nos Estados Unidos.

RESUMO

Nesta Dissertação de Mestrado, apresenta-se a proposta de um protocolo para autenticação de usuários, fazendo uso de biometria e *smartcards*.

As principais características desse conjunto de protocolos são: prover um mecanismo próprio e integrado para cadastro e autenticação, assim como permitir a existência de diferentes papéis de usuários. Com a união destas duas características, almeja-se a obtenção de protocolos que possuam níveis aprimorados, não apenas de segurança, mas também de gerenciamento.

Seu desenvolvimento é norteado pela adoção de uma metodologia própria ao projeto de protocolos de segurança. As seguintes etapas fazem parte desta metodologia: projeto inicial, prototipação, implantação, modelagem formal e verificação.

A modelagem formal é feita em Lógica de Primeira Ordem. Os modelos lógicos criados são, posteriormente, alvo de verificação com auxílio de um provador automático de teoremas (em nosso caso, o SPASS). De modo a antecipar as ações de um atacante em potencial, seus possíveis movimentos são também alvo de formalização, resultando na criação de um modelo lógico próprio. Logo após, tem início a etapa de verificação, a qual consiste no teste de conjecturas sobre os modelos lógicos. O resultado deste teste permite a extração de fatos (certezas) sobre nosso conjunto de protocolos. Em última análise, estes fatos são a comprovação da resistência do protocolo a padrões conhecidos de ataque.

Com a finalidade de propiciar uma visão geral da temática da verificação de protocolos de segurança, este trabalho também apresenta uma revisão dos métodos disponíveis, não se limitando apenas àqueles efetivamente utilizados neste estudo. Ademais, todo o material relativo às modelagens formais e respectivas provas é incluído como anexos.

Palavras-chave: Autenticação Multifator, Protocolo de Segurança, Verificação Formal, Prova Automática de Teoremas.

ABSTRACT

In this Master Thesis, a proposal for a set of authentication protocols through the use of biometrics and smartcards is presented.

The main characteristics of such protocols are: to provide their own integrated mechanism for user registration and authentication, as well as to differentiate the existent user roles. By combining these characteristics, we aim at creating protocols with, not only improved security levels, but also with flexible management.

Their development is guided by the adoption of a proper methodology for the security protocols' project. The following steps make part of it: initial design, prototyping, deployment, formal modelling and verification.

The formal modelling is carried out in First-Order Logic. The logic models created are subsequently targeted of verification with the assistance of an automated theorem prover (in this case, SPASS). To anticipate the actions from a potential attacker, his characteristics are also target of formalisation, resulting in his own logic model. After, we proceed with the verification which consists on testing conjectures upon the logic models. The results collected with those tests allow the extraction of facts about our set of protocols. In a last instance, such facts are the evidence of the protocol's resistance to well-known attack patterns.

In order to provide a broader view of the subject of security protocol verification, this work also reviews all the available methods. Not limiting to those already used in this study. In addition, every piece of material related to the formal modelling and respective proofs is included as appendices.

Keywords: Multi-factor Authentication, Security Protocol, Formal Verification, Automated Theorem Proving

RÉSUMÉ

Cette Thèse de Maîtrise présente la proposition d'un ensemble de protocoles pour l'authentification des utilisateurs, par le biais de la biométrie et des cartes à puce.

Cet ensemble de protocoles fournit un mécanisme propre et intégré pour le cadastre et l'authentification, permettant ainsi le classement des différents types d'utilisateurs. Avec la combinaison de ces caractéristiques on vise l'amélioration des niveaux de protocole en termes de sécurité, mais aussi de gestion.

Son développement est guidé par l'adoption d'une méthodologie propre au projet de protocoles de sécurité. Cela implique : un projet initial, puis le prototypage, ensuite l'implantation, le modelage formel et enfin la vérification.

Le modelage formel est réalisé dans la Logique de Première Ordre. Les modèles logiques crés seront ensuite utilisés comme cibles de vérification à l'aide d'un proveur automatique de théorèmes (dans ce cas, le SPASS). Pour prévenir les actions d'un attaquant potentiel, ses mouvements possibles seront formalisés à l'aide de la création d'un modèle logique propre. Ensuite ou après cela, l'étape de vérification commence, ce qui comprend le test de conjectures sur les modèles logiques crés. Le résultat de ceci permet l'extraction de faits (des assurances) sur notre ensemble de protocoles. Dans la dernière analyse, ces faits sont les preuves de la résistance du protocole aux modèles d'attaques connus.

Afin de fournir une vision générale de la thématique de la vérification de protocoles de sécurité, ce travail présente aussi une révision des méthodes disponibles, ne se limitant pas à ceux utilisés dans cet étude. De plus, tout le matériel relatif aux modèles formels et aux preuves logiques respectives est inclus dans ce travail en annexe.

Mots-clés : Authentification en Divers Facteurs ; Protocole de Sécurité ; Vérification Formelle ; Preuve Automatique de Théorèmes.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AES	*Advanced Encryption Standard*
AVISPA	*Automatic Validation of Internet Security Protocols and Applications*
CPF	Cadastro de Pessoa Física
CNPJ	Cadastro Nacional de Pessoa Jurídica
CSP	*Communicating Sequential Processes*
DNA	*Deoxyribonucleic Acid*
FDR	*Failures Divergence Refinement*
FIPS	*Federal Information Processing Standards*
FOL	*First-Order Logic*
GSM	*Global System for Mobile Communication*
HOL	*Higher-Order Logic*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
ICP	Infra-Estrutura de Chaves Públicas
ICP-Brasil	Infra-Estrutura de Chaves Públicas Brasileira
IEEE	*Institute of Electrical and Electronics Engineers*
INCITS	*International Committee for Information Technology Standards*
ILO	*International Labour Organization*
ISO/IEC	*International Organisation for Stardardisation/ International Electrotechnical Commission*
LabSEC	Laboratório de Segurança em Computação
NSPK	*Needham-Schroeder Public-Key Protocol*
PIN	*Private Identification Number*
PKCS	*Public-Key Cryptography Standard*
SAT	*SATisfiability*
SET	*Secure Electronic Transaction*
SDK	*Software Development Kit*
SIM	*Subscriber Identity Module*
SMTP	*Simple Mail Transfer Protocol*
SSL/TLS	*Secure Sockets Layer/Transport Layer Security*
UML	*Unified Modeling Language*

LISTA DE SÍMBOLOS

$\forall$	Para todo
$\exists$	Existe
$\wedge$	E
$\vee$	Ou
$\neg$	Não
$\rightarrow$	Implicação
$\leftrightarrow$	Equivalência
$\vdash$	Prova
$\nvdash$	Não prova
ω	Fórmula (premissa ou axioma)
ψ	Fórmula (conclusão ou conjectura)
Γ	Conjunto de Fórmulas
$\cup$	União

SUMÁRIO

1 INTRODUÇÃO

O crescente desenvolvimento da Internet registrado nas últimas décadas tem proporcionado uma maior interação entre pessoas e serviços baseados na rede. Como expoentes desta nova realidade, pode-se citar o número elevado de informações pessoais compartilhadas entre usuários através das redes sociais; a criação e constante aprimoramento de novas formas de realização de transações financeiras, contribuindo para a geração de riqueza em larga escala, e a maior integração entre diferentes sistemas e dispositivos, permitindo a atualização de dados em tempo real.

Por outro lado, a existência de tantas possibilidades de interação com consequente troca de informações sigilosas, torna os serviços envolvidos um alvo preferencial no roubo de dados. Este cenário tem contribuído para o desenvolvimento do campo de estudos da Segurança da Informação. O principal objetivo deste campo é a busca de meios que tornem as comunicações em meio eletrônico protegidas à ação de usuários mal-intencionados.

A segurança não é algo homogêneo. Quando dizemos que *"todos os sistemas devem ser seguros"*, estamos sendo muito imprecisos. Estamos nos referindo a quê, exatamente? Que as informações não caiam nas mãos de terceiros? Ou, que o sistema saiba quando está recebendo ordens de um impostor? Ou, ainda, que o sistema impeça o usuário de cometer erros por deslize? Com o devido cuidado, podemos construir sistemas levando em conta todas essas perguntas.

A construção de sistemas de informação seguros deve levar em conta cinco aspectos básicos. Esses aspectos resumem o que pode ou não ser oferecido em matéria de segurança aos usuários do sistema. Em algumas situações, devido aos altos custos de implementação envolvidos, alguns aspectos de segurança podem vir a ter maior prioridade que outros.

Inicialmente, devemos fornecer **confidencialidade** (ou sigilo) aos dados trocados entre o sistema e seus usuários. O nível de sigilo deve ser forte o suficiente para impedir a leitura dos dados por terceiros não-autorizados.

Devemos também prover garantias de **integridade** aos dados. Ou seja, garantir que os mesmos não sejam modificados enquanto trafegam pela rede. Eventual alteração deve ser de fácil detecção.

Precisamos averiguar a **autenticidade** das mensagens recebidas, isto é, verificar se as mensagens recebidas foram de fato enviadas pelo usuário apontado como emissor.

Outra questão importante é a presença de mecanismos de **autenticação** confiáveis de modo a evitar que um usuário tenha dúvidas sobre a real identidade de seus interlocutores.

E, deve-se também levar em consideração, a existência de uma política de **autorização** rígida para impedir que usuário faça uso de

serviços ou recursos fora dos previamente acordados.

Além disso, para o bom entendimento deste trabalho, é necessário a correta assimilação do conceito de **protocolo**, principalmente de **protocolo de segurança**. Segundo (RYAN; SCHNEIDER, 2000), um **protocolo de segurança**, em tradução livre, *"compreende uma sequência pré-definida de interações entre entidades para alcançar um fim especifico"*. Em diversos casos, as entidades envolvidas possuem interesses conflitantes, tornando o projeto de protocolos uma atividade ainda mais trabalhosa e complexa. É necessário ter em mente que, por mais que sejam desenvolvidos para uso em computadores, existe toda uma base conceitual que independe do meio sobre o qual é feita sua implementação. Esta base conceitual faz uso sobretudo, da matemática (em particular, da lógica) para garantir o correto atendimento dos objetivos do protocolo.

Cabe salientar ainda que protocolos podem fazer uso de dados providos por outros protocolos. Isto é bastante comum, especialmente se considerarmos o paradigma das redes de computadores, onde diferentes protocolos cuidam das tarefas de transmissão em meio físico, conexão *host-a-host*, roteamento de pacotes, apresentação da informação e criptografia de dados (TANENBAUM, 2002). Outro exemplo bastante ilustrativo desta situação é o comércio eletrônico. Ao efetuar uma compra, fazemos uso de protocolos para: exibição da página *web* da loja virtual (protocolo HTTP (FIELDING et al., 1999)); recebimento de *e-mail* de confirmação da compra (protocolo SMTP (POSTEL, 1982)) e transmissão sigilosa dos dados do cartão de crédito (protocolo SSL/TLS (DIERKS; ALLEN, 1999)).

1.1 DEFINIÇÃO DO PROBLEMA

O problema que nosso trabalho se propõe a resolver é a criação de um novo protocolo para autenticação de usuários no âmbito do Poder Judiciário.

Após uma análise detalhada, constatou-se que a versão do protocolo em uso (até o início deste projeto de pesquisa), contém algumas vulnerabilidades. Na eventualidade de terem sido exploradas, estas vulnerabilidades podem permitir o acesso de usuários não-autorizados aos sistemas envolvidos.

Apesar de ter sido criado para resolver este problema específico, o uso deste protocolo é facilmente adaptado a outros cenários. No decorrer do texto, adotamos sempre explicações genéricas de seu funcionamento. Por questões de sigilo, a versão antiga do protocolo não é objeto de discussão. Isto não acarreta em prejuízo no entendimento da proposta, tendo em vista que a mesma é feita do início, não levando em consideração o protocolo original.

Para assegurar maior confiabilidade à proposta, vislumbra-se a adição de duas características-chave. Primeiro, o uso de dispositivos de reconhecimento biométrico para maior precisão na identificação dos usuários. E, segundo, a realização de verificações matemáticas (formais)

afim de confirmar se as garantias de segurança pretendidas são realmente providas.

1.2 JUSTIFICATIVA E MOTIVAÇÃO

A realização deste trabalho justifica-se por ser uma nova tentativa na solução do problema de acesso não-autorizado a sistemas de informação. Acessos desta natureza acarretam enormes prejuízos financeiros (BATCHELOR, 2012). Não raro, os prejuízos sofridos vão além de perdas financeiras, atingindo também bens intangíveis como confiança do consumidor.

Outro fator importante a ser considerado, é a baixa quantidade de pesquisa correlata feita no Brasil, ao menos com relação à temática da verificações formais de protocolos. Durante a etapa de revisão bibliográfica, foram encontrados poucos trabalhos com tema semelhante que tiveram origem em universidades brasileiras. Um dos exemplares é a Dissertação de Mestrado de Fábio Piva (PIVA, 2009).

Logo, talvez a maior motivação para a condução deste estudo seja servir como estímulo ao desenvolvimento de trabalhos com objetos de pesquisa semelhante.

1.3 OBJETIVOS

Definimos o objetivo geral deste trabalho como sendo *a criação de um novo protocolo para autenticação de usuários com mecanismos de garantia da confiança em diversos níveis de execução*.

1.3.1 Objetivos Específicos

(a) Apresentar os protocolos de autenticação biométrica já existentes na literatura;

(b) Estudar e apresentar o estado da arte das técnicas existentes para verificação formal de protocolos de segurança;

(c) Elaborar uma versão que faça uso da certificação digital como único fator de autenticação;

(d) Estender a versão inicial do protocolo, adaptando-a para conter dois fatores de autenticação (certificação digital e biometria);

(e) Elaborar especificações em alto-nível para toda a proposta;

(f) Apresentar notações formais para todas as mensagens trocadas;

(g) Confirmar a viabilidade prática da proposta através da implementação de um protótipo;

(h) Escolher uma técnica e uma ferramenta para verificação do protocolo;

(i) Elaborar uma especificação formal do protocolo a partir de sua especificação em alto-nível;

(j) Testar cenários de ataque sobre as especificações formais criadas, com auxílio de uma ferramenta de automatização; e

(k) Avaliar o comportamento do protocolo aos testes efetuados.

1.4 CONTRIBUIÇÕES

Parte dos resultados alcançados no decorrer deste trabalho foi publicada no formato de artigo, no seguinte veículo de divulgação científica:

1. *"Towards a Formal Verification of a Multi-factor Authentication Protocol Using Automated Theorem Provers"*, em *11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom-2012)* (SANTOS et al., 2012). Neste artigo, apresentamos nossa proposta de protocolo, junto com sua formalização inicial, em Lógica de Primeira Ordem.

1.5 MÉTODO DE PESQUISA

Segundo Wazlawick, um método de pesquisa *"consiste na sequência de passos necessários para demonstrar que um objetivo de pesquisa foi atingido"* (WAZLAWICK, 2009). Neste trabalho, optou-se pela adoção de uma abordagem incremental ao projeto de protocolos de segurança. Esta abordagem, ou seja, nosso método de pesquisa, é composto das seguintes etapas: Concepção Inicial, Prototipação, Formalização, Verificação e Análise dos Resultados.

Destacam-se alguns pontos no método. O principal resultado da etapa de Concepção Inicial são especificações em alto-nível. Estas especificações são, então, validadas contra um cenário de execução real, na etapa de Prototipação. Em seguida, na fase de Formalização, a especificação de alto-nível original é convertida num modelo matemático, sobre o qual testes são realizados (Verificação). Ao fim, os resultados da verificação são analisados sob o ponto de vista da segurança (Análise dos Resultados).

1.6 LIMITAÇÕES DO TRABALHO

Na verificação do protocolo a ser proposto, não são levados em consideração os aspectos fora do fluxo de troca de mensagens. Ou seja, questões como a interação entre os participantes humanos e como estes podem conluir na tentativa de quebra do protocolo não são levadas em consideração.

A análise dessa problemática constitui um problema de pesquisa próprio e não está incluída no trabalho.

1.7 ORGANIZAÇÃO DO TRABALHO

Esta Dissertação de Mestrado está estruturada do seguinte modo.

Capítulo 2: Definições Preliminares

Inicialmente, o Capítulo 2 apresenta os conceitos básicos relacionados ao nosso trabalho. É feita uma revisão sobre os tópicos de Criptografia e Certificação Digital (Seção 2.2); Protocolos de Segurança (Seção 2.3); Biometria (Seção 2.4); e *Smartcards* (Seção 2.5). A leitura deste capítulo pode ser dispensada caso o leitor já possua conhecimento prévio na área da Segurança da Informação.

Capítulo 3: Métodos Formais para Verificação de Protocolos

Dando continuidade às explicações-chave, o Capítulo 3 faz uma revisão geral dos métodos formais existentes para a verificação de protocolos de segurança. O capítulo começa apresentando a notação formal adotada para descrição dos protocolos (Seção 3.2). Logo após, é feita uma revisão dos modelos de ameaça existentes, os quais são importantes na identificação das capacidades de ataque de um usuário mal-intencionado qualquer (Seção 3.3). O cerne do capítulo vem em seguida, com a apresentação dos métodos formais propriamente ditos (Seção 3.4). O capítulo discute a existência de limites ao uso de métodos formais na Seção 3.5. Por fim, alguns exemplos de trabalhos de verificação relacionados são apresentados na Seção 3.6.

Capítulo 4: O Protocolo Proposto

Em seguida, no Capítulo 4 alcançamos o cerne da Dissertação, apresentando nossa proposta de autenticação. Desde o início, descrevemos nossa proposta em alto-nível de abstração (Seção 4.2). Dando continuidade à metodologia de projeto de protocolos adotada, apresentamos alguns pequenos *softwares* criados para servir como prova de conceito da proposta (Seção 4.3). Em adição, também é apresentado um subproduto deste projeto de pesquisa: uma biblioteca para integração do protocolo e dispositivos biométricos (Seção 4.4).

Capítulo 5: A Verificação do Protocolo Proposto

Uma vez tendo sua viabilidade prática confirmada, o Capítulo 5 aborda a verificação formal efetuada em cima de nossa proposta. Começamos por apresentar a ferramenta escolhida na assistência do

processo de verificação (Seção 5.2). Em seguida, damos início à formalização do protocolo, transcrevendo as trocas de mensagens de sua descrição em alto-nível para um modelo lógico (Seção 5.3). Em seguida, formalizamos o comportamento de um atacante hipotético num modelo lógico próprio (Seção 5.4). Com a definição dos modelos lógicos de protocolo e atacante, passamos a investigar suas propriedades com a ajuda do provador SPASS (Seção 5.5). Ainda, nas conclusões do capítulo constam algumas ponderações acerca do processo de verificação de protocolos como um todo (Seção 5.6).

Capítulo 6: Considerações Finais

Finalmente, no Capítulo 6 é feita uma análise dos resultados alcançados no decorrer deste estudo. De modo a dar continuidade à pesquisa, algumas perspectivas de trabalhos futuros são deslumbradas (Seção 6.1).

Anexos

Na forma de anexos, encontram-se disponíveis as descrições dos modelos lógicos criados em notação própria da ferramenta de suporte adotada, bem como o resultado dos testes realizados (Anexo A).

2 DEFINIÇÕES PRELIMINARES

2.1 INTRODUÇÃO

Este capítulo dedica-se à revisão dos conceitos básicos de Segurança em Computação envolvidos no trabalho. Não se tem a pretensão de torná-lo um guia detalhado dos tópicos abordados. Em caso de necessidade, recomenda-se ao leitor fazer uso das referências presentes ao longo do texto.

2.2 CRIPTOGRAFIA E CERTIFICAÇÃO DIGITAL

Desde tempos imemoriais, o homem tem buscado formas de proteção à informação vital a sua sobrevivência. Era necessário garantir que, mesmo em mãos inimigas, seus dados não tivessem serventia. Ou seja, que seu real significado não fosse compreendido por pessoas fora do círculo íntimo do autor da mensagem (informação).

Existem duas formas básicas de contornar este problema. A primeira, mais direto, consiste no puro e simples ocultamento da mensagem. Fundos falsos, pequenos bolsos, orifícios do corpo, escrita com tinta invisível eram (e são) alguns dos meios para esconder informações. Estas técnicas recebem o nome de **esteganografia**. A segunda forma, chamada de **criptografia** age diretamente no conteúdo da mensagem, alterando seu próprio significado. Para isso, troca-se letras de posição, adiciona-se novos símbolos, remove-se outros, etc.

Os propósitos da criptografia podem ser extraídos diretamente de seu nome. Vindo do grego, *"kryptus"* (oculto, escondido) e *"graphos"* (escrita).

A história da criptografia pode ser vista em detalhes no Livro dos Códigos (SINGH, 2010). Neste livro, a história da criptografia é contada, desde suas origens na Idade Antiga até os dias atuais. Descobre-se que ela teve um papel fundamental nos principais conflitos geopolíticos da História. Também são apresentados os esforços de matemáticos na quebra (descoberta) dos textos cifrados.

A revolução digital proporcionada pela Internet levou a mudanças no uso da criptografia. Deixamos de escrever cartas e passamos a trocar *e-mails*. A criptografia então passou a atuar na forma como mensagens são armazenadas no computador: em dados binários (os *bits* e *bytes*). Passou a lidar também com o sigilo da comunicação *online*, principalmente em *sites* de comércio eletrônico.

Deste ponto em diante, assume-se que o leitor tenha conhecimento dos seguintes conceitos: **Criptografia Simétrica**; **Criptografia Assimétrica**; **Função de Resumo Criptográfico**; **Assinatura Digital** e

Certificação Digital. Este conteúdo pode ser obtido em: (STALLINGS, 2002), (SCHNEIER, 1995).

2.3 PROTOCOLOS DE SEGURANÇA

Ross Anderson lembra que protocolos de segurança podem ser simples ou complexos (ANDERSON, 2008). Em **protocolos simples**, poucas mensagens são trocadas entre um número reduzido de agentes. Um protocolo é formado pela troca, de no mínimo, duas mensagens. O ato de passar um cartão num leitor para obter acesso ao escritório é um exemplo de protocolo simples.

Por outro lado, **protocolos complexos** demandam a troca de mensagens complexas entre diversos agentes. As mensagens trocadas contêm grande quantidade de material criptográfico (dados cifrados, dados assinados, *hashes*) que devem ser corretamente validados para avançar ao passo seguinte. O uso de cartões de crédito é um exemplo bastante ilustrativo de protocolo complexo. Possuem, no mínimo quatro participantes (o banco, a bandeira do cartão, o consumidor e o comerciante), trocando informações e validando os dados uns dos outros (limite total disponível, limite parcelado disponível, valor da compra, etc) (BELLA et al., 2006).

Outro ponto é a existência de protocolos públicos e privativos. Um protocolo é considerado público quando possuir especificação aberta, isto é, disponível para qualquer interessado em descobrir como o protocolo funciona. Protocolos privativos, por definição, impedem o acesso à sua especificação, disponibilizando somente as interfaces de comunicação com o usuário final. No caso de protocolos de especificação pública, podemos citar como exemplos os protocolos SSL/TLS (DIERKS; ALLEN, 1999) que provêm sigilo às comunicações realizadas na Internet, além de protocolos de autenticação como o Kerberos (MILLER et al., 1987), amplamente utilizado em sistemas operacionais Windows. A presença de protocolos privativos se faz notar nos produtos de *hardware* e *software* comercializados. O setor bancário também possui um grande número de protocolos internos restritos aos seus próprios funcionários (ANDERSON, 2008).

2.3.1 Propriedades de Segurança

As propriedades dos protocolos de segurança são as garantias que o protocolo busca fornecer aos seus participantes durante sua execução. O alcance destas propriedades é feito através do uso de técnicas criptográficas sobre a mensagem, como a cifragem e decifragem. cálculo da função de resumo criptográfico; e aplicação de assinaturas digitais. Para que estas técnicas realmente venham a contribuir para o atendimento das propriedades esperadas para o protocolo, é necessário que sejam utilizadas de maneira racional. Seu uso indiscriminado e sem planejamento pode ter um efeito inócuo no protocolo, não contribuindo para o efetivo

aprimoramento da segurança.

Na literatura, as expressões "princípios de segurança" e "objetivos de segurança" são comumente utilizadas como sinônimos a "propriedades de segurança", não havendo prejuízo de seu significado.

- **Confidencialidade** refere-se ao provimento de sigilo às mensagens trocadas entre participantes legítimos do protocolo. Isto é, mesmo que as mensagens venham a ser interceptadas, seu conteúdo permanecerá inacessível a terceiros não autorizados. Essa propriedade é obtida através da criptografia (operações de cifragem e decifragem de dados).

- **Integridade** provê mecanismos para detecção de alterações indevidas na estrutura da mensagem. Esta propriedade é de vital importância se considerarmos o cenário no qual uma mensagem é interceptada por um atacante e este a modifica antes de reencaminhá-la ao destinatário original. Outro exemplo importante vem na forma de detecção de alterações em arquivos. Ou seja, o destinatário deve prover de meios para verificar a não-alteração da mensagem a partir de seu alegado emissor. Esta propriedade é alcançada através do uso de funções de resumo criptográfico e assinaturas digitais.

- **Não-Repúdio** visa garantir que após o efetivo envio de uma mensagem, seu emissor não possa negar tê-lo feito. Esta propriedade é de suma importância num cenário em que ambas as partes, mesmo constantemente trocando informação entre si, não confiam uma na outra. Para alcançar esta propriedade, são utilizadas assinaturas digitais.

- **Autenticação** é a propriedade que uma vez cumprida pelo protocolo, permite que seus participantes possam verificar a identidade de seus interlocutores. Existem dois tipos de autenticação: autenticação unilateral e autenticação mútua. Num protocolo de autenticação do tipo unilateral, apenas uma das partes tem sua identidade verificada perante a outra (geralmente o participante que iniciou o protocolo verifica a identidade do receptor de suas mensagens). Por outro lado, num protocolo de autenticação mútua, ambas as partes são autenticadas perante a outra; o participante que iniciou o protocolo autentica o receptor de sua mensagem e, em seguida tem sua identidade verificada pelo último.

  Outro conceito importante é o fator de autenticação utilizado. Um fator de autenticação é a informação utilizada na verificação da identidade do participante. São, em geral, em número de três: *algo que você sabe* (por exemplo, uma senha), *algo que você possui* (por exemplo, um *smartcard*) e *o que você é* (por exemplo, uma impressão digital ou íris). Adicionalmente, em (BRAINARD et al., 2006) é apresentado um quarto fator de autenticação: *alguém que você conhece*.

Certificados e assinaturas digitais são as principais técnicas utilizadas na garantia da propriedade de autenticação.

- **Autorização**, complementando a propriedade de Autenticação, é responsável pela correta atribuição de privilégios e controles de acesso aos recursos disponibilizados pelos participantes do protocolo. Assim sendo, mesmo que um participante tenha sido autenticado perante outro participante, é possível que seu acesso a determinado recurso deste (por exemplo, cadastro de novos usuários), lhe seja negado, uma vez que não foi previamente autorizado a faz. Para que esta propriedade seja eficaz, é comum a definição de diferentes papéis (níveis de responsabilidade) entre os participantes. Desta forma, alguns participantes ficam encarregados da execução de tarefas de gerenciamento, ao passo que outros papéis ocupam-se de tarefas rotineiras, de menor responsabilidade.

A lista de propriedades de segurança certamente não é exaustiva, havendo conflitos em seus significados entre autores, principalmente quando entramos num nível de detalhamento maior. Por exemplo, há autores que consideram que a propriedade de Integridade não se aplica ao participante emissor da mensagem (GOLLMANN, 1996), mas tão somente à mensagem propriamente dita. Neste caso, é definida uma nova propriedade, chamada de Autenticidade, encarregada de garantir que tanto a mensagem quanto seu emissor permaneçam inalterados durante seu trajeto entre dois participantes. Optou-se aqui pelas definições mais próximas a compreensão deste trabalho, abrindo mão de conflitos menores.

2.3.2 Ataques a Protocolos

Num primeiro momento, pode-se pensar que existem ilimitadas formas de ataque a protocolos, porém, de uma maneira geral, a despeito de suas peculiaridades, os ataques a protocolos podem se enquadrar nos seguintes tipos: homem-no-meio, repetição, reflexão e oráculo. Veremos a seguir em detalhes cada um deles.

- **Homem-no-Meio** *(Man-in-the-Middle)*. Ocorre quando um atacante posiciona-se entre dois participantes legítimos do protocolo, interagindo com cada um deles como se fosse o correto interlocutor. Ou seja, cada participante interage com o atacante de tal modo como se este fosse o destinatário real das mensagens. Nesta posição, o atacante pode comprometer a execução do protocolo de diversas formas. Além do simples não reencaminhamento das mensagens ao destinatário, o atacante pode fazer o reencaminhamento de uma versão adulterada da mensagem original, fazendo com que o destinatário responsa de maneira inapropriada.

- **Repetição** *(Replay)*. Consiste na retransmissão, num momento futuro, de mensagens capturadas. Através do monitoramento da rede,

o atacante passa a ser acesso a todas as mensagens trocadas, podendo armazená-las indefinidamente. Uma vez em posse de uma mensagem antiga, o atacante poderá reenviá-la ao seu destinatário original, na tentativa de que este dê continuidade a um novo fluxo do protocolo a partir da mensagem já utilizada. Existem diversas modalidades de ataques de repetição (SYVERSON, 1994), devendo contemplada, ainda, a hipótese do reenvio de mensagens ser feito com o protocolo ainda em execução.

Uma das técnicas existentes para prevenção dos ataques de repetição, é o uso de *nonces*. Um *nonce* (*number used only once*) é um número aleatório gerado ao acaso pelos participantes do protocolo e concatenado às mensagens enviadas.

- **Reflexão** *(Reflection)*. Como o próprio nome diz, uma reflexão de mensagens é um ataque onde o atacante envia uma mensagem qualquer ao participante do protocolo, na esperança deste fornecer-lhe a resposta correta. Uma analogia a este ataque é a seguinte situação: existe um guarda parado na frente de uma porta que, para permitir a entrada, pergunta: "qual é a senha?"; você não sabe qual é a senha e por sua vez, pergunta ao guarda "qual é a senha?"; eis que o guarda fornece-lhe a senha (RYAN; SCHNEIDER, 2000). Ataques de reflexão são comuns em protocolos em que seus participantes autentiquem-se uns aos outros (autenticação mútua).

- **Oráculo** *(Oracle)*. Ocorre quando um agente honesto é induzido pelo atacante a executar algum passo do protocolo, fornecendo inadvertidamente algum dado importante (RYAN; SCHNEIDER, 2000). O atacante pode explorar passos em diferentes instâncias de execução do protocolo ou, ainda, em diferentes protocolos.

- **Entrelaçamento (*interleave*)**. Ocorre quando o atacante sobrepõe duas ou mais instâncias de execução do protocolo (RYAN; SCHNEIDER, 2000). Executando um protocolo múltiplas vezes em paralelo, o atacante pode acumular mais rapidamente conhecimento sobre a execução do protocolo. Também e possível que dois ou mais passos venham a entrar em conflito uns com os outros. Credita-se a Lowe (LOWE, 1996), o primeiro uso bem-sucedido desta técnica de ataque.

- **Ataques algébricos (*algebraic attacks*)**. Algoritmos criptográficos são, em sua essência, operações matemáticas. Este tipo de ataque busca explorar falhas no modo como estas operações são realizadas em *hardware* ou *software*. A ocorrência de falhas nesse sentido pode levar à *identidades algébricas* que anulam o efeito da cifragem de dados. Em matemática, um exemplo de identidade algébrica é $x = x$ ("x é sempre igual a ele mesmo"). Um exemplo de ataque algébrico bem-sucedido consta em (RYAN; SCHNEIDER, 1998).

2.4 BIOMETRIA

Até o século XIX, o único método disponível para a identificação de pessoas para fins criminais ou de emissão de documentos oficiais era o reconhecimento facial *in loco*. Apesar do razoável grau de precisão quando executado por pessoa de confiança (geralmente um agente do governo) e boas condições físicas do indivíduo sendo avaliado, este método não era imune a fatores naturais de descaracterização, como envelhecimento e alterações na pele.

No século XIX, com a descoberta da propriedade de unicidade das impressões digitais, novas portas começaram a se abrir no processo de reconhecimento de indivíduos. Daquele momento em diante, não seria mais necessário confiar no julgamento de outrem, mas efetuar uma comparação entre dois registros: um coletado no momento da verificação e outro já previamente armazenado numa ocorrência anterior. A comparação entre estes dois registros possibilita o reconhecimento com precisão, embora exista margem para a ocorrência de falsos positivos e falsos negativos (ANDERSON, 2008).

Para que uma determinada característica do corpo humano seja também uma característica biométrica de qualidade, certas propriedades devem ser satisfeitas (JAIN et al., 2004):

- *Universalidade*: cada pessoa deve ter sua própria característica;
- *Distinção*: duas pessoas quaisquer devem ter características suficientemente diferentes;
- *Permanência*: a característica não pode se modificar dentro de um determinado período de tempo; e
- *Fácil medida*: a característica deve ser capaz de ser medida quantitativamente.

Como se sabe atualmente, impressões digitais não são as únicas características do corpo humano que possuem, em maior ou menor grau, as propriedades acima. Cabe ressaltar que não apenas características fisiológicas, mas também características comportamentais podem ser utilizadas como fatores biométricos. A Tabela 2.1 enumera alguns fatores biométricos.

Tabela 2.1: Exemplos de características biométricas

Características Fisiológicas	**Características Comportamentais**
DNA	Forma de andar
Impressão digital	Voz
Impressão da palma da mão	Velocidade de digitação (*Keystroke*)
Retina	Assinatura de próprio punho

2.4.1 Sistemas Biométricos

Um sistema biométrico é essencialmente um sistema de reconhecimento de padrões (JAIN et al., 2004). Seu funcionamento começa com a captura do dado biométrico em si, por exemplo, a imagem de uma impressão digital com seus vales e sulcos visíveis. Em seguida, extrai-se os traços biométricos particulares correspondentes. Então, compara-se os traços biométricos extraídos com os traços-modelo (*template*) previamente armazenados em banco de dados. Com relação à comparação de traços biométricos, esta pode ser realizada de dois modos distintos: **Verificação** e **Identificação**. A Tabela 2.2 realiza a comparação entre os modos.

Tabela 2.2: Diferentes modos para comparação dos traços biométricos

	Verificação	**Identificação**
Comparação	1:1	1:N
Reconhecimento	Positivo	Negativo
Pergunta realizada	Este traço biométrico pertence a Alice?	A quem pertence este traço biométrico?

Com relação ao tipo de **reconhecimento**, dizemos que um reconhecimento é **positivo** quando ele busca impedir que diferentes indivíduos façam uso da mesma identidade (WAYMAN, 2001). Por outro lado, um reconhecimento é **negativo** quando busca impedir que um único indivíduo faça uso de diferentes identidades.

2.4.2 Protocolos de Autenticação Multifator

Na literatura, encontramos diversos exemplos de protocolos que fazem uso de mais de um fator para oferecer níveis aprimorados de autenticação a seus participantes. Por exemplo: (BODNAR, 2008; SANYAL et al., 2010; UPMANYU et al., 2010; HARTUNG; BUSCH, 2010; YANG; YANG, 2010a; YANG; YANG, 2010b; MATHEW et al., 2011).

Também descobrimos que alguns esquemas focam no uso do *template* biométrico como entrada na geração de chaves criptográficas mais fortes. Por exemplo: (HAO et al., 2005; KANADE et al., 2010; KANADE et al., 2011; POINTCHEVAL; ZIMMER, 2008; HAO; CLARKE, 2012). Esta abordagem aparenta ser conveniente nos cenários de distribuição de chaves e geração de chaves temporárias de sessão.

2.5 *SMARTCARDS*

Para o entendimento do que são e para que servem os *smartcards*, é preciso primeiro entender o conceito de processador criptográfico. Um **processador criptográfico** é um tipo especial de processador que tem

como única responsabilidade, a execução de um conjunto pré-definido de operações criptográficas. Visando garantir que a execução desta operações seja feita de forma segura, isto é, sem interferências do meio externo, esta classe de processadores também conta com mecanismos avançados de proteção contra invasão física (*physical tamper resistance*). A presença destes mecanismos justifica-se não apenas pelo isolamento às operações criptográficas realizadas, mas também pela proteção ao material criptográfico (chaves e arquivos de configuração) armazenados em memória não-volátil anexa ao coprocessador.

A união do sigilo proporcionado pela criptografia com o uso de processadores resistentes a violações física não é nova, remontando da necessidade de garantir sigilo a comunicações militares, durante a Guerra Fria. Com o passar dos anos, processadores criptográficos passaram a ser os responsáveis por garantir a segurança em redes bancárias; armazenamento das chaves privadas de Autoridades Certificadoras em Infraestruturas de Chaves Públicas (ICPs); controle de produtos vendidos em máquinas de vendas (*self-service*); e controle, por fabricantes, dos acessórios que poderão ser utilizados em seus equipamentos.

Os **smartcards** (ou *cartões inteligentes*) são a forma mais difundida de processadores criptográficos. No final da década de 1960 e durante a década de 1970 diversos pesquisadores propuseram a criação de cartões inteligentes (ANDERSON, 2008). O mais bem-sucedido nesta tarefa (e o qual recebe o crédio pela invenção), foi o inventor francês Roland Moreno.

Inicialmente, foram usados na França no pagamento de pequenos bens ou serviços, como telecomunicações (SHELFER; PROCACCINO, 2002; NATIONAL MUSEUM OF AMERICAN HISTORY, 2012).

A partir do início dos anos 90, passaram a ser utilizados num conjunto ainda maior de aplicações, como no acesso a caixas eletrônicos; utilização de redes de telefonia móvel GSM (através dos *SIM cards*); controle do sinal emitido por operadoras de TV por assinatura (dificultar a ação de piratas); e controle de acesso de pessoas ao seu local de trabalho (em substituição a chaves e cadeados) (ANDERSON, 2008).

A Figura 2.1, extraída de (THINK ALIKE, 2012), apresenta um exemplo de *smartcard*. Pode-se notar, na figura, que o *smarcard* consiste de um *chip* colado sobre uma superfície plástica. Os *smartcards* são padronizados pela norma ISO 7816 (ISO, 2011b).

Figura 2.1: Exemplo de *smartcard*

2.6 CONCLUSÃO

Neste capítulo, vimos os conceitos fundamentais que dão suporte ao nosso trabalho. As tecnologias apresentadas possuem cada qual sua própria história. Não são, portanto, produto exclusivo do conhecimento moderno, mas sim, a evolução de décadas ou, em alguns casos, de séculos de aperfeiçoamento. Sua origem data de períodos onde o computador sequer existia.

Smartcards e biometria constituem, na opinião do autor, os expoentes tecnológicos mais conhecidos pelo grande público. Fazem parte de operações rotineiras como acesso a caixas eletrônicos e identificação pessoal. Por outro lado, a criptografia e a certificação digital, por não serem tecnologias "palpáveis", são de mais difícil assimilação pelo público leigo. Mesmo assim, são ainda mais difundidas e conferem suporte às demais tecnologias de segurança.

Outra questão importante são os protocolos de segurança. As redes de computadores (não apenas a Internet) possibilitaram que usuários trocassem informações (mensagens) entre si, mesmo fisicamente separados. Para chegar de um usuário a outro, a mensagem precisa passar por inúmeros pontos (ou nós) da rede. Em cada um destes pontos, pode haver usuários mal-intencionados (atacantes), buscando fazer uso indevido da mensagem original. Protocolos de segurança buscam impedir que isto aconteça, levando em conta a finalidade das mensagens trocadas (se para autenticação, se para troca de chaves criptográficas, entre outros).

Encerramos, neste ponto, a primeira etapa de nossa revisão bibliográfica. O capítulo seguinte continua com a revisão dos métodos formais existentes para verificação de protocolos de segurança.

3 MÉTODOS FORMAIS PARA VERIFICAÇÃO DE PROTOCOLOS

3.1 INTRODUÇÃO

Este capítulo apresenta os principais conceitos e técnicas e relacionados à verificação de protocolos de segurança. Tal como ocorre com os fundamentos teóricos da Ciência da Computação, a matemática constitui o principal alicerce desses métodos. Em teoria, é possível realizar estas verificações formais manualmente, porém, atualmente dá-se maior ênfase ao uso de ferramentas de automatização. Sempre que possível, daremos exemplos de ferramentas existentes para cada técnica, sem nos aprofundar muito.

Além de atuarem na verificação de protocolos, os métodos formais também são empregados em outras áreas dentro da Computação. Em Arquitetura de Computadores, métodos formais são empregados na verificação de projetos de circuitos de *hardware* (KROPF, 2010), ao passo que, na Engenharia de Software, métodos formais são usados na especificação e teste de *software* (PRESSMAN, 2009). Dependendo do problema tratado, o certas técnicas são preferíveis em detrimento de outras.

Iniciamos nosso estudo dos métodos formais aplicados com a apresentação da notação empregada para representação de protocolos.

3.2 NOTAÇÕES FORMAIS

O primeiro passo a ser adotado na verificação de protocolos é a transcrição de sua especificação de alto-nível para uma notação precisa e formal (matemática). O objetivo é evitar ambiguidades na interpretação das trocas de mensagens envolvidas no protocolo.

Além disso, o uso de notações formais torna mais fácil a obtenção de código-fonte e especificações de protocolos. Algumas ferramentas de verificação utilizam linguagens próprias para representação dos problemas a serem tratados.

$$1. \quad A \quad \rightarrow \quad B \quad : \qquad \{|\{|Dado1, Dado2|\}_{sKr_A}, Dado3|\}_{eKu_B},$$

Figura 3.1: Exemplo de notação formal

A Figura 3.1 apresenta um exemplo da notação formal adotada. Cada linha desta notação representa um único passo na execução do protocolo, o qual é subdividido em três partes.

A primeira parte, indicada através de um número seguido por "." (ponto), representa a posição (primeiro, segundo, etc) do passo no

protocolo. Na segunda parte, indica-se o fluxo de envio da mensagem: do agente emissor (antes da seta) ao agente receptor (após a seta). Cada agente é representado por uma letra maiúscula, sendo proibida a repetição da mesma letra para diferentes tipos de agente. A terceira e última parte, mais complexa, inicia após o sinal de dois-pontos ":" e contempla a mensagem trocada entre os agentes.

Tomando como exemplo o primeiro passo da Figura 3.1, $Dado1$ e $Dado2$ são dados genéricos concatenados. O operador de concatenação utilizado é a vírgula ",". O conjunto de dados concatenados, de qualquer tamanho, recebe o nome de ***token***. Como pode-se observar, o *token* $Dado1, Dado2$ está envolto por chaves grossas, "$\{|$"e "$|\}$". Estas chaves indicam a realização de alguma operação criptográfica sob os dados nela contidos.

Eis uma regra prática para determinação das operações criptográficas: *e* indica *encryption* (cifragem), enquanto que *s* indica *signature* (assinatura digital). E com relação aos tipos de chaves criptográficas, adota-se: *Ku* denota *public key* (chave pública); *Kr* denota *private key* (chave privada); e *K* representa uma chave simétrica. Em alguns casos, *K* pode vir acompanhada de um *nonce*, cujo comportamento é idêntico ao de uma chave simétrica. *Nonces* são representados por N. O identificador do agente dono de chave ou *nonce* também é indicado. A operação criptográfica é representada por um rótulo subscrito, logo após a chave grossa de fechamento ("$|\}$"). Alguns exemplos de operações criptográficas:

- eKu_A – Cifragem com a chave pública do agente A;
- eK_{N_B} – Cifragem com o *nonce* gerado pelo agente B;
- eK_{AC} – Cifragem com a chave simétrica compartilhada entre os agentes A e C; e
- sKr_D – Assinatura digital com a chave privada do agente D.

3.3 MODELOS DE AMEAÇAS

Um modelo de ameaça (modelo de atacante) é responsável pela definição das **capacidades de um atacante** em sua tentativa de desvirtuar um protocolo. Aqui, *capacidades* referem-se às possíveis interações com o fluxo normal de troca de mensagens, bem como o relacionamento com outros atacantes (MARTINA, 2011; NEEDHAM; SCHROEDER, 1978).

Atribui-se a Needham e Schroeder (NEEDHAM; SCHROEDER, 1978), a primeira definição de um modelo de atacante. Nesta primeira concepção, o atacante é dotado das seguintes capacidades:

1. Entrepor-se entre dois agentes em todos os caminhos de comunicação possíveis;
2. Alterar e copiar partes das mensagens trocadas;

3. Repetir mensagens; e
4. Enviar mensagens falsas.

Por outro lado, o atacante também possui a seguinte limitação:

5. O atacante é incapaz de descobrir chaves por busca exaustiva ou análise criptográfica.

As capacidades de um atacante tal como vislumbradas por Needham e Schroeder são, ainda, insuficientes. Sua principal inconsistência reside na não contemplação da possibilidade do atacante realizar escutas ativas (*active eavesdropping*) na rede, ou seja, alterando mensagens antes que estas cheguem ao seu destino. Com esta constatação em mãos, Danny Dolev e Andrew Yao, publicaram um trabalho onde demonstram a viabilidade de ataques nesse sentido (DOLEV; YAO, 1983).

A lista de capacidades do atacante é então extendida, passando a contar com as seguintes ações:

6. Gravar qualquer mensagem sendo transmitida pela rede;
7. Iniciar troca de mensagens diretamente com outros participantes (o atacante também é um agente na rede); e
8. Receber diretamente a resposta da comunicação iniciada com outros agentes.

Este conjunto estendido de capacidades do atacante recebeu o nome de modelo de ameaça Dolev-Yao em homenagem a seus criadores. Mesmo com sua ampla aceitação como padrão no projeto de protocolos, o desenvolvimento de novos modelos de ameaça continua sendo objeto de pesquisa. Isto justifica-se pelo caráter dinâmico das ameaças: novas ameaças podem surgir com o passar do tempo.

Como exemplos de novos modelos de ameaça, cita-se o modelo Multi-Atacante (ARSAC et al., 2011) e o modelo BUG (*Bad, Ugly and Good*, em tradução livre "o mal, o feio e o bom") (BELLA et al., 2006).

No modelo Multi-Atacante, agentes legítimos podem comportar-se como atacantes, conspirando contra seus próprios pares e recusando troca de informações. É adotado um cenário de competição, cujo único objetivo é o ganho pessoal de cada agente.

Já no modelo BUG, são definidos três tipos de agente: aquele que segue as regras (agente Bom), aquele que não segue as regras de jeito nenhum (agente Mal) e aquele sem comportamento preciso, podendo seguir ou desobedecer as regras a critério próprio (agente Feio). Esta distinção de papeis permite a análise de ataques de retaliação, onde a ocorrência de um ataque, necessariamente, implica na ocorrência de um segundo ataque mais forte que o primeiro.

Mudanças no modelo de ameaças podem levar a descoberta de novos ataques a protocolos já bem-estabelecidos. Um exemplo clássico

é o ataque de Lowe sobre o protocolo de Needham-Schroeder para autenticação por chaves públicas (protocolo NSPK) (LOWE, 1995). O modelo de ameaça adotado originalmente por seus autores, não previa que o atacante pudesse gravar as mensagens trocadas para posterior replicação em outra instância de execução do protocolo.

O projeto e a verificação do nosso protocolo segue o modelo de ameaça Dolev-Yao. Na Seção 5.4, é feita a tradução das ações do atacante de sua descrição textual para seu equivalente em fórmulas lógicas.

3.4 MÉTODOS FORMAIS PARA VERIFICAÇÃO DE PROTOCOLOS

Iniciamos aqui o estudo dos métodos formais para verificação de protocolos. Por existirem diferentes abordagens de encarar este processo, é comum sua divisão em três grandes grupos. São eles: Lógicas de Crenças, Exploração de Estados, e Provadores de Teoremas. Cada um destes possui bases teóricas fincadas sobre fundamentos distintos e com propósitos diferentes.

Algumas abordagens dão maior ênfase à confirmação de corretude, isto é, determinar se o protocolo de fato cumpre com seus objetivos. Outros, dão maior ênfase à descoberta de ataques ao protocolo.

Esta seção tem o propósito de apresentar em linhas gerais, as principais características das diferentes abordagens existentes. Para complementação da leitura, recomenda-se consultar as respectivas referências. A Tese de Martina (MARTINA, 2011) complementa esta revisão em alguns pontos.

3.4.1 Lógica de Crenças

Introduzida em 1990 por Burrows, Abadi e Needham, as Lógicas de Crenças foram a primeira tentativa de conceder tons formais à problemática da verificação de protocolos (BURROWS et al., 1990). Em especial, de protocolos de autenticação. Este formalismo ficou conhecido como *"Lógica BAN",* em referência às iniciais dos sobrenomes de seus autores.

Conforme enfatizado por seus autores, o principal objetivo deste formalismo não é demonstrar a resistência do protocolo a ataques, mas sim, comprovar que ele alcança os objetivos para os quais foi construído (corretude).

Para tanto, o formalismo requer que as crenças (conhecimento assumido como verdade) pelos agentes envolvidos sejam expressos através de **construtores**. Estes construtores representam as informações relativas a troca de mensagens entre agentes, bem como o compartilhamento de chaves e segredos. Em seguida, alguns **postulados lógicos** são definidos. Estes postulados são aplicados sobre as crenças para a obtenção de provas.

Em seu trabalho original, os autores aplicaram este formalismo na verificação da propriedade de autenticação de quatro protocolos, dentre os quais, o *Kerberos* (MILLER et al., 1987) e o protocolo NSPK (NEEDHAM; SCHROEDER, 1978).

Há críticas à Lógica BAN. Por considerar todos os agentes como honestos, não prevendo a existência de intrusos, seu escopo de atuação tende a ser limitado com relação a certas propriedades (GLIGOR et al., 1991). Mesmo assim, provou ser um formalismo eficiente na verificação de protocolos de troca de chaves e autenticação mútua (ANDERSON, 1997).

Diversas extensões foram propostas de modo a aumentar sua expressividade. Como por exemplo, (KESSLER; WEDEL, 1994). A ausência de ferramentas de apoio foi solucionada por Schumann (SCHUMANN, 1997), que traduziu a Lógica BAN para fórmulas em Lógica de Primeira Ordem, sendo em seguida, processadas por seu provador automático de teoremas.

3.4.2 Exploração de Estados

As técnicas que fazem parte deste grupo buscam a verificação do protocolo até exaurir todas as possibilidades. Isto é, percorrem todos os possíveis estados de execução do protocolo em busca de algum caminho que invalide suas propriedades iniciais.

A principal desvantagem a esta abordagem é o seu grande consumo de recursos, podendo levar à explosão de estados (CLARKE, 2008). Em se tratando de protocolos de segurança, é necessário não apenas percorrer todos os caminhos possíveis, mas também lidar com todos os possíveis conjuntos de conhecimento e interações entre agentes.

No entanto, este grupo é o que possui maior número de ferramentas de suporte à disposição. O uso destas ferramentas permite a condução de verificações com alto nível de automatização.

A exploração de estados pode ser subdividida em duas correntes principais: a verificação de modelos (*model checking*) e o método de espaços de fitas (*strand spaces model*).

3.4.2.1 Verificação de Modelos

A verificação de modelos, é, em tradução livre, *"uma técnica que se baseia na construção de um modelo finito para um sistema e consequente verificação das propriedades contidas no modelo criado"* (CLARKE; WING, 1996).

A principal dificuldade nesta abordagem está na criação de algoritmos e estruturas de dados robustos o suficiente para suportar grandes espaços de busca(CLARKE; WING, 1996). Por outro lado, sua principal vantagem reside em seu alto grau de automatização e rapidez se comparada a outras abordagens.

O uso pioneiro da verificação de modelos aplicado a protocolos de segurança é creditado a Gavin Lowe (LOWE, 1995) em seu ataque ao protocolo NSPK. Para tanto, Lowe fez uso combinado da linguagem para descrição de padrões CSP (*Communicating Sequential Processes*) (HOARE, 1985) e do verificador de modelos de propósito geral FDR (*Failures Divergences Refinement*) (FORMAL METHODS (EUROPE) LTD.; OXFORD UNIVERSITY, 2010).

O ataque bem-sucedido de Lowe serviu como impulso à comunidade científica no desenvolvimento de ferramentas de verificação.

Neste contexto, destacamos o projeto AVISPA (*Automatic Validation of Internet Security Protocols and Applications*) (AVISPA, 2012). Este projeto foca no desenvolvimento de uma ferramenta para validação automática de protocolos.

O funcionamento interno do AVISPA se dá pela integração entre quatro diferentes verificadores de modelos. A saber: *On-the-Fly Model Checker* (BASIN et al., 2003), o *Constraint-Logic-Based Attack Searcher* (CHEVALIER; VIGNERON, 2002), o *SAT-based Model Checker* (ARMANDO; COMPAGNA, 2007) e o *Tree Automata-based Protocol Analyser* (BOICHUT et al., 2005). Cada um destes verificadores pode ser executado isoladamente, porém são integrados no AVISPA.

Além disso, este projeto também conta com uma linguagem de especificação de protocolos de alto-nível (AVISPA, 2006)

Cabe ressaltar que apesar de seu elevado grau de automatização, verificadores de modelos são usados na detecção de *bugs*. A comprovação de corretude não é usualmente tratada por esta abordagem de verificação.

3.4.2.2 Modelo de Espaços de Fitas

Verificadores de modelos focam na detecção de *bugs* em protocolos. Motivados pelo problema da comprovação da corretude, Fábrega, Herzog e Guttman introduzem o Modelo de Espaços de Fitas (*Strand Space Model*) (FABREGA et al., 1998). A tradução do nome para o português segue a mesma adotada por Piva (PIVA, 2009).

Uma **fita** (*strand*) é uma sequência de eventos em que um participante pode se envolver. Representam tanto as ações de partes legítimas quanto de intrusos.

Em participantes legítimos, suas fitas representam as mensagens enviadas e recebidas (mas não seu interlocutor).

Já com relação a intrusos, suas fitas representam ações comuns a um atacante Dolev-Yao, como: receber duas mensagens e enviar a concatenação de ambas; receber uma chave simétrica e uma mensagem cifrada com esta chave e então, enviar a mensagem decifrada, por exemplo. Um **espaço de fitas** (*strand space*), é o nome dado ao conjunto formado pela união das fitas de participantes legítimos e intrusos.

Outro conceito importante é o de **feixe** (*bundle*). Um feixe é um subconjunto do espaço de fitas. Neste subconjunto, cada fita de envio de

mensagem encontra-se ligada (enganchada) a outra fita de recebimento – relação de **precedência causal**.

Quando feixes são estabelecidos para cada fita de cada participante, considera-se o protocolo como **correto**. De fato, é necessário que *nonces*, chaves de sessão e outros dados contidos nas fitas, tenham sido acordados entre os participantes. Fitas de intrusos também podem fazer parte dos feixes, não impedindo que os demais participantes estabeleçam uns com os outros, os valores dos dados trocados.

As definições matemáticas por trás desta abordagem podem ser encontradas no trabalho original (FABREGA et al., 1998). Extensões a este método também têm sido propostas: uma delas permite a comprovação da corretude para grupos de protocolos em execução concorrente num mesmo ambiente (FABREGA et al., 1999).

As provas nesta abordagem foram originalmente idealizadas para serem feitas com papel e caneta. Mas atualmente já existem ferramentas de automatização. Como exemplo, temos o Athena (SONG et al., 2001). Esta ferramenta faz uso de um modelo de espaços de fitas estendido combinando as técnicas da verificação de modelos e provadores de teoremas. Além de ser totalmente automática, permite a verificação de protocolos com número arbitrário de execuções concorrentes.

3.4.3 Prova de Teoremas

A prova de teoremas é a abordagem adotada para verificação do protocolo proposto neste trabalho. Para o seu entendimento, é interessante fazermos uma breve revisão a respeito da Lógica como ciência.

Descoberta por Aristóteles na Grécia Antiga, a principal finalidade da **Lógica** é o estudo dos argumentos e de sua estrutura (COOK, 2009). Argumentos referem-se a afirmações em linguagem natural (Português). Com a aplicação de regras de raciocínio, podemos extrair conclusões a partir dos argumentos apresentados.

Um exemplo prático é apresentado em (HUTH; RYAN, 2004) (em tradução livre):

Está chovendo.	:	P
Jane está com seu guarda-chuva.	:	Q
Jane ficará molhada.	:	R

O seguinte raciocínio:

Se "Estiver chovendo"e "Jane não está com seu guarda-chuva", então "Jane ficará molhada": "Jane não está molhada". Mas sim, "está chovendo". Portanto, "Jane está com seu guarda-chuva".

Substituindo as afirmações pelas letras correspondentes, podemos reduzir o raciocínio acima para um formato mais compacto. Ficando:

Se P e não Q, então R. Não R. P. Portanto Q.

Substituindo as palavras que ficaram de fora por símbolos matemáticos, obtemos:

$$(P \wedge \neg Q) \rightarrow R, \neg R, P \vdash Q$$

Este é um exemplo de raciocínio cuja comprovação (verdade) é facilmente identificado pelo senso comum. Entretanto, o mesmo pode não ser o caso para argumentos mais complexos. A Lógica conta com um conjunto de regras bem-estabelecidas para comprovar a veracidade de argumentos de qualquer tamanho ou complexidade. Este conjunto de regras recebe o nome de **regras de inferência**.

Na argumentação acima, as letras P, Q e R são chamadas de **letras sentenciais** e representam, cada uma delas, argumentos simples. Os símbolos $\neg$ – "não", $\wedge$ – "e", $\vee$ – "ou", $\rightarrow$ – "implica", são chamados de *conectivos*. A principal tarefa dos conectivos é a formação de argumentos complexos com maior **expressividade** a partir de argumentos simples. O uso dos conectivos altera o valor dos argumentos simples. Existem outros conectivos mas não serão usados neste trabalho.

Cada um dos argumentos $((P \wedge \neg Q) \rightarrow R)$, $\neg R$, P e Q é chamado de **fórmula**. Fórmulas podem assumir um dos seguintes valores verdadeiro (V) ou falso (F). As fórmulas presentes antes do símbolo $\vdash$ são consideradas, por definição, como verdadeiras e são chamadas de **premissas** (ou ainda, **axiomas**).

As fórmulas após o símbolo $\vdash$ indicam a **conclusão** a que queremos chegar a partir das premissas. Conclusões que ainda não sabemos se são alcançáveis ou não a partir das premissas, recebem o nome de **conjecturas**. Quando é possível chegar a conclusão a partir das premissas, dizemos que *uma prova é encontrada*. O símbolo $\nvdash$ indica que não é possível chegar à conclusão a partir das premissas (a conjectura não se transformou em conclusão).

Ao sistema lógico cujo raciocínio é feito com base em afirmações simples, através do uso de letras sentenciais e conectivos (nos moldes acima), é dado o nome de **Lógica Proposicional**.

Embora permita a expressão de raciocínios complexos, a Lógica Proposicional ainda é insuficiente para tratar o problema da verificação de protocolos. É necessário recorrer a outros sistemas lógicos, que estendam as capacidades do raciocínio sobre afirmações. Todavia, uma das principais aplicações da Lógica Proposicional é a verificação de *hardware* (PAULSON, 2006).

Ao nosso propósito de verificação de protocolos, os sistemas lógicos mais importantes são: a Lógica de Primeira Ordem (*First- Order Logic – FOL*) e a Lógica de Ordem-Mais-Alta (*Higher-Order Logic – HOL*).

O processo de encontrar provas a partir das especificações lógicas pode ser conduzido com papel e caneta. Porém, ferramentas chamadas de **provadores de teoremas** permitem que este procedimento seja realizado também por computadores.

Existem dois tipos básicos de provadores de teoremas: os automáticos e os interativos. **Provadores automáticos** não requerem intervenção humana para obtenção das provas; sendo capazes de obtê-las sozinhos. Por outro lado, **provadores interativos** necessitam de direcionamento do usuário para que as provas sejam encontradas.

Veremos, agora, os sistemas lógicos mais importantes para a verificação de protocolos. Também apontaremos para alguns exemplos de provadores de teoremas.

3.4.3.1 Lógica de Primeira Ordem

A Lógica de Primeira Ordem, também conhecida como *lógica de predicados* ou *lógica quantificacional* estende a Lógica Proposicional. Neste sistema lógico, afirmações são feitas sobre elementos do universo de discurso, um conjunto não-vazio (PAULSON, 2006).

Este sistema busca complementar a semântica da Lógica Proposicional, cuja principal característica é o raciocínio sobre afirmações simples (HUTH; RYAN, 2004).

Um exemplo de argumentos em FOL (extraído de (HUTH; RYAN, 2004), em tradução livre):

$$\begin{array}{lcl} F(x,y) & : & x \text{ é um filho de } y \\ I(x,y) & : & x \text{ é um irmão de } y \\ p(x) & : & \text{retorna o pai de } x \end{array}$$

E a seguinte fórmula, obtida pela combinação dos símbolos acima:

$$\forall x[F(x, p(m) \rightarrow I(x, m))]$$

Que é interpretada como:

"Para todo x, se x é filho do pai de m, então x é irmão de m."

A Lógica de Primeira Ordem possui diversos elementos que não fazem parte da Lógica Proposicional.

Predicados são as afirmações que fazemos sobre os elementos do domínio do discurso. Têm comportamento semelhante às letras sentenciais, com a distinção de que recebem parâmetros. Tal como as letras, predicados recebem valores de veracidade (verdadeiro ou falso). No exemplo acima, $F(x, y)$ e $I(x, y)$ são predicados que recebem dois parâmetros cada um. Predicados são definidos com letra maiúscula ($Sabe(x, y, z)$, $A(x)$, ...).

Variáveis *"são expressões lógicas que não recebem valores fixos, mas que podem ser usadas para estender todo o universo de discurso adotado"* (em tradução livre) (COOK, 2009). No exemplo acima, x é uma variável. Não deve-se confundir com os parâmetros de predicados e funções ($P(x, y)$) que, neste caso, indicam parâmetros

e não variáveis em si. Variáveis são definidas com letras minúsculas do final do alfabeto (x, y, z, ...).

Quantificadores determinam a abrangência das variáveis aos elementos do universo de discurso. Em FOL, existem dois quantificadores: o quantificador universal – $\forall$ ("para todo") e o quantificador existencial – $\exists$ ("existe"). O primeiro indica que todas os elementos do discurso devem ser considerados para que o predicado seja verdadeiro. Já no segundo, é necessário que exista no mínimo um único elemento. É fácil verificar a veracidade de nosso exemplo: como todas as pessoas têm obrigatoriamente um pai e se duas destas pessoas são filhas de um mesmo pai, então, elas obrigatoriamente são irmãs uma da outra.

Funções são entidades matemáticas que, a partir de elementos concretos do domínio de discurso (seus argumentos), retornam um elemento concreto novo (a imagem da função) (COOK, 2009). Em nosso exemplo, $p(m)$ é uma função que retorna o pai do elemento concreto m. É possível, ainda, usar funções de funções: $p(p(m))$ (retorna o pai do pai de m – o avô de m). Funções são definidas com letras minúsculas a partir do meio do alfabeto (f, g, h, ...).

Constantes são funções com zero parâmetros (PAULSON, 2006). Representam elementos concretos do domínio de discurso. No exemplo acima, m é uma constante.

As demais definições da Lógica Proposicional também são válidas na Lógica de Primeira Ordem. Há, porém, uma diferença fundamental entre os dois sistemas no que diz respeito à obtenção de provas. Em FOL, não é possível predizer se um determinado problema lógico (axiomas e conjecturas) retornará um resultado. Neste caso, qualquer resultado: tanto a confirmação, quanto a refutação da conjectura. A este fenômeno (incerteza quanto ao recebimento de uma resposta) dá-se o nome de **indecidibilidade** (HUTH; RYAN, 2004).

O fenômeno da indecidibilidade é facilmente perceptível com o uso de provadores automáticos de teoremas. A execução de um problema lógico indecidível nunca termina, roda indefinidamente.

Em adição, ao nosso estudo, cabe destacar uma classe especial de fórmulas. Fórmulas do tipo: $P(x) \wedge Q(x) \rightarrow P(x)$ são chamadas de **Fórmulas Horn**. A aplicação das regras de inferência faz com que estas fórmulas sejam sempre *verdadeiras*, independente dos valores atribuídos a $P(x)$ e $Q(x)$. Esta classe de fórmulas é de especial importância na determinação da **satisfazibilidade**. Isto é, se existe ao menos uma interpretação em que a fórmula resulte em *verdadeiro*. Aqui, interpretação refere-se aos valores individuais de cada predicado (por exemplo, $P(x)$ =*verdadeiro*, e $Q(x)$ =*falso*. Um exemplo de fórmula não-satisfazível é $P(x) \wedge \neg P(x)$ (ou, "não é possível que uma afirmação $P(x)$ seja verdadeira e falsa ao mesmo tempo").

Como exemplos de provadores de teoremas em FOL, temos o SPASS (WEIDENBACH et al., 2009) e o ProverIf (BLANCHET, 2001). O primeiro é usado como suporte na verificação do protocolo proposto. Seu funcionamento é apresentado na Seção 5.2.

No Capítulo 5, vemos em detalhes como se dá a tradução de um protocolo criptográfico para fórmulas em Lógica de Primeira Ordem.

3.4.3.2 Lógica de Ordem Mais Alta

Além da Lógica de Primeira Ordem, outro sistema lógico bastante utilizado na verificação de protocolos é a Lógica de Ordem-Mais-Alta. Este sistema lógico estende, ainda mais, a expressividade da lógica de predicados.

A principal diferença entre os dois sistemas é a extensão ao uso de quantificadores. Em HOL, pode haver quantificação não apenas de variáveis, mas também de predicados, funções e conjuntos (BELLA, 2007). Por exemplo, em FOL, temos::

$$\exists x A(x) \wedge B(x) \quad : \quad x \textit{ é, ao mesmo tempo, } A \textit{ e } B$$

Em HOL, também é possível (COOK, 2009):

$$\exists X\, X(a) \wedge X(b)$$

Que é lido, como:

Dois elementos concretos a e b têm algo X em comum.

Como exemplo de provadores de teoremas em HOL, temos o Isabelle/HOL (NIPKOW et al., 2002) e Coq (INRIA, 2012). Ao contrário dos provadores em FOL, provadores em HOL são interativos e não automáticos – requerem assistência humana na geração das provas.

O provador Isabelle, combinado com o Método Indutivo de Paulson (PAULSON, 1998), tem sido de especial importância na verificação de protocolos de segurança. Alguns exemplos: verificação do protocolo de compras do SET (*Secure Electronic Transaction*) – o SET é um conjunto de protocolos voltado ao comércio eletrônico (BELLA et al., 2003); e a verificação do uso de *smartcards* em protocolos (BELLA, 2003).

Devido a sua curva de aprendizado íngreme, o uso do Isabelle na verificação do protocolo proposto não é abordado neste trabalho.

3.5 LIMITES AO USO DOS MÉTODOS FORMAIS

Métodos formais não são a resposta definitiva à verificação de protocolos. Não levam em consideração o meio sobre o qual o protocolo se encontra, nem suposições que vão além da descrição do protocolo em si.

No primeiro caso, o atacante pode estar monitorando, além do tráfego na rede, as emissões eletromagnéticas de dispositivos e as

flutuações de tensão (RYAN; SCHNEIDER, 2000). O modelo do atacante adotado deve de fato refletir o ambiente sobre o qual o protocolo é implantado.

Para o segundo caso, assumimos que a chave privada não está disponível a alguém que possa utilizá-la de maneira não-autorizada (ANDERSON, 2008). Contudo, nem sempre isso é uma verdade de fato. Mesmo o uso de dispositivos de proteção a intrusão (*tamper resistance*) em nível de *hardware* (exemplo: *smartcards*) pode vir a ser inócuo se seu *software* de comunicação contiver *bugs*. Nestes casos, as suposições feitas são apenas relativas, nunca constantes.

Ryan e Schneider (RYAN; SCHNEIDER, 2000) notam que mesmo quando protocolos são considerados formalmente seguros, a existência de *bugs* de implementação podem, ainda assim, comprometer sua segurança. Em (PAULSON, 1997), Paulson analisa formalmente um protocolo de autenticação, concluindo que a troca de chaves é realizada de modo seguro. Porém, mais tarde, verifica-se que a forma de implementação da operação de cifragem abre caminho para um ataque algébrico (RYAN; SCHNEIDER, 1998).

Estes limites da análise formal levaram pesquisadores a deslumbrar princípios gerais de construção de protocolos. Aqui, o propósito é não mais buscar garantias de segurança após o protocolo ter sido projetado (e em alguns casos, implantado), mas sim, em projetá-lo de modo robusto desde seu início. Estes princípios recebem o nome de **princípios de robustez**.

Retirado de (ABADI; NEEDHAM, 1996), alguns exemplos de princípios de robustez são: *"se a identidade de um agente é essencial ao significado de uma mensagem, é essencial que seu nome esteja explícito na mensagem"*; *"cifragem não é sinônimo de segurança e seu uso sem critério pode levar a redundância e, em consequência, a erros"*; e, *"nonces que são contadores previsíveis devem ser transmitidos sempre cifrados"*. Em (ANDERSON; NEEDHAM, 1995) esta discussão é ampliada com princípios para protocolos de chaves pública.

3.6 VERIFICAÇÕES FORMAIS RELACIONADAS

Como visto na Seção 2.4.2, o uso de biometria em protocolos de segurança não é algo novo. No entanto, não foi possível encontrar trabalhos de verificação formal em exemplares desta classe de protocolos. De fato, existem alguns exemplos de verificações, porém estes não fazem uso dos métodos matemáticos aqui apresentados. Por exemplo, (YANG; YANG, 2010b) e (HAO; CLARKE, 2012).

O trabalho mais próximo de nossos objetivos de verificação foi realizado por Jan Jurjens ((JURJENS, 2005b; JURJENS, 2005a)). Nestes trabalhos, seu autor usou um protocolo biométrico como estudo de caso na avaliação de sua abordagem para verificação de corretude de propriedades de segurança. A verificação é feita não sobre o fluxo de troca de mensa-

gens do protocolo, mas sim, sobre código C e diagramas UML. Apesar dos resultados sólidos encontrados, não há maneira de afirmar se estes também são válidos no alto-nível do protocolo.

3.7 CONCLUSÃO

Neste capítulo, apresentamos os elementos teóricos que dão suporte a este estudo. Foram abordados não apenas os métodos formais em si, mas também aspectos preliminares, como a notação formal adotada e o modelo de ameaças escolhido. Sem que estes pré-requisitos sejam levados em consideração, todo o processo de verificação pode falhar.

No primeiro caso, notações inconsistentes ou ambíguas aumentam o risco de erro de interpretação por parte dos analistas. No segundo caso, a definição incorreta do modelo de ameaça leva a suposições irrealistas dos meios que um atacante pode se valer para atacar o protocolo.

Com relação aos métodos formais apresentados, enfatizamos que cada um deles tem suas vantagens e desvantagens. Para uma escolha acertada, é necessário definir o propósito da verificação: se provar corretude ou descobrir ataques.

O uso de ferramentas automáticas na busca por ataques é algo positivo. Entretanto, para que os ataques encontrados sejam de fato factíveis, deve-se confrontá-los com o ambiente de implantação. Esta cautela busca evitar a ocorrência de falsos positivos (ataques teóricos, mas que na prática, não representam ameaça).

Ainda, é necessário certificar-se que se possui o conhecimento (e experiência) necessário ao manuseio das ferramentas. É interessante destacar também o estudo da ferramenta. Em alguns casos, sua curva de aprendizado tende a ser áspera.

Também foi visto que, apesar dos benefícios obtidos com os métodos de verificação, existem limites a seu uso. Mesmo protocolos formalmente analisados podem vir a apresentar problemas. A necessidade de prover segurança aos protocolos não termina com a verificação formal destes. Como foi visto neste capítulo, há diversos fatores que, por sua difícil (ou impossível) formalização, não são levados em consideração. Esta etapa é somente uma dentre as muitas etapas existentes na direção de protocolos seguros e confiáveis.

Com o término da revisão bibliográfica, nos encaminhamos agora ao desenvolvimento.

4 O PROTOCOLO PROPOSTO

4.1 INTRODUÇÃO

Deste momento em diante, apresentamos a parte central de nosso trabalho, ilustrando suas principais contribuições. Para seu bom entendimento, faz-se necessário que o leitor tenha sido capaz de assimilar os conceitos vistos nos Capítulos 2 e 3.

Por ter como alvo o reconhecimento da identidade de usuários, nosso protocolo é considerado como sendo um protocolo de autenticação. O correto provimento desta propriedade de segurança foi o principal requisito que delineou seu projeto. Outras propriedades de segurança, (por exemplo: privacidade), não foram levadas em consideração durante o desenvolvimento desta proposta.

Como principais características de nosso protocolo, destacamos a presença de diferentes papeis de usuário, visando permitir alto grau de gerenciamento. Além disso, busca-se oferecer diferentes níveis de força ao serviço de autenticação. Desta forma, são elaboradas duas versões do mesmo protocolo: uma versão com e outra versão sem o uso de dispositivos de reconhecimento biométrico. Apesar de ter sido desenvolvido tendo em mente impressões digitais, o protocolo pode ser facilmente estendido de modo a englobar outros esquemas de identificação biométrica, como a íris.

4.2 DESCRIÇÃO EM ALTO-NÍVEL

Inicialmente definimos as categorias de agentes envolvidas. **Agentes** referem-se não apenas aos usuários humanos que interagem com o protocolo, mas também a outros sistemas que nele participam (por exemplo, o banco de dados). Como sinônimo a *agentes*, temos: *entidades* e *atores*.

A descrição de cada agente e sua respectiva função é definida como segue:

1. **Tomadores de Decisão**: são os participantes responsáveis pela tomada de decisões gerenciais em alto nível da organização. Em nossa concepção, deve haver mais de um usuário nesta categoria. Esta medida busca garantir que no âmbito do protocolo, as decisões sejam tomadas em conjunto com os demais colegas. Não há necessidade dos representantes deste grupo possuírem conhecimento técnico em autenticação de usuários;

2. **Administradores**: são os participantes responsáveis pela operação dos serviços de Tecnologia da Informação. Novamente, é recomendável que haja mais de um usuário presente neste grupo.

Sendo este grupo formado por um único indivíduo, há tendência de acumulação de poderes, abrindo portas para corrupção. Ao contrário do grupo anterior, Administradores devem, sim, possuir conhecimento em matéria de autenticação;

3. **Clientes**: são os usuários ordinários do sistema e aos quais, em última análise, se destinam os mecanismos de autenticação presentes no protocolo. Participam de modo ativo do processo de cadastro, não se limitando ao simples fornecimento de informações pessoais. Sua autenticação no protocolo é feita de maneira autônoma, sem a necessidade de intermédio de um Administrador;

4. **Servidor**: é a porta de entrada do Cliente durante sua autenticação. Nas etapas de cadastro, atua como redirecionador (*proxy*) das mensagens trocadas entre todos os agentes, preservando suas localizações. Não é necessário que seja o mesmo equipamento (*host*) para o cadastro e autenticação; e

5. **Banco de Dados**: é responsável pelo armazenamento a longo prazo dos dados dos Clientes cadastrados. Aceita solicitações feitas somente através do agente Servidor.

Não existe hierarquia de agentes em nosso protocolo. Caso um Administrador queira autenticar-se no protocolo, o mesmo terá que estar previamente cadastrado como um Cliente. O mesmo vale para agentes Tomadores de Decisão. Existe uma clara divisão de responsabilidades entre os poderes e deveres de cada agente no protocolo.

O papel de cada agente será melhor visualizado com a descrição da troca de mensagens. Antes, porém, definimos algumas premissas básicas sobre o funcionamento do protocolo como um todo.

4.2.1 Premissas Iniciais

As premissas envolvidas no funcionamento de nosso protocolo são: distribuição da confiança e ancoramento de confiança. Veremos a seguir cada uma delas.

4.2.1.1 Distribuição da Confiança

Uma das premissas básicas que atuam como pilares de nossa proposta é que nosso protocolo confia em, não apenas uma, mas em várias chaves privadas.

De um ponto de vista genérico, esse sempre foi o procedimento usual para protocolos. Se um protocolo faz uso de chaves públicas e privadas, este deve confiar em todos as chaves de seus agentes legítimos. Afinal, não faz sentido discriminarmos chaves entre agentes legítimos. Protocolos clássicos e recentes, como o NSPK (NEEDHAM; SCHROEDER, 1978) e SSL/TLS (DIERKS; ALLEN, 1999), compactuam desta ideia.

Em nosso protocolo, a distribuição da confiança reside nas garantias fornecidas aos dados de identificação do Cliente uma vez concluído seu registro. Esses dados são assinados digitalmente duas vezes: a primeira, pela chave privada do próprio Cliente; e a segunda, pela chave privada do Administrador responsável pelo cadastro. Assim, para que esses dados sejam comprometidos, é necessário que um atacante tenha acesso simultâneo às chaves privadas de ambos os agentes.

Esta premissa é melhor visualizada nas notações formais presentes no decorrer deste capítulo.

4.2.1.2 Ancoramento da Confiança

De forma a garantir que o sistema aceite somente certificados digitais emitidos por uma determinada ICP (Infraestrutura de Chaves Públicas), recomenda-se que esta definição seja embarcada a nível de código-fonte. Não deve ser possível alterá-la através de interface gráfica ou manipulação de arquivos de configuração. Desta maneira, um atacante torna-se incapaz de modificar esta diretiva pela simples manipulação de parâmetros ou injeção de códigos maliciosos. Para ser bem sucedido, o atacante teria que obter o código-fonte do sistema, realizar as alterações, recompilá-lo e executar o sistema modificado novamente. Apesar de não ser zero, o risco da ocorrência deste cenário é mitigado pela adoção de políticas de controle de acesso e registro de eventos (*logs*).

A restrição dos certificados digitais pode ser feita através do identificador Nome Distinto (*Distinguished Name – DN*) contido no certificado digital ou, através da utilização de extensões de certificado específicas. Em ambas as alternativas, deve-se também levar em conta a âncora de confiança do certificado digital a ser aceito. No âmbito da ICP-Brasil, temos a presença dos certificados digitais *e-CNPJ* que podem ser utilizados neste contexto. Em ambos os casos, a restrição deve ser feita de forma a aceitar somente os certificados digitais emitidos para a entidade responsável pela implantação do protocolo em ambiente de produção.

No momento da instalação do protocolo, a chave privada correspondente a um certificado digital aceito, cria e assina o anel de certificados dos agentes Tomadores de Decisão. Esse procedimento é feito pela entidade responsável pela implantação do protocolo em ambiente final, geralmente uma empresa terceirizada. Não é necessário que todos os dados contidos nos certificados digitais de cada agente estejam presentes neste anel, ao invés disso, pode-se incluir o Nome Distinto ou o valor de alguma extensão de certificado comum a todos os agentes. Os agentes desta categoria são, em última instância, os responsáveis pela operação do protocolo. Portanto, devem ser os primeiros a serem incluídos no protocolo.

Em seguida, procede-se de maneira semelhante com a criação do anel de Administradores. Ao contrário do que ocorre com a criação do anel dos Tomadores de Decisão, o anel dos Administradores é criado por

um representante do anel de Tomadores de Decisão. Cada elemento do anel de Tomadores de Decisão pode adicionar Administradores ao anel de Administradores. E por último, cada elemento do anel de Administradores, passa também a ser capaz de adicionar usuários (Clientes) ao protocolo.

Este processo de ancoramento da confiança é ilustrado na Figura 4.1.

Passamos a descrever o protocolo em termos da troca de mensagens que ocorre entre agentes.

4.2.2 Cadastro de Usuários

O projeto do protocolo de cadastro apresenta três versões. A primeira versão, denominada **Prova de posse**, busca verificar se o usuário sendo cadastrado tem a posse de sua chave privada. A segunda versão, chamada de **Cadastro com Dois Fatores**, complementa a prova de posse com a adição dos fatores biométricos do usuário. Por sua vez, a terceira versão, conhecida como **Cadastro simplificado**, apresenta uma versão reduzida da prova de posse, sem a necessidade de interagir com o Cliente. O cadastro simplificado não permite o uso de fatores biométricos.

4.2.2.1 Prova de Posse

Em primeiro lugar, precisamos definir os componentes individuais (*tokens*) que fazem parte de cada mensagem do protocolo. Essa definição se mantém intacta no decorrer do capítulo, com alguns acréscimos sendo feitos à medida que aparecerem pela primeira vez. As mensagens trocadas em nosso protocolo podem ser formadas pelos seguintes dados:

- A, S, C e B: representam os agentes participantes do protocolo: Administrador, Servidor, Cliente e Banco de Dados, respectivamente. É comum sua utilização como *identificadores de roteamento* nas mensagens enviadas ao Servidor. Nesse caso, esses identificadores são usados pelo Servidor para determinar qual agente é o destinatário da mensagem;

- CPF_C: indica o número do CPF (Cadastro de Pessoa Física) do Cliente. Esse número é responsável pela identificação individual de pessoas no Brasil. O CPF é formado por uma sequência de 11 dígitos separados por pontos e um traço, por exemplo "123.456.789-10"(RECEITA FEDERAL, 2012a);

- Cod_C: indica o código de usuário (*login*) do Cliente. Esse código de usuário serve para identificação do Cliente no sistema em que o protocolo é implantado. Ainda, esse código de usuário pode ser formado por letras e números, por exemplo: "usuario36"; e

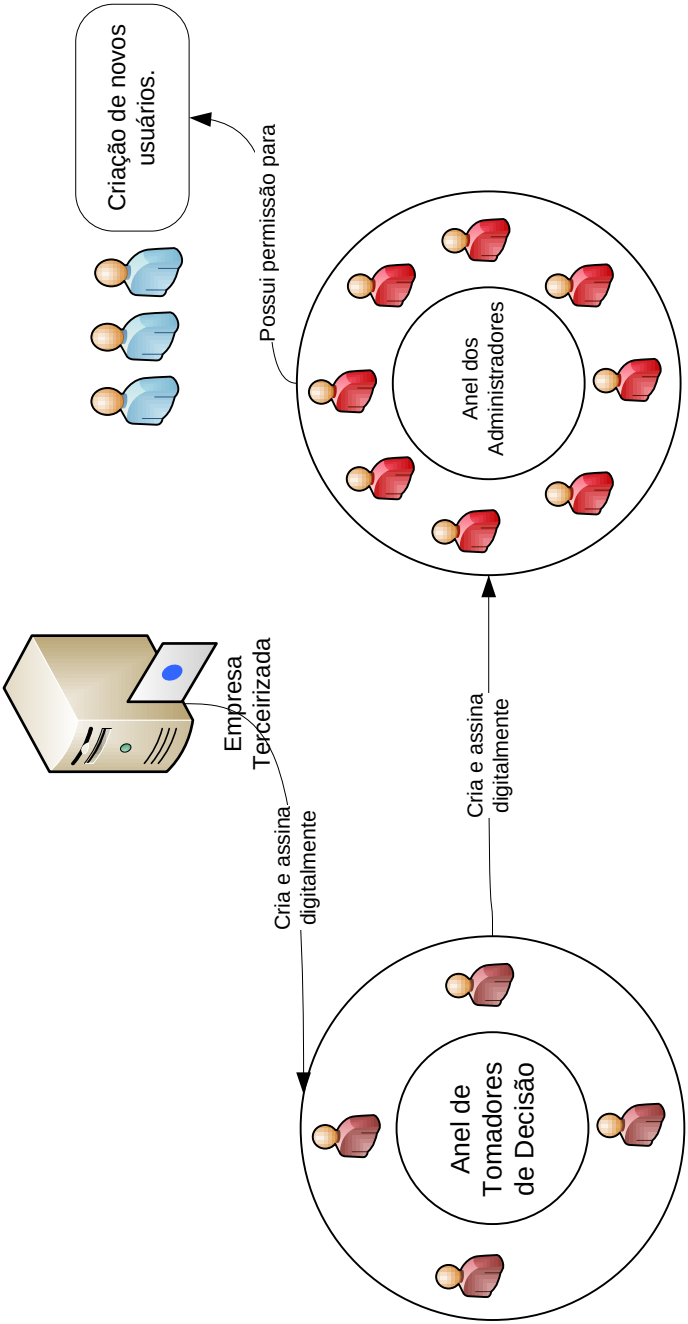

Figura 4.1: Ancoramento de confiança

- N_A: indica um *nonce* (número aleatório) gerado pelo Administrador.

Os agentes Tomadores de Decisão não possuem um identificador próprio pois não participam do processo de troca de mensagens. Estes atuam somente nas premissas que dão suporte ao protocolo (ancoramento da confiança).

O protocolo utilizado para cadastro de usuários apresenta os passos abaixo. Esse protocolo é ilustrado na Figura 4.2. Nessa figura (e nas demais ilustrações deste capítulo), as setas representam os passos do protocolo, ao passo que os quadros com linha tracejada indicam processamento interno dos agentes envolvidos em cada passo. Setas e quadros são numerados com o número do passo correspondentes.

1. *Administrador inicia cadastro.* O Administrador solicita o cadastro do Cliente. Para tanto, o Administrador gera um número aleatório, *nonce* (N_A), e o concatena ao CPF do Cliente. Esta concatenação é cifrada com a chave pública do Cliente e enviada ao Servidor para encaminhamento ao Cliente;

2. *Redirecionamento do Servidor.* O Servidor, ao receber a mensagem, descarta o identificador de roteamento C e a encaminha ao Cliente correspondente;

3. *Cliente fornece dados de identificação.* O Cliente decifra o conteúdo da mensagem com sua chave privada e verifica se o CPF contido dentro dela é, de fato, o seu próprio. Após, gera o *token* $\{|CPF_C, Cod_C|\}$ (CPF e código de usuário) e o assina com sua chave privada. Este *token* é então concatenado com o *nonce* N_A. O resultado da concatenação é então cifrado com a chave pública do Administrador e enviada ao Servidor para posterior encaminhamento ao Administrador;

4. *Redirecionamento do Servidor.* O Servidor, ao receber a mensagem, descarta o identificador de roteamento A e a encaminha ao Administrador;

5. *Administrador completa cadastro.* O Administrador decifra a mensagem com sua chave privada e compara o *nonce* N_A recebido com aquele gerado no passo 1, confirmando a posse da chave privada pelo Cliente. O Administrador, então, assina o *token* $\{|CPF_C, Cod_C|\}_{sKr_C}$ com sua chave privada, tornando-o duplamente assinado. Este *token* duplamente assinado é enviado ao Servidor para encaminhamento ao Banco de Dados; e

6. *Banco de Dados armazena informações.* O Servidor, ao receber o *token* duplamente assinado, descarta o identificador de roteamento B e encaminha a mensagem ao Banco de Dados para armazenamento de longo prazo.

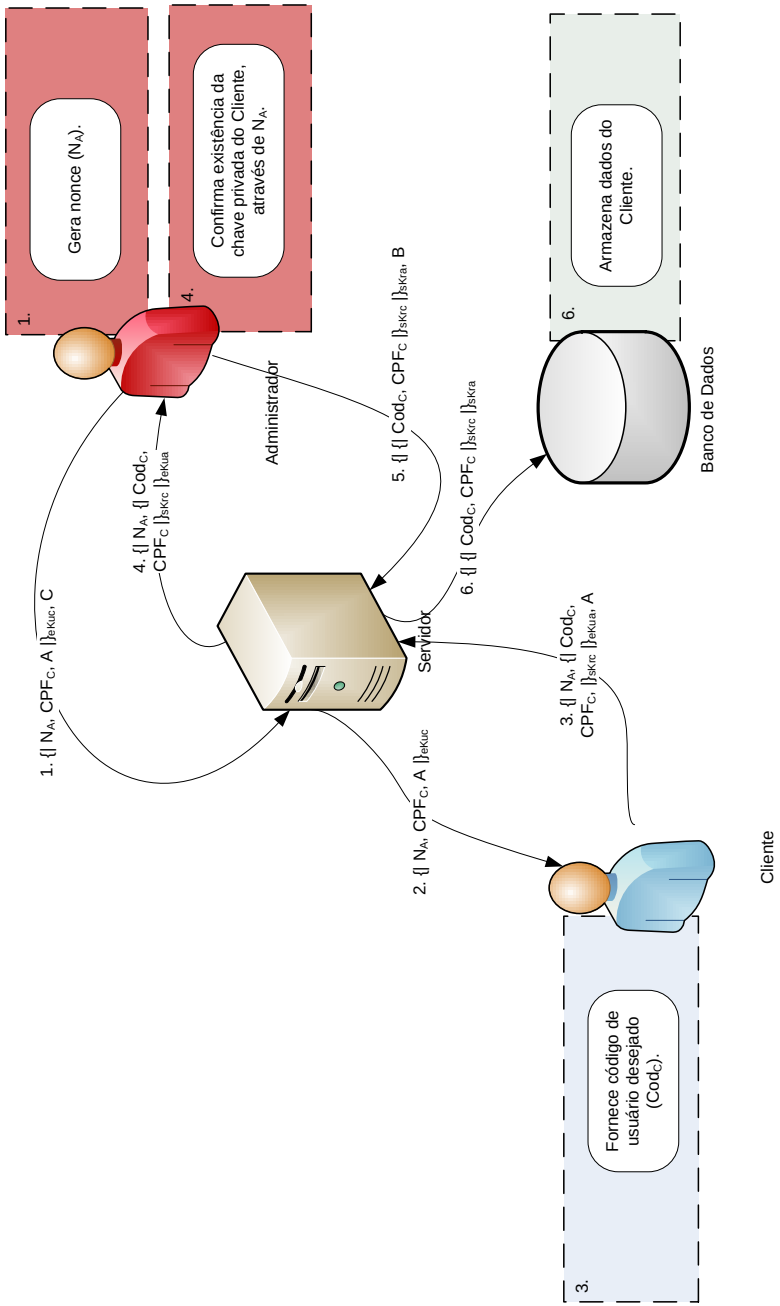

Figura 4.2: Prova de posse do cadastro

Adicionalmente, temos a notação formal desta versão do protocolo, representada na Figura 4.3. As regras de construção deste tipo de notação são definidas na Seção 3.2.

1.	A	$\rightarrow$	S	:	$\{\|N_A, CPF_C, A\|\}_{eKu_C}, C$
2.	S	$\rightarrow$	C	:	$\{\|N_A, CPF_C, A\|\}_{eKu_C}$
3.	C	$\rightarrow$	S	:	$\{\|N_A, \{\|Cod_C, CPF_C\|\}_{sKr_C}\|\}_{eKu_A}, A$
4.	S	$\rightarrow$	A	:	$\{\|N_A, \{\|Cod_C, CPF_C\|\}_{sKr_C}\|\}_{eKu_A}$
5.	A	$\rightarrow$	S	:	$\{\|\{\|Cod_C, CPF_C\|\}_{sKr_C}\|\}_{sKr_A}, B$
6.	S	$\rightarrow$	B	:	$\{\|\{\|Cod_C, CPF_C\|\}_{sKr_C}\|\}_{sKr_A}$

Figura 4.3: Notação formal da prova de posse no cadastro

4.2.2.2 Cadastro com Dois Fatores

De uma maneira geral, a adição de fatores biométricos ao protocolo de cadastro, causa pequenas mudanças em sua estrutura. Basicamente, estas modificações restringem-se ao terceiro passo, onde o agente Cliente deve fornecer seu *template* biométrico junto com seu código de usuário e CPF.

O funcionamento desta segunda versão do protocolo de cadastro é definido nos passos abaixo. O *template* biométrico do Cliente é indicado por $Template_C$. As Figuras 4.4 e 4.5 apresentam sua visão geral e correspondente notação.

1. *Administrador inicia cadastro.* O Administrador solicita o cadastro do Cliente. Para tanto, o Administrador gera um *nonce* (N_A) e o concatena ao CPF do Cliente. Esta concatenação é cifrada com a chave pública do Cliente e enviada ao Servidor para encaminhamento ao Cliente;

2. *Redirecionamento do Servidor.* O Servidor, ao receber a mensagem, descarta o identificador de roteamento C e encaminha a mensagem ao Cliente correspondente;

3. *Cliente fornece dados de identificação.* O Cliente decifra o conteúdo da mensagem com sua chave privada e verifica se o CPF contido dentro dela é, de fato, o seu próprio. Após, gera o *token* $\{|CPF_C, Cod_C, Template_C|\}$ e o assina com sua chave privada. Este *token* é então concatenado com o *nonce* N_A. Esta concatenação é cifrada com a chave pública do Administrador e enviada ao Servidor para encaminhamento;

4. *Redirecionamento do Servidor.* O Servidor, ao receber a mensagem, descarta o identificador de roteamento A e encaminha a mensagem ao Administrador;

5. *Administrador completa cadastro.* O Administrador decifra a mensagem com sua chave privada e compara o *nonce* N_A recebido com aquele gerado no passo 1, confirmando a posse da chave privada pelo Cliente. O Administrador, então, assina o *token* $\{|CPF_C, Cod_C, Template_C|\}_{sKr_C}$ com sua chave privada, tornando-o duplamente assinado. Este *token* duplamente assinado é enviado ao Servidor para posterior encaminhamento ao Banco de Dados; e

6. *Banco de Dados armazena informações.* O Servidor, ao receber o *token* duplamente assinado, descarta o identificador de roteamento B e o encaminha ao Banco de Dados para armazenamento de longo prazo.

4.2.2.3 Cadastro Simplificado

Foi também idealizada uma versão reduzida do protocolo de cadastro. A principal diferença desta versão para as anteriores (prova de posse e cadastro com dois fatores) se faz sentir pela ausência de participação ativa do Cliente. O processo de cadastro passa a ser de inteira responsabilidade do Administrador. Todavia, o uso das versões completas deve ser preferido, de modo a suavizar problemas de confiança quando da troca de Administradores dentro de seu anel.

Os passos que fazem parte deste cadastro reduzido são definidos a seguir. As Figuras 4.6 e 4.7 apresentam uma visão gráfica da troca de mensagens e correspondente notação formal, respectivamente.

1. *Administrador inicia cadastro.* O Administrador obtém (ou define ele próprio) o código de usuário e CPF do Cliente, concatenando-os e assinando-os com sua chave privada. Este *token* assinado é encaminhado ao Servidor para encaminhamento ao Banco de Dados; e

2. *Banco de Dados armazena informações.* O Servidor, ao receber o *token* assinado, descarta o identificador de roteamento B e o encaminha ao Banco de Dados para armazenamento de longo prazo.

4.2.3 Autenticação de Usuários

Nosso protocolo de autenticação é baseado numa estratégia de *desafio-resposta*. Esta estratégia é feita de forma a permitir que o Servidor confirme a identidade do Cliente. Para isso, o Servidor busca verificar se a chave privada do Cliente está realmente sendo utilizada.

Semelhante ao protocolo de cadastro, foram concebidas duas versões do protocolo de autenticação: **Prova de Posse** e **Autenticação com Dois Fatores**. Cada uma dessas versões deve ser integrada com sua

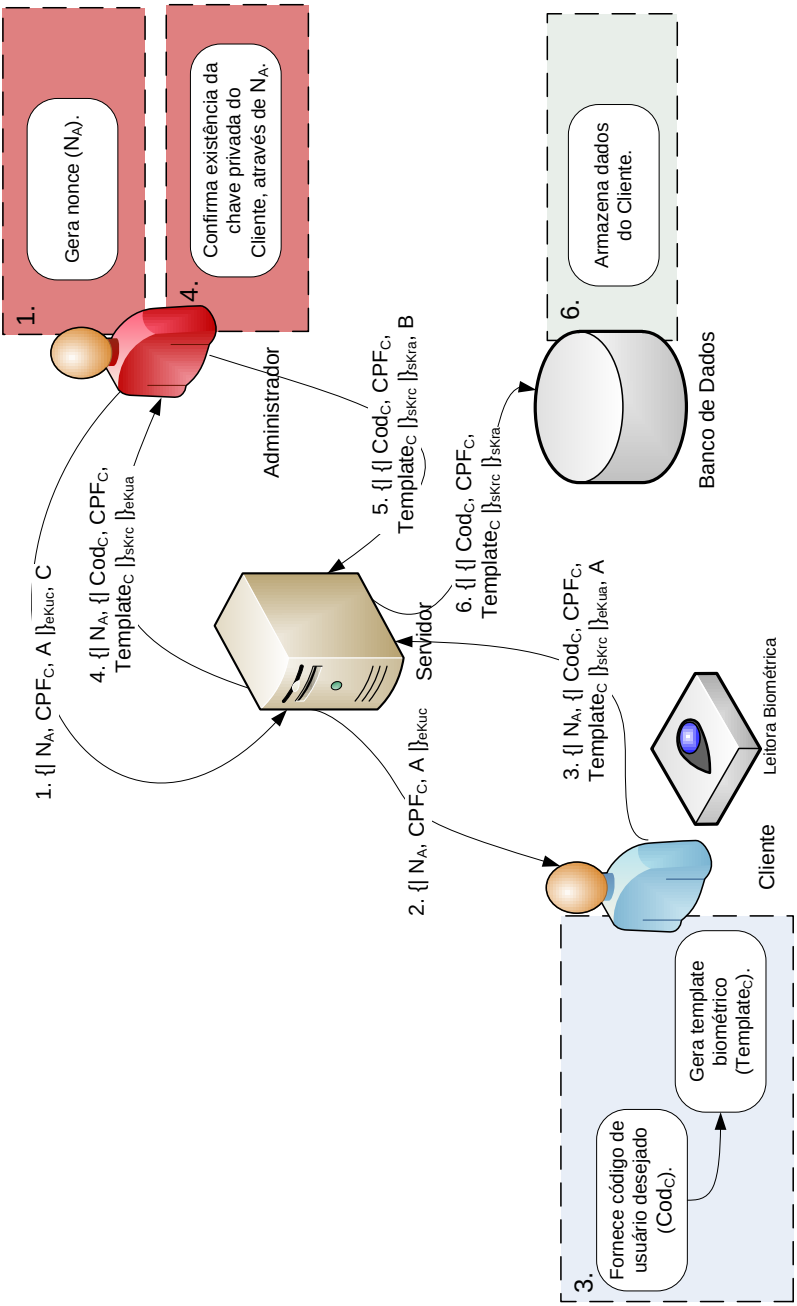

Figura 4.4: Cadastro de usuários com dois fatores

1.	A	$\rightarrow$	S	:	$\{\|N_A, CPF_C, A\|\}_{eKu_C}, C$
2.	S	$\rightarrow$	C	:	$\{\|N_A, CPF_C, A\|\}_{eKu_C}$
3.	C	$\rightarrow$	S	:	$\{\|N_A, \{\|Cod_C, CPF_C, Template_C\|\}_{sKr_C}\|\}_{eKu_A}, A$
4.	S	$\rightarrow$	A	:	$\{\|N_A, \{\|Cod_C, CPF_C, Template_C\|\}_{sKr_C}\|\}_{eKu_A}$
5.	A	$\rightarrow$	S	:	$\{\|\{\|Cod_C, CPF_C, Template_C\|\}_{sKr_C}\|\}_{sKr_A}, B$
6.	S	$\rightarrow$	B	:	$\{\|\{\|Cod_C, CPF_C, Template_C\|\}_{sKr_C}\|\}_{sKr_A}$

Figura 4.5: Notação formal do cadastro com dois fatores

correspondente no protocolo de cadastro. Por exemplo: a versão prova de posse do protocolo de cadastro deve ser integrada à versão prova de posse do protocolo de autenticação. O cadastro simplificado de usuários, por não envolver o uso de fatores biométricos, deve ser integrado com a prova de posse da autenticação.

4.2.3.1 Prova de Posse

O Cliente é o responsável pelo desejo de se conectar ao Servidor e, para isso, inicia uma conexão segura (através do protocolo SSL/TLS). Após o estabelecimento dessa conexão, o Servidor inicia o processo de troca de mensagens que busca a autenticação do Cliente. Em caso de não recebimento de resposta do Cliente, a conexão é automaticamente recusada. Isto acontece tanto na versão prova de posse quanto autenticação com dois fatores.

Os passos da versão prova de posse do protocolo de autenticação são definidos a seguir:

1. *Servidor envia desafio.* O Servidor inicia o processo de autenticação através da geração do *nonce*, N_S, para atuar como desafio. Este *nonce* é enviado ao Cliente;

2. *Cliente fornece dados de identificação.* O Cliente concatena seu CPF e o identificador S ao desafio recebido. Este *token* é assinado com a chave privada do Cliente e enviado ao Servidor.

3. *Servidor realiza consulta.* O Servidor verifica a assinatura do Cliente, confirmando que o mesmo está em posse de sua chave privada. Então, CPF_C é enviado ao Banco de Dados; e

4. *Banco de Dados retorna informações armazenadas.* O Banco de Dados retorna o *token* duplamente assinado $((\{\|\{\|Cod_C, CPF_C\|\}_{sKr_C}\|\}_{sKr_A}))$ ao Servidor. Este *token* foi armazenado no fim no protocolo de cadastro. Ao receber a mensagem, o Servidor verifica se os dados armazenados no banco e fornecidos pelo Cliente são iguais.

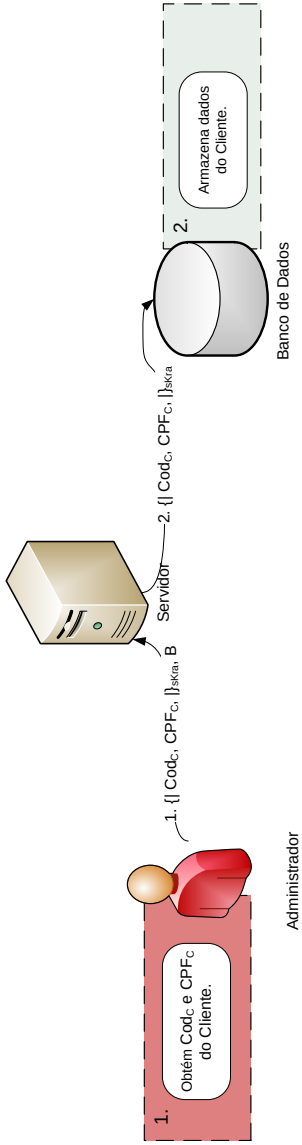

Figura 4.6: Cadastro simplificado

1.	A	$\rightarrow$	S	:	$\{\|Cod_C, CPF_C\|\}_{sKr_A}, B$
2.	S	$\rightarrow$	B	:	$\{\|Cod_C, CPF_C\|\}_{sKr_A}$

Figura 4.7: Notação formal do cadastro simplificado

Para facilitar o entendimento, a Figura 4.8 apresenta uma visão geral desta primeira versão do protocolo de autenticação. Em termos de notações, a prova de posse do protocolo de autenticação é representada na Figura 4.9.

4.2.3.2 Autenticação com Dois Fatores

De maneira semelhante à sua contraparte no protocolo de cadastro, a adição de fatores biométricos ao protocolo de autenticação requer pequenas modificações em sua estrutura. Novamente, estas mudanças limitam-se ao envio do *template* biométrico do Cliente. Ao final, o Servidor deve também verificar a correspondência entre os *templates* biométricos (fornecido e previamente armazenado). Esta verificação é feita inteiramente pelo Servidor, sem participação do Cliente.

No entanto, há uma diferença substancial entre esta versão e a prova de posse da autenticação. A forma como o *nonce* N_S de desafio é transmitido no primeiro passo.

Na primeira versão, este *nonce* N_S era enviado em texto claro ao Cliente. Para comprovar a posse de sua chave privada, o Cliente deveria enviar o *nonce* novamente dentro da mensagem seguinte.

Por outro lado, nesta nova versão, o *nonce* N_S é cifrado com a chave pública do Cliente antes de ser enviado pelo Servidor. Ainda, o Servidor continua sendo o responsável pelo início da troca de mensagens com o Cliente, de modo idêntico à versão anterior. Na mensagem posterior, ao invés de concatenar o *nonce* decifrado aos seus dados, o Cliente cifra seus dados com o *nonce* recebido.

Ambos os modos permitem a comprovação de posse da chave privada pelo Cliente, sem diminuir o nível de confiança no processo. Com isso, provê-se o sigilo dos dados biométricos em trânsito.

Os passos do protocolo de autenticação com fatores biométricos são dados a seguir:

1. *Servidor envia desafio.* O Servidor inicia o processo de autenticação através da geração de um *nonce*, N_S, que é enviado ao Cliente. Antes de ser enviado, esse *nonce* é cifrado com a chave pública do Cliente;

2. *Cliente fornece dados de identificação.* O Cliente decifra o *nonce* recebido com sua chave privada. Então, concatena seu CPF, a identificação do Servidor e seu *template* biométrico. Este *token* é cifrado com o *nonce* recebido e enviado ao Servidor. Neste passo,

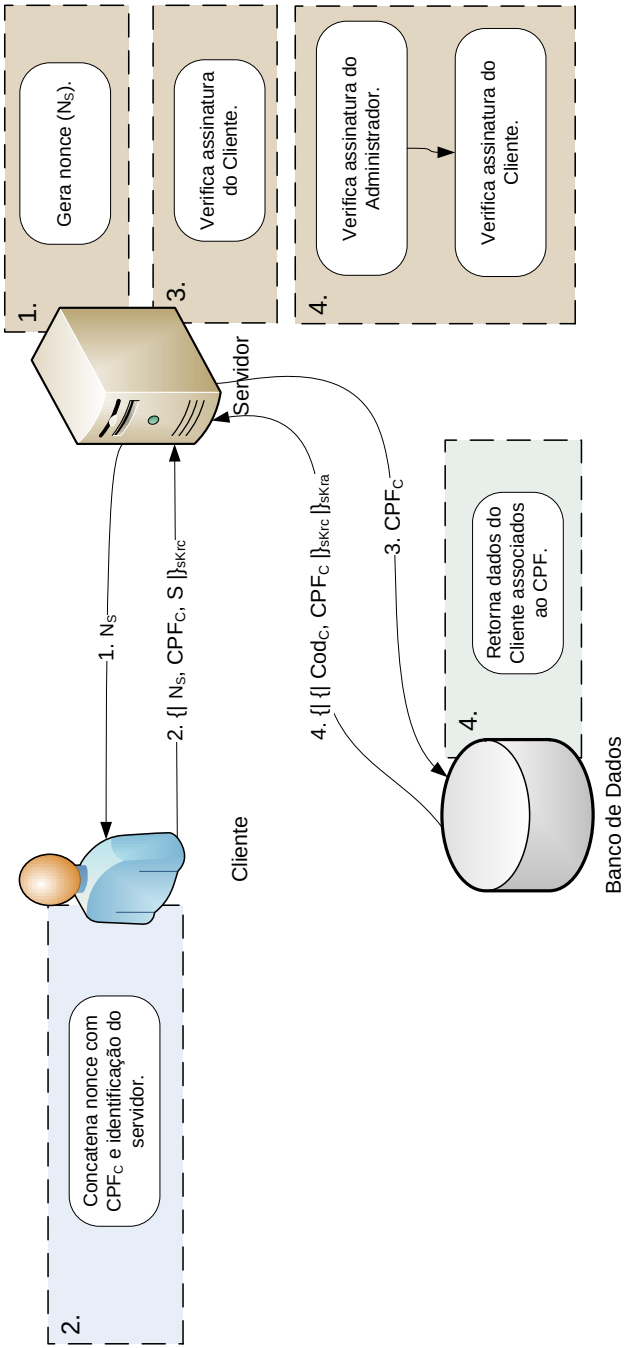

Figura 4.8: Prova de posse da autenticação

1.	S	$\rightarrow$	C	:	N_S
2.	C	$\rightarrow$	S	:	$\{\|N_S, CPF_C, S\|\}_{sKr_C}$
3.	S	$\rightarrow$	B	:	CPF_C
4.	B	$\rightarrow$	S	:	$\{\|\{\|Cod_C, CPF_C\|\}_{sKr_C}\|\}_{sKr_A}$

Figura 4.9: Notação formal da prova de posse da autenticação

a concatenação da origem (CPF) e destino (S) da mensagem é necessária para evitar ataques de oráculo;

3. *Servidor realiza consulta.* O Servidor decifra o *token* recebido, confirmando que o Cliente de fato está em posse de sua chave privada. Então, CPF_C é enviado ao Banco de Dados para realização de consulta; e

4. *Banco de Dados retorna informações armazenadas.* O Banco de Dados retorna o *token* duplamente assinado $((\{\|\{\|Cod_C, CPF_C, Template_C\|\}_{sKr_C}\|\}_{sKr_A}))$ ao Servidor. Ao receber a mensagem, o Servidor verifica as assinaturas e se há correspondência entre os dados previamente armazenados no banco e fornecidos pelo Cliente.

A Figura 4.10 apresenta a autenticação com dois fatores. A notação formal para esta versão do protocolo de autenticação encontra-se na Figura 4.11.

Terminando a descrição em alto-nível de nosso protocolo, passamos a ter uma visão ampla de seu funcionamento. Embora esta seja uma etapa muito importante, não podemos nos esquecer que, não raro, detalhes técnicos podem comprometer o funcionamento do todo. Para termos certeza da viabilidade prática de nossa proposta, é necessário a criação de uma prova de conceito.

4.3 PROVA DE CONCEITO

Também conhecidas como **protótipos**, as provas de conceito buscam avaliar a viabilidade prática de uma especificação após sua concepção inicial. Sua principal finalidade é certificar-se de que a especificação inicial não contenha lacunas em aberto ou mesmo pontos conflitantes.

Provas de conceito de protocolos assumem a forma de implementações em linguagem de programação. Nestas implementações são levados em conta apenas os aspectos centrais da proposta de novo protocolo. Ou seja, não são levadas em consideração as características próprias do sistema ao qual o protocolo é futuramente integrado. Desta forma, evita-se o aumento desnecessário da complexidade da implementação, permitindo uma avaliação mais objetiva da execução do protocolo.

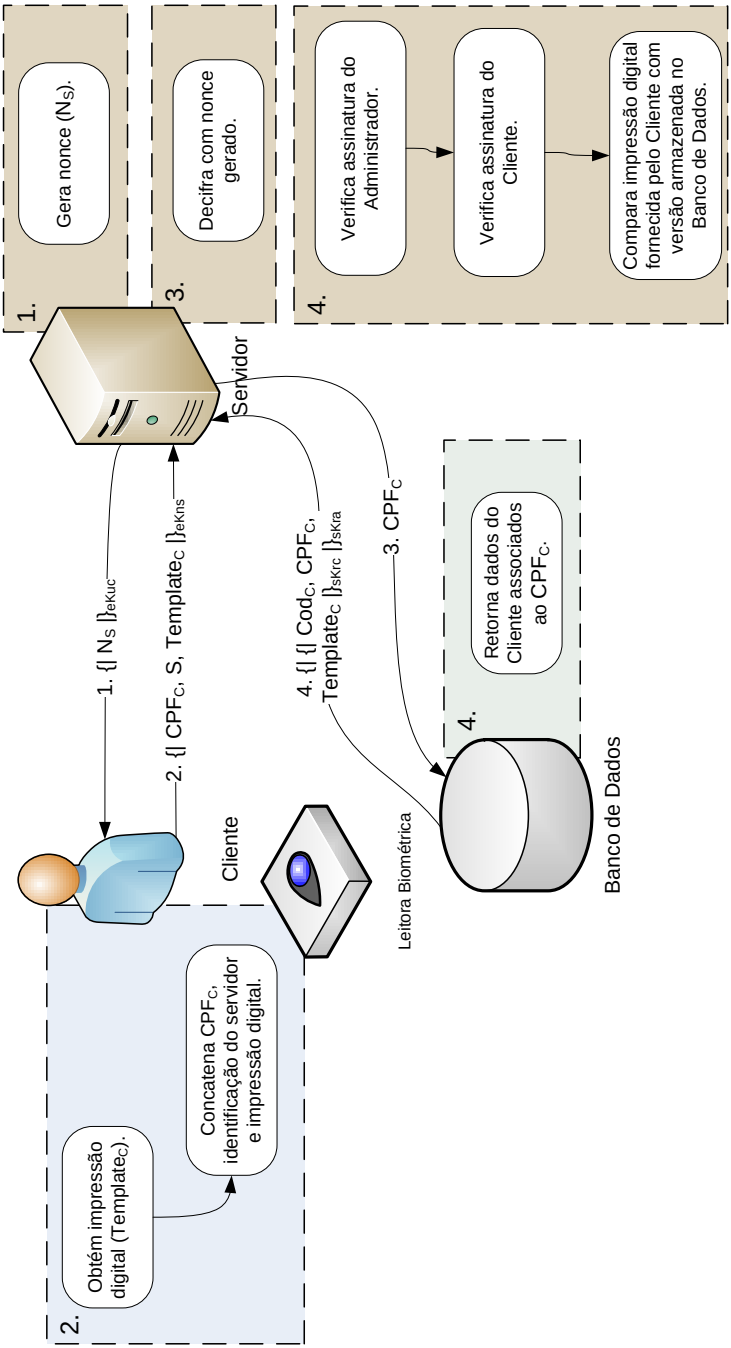

Figura 4.10: Autenticação com dois fatores

1.	S	$\rightarrow$	C	:	$\{\|N_S\|\}_{eKu_C}$
2.	C	$\rightarrow$	S	:	$\{\|CPF_C, S, Template_C\|\}_{eK_{N_S}}$
3.	S	$\rightarrow$	B	:	CPF_C
4.	B	$\rightarrow$	S	:	$\{\|\{\|Cod_C, CPF_C, Template_C\|\}_{sKr_C}\|\}_{sKr_A}$

Figura 4.11: Notação formal da autenticação com dois fatores

No âmbito de nosso trabalho, foram criados dois conjuntos de aplicações para testar a viabilidade de nossa proposta. O primeiro conjunto buscou avaliar o uso de dispositivos biométricos e *smartcards* integrados à linguagem de programação Java. Por sua vez, o segundo conjunto de aplicações serviu como prova de conceito dos protocolos propostos em si.

4.3.1 Comunicação com Dispositivos Biométricos e *Smartcards*

A falta de *expertise* por parte do autor com relação à implementação de sistemas envolvendo dispositivos de autenticação biométrica e acesso a *smartcards* tornou imperativo a realização de um estudo preliminar para avaliar as possibilidades de desenvolvimento.

A adoção da plataforma Java como ferramente básica de desenvolvimento levou ao uso da tecnologia de *Applets* em algumas etapas dos protocolos. *Applets* Java são pequenas aplicações embarcadas em páginas *web* que permitem a execução de aplicações completas. Aumenta-se, assim, as possibilidades de interação com usuários (ORACLE, 2012b). O uso deste tipo de aplicação é comum quando há a necessidade de conexão com dispositivos de *hardware* do lado do usuário cliente (e não do lado servidor *web*).

Era necessário garantir o acesso às operações da leitora biométrica nestas condições. Um conjunto mínimo de operações deveria ser suportado: cadastro de *templates* e reconhecimento um-para-um (1:1). A operação de identificação um-para-muitos (1:N) não é essencial ao nosso propósito, tendo em vista que o servidor compara *templates* em pares em cada tentativa de autenticação.

Quanto ao uso de *smartcards*, não sabíamos dizer de antemão se os mesmos poderiam ser de fato integrados ao protocolo através da linguagem Java. As seguintes operações deveriam ser executadas diretamente no cartão: geração de números aleatórios (para os *nonces*); cifragem simétrica; cifragem assimétrica; assinatura digital; verificação de assinatura; e exportação de certificados digitais.

O projeto que envolveu a criação deste conjunto inicial de aplicações recebeu o nome de Prova de Tecnologias (*Proof of Technologies*). As duas aplicações desenvolvidas demonstraram ser possível a plena integração da linguagem Java com leitoras biométricas e *smartcards*.

O acesso à leitora biométrica e *smartcards* foi bem-sucedido. Em ambos os casos, foi possível demonstrar a viabilidade de todas as operações previamente estabelecidas. Além disso, sua integração com *Applets* demonstrou também ser viável.

A Figuras 4.12 apresenta a aplicação que atuou como Prova de Tecnologia da biometria. Nessa figura, é apresentado um *Applet* Java capturando impressões digitais a partir da leitora biométrica. Essa aplicação está organizada do seguinte modo: no lado esquerdo, encontra-se a imagem da impressão digital capturada pela leitora; e no lado direito, há uma lista das impressões digitais cadastradas. Na parte central da figura, existem três botões de ação: *Enroll* (Cadastrar), *Verify* (Verificar) e *Identify* (Identificar). O botão *Verify* faz o reconhecimento biométrico um-para-um (1:1), ao passo que o botão *Identify* faz o reconhecimento um-para-muitos (1:N). Já na parte inferior da figura, há um espaço para registro dos eventos internos (*logs*) da aplicação.

Na Figura 4.13 é apresentada a aplicação que atuou como Prova de Tecnologia de *smartcards*. Essa figura apresenta o resultado de uma função de resumo criptográfico (*hash*) de um arquivo texto simples, calculado diretamente por um *smartcard*. No lado direito da aplicação, estão listados os certificados digitais contidos no cartão; na parte central há alguns botões de ação representando algumas operações criptográficas a serem feitas diretamente no cartão. As operações disponíveis são: obter informações acerca dos algoritmos e tamanhos de chave suportados (botão *Info*); atualizar (carregar novamente) a lista de certificados digitais (botão *Update*); cifrar arquivo (botão *Cipher File*); obter o texto em claro de um arquivo cifrado (*Decipher File*); calcular o resumo criptográfico (botão (*Hash*); e assinar digitalmente um arquivo (botão *Sign File*). Apesar de não aparecer na figura, há também uma opção para verificação de assinatura digital (botão *Verify Signature*). Também há um registro de eventos (*logs*) na parte inferior da aplicação.

Na eventualidade de não ser possível a integração das capacidades biométricas com a tecnologia de *Applets*, outras alternativas de implementação teriam que ser pesquisadas. Num cenário hipotético, ocorreria a substituição do *Applet* por uma aplicação convencional. Esta aplicação seria descarregada automaticamente no momento do acesso do usuário.

A adequação das tecnologias escolhidas para o cumprimento dos requisitos mais críticos de nossa proposta foi comprovada. Iniciamos, então, a segunda etapa da prova de conceito.

4.3.2 Protótipo dos Protocolos

No desenvolvimento do protótipo dos protocolos, toma-se como base unicamente a versão com dois fatores dos protocolos. Como visto anteriormente, são poucas as diferenças entre ambas as versões. Uma delas é a presença do *token* biométrico nos dados de identificação do Cli-

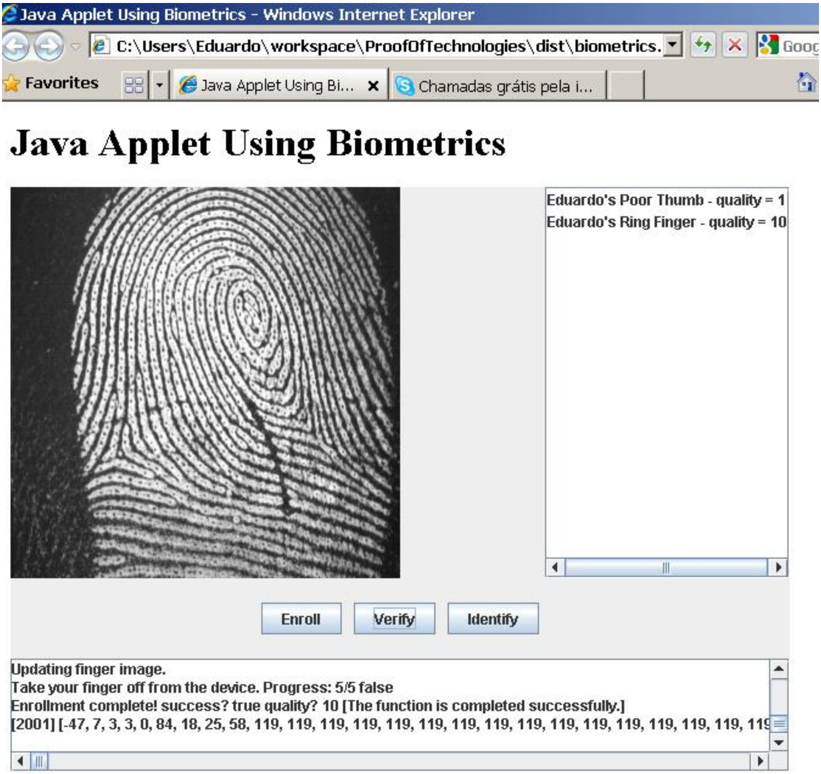

Figura 4.12: Prova de tecnologia: uso da biometria

ente. Outra, é a forma como o *nonce* N_S é utilizado na autenticação. Na opinião do autor, estas sutilezas, não justificam por si só, a criação de protótipos próprios para cada versão.

É interessante adicionar uma nova limitação ao protótipo. Este protótipo busca avaliar a capacidade prática de implantação do protocolo, em termos da troca de mensagens entre agentes. Assim, as premissas iniciais (Seção 4.2.1) que dão sustentação ao seu funcionamento não foram incluídas neste escopo.

Em resumo, os pontos que ficaram de fora do escopo deste protótipo são:

- Ancoramento e distribuição da confiança;
- Prova de posse do cadastro e autenticação; e
- Cadastro simplificado.

A implementação do protótipo foi feita com o uso das seguintes

Figura 4.13: Prova de tecnologia: uso de *smartcards*

tecnologias: plataforma Java (ORACLE, 2012c), biblioteca de serviços criptográficos *BouncyCastle* (BOUNCY CASTLE, 2012), banco de dados *MySQL* (MYSQL, 2012) e SDK (*Software Development Kit*) disponibilizado pela leitora biométrica Futronic (FUTRONIC, 2012a).

Os agentes Administrador e Cliente são implementados com interface gráfica. Por sua vez, o agente Servidor é implementado em interface texto. O agente Banco de Dados se faz presente como uma conexão na aplicação do agente Servidor. Além disso, a comunicação entre as aplicações de cada agente ocorre através da rede sobre uma conexão SSL. Para simplificação, todo o material criptográfico é armazenado dentro de um único *smartcard*, que é acessado simultaneamente pelas aplicações.

A interação com o protótipo segue a ordem prevista na descrição de alto-nível. O funcionamento do protocolo de cadastro ocorre do seguinte modo:

1. Primeiro, na aplicação Administrador, o usuário solicita o cadastro

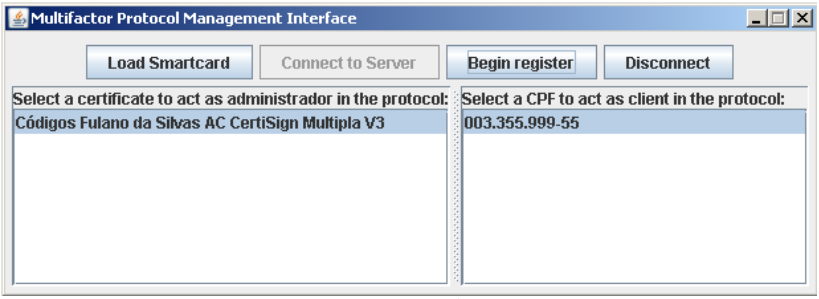

Figura 4.14: Prova de conceito do protótipo: aplicação Administrador

de um novo Cliente;

2. O fluxo é então transferido para a aplicação Servidor e daí para a aplicação Cliente, onde é feito a coleta dos dados de identificação;

3. Em seguida, o fluxo retorna à aplicação Servidor e daí, novamente, é reencaminhado para a aplicação Administrador;

4. Na aplicação Administrador, o usuário finaliza o cadastro; e

5. Por último, a aplicação Administrador encaminha, à aplicação Servidor, os dados para armazenamento no Banco de Dados.

A Figura 4.14 apresenta o protótipo da aplicação usada pelo agente Administrador. Essa aplicação está dividida em duas partes principais: botões de comando (parte superior) e listas de certificados digitais e CPFs. Os seguintes comandos estão disponíveis *Load Smartcard* (carregar o conteúdo do *smartcard* para as listas), *Connect to Server* (conectar ao agente Servidor); *Begin register* (iniciar protocolo de cadastro); e *Disconnect* (desconectar do agente Servidor). A lista de certificados exibe todos os certificados digitais contidos no *smartcard*, por sua vez, a lista de CPFs é preenchida apenas com os certificados digitais do tipo *e-CPF*.

Com no mínimo um Cliente cadastrado, o protocolo de autenticação é disponibilizado para uso na aplicação Cliente. Segue seu funcionamento.

1. Primeiro, a aplicação Servidor envia o *token* $\{N_S\}_{eKu_C}$. O acesso ao certificado digital do Cliente no momento do estabelecimento da conexão SSL;

2. A aplicação Servidor também envia uma lista dos CPFs dos usuários cadastrados (este passo foi adicionada apenas para simplificar a interação do usuário com o protótipo);

3. Então, na aplicação Cliente, o usuário seleciona seu CPF, fornece sua impressão digital e retorna estes dados à aplicação Servidor;
4. De posse destas informação, a aplicação Servidor realiza uma consulta ao Banco de Dados;
5. No retorno da consulta, a aplicação Servidor verifica a autenticidade das assinaturas digitais e compara os *templates* biométricos.
6. Dependendo do resultado destas verificações, a aplicação Servidor toma a decisão de aceitar ou recusar a conexão.
7. Por último, a aplicação Cliente exibe uma caixa de diálogo informando ao usuário se a conexão foi estabelecida ou encerrada.

A Figura 4.15 apresenta o protótipo da aplicação usada pelo agente Cliente. Essa aplicação consiste de uma área para exibição das impressões digitais capturadas pela leitora biométrica conectada ao computador do Cliente. Além disso, também estão presentes quatro botões de ação: *Connect* (conecta ao agente Servidor); *Register* (dar continuidade ao protocolo de cadastro); *Authenticate* (simular o protocolo de autenticação junto com o Servidor); e *Disconnect* (desconectar do Servidor). Um detalhe: nessa figura aparece a frase *Put finger on reader for template 3/5*, que significa "coloque o dedo na leitora para captura da terceira de um total de cinco impressões digitais". Essa medida é importante afim de tornar o *template* biométrico resultante o mais preciso possível. É comum que haja variação na qualidade do *template*.

4.3.3 Adaptações Efetuadas

De modo geral, a prova de conceito não levou a modificações significativas na estrutura do protocolo. Contudo, adaptações pontuais foram feitas para contornar pequenos entraves durante sua implementação. Apresentamos, a seguir, algumas modificações e impressões.

- **Formato do identificador de roteamento.** Os identificadores de roteamento (A e C) devem assumir a forma de um certificado digital ou *alias* de certificado digital. No protótipo, estes identificadores são utilizados para determinar qual chave (dentre todas presentes no *smartcard*) é usada para cifragem e assinatura. Esta medida é necessária para permitir que cada agente seja identificado de maneira única na ICP utilizada (neste caso, a ICP-Brasil). Na hipótese deste identificador assumir outro critério, como o nome do agente, podem ocorrer conflitos com outros usuários com o mesmo identificador.
- **Manutenção do identificador de roteamento.** No protocolo de cadastro de usuários, o identificador de roteamento (A, C e B)

Figura 4.15: Prova de conceito do protótipo: aplicação Cliente

é removido em redirecionamentos. Em se tratando dos destinatários Administrador e Cliente, é necessário que suas respectivas aplicações tenham as chaves correspondentes a cada agente. Sem esta informação, não há como a aplicação saber qual dos (vários) Administradores foi o responsável pelo início do cadastro (passo 4 da Figura 4.3).

- **Tamanho do *nonce* N_S.** Na versão com dois fatores do protocolo de autenticação, o *nonce* N_S é usado como chave (passo 2 da Figura 4.11). O *nonce* então deve servir como entrada a um algoritmo criptográfico. É necessário que o *nonce* gerado tenha tamanho igual ou superior ao menor tamanho possível de chave para o algoritmo criptográfico escolhido. No caso do AES (*Advanced Encryption Standard*), o menor tamanho permitido para chaves é 128 bits (NIST, 2001). Outra possibilidade, ainda, seria o uso do *nonce* como entrada a uma função de derivação de chaves.

A partir deste momento, passaremos a tratar um problema detectado durante o desenvolvimento da primeira prova de conceito. Em organizações de grande porte, é comum a existência de mais de um local (diversas entradas) onde a presença de fortes mecanismos de autenticação são necessários. O mesmo se aplica a situações onde uma matriz conta com inúmeras filiais geograficamente espalhadas.

Levando isso em consideração, é possivel que haja uma grande diversidade de dispositivos de *hardware* e *software*. Para evitar problemas de interoperabilidade, sugere-se o uso de interfaces de comunicação padronizadas.

Criados por organismos internacionais ou mesmo de iniciativas da própria indústria, estes **padrões** possibilitam um alto-nível de integração. Este aumento de integração não ocorre apenas entre diferentes produtos, mas também entre diferentes fabricantes.

A ISO (*International Organisation for Stardardisation*) (ISO, 2012) representa o exemplo mais conhecido de instituição voltada ao desenvolvimento de padrões. Como exemplo de iniciativa da indústria com este fim, temos o *Trust Computing Group* (TRUSTED COMPUTING GROUP, 2012).

Em nossa proposta, existem dois componentes críticos de *hardware* que demandam um tratamento padronizado para sua plena integração com demais sistemas. São eles:

- Leitores de *smartcards*; e
- Dispositivos de autenticação biométrica.

No caso dos leitores de *smartcards*, a comunicação entre sistema operacional e leitor é feita através da interface PKCS#11 (*Public-Key Cryptography Standard Number 11*) (RSA LABORATORIES, 2009). Na implementação das provas de conceito, fez-se uso da biblioteca *SunPKCS#11* (ORACLE, 2012a) que atua como ponte entre programa escritos na linguagem Java e leitoras de cartões que suportam este padrão.

Por outro lado, os dispositivos de autenticação biométrica não contam com interfaces de comunicação padronizadas. A comunicação entre leitora biométrica e aplicação depende exclusivamente dos SDKs (*Software Development Kits*) disponibilizados por seus respectivos fabricantes. Este processo porém, não é portável, fazendo com que haja um forte acoplamento com as implementações de cada fabricante.

Um produto extra deste trabalho é a criação de uma nova biblioteca para integrar o acesso a diferentes dispositivos de autenticação biométrica.

4.4 A BIBLIOTECA DE INTEGRAÇÃO

A utilização de biometria na autenticação de usuários em sistemas de informação é algo complexo, envolvendo não apenas fatores de ordem técnica, mas também fatores de ordem organizacional. Além disso,

também deve-se levar em consideração questões relativas à privacidade dos dados pessoais envolvidos.

Neste sentido, (CAVOUKIAN, 2008) apresenta uma lista de questionamentos que devem ser levados em consideração antes do desenvolvimento desses sistemas. Busca-se antecipar problemas de projeto do sistema, processo de cadastro, processo de autenticação, *templates* de impressão digital, armazenamento e segurança e política de retenção de dados. Apesar de ter sido elaborado com foco em sistemas biométricos baseados no reconhecimento de impressões digitais, as orientações contidas nesse documento podem ser estendidas a outras formas de reconhecimento, como a íris.

Um caso particular ocorre em organizações com grandes instalações ou com necessidade de fornecer níveis diferenciados de segurança a seus departamentos internos. Nestas circunstâncias, é preferível que os dados de identificação sejam armazenados num servidor central. Este servidor também se ocupa das tarefas de comparação de *templates* e cadastro de novos usuários. Basicamente, o mesmo modelo adotado neste trabalho.

Nestas condições, algumas das recomendações de (CAVOUKIAN, 2008) são imperativas, (em tradução livre):

- Descreva a arquitetura do sistema em detalhes. Em particular, o cadastro e autenticação são feitos localmente ou num servidor?
- Quais produtos de *hardware* e *software* do fabricante são usados? Você é capaz de fornecer especificações a estes produtos, além daquelas disponíveis publicamente na página *web* do fabricante?
- Os *templates* de impressão digital são compatíveis, ou podem ser tornados compatíveis com algum dos seguintes padrões: INCITS 378 (INCITS, 2009), ISO/IEC 19794-2 (ISO, 2011a), FIPS 201 (NIST, 2006) ou ILO SID-0002 (ILO, 2006)?
- Onde os *templates* são armazenados (por exemplo, localmente ou num servidor)?

Com a perspectiva de que nossa proposta de protocolos possa vir a ser utilizada nos moldes acima, buscou-se a criação de uma solução própria para os problemas de integração. Outras iniciativas do mercado já existem nesse sentido, sob a forma de *software* proprietário com respectiva aquisição de licença para seu uso. São exemplos desta seara: Griaule (GRIAULE, 2012) e VeriFinger (VERIFINGER, 2012). Devido a restrições financeiras deste trabalho de Mestrado, o uso de ferramentas proprietárias foi descartado desde o primeiro momento.

4.4.1 Requisitos

O desenvolvimento desta biblioteca foi norteado pelo seguintes requisitos em sua primeira versão:

1. A biblioteca deve prover flexibilidade na inclusão de novas leitoras de diferentes fabricantes;
2. A biblioteca deve suportar um conjunto mínimo de leitoras;
3. A biblioteca deve suportar as seguintes operações: obtenção de *template* para cadastro; verificação (um-para-um); e identificação (um-para-muitos); e
4. A biblioteca deve prover compatibilidade a *templates* gerados em diferentes leitoras.

A Tabela 4.1 apresenta as leitoras definidas como integrantes do conjunto de suporte inicial mínimo (Requisito 2).

Tabela 4.1: Leitoras biométricas suportadas pela biblioteca de integração

Fabricante e Modelo	**Referência**	**Imagem**
DigitalPersona U.are.U 4000B	(DIGITAL PERSONA, 2005)	
NitGen FingKey Hamster DX	(FINGERTECH, 2012)	
Futronic FS 82	(FUTRONIC, 2012b)	

Com relação aos critérios adotados para a escolha dos fabricantes e modelos de leitoras, este não foi, inicialmente, pautada por critérios técnicos. Mas sim, pela facilidade de compra no mercado brasileiro. Entretanto, durante o desenvolvimento da biblioteca de integração, verificou-se que alguns critérios mínimos de escolha deveriam ter sido adotados. Principalmente, a aderência a padrões como o ISO/IEC 19794-2 (ISO, 2011a) é um requisito essencial em contexto de integração.

Mesmo tendo sido utilizada com sucesso na implementação das provas de conceito (Seção 4.3), a leitora Futronic FS82 não pode ser integrada às demais leitoras. A mesma não oferecia compatibilidade com os padrões de intercâmbio de *templates* mencionados anteriormente. Em virtude disso, esta leitora acabou sendo descartada do projeto.

Infelizmente, só foi possível detectar essa restrição após o início do desenvolvimento da biblioteca. Isto nos chama a atenção para as

recomendações transcritas anteriormente para a boa implantação de sistemas biométricos.

4.4.2 Desenvolvimento

O desenvolvimento da biblioteca fez uso dos diferentes SDKs disponibilizados pelos fabricantes de cada leitora. Foi criado um nível adicional de abstração às operações de cada SDK.

Em paralelo ao desenvolvimento de mecanismos para tornar as leitoras biométricas interoperáveis, teve início uma iniciativa para transformação de nossa proposta de protocolo em, também, uma biblioteca. Pelo forte grau de relacionamento entre ambas as iniciativas, optamos por sua fusão.

A presença do intercâmbio entre leitoras está presente através da interface *BiometricApp*. Esta interface representa um conjunto de métodos básicos que devem ser implementados por qualquer leitora que busque ser integrada à biblioteca. A Figura 4.16 apresenta as classes envolvidas na interoperação de leitoras.

Por outro lado, as classes *Nonce*, *Token* e *AuthReply* (e respectivas subclasses), delineiam nosso protocolo de autenticação, como mostra a Figura 4.17.

4.5 CONCLUSÃO

Neste capítulo, discorremos sobre a parte central de nosso estudo. Foi apresentado um protocolo para cadastro e autenticação de usuários. Duas versões do mesmo protocolo foram apresentadas: prova de posse e dois fatores. A versão com dois fatores complementa a versão prova de posse pela adição de dispositivos de reconhecimento biométrico.

Além da proposta teórica, buscamos demonstrar sua viabilidade prática, através da elaboração de provas de conceito: tanto dos protocolos quanto das tecnologias envolvidas. Como subproduto adicional, desenvolvemos uma biblioteca para integração de diferentes leitoras e uso do protocolo em qualquer sistema.

Dando continuidade à metodologia de projeto de protocolos adotado neste trabalho, nos encaminhamos em direção à próxima etapa que consiste na verificação formal dos protocolos aqui apresentados.

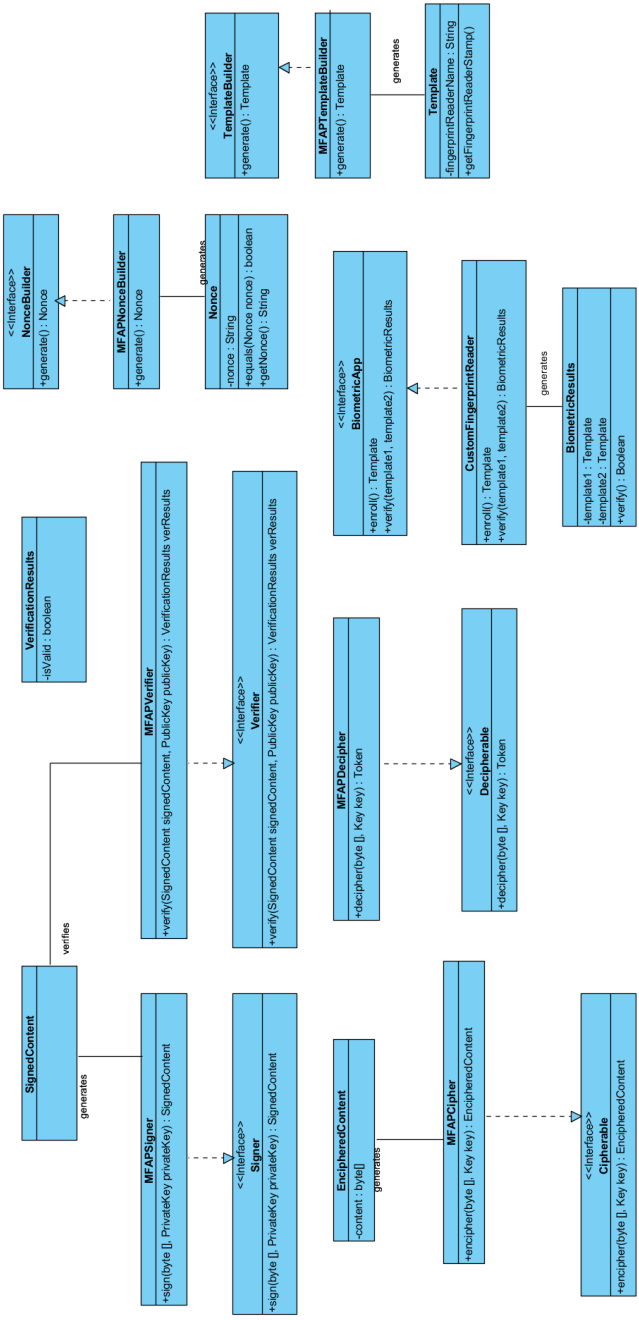

Figura 4.16: Arquitetura da biblioteca de integração: Parte 1

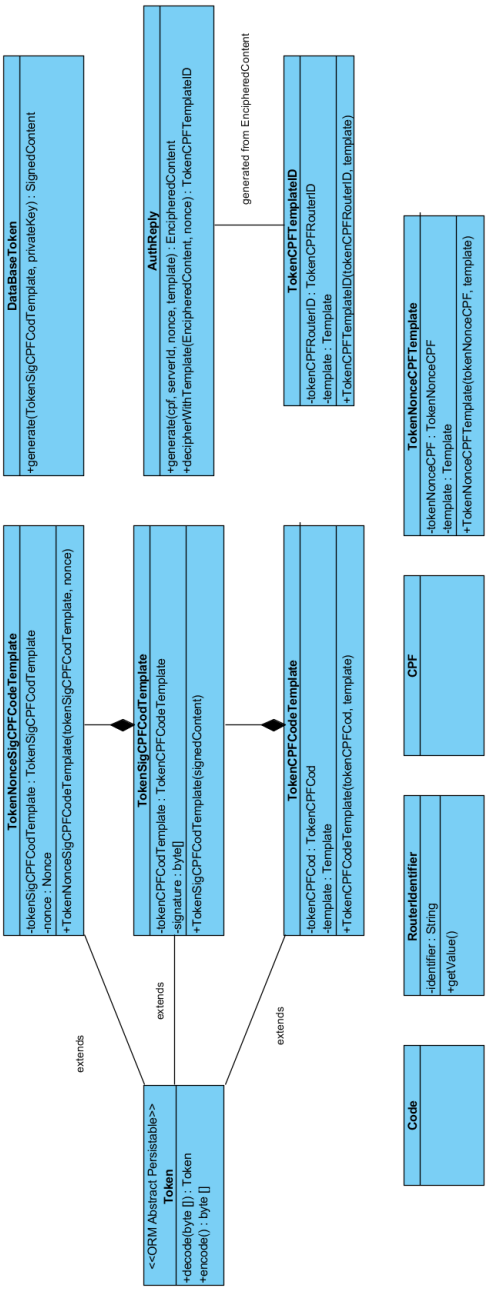

Figura 4.17: Arquitetura da biblioteca de integração: Parte 2

5 A VERIFICAÇÃO DO PROTOCOLO PROPOSTO

5.1 INTRODUÇÃO

Tendo finalizado a especificação em alto-nível de nossos protocolos, avançamos à próxima etapa, que consiste na verificação de suas propriedades. As notações formais apresentadas no decorrer do capítulo anterior são o principal objeto de entrada a este processo de verificação.

Conforme pode ser observado no decorrer do capítulo anterior, existem poucas diferenças entre as versões de nosso protocolo (prova de posse e uso de dois fatores). A única diferença significativa está na presença ou não do *template* biométrico nas mensagens trocadas. Por ser a versão mais completa, a versão do protocolo com adição de fatores biométricos é abordada neste processo de verificação.

Ainda, na opinião do autor, as diferenças na forma de uso do *nonce* N_S entre as diferentes versões do protocolo de autenticação (Figuras 4.9 e 4.11), não justificam por si só, o aumento do escopo da verificação. Além disso, tal como feito com o protótipo desenvolvido, as premissas básicas (ancoramento e distribuição da confiança), não são verificadas formalmente. A justificativa para esta decisão é que estas premissas não fazem parte do protocolo propriamente dito, mas sim das cerimônias de segurança envolvidas.

Para que possamos verificar as propriedades de segurança de nosso protocolo, é necessário antes, formalizá-lo. O processo de formalização adotado segue o modelo proposto por Weidenbach (WEIDENBACH, 1999). Neste processo, um modelo descritivo do protocolo em Lógica de Primeira Ordem é combinado com um modelo descritivo do atacante. As ações permitidas ao atacante fazem parte do modelo de ameaça Dolev-Yao (DOLEV; YAO, 1983). Como ferramenta de suporte a execução de testes sobre estes modelos, adotaremos o provador automático de teoremas SPASS versão 3.5 (WEIDENBACH et al., 2009).

Sob o ponto de vista da Lógica, o processo de formalização consiste na descrição do predicado $M(x)$ de todas as mensagens enviadas durante a execução do protocolo. Cada mensagem é representada através dos elementos que fazem parte de qualquer fórmula em Lógica de Primeira Ordem, como: quantificadores, predicados, funções, constantes e conectivos. Para relembrar os conceitos relacionados, recomenda-se rever a Seção 3.4.3.1 (página 24).

Antes de iniciar a formalização de nosso protocolo, apresentamos o provador automático de teoremas SPASS.

5.2 O PROVADOR DE TEOREMAS SPASS

Criado por Christoph Weidenbach, o SPASS é um provador automático de teoremas para Lógica de Primeira Ordem(MAX-PLANCK, 2012b). Sua primeira versão data de 1999 (WEIDENBACH et al., 1999), sendo que em 1994 começaram a ser feitas as primeiras implementações que viriam a dar origem ao provador de teoremas (MAX-PLANCK, 2012b). É um dos projetos que fazem parte do grupo de pesquisa em Automação da Lógica do Instituto Max-Planck, na Alemanha (MAX-PLANCK, 2012a).

Ao contrário do que pode parecer à primeira vista, o SPASS é um provador de teoremas de propósito geral. Isto é, não se limita à avaliação da problemática da segurança em protocolos. Qualquer problema lógico pode ser tratado no SPASS. Tudo depende da maneira como o mesmo é transcrito em termos de fórmulas lógicas, bem como os fatos que queremos extrair do problema. Na página *web* do SPASS podemos encontrar diversos experimentos e tutoriais para aprendizado desta ferramenta (MAX-PLANCK, 2012b).

Com relação a avaliação de protocolos de segurança em particular, o SPASS foi empregado na verificação de alguns exemplares. Como por exemplo: o protocolo Neuman-Stubblebine para troca de chaves (WEIDENBACH, 1999); e os protocolos de envio e confirmação de processamento da Nota Fiscal Eletrônica brasileira (MARTINA; BOAL, 2008). Demonstrando as possibilidades do SPASS, outro exemplo de trabalho é a verificação de infraestruturas virtualizadas e nuvens (BLEIKERTZ et al., 2011).

A forma de obtenção das provas pelo SPASS ocorre por meio de **refutação**, com a negação da disjunção das conjecturas. Ou em notação matemática:

$$\omega_1, \omega_2, \ldots, \omega_n \vdash \neg(\psi_1 \vee \ldots \vee \psi_n)$$

A principal justificativa da escolha do SPASS deveu-se, principalmente, a uma questão de aprendizado contínuo. A Lógica de Primeira Ordem é um modelo formal relativamente simples, estando na base de outros formalismos mais complexos. Dessa forma, a experiência a ser adquirida na modelagem formal do protocolo no SPASS tende a suavizar sua adaptação para outras estratégias de prova de teoremas, como a Lógica de Ordem Mais Alta.

A partir deste momento, passamos a apresentar os modelos lógicos representativos do protocolo escolhido a ser formalizado e verificado. Por enquanto, é dada maior ênfase à representação em notação matemática convencional. A representação dos modelos lógicos na linguagem própria do SPASS, bem como a saída da execução dos problemas, podem ser encontrados no Anexo A.

5.3 MODELO LÓGICO PRINCIPAL

Os protocolos de cadastro e autenticação são modelados em conjunta, devido ao forte relacionamento. A adoção dessa estratégia torna o teste de conjecturas mais simples na etapa de verificação, já que estamos lidando com um protocolo síncrono. Relembrando, num protocolo síncrono, para que um dado passo n_k seja executado, é necessário que os passos anteriores $(1, \ldots, n_{k-2}, n_{k-1})$ tenham sido também executados com sucesso. Em outras palavras, o fluxo de execução do protocolo deve ser seguido rigorosamente.

Devemos salientar que apesar de não caracterizar uma sequência direta na especificação de alto nível, o protocolo de autenticação é modelado imediatamente após o protocolo de cadastro. Esta decisão justifica-se no sentido de que um usuário só pode autenticar-se uma vez que ele esteja previamente cadastrado. Antes disto, todas as premissas relativas ao protocolo de autenticação são falsas.

5.3.1 Formalização Inicial

O primeiro ítem a ser levado em consideração ao modelarmos formalmente qualquer protocolo, é a definição de seus agentes (entidades) e as respectivas interações com suas bases de conhecimento.

Estas e outras ações são definidas através dos predicados abaixo.

- $E(x)$: x é uma entidade (um agente) no protocolo;
- $Stores(x, y)$: o dado x é armazenado pela entidade y;
- $Knows(x, y)$: o dado x é conhecido pela entidade y; e
- $M(x)$: a mensagem x é enviada no protocolo.

Nosso modelo lógico também faz uso das seguintes classes de funções:

- Funções de composição de mensagens: agrupam diversas mensagens individuais numa única mensagem.
  - $pair(x, y)$: agrupa as mensagens x e y; e
  - $triple(x, y, z)$: agrupa as mensagens x, y e z.
- Funções de troca de mensagens: caracterizam a troca de mensagens entre duas entidades.
  - $sent(x, y, z)$: a entidade x envia à entidade y, a mensagem z.
- Funções de chaves: indicam as chaves criptográficas e seus respectivos relacionamentos com entidades.
  - $krkey(x, y)$: a chave privada x pertence à entidade y;

  - $kukey(x, y)$: a chave pública x pertence à entidade y; e
  - $kp(x, y)$: a chave privada x e a chave pública y formam um par de chaves.

- Funções de *nonce*: indicam os *nonces* e seus respectivos relacionamentos com entidades.

  - $nonce(x, y)$: o *nonce* x é gerado pela entidade y.

- Funções de segurança: indicam o uso de primitivas criptográficas.

  - $encr(x, y)$: a mensagem x é cifrada usando a chave y; e
  - $sign(x, y)$: a mensagem x é assinada digitalmente usando a chave y.

  São também definidas algumas constantes (funções zerárias).

- Entidades (agentes) presentes no protocolo:

  - a: Administrador;
  - c: Cliente;
  - s: Servidor; e
  - d: Banco de Dados.

- Identificadores de roteamento (usados pelo servidor no redirecionamento de mensagens):

  - ida: para o Administrador;
  - idc: para o Cliente;
  - ids: para o Servidor; e
  - idd: para o Banco de Dados.

- Chaves privadas e certificados de chave pública. Aqui, a denominação *"certificado de chave pública"* é usada para enfatizar que, em nosso contexto, a chave pública está intimamente relacionada ao certificado digital. O objeto certificado digital em si não é formalizado. Para tanto, seria necessário levar em consideração todos os atributos que fazem parte de um certificado digital (dados do emissor, período de validade, etc).

  em detrimento a, somente, *"chave pública"* para enfatizar o fato da chave.

  - kra: chave privada do Administrador;
  - $cera$: certificado de chave pública do Administrador;
  - krc: chave privada do Cliente; e
  - $cerc$: certificado de chave pública do Cliente.

- *Nonces*:
  - na: *nonce* gerado pelo Administrador no cadastro; e
  - ns: *nonce* gerado pelo Servidor na autenticação.
- Dados de identificação do Cliente:
  - $codc$: código de identificação;
  - $cpfc$: o CPF; e
  - $templatec$: o *template* biométrico.

Uma vez tendo definido os elementos (predicados, funções e constantes) que fazem parte de nosso modelo lógico, passamos a definir suas fórmulas. Num primeiro momento, definimos as fórmulas que constroem a base de conhecimento inicial de cada agente.

O conhecimento do agente Administrador é representado pelas Fórmulas 1 a 7. Na primeira fórmula, indicamos que a constante a representa um agente (entidade) no protocolo.

Tanto o conhecimento privativo do próprio Administrador quanto o conhecimento público de outros agentes deve ser representado como fórmula. Por exemplo: na Fórmula 3 indicamos que o Administrador conhece sua chave privada, ao passo que, na Fórmula 5, indicamos que o Administrador conhece a chave pública do Cliente. Outros exemplos são o conhecimento do CPF do Cliente (Fórmula 6) e *nonce* N_A (Fórmula 6).

Por questões de redundância, criamos uma fórmula enfatizando o conhecimento do seu par de chaves (Fórmula 2), além das fórmulas individuais para cada uma das chaves (Fórmulas 3 e 4).

1. $E(a)$
2. $Knows(kp(krkey(kra, a), kukey(cera, a)), a)$
3. $Knows(krkey(kra, a), a)$
4. $Knows(kukey(cera, a), a)$
5. $Knows(kukey(cerc, c), a)$
6. $Knows(cpfc, a)$
7. $Knows(nonce(na, a), a)$

Em seguida, nas Fórmulas 8 a 16, descrevemos a base de conhecimento do agente Cliente. Sua descrição é, em linhas gerais, idêntica ao feito para o agente Administrador.

Novamente, convém definir o conhecimento de cada dado de identificação do Cliente em conjunto (Fórmula 13) e também em separado (Fórmulas 14 a 16).

8. $E(c)$
9. $Knows(kp(krkey(krc, c), kukey(cerc, c)), c)))$
10. $Knows(krkey(krc, c), c)$

11. $Knows(kukey(cerc, c), c)$
12. $Knows(kukey(cera, a), c)$
13. $Knows(triple(codc, cpfc, templatec), c)$
14. $Knows(codc, c)$
15. $Knows(cpfc, c)$
16. $Knows(templatec, c)$

A descrição da base de conhecimento do agente Servidor vem na sequência através das Fórmulas 17 a 20. Seu conhecimento está restrito às chaves públicas do Administrador (Fórmula 18) e Cliente (Fórmula 19), e ao *nonce* N_S gerado na autenticação (Fórmula 20).

17. $E(s)$
18. $Knows(kukey(cera, a), s)$
19. $Knows(kukey(cerc, c), s)$
20. $Knows(nonce(ns, s), s)$

O agente Banco de Dados não possui base de conhecimento inicial prévia (ela é preenchida durante a execução do protocolo). Desta forma, sua descrição lógica limita-se a afirmar que a constante d é um agente (Fórmula 21).

21. $E(d)$

O processo de definição dos axiomas não termina com as fórmulas básicas acima. Estende-se também a todos os passos de nosso protocolo, tanto nas fases de cadastro quanto de autenticação.

5.3.2 Formalização do Cadastro

Deste ponto em diante, começamos a formalizar a troca de mensagens entre os participantes do protocolo. Da mesma forma como ocorre com sua descrição em alto-nível, que possui um total de 6 passos, a formalização do cadastro conta com 6 fórmulas. Cada uma destas fórmulas representa um passo no protocolo.

No primeiro passo do cadastro (Fórmula 22), precisamos ter certeza que o Administrador a está realmente em posse de sua base de conhecimento inicial (vide Fórmulas 2 a 7). Isto é feito através da conjunção ($\wedge$) entre os predicados $Knows()$ que têm o agente a como segundo parâmetro.

Com o cumprimento desse pré-requisito, a função $sent()$ representa a mensagem trocada entre o agente a e o Servidor (agente s). O resultado dessa função é então adicionado ao conjunto de todas as mensagens enviadas durante a execução do protocolo (predicado $M()$). Através do predicado $Stores()$, o agente a armazena um novo conhecimento em sua base de armazenamento local: a correspondência entre o *nonce* na e $cpfc$ usados.

22. $Knows(kukey(cera, a), a) \wedge$
$Knows(kp(krkey(kra, a), kukey(cera, a)), a) \wedge$
$Knows(kukey(cerc, c), a) \wedge$
$Knows(cpfc, a) \wedge$
$Knows(nonce(na, a), a)$
$\rightarrow$
$M(sent(a, s, pair(encr(triple(na, cpfc, ida), cerc), idc))) \wedge$
$Stores(pair(na, cpfc), a)$

Em seguida, na Fórmula 23, notamos que a única diferença desta para a fórmula anterior, é sua pós-condição (lado direito da seta de implicação). Isto acontece por estarmos lidando com um protocolo síncrono, o qual requer que o fluxo de execução seja seguido de modo rigoroso. Em outras palavras, não é possível que uma dada Fórmula n seja validada como *verdadeiro* sem que todas as fórmulas anteriores também o tenham sido. A ocorrência desta situação já foi exposta anteriormente, mas é a partir deste momento que esta situação fica mais nítida em termos de fórmulas lógicas. Em Lógica, esta propriedade de sincronicidade é conhecida como **transitividade**.

O agente s pode receber mensagens de qualquer outro agente, inclusive de agentes desonestos (atacantes). Uma vez que seu único papel é o de atuar como redirecionador das mensagens recebidas, o agente s não provê de meios para saber a real origem de suas mensagens. Relembrando, na Seção 2.3.2 foram apresentados diversos exemplos de ataques.

Porém, o agente s tem, sim, como determinar se a mensagem recebida é uma mensagem válida e, neste caso, qual é seu destinatário. A validade da mensagem é verificada a partir da posição das funções e constantes que dela fazem parte. Por sua vez, o destinatário da mensagem é obtido através do identificador de roteamento (nesta fórmula, idc). Na eventualidade de um agente receber uma mensagem para a qual ele não estava preparado, isto resulta em negação da pré-condição (lado esquerdo da implicação).

23. $\forall xa[$
$M(sent(xa, s,$
$pair(encr(triple(na, cpfc, ida), cerc), idc)))$
$\rightarrow$
$M(sent(s, c, encr(triple(na, cpfc, ida), cerc))))]$

Revendo a Fórmula 23, verificamos que o emissor da mensagem ao agente s na pré-condição foi identificado pela variável xa. Em conjunto com o quantificador universal ($\forall$), esta variável indica o conjunto formado por todos os possíveis elementos que podem ter enviado a mensagem. Após, o agente s identifica o destinatário da mensagem através do identificador de roteamento idc. Então, na pós-condição, o agente s remove o identificador de roteamento e reenvia a mensagem ao agente correspondente, neste caso, ao Cliente (agente c).

Todos os passos do protocolo de cadastro que envolvam encaminhamento de mensagens terão tradução semelhante em fórmulas.

Continuando o fluxo de troca de mensagens, na Fórmula 24, o agente c verifica sua base de conhecimento, de maneira idêntica ao feito pelo agente a na Fórmula 22. Ainda, há uma conjunção na pré-condição que enfatiza a mensagem que deve ser recebida. O significado da variável xs é idêntico à variável xa na fórmula anterior.

A resposta como pós-condição, vem na forma de uma nova mensagem enviada ao agente s para posterior encaminhamento ao agente a. Desta vez, o destinatário final, o Administrador, é indicado pelo identificador de roteamento ida.

24. $\forall xs[$
$Knows(kp(krkey(krc, c), kukey(cerc, c)), c) \wedge$
$Knows(kukey(cera, a), c) \wedge$
$Knows(triple(codc, cpfc, templatec), c) \wedge$
$M(sent(xs, c, encr(triple(na, cpfc, ida), cerc)))$
$\rightarrow$
$M(sent(c, s, pair(encr(pair(na, sign(triple(codc, cpfc,$
$templatec), krc)), cera), ida)))]$

A Fórmula 25 redireciona a mensagem recebida pelo agente s para o agente a. O processo é idêntico ao apresentado pela Fórmula 23, apenas trocando o nome das variáveis e identificadores.

25. $\forall xc[$
$M(sent(xc, s, pair(encr(pair(na, sign(triple(codc, cpfc,$
$templatec), krc)), cera), ida)))$
$\rightarrow$
$M(sent(s, a, encr(pair(na, sign(triple(codc, cpfc,$
$templatec), krc)), cera)))]$

Na Fórmula 26, o corpo de conhecimento do agente a é verificado uma segunda vez, de modo a garantir que não houve mudanças desde a Fórmula 22. Através do predicado $Stores()$ é possível verificar se o *nonce* na foi utilizado em conjunto com o $cpfc$ no início do cadastro (Fórmula 22). Também está presente na pré-condição, uma conjunção para testar a mensagem recebida a partir de algum agente xs.

A pós-condição vem na forma de uma nova função $sign(\ldots, kra)$ ao redor da função $sign(triple(\ldots), krc)$ recebida originalmente. Então, a mensagem duplamente assinada, $sign(sign(\ldots, krc), kra)$ é concatenada com o identificador de roteamento idd, e enviada ao agente s.

26. $\forall xs[$
$Knows(nonce(na, a), a) \wedge$
$Knows(cpfc, a) \wedge$

$Knows(kp(krkey(kra, a), kukey(cera, a)), a) \wedge$
$Knows(kukey(cerc, c), a) \wedge$
$Stores(pair(na, cpfc), a) \wedge$
$M(sent(xs, a, encr(pair(na, sign(triple(codc, cpfc,$
$templatec), krc)), cera)))$
$\rightarrow$
$M(sent(a, s, pair(sign(sign(triple(codc, cpfc, templatec),$
$krc), kra), idd)))]$

No último passo do protocolo de cadastro (Fórmula 27), o agente s efetua o encaminhamento da mensagem de maneira idêntica às Fórmulas 23 e 25.

Porém, ocorre uma diferença substancial em comparação aos encaminhamentos anteriores. Quando o Banco de Dados (agente d) recebe a mensagem, este também armazena o *token* duplamente assinado ($sign(sign(triple(codc, cpfc, templatec), krc), kra)$) e o $cpfc$ em seu conjunto de armazenamento (predicado $Stores()$). Deste modo, transpomos para o modelo lógico, a principal característica do agente d, que é o armazenamento de longo prazo.

27. $\forall xa[$
$M(sent(xa, s, pair(sign(sign(triple(codc, cpfc, templatec),$
$krc), kra), idd)))$
$\rightarrow$
$M(sent(s, d, sign(sign(triple(codc, cpfc, templatec),$
$krc), kra))) \wedge$
$Stores(cpfc, d) \wedge$
$Stores(sign(sign(triple(codc, cpfc, templatec),$
$krc), kra), d)]$

5.3.3 Formalização da Autenticação

Agora começamos a modelar a fase de autenticação de nosso protocolo. Da modo semelhante à formalização da fase de cadastro, cada passo na descrição de alto-nível é representado através de uma fórmula.

A modelagem do protocolo de autenticação inicia com a comprovação da posse da base de conhecimento do agente s (Fórmula 28).

Com sua base de conhecimento verificada, o agente s é capaz de enviar ao agente c, o *nonce* ns cifrado com o certificado de chave pública $cerc$. Além disso, a correspondência do *nonce* ns e o certificado digital $cerc$ do cliente em conexão, é armazenado no predicado $Stores()$ do agente s.

28. $Knows(kukey(cerc, c), s) \wedge$
$Knows(nonce(ns, s), s)$
$\rightarrow$

$M(sent(s, c, encr(ns, cerc))) \wedge$
$Stores(pair(ns, cerc), s)$

Continuando, na Fórmula 29, o agente c verifica se está em posse dos dados para sua identificação ($cpfc$ e $templatec$), além de confirmar se está em posse de seu par de chaves.

Em resposta (pós-condição), o agente c armazena o *nonce* ns em seu predicado $Stores()$. Ele também envia ao agente s, a concatenação dos dados $cpfc$, ids e $templatec$. Esta concatenação é cifrada (função $encr()$) com o *nonce* ns recebido na fórmula anterior.

29. $\forall xs[$
$Knows(cpfc, c) \wedge$
$Knows(templatec, c) \wedge$
$Knows(kp(krkey(krc, c), kukey(cerc, c)), c) \wedge$
$M(sent(xs, c, encr(ns, cerc)))$
$\rightarrow$
$Stores(pair(ns, xs), c) \wedge$
$M(sent(c, s, encr(triple(cpfc, ids, templatec), ns)))]$

Em seguida, na Fórmula 30, o agente s é capaz de decifrar o conteúdo da mensagem recebida com o *nonce* ns. Este *nonce* foi armazenado no predicado $Stores()$ no primeiro passo da autenticação (Fórmula 28).

Em posse do $cpfc$, o agente s, realiza uma consulta ao Banco de Dados d, através do envio de uma mensagem simples.

30. $\forall xc[$
$Stores(pair(ns, cerc), s) \wedge$
$M(sent(xc, s, encr(triple(cpfc, ids, templatec), ns)))$
$\rightarrow$
$M(sent(s, d, cpfc))]$

O último passo da autenticação (Fórmula 31) é direto. O agente d recebe o $cpfc$ e verifica se este e o *token* duplamente assinado, $sign(sign(triple(codc, cpfc, templatec), krc), kra)$, encontram-se em seu conjunto de armazenamento (predicado $Stores()$). Em caso positivo, isso indica que o protocolo de cadastro foi executado com sucesso para o agente c (Fórmula 27).

Ao contrário do que pode parecer à primeira vista, a decisão de autenticação compete ao agente s e não ao agente d. Ao agente d compete somente o envio do *token* duplamente assinado ao agente s (pós-condição).

31. $\forall xs[$
$Stores(cpfc, d) \wedge$
$Stores(sign(sign(triple(codc, cpfc,$

$templatec), krc), kra), d) \wedge$
$M(sent(xs, d, cpfc))$
$\rightarrow$
$M(sent(d, s, sign(sign(triple(codc, cpfc, templatec),$
$krc), kra))))]$

Mesmo não expresso no modelo lógico, o agente s é o responsável pelo efetivo aceite ou recusa da conexão. Internamente, ele faz a verificação das assinaturas digitais do último *token* e a comparação um-para-um dos *templates* biométricos. Quando todas estas verificações são bem-sucedidas, o agente c é considerado genuíno e sua conexão é aceita.

Em notação matemática, representamos o conjunto de fórmulas que compõem nosso modelo lógico principal por:

$$\Gamma_{principal} = \{ \text{ Fórmula 1}, \ldots, \text{Fórmula 31 } \}$$

5.4 MODELO LÓGICO DO ATACANTE

O modelo do atacante refere-se ao conjunto de ações e conhecimento a partir dos quais um atacante pode fazer uso na tentativa de comprometer o correto funcionamento do protocolo.

Para o entendimento do formalismo que segue, é necessário definir dois novos elementos lógicos, que serão adicionados à formalização inicial. São eles:

- A constante i, indicando o agente Atacante;
- As constantes $icodc$, $icpfc$ e $itemplatec$, representando dados do atacante enquanto atuando como um agente válido; e
- O predicado $Im(x)$, representando o conhecimento adquirido pelo atacante. Este predicado tem funcionamento idêntico ao predicado $M()$.

A seguir, começamos a elaborar as fórmulas que fazem parte do modelo lógico do atacante adotado neste trabalho, as quais também entram na definição dos axiomas. Este modelo segue em linhas gerais, o modelo proposto por Weidenbach (WEIDENBACH, 1999), fazendo uso das capacidades de ataque definidas pelo modelo de ameaça Dolev-Yao (DOLEV; YAO, 1983). Algumas extensões são adicionadas devido às características particulares de nosso protocolo (como cifragem com *nonces*).

Primeiramente, definimos todas as informações publicamente disponíveis ao atacante (Fórmulas 33 e 34). Os dados de identificação do atacante encontram-se desde o início, presentes no predicado $Im()$ (Fórmula 35).

Em seguida, transcrevemos para fórmulas lógicas todas as habilidades do atacante com relação ao monitoramento e manipulação do tráfego de rede. Por exemplo: gravar todas as mensagens (Fórmula 36); dividir

uma mensagem em pedaços menores (Fórmulas 37 e 38); montar uma nova mensagem a partir de pedaços menores (Fórmulas 39 e 40); e enviar mensagens falsas a partir do conhecimento adquirido (Fórmula 41).

32. $E(i)$
33. $Knows(kukey(cerc, c), i)$
34. $Knows(kukey(cera, a), i)$
35. $Im(triple(icodc, icpfc, itemplatec))$
36. $\forall x, y, w[M(sent(x, y, w)) \rightarrow Im(w)]$
37. $\forall u, v[Im(pair(u, v)) \rightarrow Im(u) \wedge Im(v)]$
38. $\forall u, v, w[Im(triple(u, v, w)) \rightarrow Im(u) \wedge Im(v) \wedge Im(w)]$
39. $\forall u, v[Im(u) \wedge Im(v) \rightarrow Im(pair(u, v))]$
40. $\forall u, v, w[Im(u) \wedge Im(v) \wedge Im(w) \rightarrow Im(triple(U, V, W))]$
41. $\forall u, x, y[Im(u) \wedge E(x) \wedge E(y) \rightarrow M(sent(x, y, u))]$

Em segundo lugar, nas fórmulas abaixo, são definidas as operações criptográficas possíveis de utilização por um atacante na tentativa de quebra do protocolo.

Por ser capaz de gravar todas as mensagens trocadas (Fórmula 36), o atacante pode, exaustivamente, testar todas as possibilidades de combinação até encontrar uma chave (Fórmulas 42 e 43).

Nosso protocolo conta com *nonces* sendo usados como chaves em alguns momentos. Desta forma, também é interessante adicionar uma operação de teste exaustivo em busca de *nonces* (Fórmula 44).

Além disso, o atacante também é capaz de criar mensagens cifradas ou assinadas com as chaves que ele possui em seu conhecimento (Fórmulas 45 e 46).

42. $\forall u, v[Im(u) \wedge E(v) \rightarrow Knows(krkey(u, v), i)]$
43. $\forall u, v[Im(u) \wedge E(v) \rightarrow Knows(kukey(u, v), i)]$
44. $\forall u, v[Im(u) \wedge E(v) \rightarrow Knows(nonce(u, v), i)]$
45. $\forall u, v, x[Im(u) \wedge Knows(kukey(v, x), i) \wedge E(x) \rightarrow Im(encr(u, v))]$
46. $\forall u, v, x[Im(u) \wedge Knows(krkey(v, x), i) \wedge E(x) \rightarrow Im(sign(u, v))]$

Ademais, a Fórmula 47 permite ao atacante aprender o conteúdo das mensagens cifradas a partir das chaves que possui acesso. A mesma operação também vale para *nonces* (Fórmula 48). Já a Fórmula 49 permite ao atacante ter acesso ao conteúdo de mensagens assinadas. Isto não implica na verificação da assinatura em si, mas somente no acesso ao conteúdo que foi assinado.

47. $\forall u, v, w, x[Im(encr(u, v)) \wedge Knows(kp(krkey(w, x), kukey(v, x)), i) \wedge E(x) \rightarrow Im(u))]$

48. $\forall u, v, w[Im(encr(u, v)) \wedge Knows(nonce(v, w), i) \wedge E(w) \rightarrow Im(u))]$

49. $\forall u, v[Im(sign(u, v)) \rightarrow Im(u)]$

Em notação matemática, representamos o conjunto de fórmulas que fazem parte do modelo do atacante, como:

$$\Gamma_{atacante} = \{ \text{ Fórmula 32, . . . ,Fórmula 49 } \}$$

E nosso modelo lógico estendido:

$$\Gamma_{estendido} = \Gamma_{principal} \cup \Gamma_{atacante}$$

De modo complementar, estabelecemos duas hipóteses a serem testadas sobre nosso modelo lógico estendido. O principal objetivo é descobrir se o atacante possui vantagens reais na eventualidade de conseguir acesso ao material criptográfico sigiloso.

O uso destas hipóteses é apenas pontual, não sendo levadas em consideração no modelo lógico estendido.

50. $Knows(krkey(kra, a), i) \wedge Knows(kp(krkey(kra, a), kukey(cera, a)), i)$

51. $Knows(krkey(krc, c), i) \wedge Knows(kp(krkey(krc, c), kukey(cerc, c)), i)$

Em síntese, buscamos verificar se o atacante consegue comprometer o protocolo estando em posse das chaves privadas do Administrador (Fórmula 50) ou do Cliente (Fórmula 51). Queremos ter certeza que nosso protocolo permanece seguro mesmo com algumas premissas violadas. Neste caso, a premissa violada é *"somente os donos dos smartcards possuem acesso à sua respectiva chave privada"*.

Terminamos aqui a definição do modelo lógico de nosso protocolo. Sua representação no provador de teoremas SPASS encontra-se na Seção A.2 (modelo lógico principal) e Seção A.3 (modelo lógico estendido).

Passamos agora à próxima etapa da verificação formal, que consiste na elaboração e teste de conjecturas.

5.5 ELABORAÇÃO DE CONJECTURAS

Uma vez que as conjecturas sejam provadas, isto é, consideradas verdadeiras, podemos estabelecer fatos (afirmações certas) sobre nosso protocolo. Nem todos os fatos extraídos são, necessariamente, resultado da execução de conjecturas. A saturação do modelo lógico é um exemplo de fato importante obtido a partir da execução do modelo sem conjecturas. Conjecturas provadas são denotadas pelo símbolo $\vdash$ e, não provadas, pelo símbolo $\nvdash$.

Um total de 7 fatos são extraídos a partir dos modelos lógicos criados. Veremos cada um deles a seguir. A saída retornada pelo SPASS para cada fato encontra-se na Seção A.4.

Fato 1: Saturação do modelo lógico principal (Fórmulas 1 a 31).

O primeiro fato extraído de nosso modelo não é propriamente o resultado do teste de uma conjectura. Mas sim, a saturação do modelo lógico. Isto indica que as características do modelo permanecem estáveis durante sua execução sem conjecturas.

A ocorrência deste fato é de especial importância para avaliar a decidibilidade do modelo, ou seja, se o mesmo retorna alguma resposta ou executa indefinidamente.

Fato 2: $\Gamma_{principal} \vdash \exists u[M(sent(d, s, sign(sign(triple(codc, u, templatec), krc), kra)))]$

O segundo fato representa um teste para termos certeza que nosso protocolo foi modelado corretamente. Deve ser possível confirmar o fato básico presente no protocolo: o cadastro e a consequente autenticação de um usuário qualquer.

Para isso, criamos , uma conjectura que representa a execução da pós-condição presente na última implicação do modelo lógico principal (Fórmula 31). Pela propriedade da transitividade, esta conjectura só pode ser confirmada como fato quando todas as implicações entre fórmulas estejam corretas. Até chegarmos a esta conjectura final, criamos diversas conjecturas intermediárias de forma a testar cada passo isoladamente.

Como o SPASS é realmente capaz de encontrar uma prova para esta conjectura, comprovamos a correta modelagem de nosso protocolo.

Fato 3: Saturação do modelo lógico estendido (Fórmulas 1 a 49).

A execução do modelo lógico estendido também resulta em sua saturação. Este fato é bastante positivo pois, *a priori*, o protocolo não sofre de problemas estruturais graves. Um exemplo desta situação pode ser encontrado na verificação do protocolo Neuman-Stubblebine, onde a não saturação do modelo lógico estendido levou a confirmação de ataques (WEIDENBACH, 1999).

Fato 4: $\Gamma_{estendido} \nvdash M(sent(i, d, sign(sign(triple(icodc, icpfc, itemplatec), krc), kra)))$

Neste fato, comprova-se que o atacante não tem capacidade para gerar mensagens falsas de conclusão de cadastro. Ele não consegue criar um *token* duplamente assinado com dados forjados e encaminhá-lo ao Banco de Dados (agente d).

Fato 5: $\Gamma_{estendido} \cup \{$ Fórmula 50 $\} \nvdash M(sent(i, d, sign($ $sign(triple(icodc, icpfc, itemplatec), krc), kra)))$

Adicionamos a hipótese do atacante ter acesso à chave privada do Administrador (Fórmula 50). Mesmo com esta vantagem, o atacante ainda não é capaz de gerar um registro com dados falsos.

O atacante busca, através das operações de ataque disponíveis ($\Gamma_{atacante}$), ter acesso à chave privada krc para gerar a primeira assinatura. Isto não ocorre, e a posse da chave privada do administrador (kra) é insuficiente para que o protocolo seja comprometido.

Outra linha de ação é o atacante assinar, no lugar de krc com, digamos kri. Para que isto funcione, o atacante é obrigado a utilizar um certificado e-CPF. Segundo as premissas de nosso protocolo (Seção 4.2.1), somente certificados emitidos pela ICP-Brasil são aceitos. Ao fazer uso de um certificado e-CPF, o atacante acaba por divulgar sua real identidade (através do CPF dentro do cartão).

Fato 6: $\Gamma_{estendido} \cup \{$ Fórmula 51 $\} \nvdash M(sent(i, d, sign($
$sign(triple(icodc, icpfc, itemplatec), krc), kra)))$

Testamos agora a hipótese inversa: o atacante tem acesso à chave privada do Cliente. Novamente, o atacante não consegue gerar uma solicitação de cadastro contendo seus dados forjados.

Fato 7: $\Gamma_{estendido} \cup \{$ Fórmula 50, Fórmula 51 $\} \vdash M(sent(i, d,$
$sign(sign(triple(icodc, icpfc, itemplatec), krc), kra)))$

O último fato faz uso das duas hipóteses em conjunto: o atacante tem acesso simultâneo às chaves privadas do Administrador e Cliente. Neste cenário, ele consegue gerar uma mensagem com dados de identificação falsos.

Também é possível encaminhar a mensagem ao Banco de Dados, porém não há certeza quanto ao aceite ou recusa do agente d em armazenar o *token* duplamente assinado. Lembrando que, em nosso modelo, o agente c é considerado registrado somente quando seu *token* duplamente assinado é inserido no predicado $Stores()$ do agente d (Fórmula 27).

Para que essa possibilidade de teste de conjectura seja possível, é necessário fazer mudanças estruturais em nosso modelo lógico. Particularmente, no que se refere a presença de variáveis no lugar de constantes na formalização do cadastro e autenticação (Fórmulas 22 a 31).

Mudanças como essa, envolvem um nível significativo de esforço e análise lógica. Por esta razão, optou-se por deixar eventuais aprimoramentos da especificação formal como trabalho futuro (Seção 6.1). Mesmo com esta aparente limitação, fomos capazes de obter um total de 7 fatos a respeito das principais propriedades de nosso protocolo.

5.6 CONCLUSÃO

O capítulo que se encerra apresentou o processo de verificação formal do protocolo proposto sob a ótica da Lógica de Primeira Ordem, com assistência do provador automático de teoremas SPASS.

Este processo envolveu, não apenas a formalização do protocolo, mas também a formalização das capacidades de um atacante qualquer. Em seguida, procedemos com o teste de conjecturas sobre os modelos lógicos criados. O resultado dos testes de conjecturas permitiu extrair fatos sobre a segurança do protocolo como um todo.

Porém, a verificação realizada não foi exaustiva. Uma limitação aos modelos lógicos atuais reside na impossibilidade de expressar situações de paralelismo. Por exemplo: *"dois Administradores iniciam o cadastro de um mesmo Cliente"* ou *"dois Clientes informam o mesmo* Cod_C*"*. Para expressar situações desse tipo, é necessário fazer mudanças estruturais significativas: criação de novos predicados e substituição das constantes por variáveis sempre que o predicado $M(x)$ é utilizado. Estas modificações foram consideradas propostas de extensão e, por isso, não foram efetivamente incluídas no trabalho.

Ainda, para que um protocolo de segurança seja de fato considerado livre de falhas, é necessário expandir a lista de métodos de verificação usados. Logo, deve-se ir além da Lógica de Primeira Ordem. Pode-se usar, por exemplo, outros métodos de verificação de protocolos, conforme apresentado na Seção 3.4.

O uso concomitante de diferentes técnicas de verificação, permite que haja um melhor balanceamento entre seus pontos fortes e fracos. O uso de diferentes ferramentas dentro de uma mesma técnica também é aconselhável, atuando como um segundo nível de confirmação dos fatos obtidos.

Mesmo que um ataque venha a ser descoberto, é possível que este não represente ameaça real. É necessário validá-lo contra premissas externas relativas ao ambiente no qual o protocolo é implantado. A impossibilidade do atacante utilizar um certificado digital próprio constitui uma premissa forte que impede avanços do atacante (conforme explicação do Fato 5). Ainda, é importante estar atento às limitações da análise formal Seção 3.5.

Com o término do desenvolvimento deste trabalho, nos encaminhamos ao seu encerramento.

6 CONSIDERAÇÕES FINAIS

Neste trabalho, apresentamos um conjunto de protocolos para cadastro e autenticação de usuários. A partir do uso de diferentes fatores de autenticação, este protocolo provê níveis adicionais de segurança. Como suas principais características, destacamos: a presença de diferentes participantes, de forma a não apenas prover compartilhamento de responsabilidades, mas também prevenir ataques baseados num único ponto fraco; e a adoção de premissas vinculadas a diferentes níveis de execução do protocolo (distribuição da confiança, ancoramento da confiança e prova de posse).

Além disso, para nos ajudar a determinar se o protocolo contém alguma vulnerabilidade em sua descrição de alto-nível (troca de mensagens), construímos seu modelo lógico. Desenvolvido em Lógica de Primeira Ordem, este modelo permite a avaliação das propriedades de segurança através do teste de conjecturas representando ações de um atacante potencial. A confirmação ou refutação destas conjecturas (e em consequência, dos ataques) é feita de forma automatizada, com o auxilio de um provador de teoremas, em nosso caso, o SPASS (WEIDENBACH et al., 2009).

Embora nenhuma vulnerabilidade tenha sido encontrada até o momento, nossa proposta ainda requer uma análise mais profunda para termos certeza que o conjunto de protocolos fornece um nível de segurança adequado. Mesmo não sendo considerado definitivo, o modelo lógico criado servirá como ponto de partida à uma eventual expansão do processo de verificação formal.

Relembrando a revisão da literatura, não foram encontradas referências a outras propostas de protocolos com características semelhantes ao nosso. Outro ponto positivo, é que, até onde sabemos, este é o primeiro trabalho a avaliar um protocolo biométrico sob a ótica da Lógica Formal.

Desde o início deste projeto de pesquisa, diversas possibilidades de trabalhos de extensão vêm sendo identificadas. Por questões de espaço ou por fugirem um pouco do escopo, não foram incluídas em nossa proposta original. São então apresentadas a seguir, na forma de trabalhos futuros.

6.1 TRABALHOS FUTUROS

O primeiro trabalho futuro vislumbrado é a adição de novas conjecturas de ataque ao modelo lógico criado. Podemos também refinar o modelo, tornando-o ainda mais preciso do ponto de vista semântico, evitando redundância e contribuindo para a identificação de ataques, não apenas teóricos, mas também práticos.

Tal como mencionado na Seção 5.6, para que uma verificação formal seja considerada exaustiva, levando a fortes garantias de segurança, é necessário também que diferentes técnicas sejam utilizadas.

Neste sentido, a formalização do protocolo em Lógica de Ordem-Mais-Alta com assistência do provador de teoremas Isabelle (NIPKOW et al., 2002) representa um passo importante na obtenção de um maior entendimento das reais capacidades e limitações do protocolo. Não podemos nos esquecer, contudo, da existência de outras técnicas de verificação, como *Model Checking* e *Strand Spaces Model*.

Ainda, vislumbramos a extensão do escopo da verificação tratada neste trabalho. Uma das frentes em que isso pode ser feito, é a análise da cerimônias envolvidas no protocolo. Cerimônia é uma extensão ao conceito de protocolo na qual a interação entre agentes humanos também é levada em consideração (ELLISON, 2007). As premissas iniciais de nosso protocolo, Distribuição e Ancoramento da Confiança, são os exemplo mais nítidos de cerimônias neste trabalho.

Outro aspecto importante a ser analisado é a questão da privacidade dos *templates* biométricos dos usuários cadastrados. Na proposta atual, os mesmos são armazenados em bancos de dados remotos, porém esta abordagem não é a mais apropriada em termos de proteção de dados pessoais (TANG et al., 2008). Uma alternativa a esta abordagem, conhecida como *match on card*, consiste no armazenamento e comparação do dado biométrico no próprio *smartcard* do usuário (SANCHEZ-REILLO et al., 2002).

Neste contexto, o RIC (Registro de Identidade Civil) (BRASIL, 2012), novo método de identificação pessoal criado pelo Governo Brasileiro, permite a implementação de uma abordagem *match on card*. Porém, o mesmo ainda não está suficientemente disseminado na população, inviabilizando por ora, a criação de aplicações em larga escala que dele façam uso.

O funcionamento do protocolo também pode ser otimizado, principalmente em termos de velocidade. A maior parte de nossa proposta baseia-se na criptografia assimétrica, a qual é particularmente lenta nas operações de cifragem e assinatura digital se comparada à criptografia simétrica. Dessa forma, a elaboração de uma versão alternativa que busque reduzir o uso das chaves públicas e privadas, representa um novo trabalho de extensão. Apesar de parecer remoto, esse problema tende a adquirir relevância quando o protocolo passa a lidar com um grande número de usuários (na ordem de centenas, ou mesmo, milhares).

Por fim, qualquer protocolo, quer seja ele, de segurança ou não, público ou privativo, deve contar com verificações formais em seu processo de desenvolvimento. Esta iniciativa, apesar de possuir limites (Seção 3.5) fornece uma linha de defesa adicional a ataques. Em particular, isto é de especial interesse em protocolos criados e mantidos por governos que, por lidarem com dados de milhões de cidadãos, demandam altos níveis de aderência a seus objetivos.

No cenário brasileiro, apontamos os seguintes protocolos como potenciais beneficiários dos resultados de verificações formais: o protocolo de transmissão da Declaração de Imposto sobre a Renda da Pessoa Física (IRPF) (RECEITA FEDERAL, 2012b); a Urna Eletrônica (TSE, 2010); o ASI-HSM (LABSEC, 2012); e protocolos internos de órgãos públicos.

REFERÊNCIAS

ABADI, M.; NEEDHAM, R. Prudent engineering practice for cryptographic protocols. *Software Engineering, IEEE Transactions on*, v. 22, n. 1, p. 6 –15, jan. 1996. ISSN 0098-5589.

ANDERSON, R. The formal verification of a payment system. In: *Industrial-Strength Formal Methods in Practice*. [S.l.]: Springer, 1997. p. 43–52.

ANDERSON, R.; NEEDHAM, R. Robustness principles for public key protocols. In: . [S.l.]: Springer-Verlag, 1995. p. 236–247.

ANDERSON, R. J. *Security engineering - a guide to building dependable distributed systems (2. ed.)*. [S.l.]: Wiley, 2008. I-XL, 1-1040 p. ISBN 978-0-470-06852-6.

ARMANDO, A.; COMPAGNA, L. SAT-based model-checking for security protocols analysis. *International Journal of Information Security*, v. 7, n. 1, p. 3–32, set. 2007. ISSN 1615-5262, 1615-5270. Disponível em: <http://www.springerlink.com/index/10.1007/s10207-007-0041-y>.

ARSAC, W. et al. Multi-attacker protocol validation. *Journal of Automated Reasoning*, v. 46, n. 3, p. 353–388, 2011. ISSN 0168-7433. 10.1007/s10817-010-9185-y. Disponível em: <http://dx.doi.org/10-.1007/s10817-010-9185-y>.

AVISPA. *The HLPSL Tutorial A Beginner's Guide to Modelling and Analysing Internet Security Protocols (Version 1.1)*. jun. 2006. Acesso em: 25 de Julho de 2012. Disponível em: <http://www.avispa-project-.org/package/tutorial.pdf>.

AVISPA. *The AVISPA Project*. 2012. Acesso em: 25 de Julho de 2012. Disponível em: <http://www.avispa-project.org/>.

BASIN, D.; MöERSHEIM, S.; VIGANò, L. An on-the-fly model-checker for security protocol analysis. In: *Eight ESORICS*. [S.l.]: Springer-Verlag, Berlin Germany, 2003. (Lecture Notes in Computer Science, v. 2808), p. 253–270.

BATCHELOR, C. Real costs: Targeted attacks drive growth in the hacking economy - FT.com. *Financial Times*, maio 2012.

BELLA, G. Inductive verification of smartcard protocols. *IOS Journal of Computer Security*, v. 11, n. 1, p. 87–132, 2003. Disponível em: <http://www.cl.cam.ac.uk/%7Elcp/papers/Bella/smartcards-jcs.pdf>.

BELLA, G. *Formal Correctness of Security Protocols*. [S.l.]: Springer, 2007. (Information Security and Cryptography).

BELLA, G.; BISTARELLI, S.; MASSACCI, F. Retaliation: Can we live with flaws? In: THOMAS, J.; ESSAAIDI, M. (Ed.). *IOS "Information Assurance and Computer Security"*. [S.l.]: IOS Press, 2006, (NATO Security through Science Series, v. 6). ISBN 1-58603-678-5.

BELLA, G.; MASSACCI, F.; PAULSON, L. Verifying the SET registration protocols. *Selected Areas in Communications, IEEE Journal on*, v. 21, n. 1, p. 77 – 87, jan. 2003. ISSN 0733-8716.

BELLA, G.; MASSACCI, F.; PAULSON, L. Verifying the SET purchase protocols. *Journal of Automated Reasoning*, v. 36, n. 1, p. 5–37, 2006. ISSN 0168-7433. 10.1007/s10817-005-9018-6. Disponível em: <http://dx.doi.org/10.1007/s10817-005-9018-6>.

BLANCHET, B. An efficient cryptographic protocol verifier based on prolog rules. In: *Computer Security Foundations Workshop, 2001. Proceedings. 14th IEEE*. [S.l.: s.n.], 2001. p. 82 –96.

BLEIKERTZ, S.; GROß, T.; MöDERSHEIM, S. Automated verification of virtualized infrastructures. In: *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. New York, NY, USA: ACM, 2011. (CCSW '11), p. 47–58. ISBN 978-1-4503-1004-8. Disponível em: <http://doi.acm.org/10.1145/2046660.2046672>.

BODNAR, P. A solution to remote biometric identification. In: *Information Technology, 2008. IT 2008. 1st International Conference on*. [S.l.: s.n.], 2008. p. 1 –4.

BOICHUT, Y.; HéAM, C.; KOUCHNARENKO, O. *Automatic Verification of Security Protocols Using Approximations*. [S.l.], out. 2005. Disponível em: <http://www.avispa-project.org/papers/RR-5727-.ps>.

BOUNCY CASTLE. *The Legion of the Bouncy Castle*. ago. 2012. Acesso em: 1o de Agosto de 2012. Disponível em: <http://www-.bouncycastle.org/>.

BRAINARD, J. et al. Fourth-factor authentication: somebody you know. In: *Proceedings of the 13th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2006. (CCS '06), p. 168–178. ISBN 1-59593-518-5.

BRASIL. *Novo Registro de Identidade Civil (RIC)* -. 2012. Acesso em: 8 de Agosto de 2012. Disponível em: <http://www.brasil.gov.br/para-/servicos/documentacao/conheca-o-novo-registro-de-identidade-civil-ric>.

BURROWS, M.; ABADI, M.; NEEDHAM, R. A logic of authentication. *ACM Trans. Comput. Syst.*, v. 8, n. 1, p. 18–36, fev. 1990. ISSN 0734-2071. Disponível em: <http://doi.acm.org/10.1145/77648.77649>.

CAVOUKIAN, A. *Fingerprint Biometric Systems: Ask the Right Questions Before You Deploy*. Ontario, Canada, jul. 2008. Disponível em: <http://www.ipc.on.ca/images/Resources/fingerprint-biosys.pdf>.

CHEVALIER, Y.; VIGNERON, L. Automated unbounded verification of security protocols. In: BRINKSMA, E.; LARSEN, K. G. (Ed.). *Computer Aided Verification*. [S.l.: s.n.], 2002. (Lecture Notes in Computer Science, v. 2404), p. 324–337.

CLARKE, E. Model checking – my 27-year quest to overcome the state explosion problem. In: CERVESATO, I.; VEITH, H.; VORONKOV, A. (Ed.). *Logic for Programming, Artificial Intelligence, and Reasoning*. Springer Berlin / Heidelberg, 2008, (Lecture Notes in Computer Science, v. 5330). p. 182–182. ISBN 978-3-540-89438-4. 10.1007/978-3-540-89439-1_13. Disponível em: <http://dx.doi.org/10.1007/978-3-540-89439-1_13>.

CLARKE, E. M.; WING, J. M. Formal methods: state of the art and future directions. *ACM Computing Surveys*, v. 28, n. 4, p. 626–643, dez. 1996. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145-/242223.242257>.

COOK, R. T. *A Dictionary of Philosophical Logic*. [S.l.]: Edinburgh University Press, 2009. ISBN 978 0 7486 2559 8.

DIERKS, T.; ALLEN, C. *RFC 2246: The TLS Protocol Version 1*. jan. 1999.

DIGITAL PERSONA. *DigitalPersona® U.are.U® 4000B Reader USB Fingerprint Reader*. 2005. Acesso em: 1o de Agosto de 2012. Disponível em: <http://www.signtechbiometric.com/pdf/digitalpersona-/uareu4000breader.pdf>.

DOLEV, D.; YAO, A. On the security of public key protocols. *Information Theory, IEEE Transactions on*, v. 29, n. 2, p. 198 – 208, mar. 1983. ISSN 0018-9448.

ELLISON, C. *Ceremony Design and Analysis*. 2007. Cryptology ePrint Archive, Report 2007/399. Acesso em: 29 de Agosto de 2012. Disponível em: <http://eprint.iacr.org/>.

FABREGA, F.; HERZOG, J.; GUTTMAN, J. Strand spaces: Why is a security protocol correct? In: *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*. [S.l.: s.n.], 1998. p. 160 –171.

FABREGA, F.; HERZOG, J.; GUTTMAN, J. Mixed strand spaces. In: *Computer Security Foundations Workshop, 1999. Proceedings of the 12th IEEE*. [S.l.: s.n.], 1999. p. 72 –82.

FIELDING, R. et al. *RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*. 1999. Disponível em: <http://www.rfc.net/rfc2616.html>.

FINGERTECH. *FingKey Hamster DX*. 2012. Acesso em: 1o de Agosto de 2012. Disponível em: <http://www.nitgen.com.br/produtos/18-fingkey-hamster-dx>.

FORMAL METHODS (EUROPE) LTD.; OXFORD UNIVERSITY. *Failures-Divergences Refinement – FDR2 User Manual*. 9. ed. [S.l.: s.n.], 2010.

FUTRONIC. *Fingerprint Recognition Software Development Kit(SDK) for MS Windows*. 2012. Acesso em: 1o de Agosto de 2012. Disponível em: <http://www.futronic-tech.com/product_sdk_win.html>.

FUTRONIC. *Futronic's FS82 USB2.0 Fingerprint Smart Card Reader*. 2012. Acesso em: 1o de Agosto de 2012. Disponível em: <http://www.futronic-tech.com/product_fs82.html>.

GLIGOR, V. et al. Logics for cryptographic protocols-virtues and limitations. In: *Computer Security Foundations Workshop IV, 1991. Proceedings*. [S.l.: s.n.], 1991. p. 219 –226.

GOLLMANN, D. What do we mean by entity authentication? In: *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. [S.l.: s.n.], 1996. p. 46 –54.

GRIAULE. *Griaule Biometrics*. 2012. Acesso em: 1o de Agosto de 2012. Disponível em: <http://www.griaulebiometrics.com/>.

HAO, F.; ANDERSON, R.; DAUGMAN, J. *Combining cryptography with biometrics effectively*. [S.l.], jul. 2005.

HAO, F.; CLARKE, D. *Security Analysis of a Multi-Factor Authenticated Key Exchange Protocol*. [S.l.], fev. 2012.

HARTUNG, D.; BUSCH, C. Biometric transaction authentication protocol. In: *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*. [S.l.: s.n.], 2010. p. 207 –215.

HOARE, C. *Communication Sequential Processes*. Prentice-Hall, 1985. Acesso em: 26 de Julho de 2012. Disponível em: <http://www.usingcsp-.com/cspbook.pdf>.

HUTH, M.; RYAN, M. *Logic in Computer Science - Modelling and Reasoning about Systems*. 2. ed. [S.l.]: Cambridge University Press, 2004. ISBN 978-0-521-54310-1.

ILO. *Seafarers' Identity Documents Convention (Revised), 2003 (No. 185)*. Geneva, 2006. Disponível em: <http://www.ilo.org/wcmsp5-/groups/public/—ed_norm/—normes/documents/genericdocument-/wcms_144265.pdf>.

INCITS. *INCITS 378-2009: Information Technology - Finger Minutiae Format for Data Interchange*. [S.l.: s.n.], 2009.

INRIA. *Welcome ! | The Coq Proof Assistant*. 2012. Acesso em: 13 de Agosto de 2012. Disponível em: <http://coq.inria.fr/>.

ISO. *ISO/IEC 19794-2:2011 Information technology – Biometric data interchange formats – Part 2: Finger minutiae data*. Switzerland: [s.n.], 2011.

ISO. *ISO/IEC 7816-1:2011 Identification cards – Integrated circuit cards – Part 1: Cards with contacts – Physical characteristics*. Switzerland: [s.n.], 2011.

ISO. *ISO - International Organization for Standardization*. 2012. Acesso em: 1o de Agosto de 2012. Disponível em: <http://www.iso.org/iso-/home.html>.

JAIN, A.; ROSS, A.; PRABHAKAR, S. An introduction to biometric recognition. *Circuits and Systems for Video Technology, IEEE Transactions on*, v. 14, n. 1, p. 4 – 20, jan. 2004. ISSN 1051-8215.

JURJENS, J. Code security analysis of a biometric authentication system using automated theorem provers. In: *Computer Security Applications Conference, 21st Annual*. [S.l.: s.n.], 2005. p. 10 pp. –149.

JURJENS, J. Verification of low-level crypto-protocol implementations using automated theorem proving. In: *Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design*. Washington, DC, USA: IEEE Computer Society, 2005. (MEMOCODE '05), p. 89–98. ISBN 0-7803-9227-2. Disponível em: <http://dx.doi.org/10.1109/MEMCOD.2005.1487898>.

KANADE, S.; PETROVSKA-DELACRETAZ, D.; DORIZZI, B. Generating and sharing biometrics based session keys for secure cryptographic applications. In: *Biometrics: Theory Applications and Systems (BTAS), 2010 Fourth IEEE International Conference on*. [S.l.: s.n.], 2010. p. 1 –7.

KANADE, S.; PETROVSKA-DELACRéTAZ, D.; DORIZZI, B. Multi-biometrics based crypto-biometric session key generation and sharing protocol. In: *Proceedings of the thirteenth ACM multimedia workshop on Multimedia and security*. New York, NY, USA: ACM, 2011. (MM & Sec '11), p. 109–114. ISBN 978-1-4503-0806-9.

KESSLER, V.; WEDEL, G. AUTLOG-an advanced logic of authentication. In: *Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings*. [S.l.: s.n.], 1994. p. 90 –99.

KROPF, T. *Introduction to Formal Hardware Verification*. [S.l.]: Springer, 2010. ISBN 978-3642084775.

LABSEC. *ASI-HSM*. 2012. Acesso em: 10 de Julho de 2012. Disponível em: <https://projetos.labsec.ufsc.br/openhsmd>.

LOWE, G. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, v. 56, n. 3, p. 131 – 133, 1995. ISSN 0020-0190. Disponível em: <http://www.sciencedirect.com-/science/article/pii/0020019095001442>.

LOWE, G. Breaking and fixing the needham-schroeder public-key protocol using FDR. In: MARGARIA, T.; STEFFEN, B. (Ed.). *Tools and Algorithms for the Construction and Analysis of Systems*. Springer Berlin / Heidelberg, 1996, (Lecture Notes in Computer Science, v. 1055). p. 147–166. ISBN 978-3-540-61042-7. 10.1007/3-540-61042-1_43. Disponível em: <http://dx.doi.org/10.1007/3-540-61042-1_43>.

MARTINA, J. E. *Verification of Security Protocols Based on Multicast Communication*. Tese (Doutorado) — University of Cambridge, Cambridge, UK, fev. 2011.

MARTINA, J. E.; BOAL, L. A. C. Uma análise formal automatizada dos protocolos de envio e confirmaação de processamento da nota fiscal eletrônica brasileira. In: *Proceedings of the VIII Brazilian Symposium on Information and Computing Systems Security (SBSeg'08)*. Gramado, Brazil: [s.n.], 2008.

MATHEW, H. et al. An improved three-factor authentication scheme using smart card with biometric privacy protection. In: *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*. [S.l.: s.n.], 2011. v. 3, p. 220 –223.

MAX-PLANCK. *Research Group 1: Automation of Logic*. 2012. Acesso em: 19 de Julho de 2012. Disponível em: <http://www.mpi-inf.mpg.de-/departments/rg1/>.

MAX-PLANCK. *Welcome to the SPASS Home Page*. 2012. Acesso em: 9 de Julho de 2012. Disponível em: <http://www.spass-prover.org/index-.html>.

MILLER, S. et al. *Kerberos authentication and authorization system*. Cambridge, Massachussets, jul. 1987.

MYSQL. *MySQL :: The world's most popular open source database*. 2012. Acesso em: 8 de Agosto de 2012. Disponível em: <http://www.mysql.com/>.

NATIONAL MUSEUM OF AMERICAN HISTORY. *Your Money in Transition*. ago. 2012. Acesso em: 19 de Agosto de 2012. Disponível em: <http://americanhistory.si.edu/collections/numismatics/moneyintrans-/nncstory2f.htm>.

NEEDHAM, R. M.; SCHROEDER, M. D. Using encryption for authentication in large networks of computers. *Commun. ACM*, v. 21, n. 12, p. 993–999, dez. 1978. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/359657.359659>.

NIPKOW, T.; WENZEL, M.; PAULSON, L. C. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Berlin, Heidelberg: Springer-Verlag, 2002. ISBN 3-540-43376-7.

NIST. *Federal Information Processing Standards Publication 197: Advanced Encryption Standard (AES)*. Gaithersburg, MD: National Institute of Standards (NIST), 2001. Acesso em: 1o de Agosto de 2012. Disponível em: <http://scholars.indstate.edu/handle/10484/3730>.

NIST. *Federal Information Processing Standards Publication 201-1: Personal Identity Verification (PIV) of Federal Employees and Contractors*. Gaithersburg, MD: National Institute of Standards (NIST), 2006. Acesso em: 1o de Agosto de 2012. Disponível em: <http://csrc-.nist.gov/publications/fips/fips201-1/FIPS-201-1-chng1.pdf>.

ORACLE. *Java PKCS#11 Reference Guide*. 2012. Acesso em: 8 de Junho de 2012. Disponível em: <http://docs.oracle.com/javase/1.5.0-/docs/guide/security/p11guide.html>.

ORACLE. *Lesson: Java Applets (The Java™ Tutorials > Deployment)*. ago. 2012. Acesso em: 1o de Agosto de 2012. Disponível em: <http://docs.oracle.com/javase/tutorial/deployment/applet/>.

ORACLE. *Oracle Technology Network for Java Developers*. 2012. Acesso em: 8 de Agosto de 2012. Disponível em: <http://www.oracle-.com/technetwork/java/index.html>.

PAULSON, L. Mechanized proofs for a recursive authentication protocol. In: *Computer Security Foundations Workshop, 1997. Proceedings., 10th*. [S.l.: s.n.], 1997. p. 84 –94.

PAULSON, L. C. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, v. 6, n. 1, p. 85–128, 1998.

PAULSON, L. C. *Logic and Proof: Computer Science Tripos Part 1B – Michaelmas Term*. Cambridge: [s.n.], 2006.

PIVA, F. R. *Verificação formal de protocolos de trocas justas utilizando o método de espaços de fitas*. Tese (Mestrado) — Universidade Estadual de Campinas, Campinas, 2009.

POINTCHEVAL, D.; ZIMMER, S. Multi-factor authenticated key exchange. In: BELLOVIN, S. et al. (Ed.). *Applied Cryptography and Network Security*. [S.l.]: Springer Berlin / Heidelberg, 2008, (Lecture Notes in Computer Science, v. 5037). p. 277–295. ISBN 978-3-540-68913-3.

POSTEL, J. *RFC 821: Simple Mail Transfer Protocol*. ago. 1982.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 7. ed. New York: McGraw-Hill, 2009. ISBN 978-0071267823.

RECEITA FEDERAL. *CPF - Perguntas e Respostas*. 2012. Acesso em: 29 de Agosto de 2012. Disponível em: <http://www.receita.fazenda.gov-.br/PessoaFisica/CPF/PerguntasRespostas/PerguntasRespostas.htm>.

RECEITA FEDERAL. *Programa Imposto sobre a Renda da Pessoa Física – Perguntas e Respostas*. Brasília: [s.n.], 2012. Acesso em: 10 de Julho de 2012. Disponível em: <http://www.receita.fazenda.gov.br-/publico/perguntao/Irpf2012/PerguntaseRespostasIRPF2012.pdf>.

RSA LABORATORIES. *PKCS #11 v2.30: Cryptographic Token Interface Standard*. [S.l.], abr. 2009. 7 p. Disponível em: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-30/pkcs-11v2-30-d1.pdf>.

RYAN, P.; SCHNEIDER, S. *The modelling and analysis of security protocols: the csp approach*. 1. ed. [S.l.]: Addison-Wesley Professional, 2000. ISBN 0-201-67471-8.

RYAN, P. Y. A.; SCHNEIDER, S. A. An attack on a recursive authentication protocol a cautionary tale. *Information Processing Letters*, v. 65, n. 1, p. 7 – 10, 1998. ISSN 0020-0190. Disponível em: <http:/-/www.sciencedirect.com/science/article/pii/S0020019097001804>.

SANCHEZ-REILLO, R.; SANCHEZ-AVILA, C.; MENGIBAR-POZO, L. Microprocessor smart cards with fingerprint user authentication. In: *Security Technology, 2002. Proceedings. 36th Annual 2002 International Carnahan Conference on*. [S.l.: s.n.], 2002. p. 46 – 49.

SANTOS, E. dos; MARTINA, J. E.; CUSTóDIO, R. F. Towards a formal verification of a multi-factor authentication protocol using automated theorem provers. In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*. [S.l.: s.n.], 2012. p. 84–91.

SANYAL, S.; TIWARI, A.; SANYAL, S. A multifactor secure authentication system for wireless payment. In: CHBEIR, R. et al. (Ed.). *Emergent Web Intelligence: Advanced Information Retrieval*. Springer London, 2010, (Advanced Information and Knowledge Processing). p. 341–369. ISBN 978-1-84996-074-8. 10.1007/978-1-84996-074-8_13. Disponível em: <http://dx.doi.org/10.1007/978-1-84996-074-8_13>.

SCHNEIER, B. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. New York, NY, USA: John Wiley & Sons, Inc., 1995. ISBN 0-471-11709-9.

SCHUMANN, J. Automatic verification of cryptographic protocols with SETHEO. In: McCune, W. (Ed.). *Automated Deduction—CADE-14*. Springer Berlin / Heidelberg, 1997, (Lecture Notes in Computer Science, v. 1249). p. 87–100. ISBN 978-3-540-63104-0. 10.1007/3-540-63104-6_12. Disponível em: <http://dx.doi.org/10.1007/3-540-63104-6_12>.

SHELFER, K. M.; PROCACCINO, J. D. Smart card evolution. *Commun. ACM*, v. 45, n. 7, p. 83–88, jul. 2002. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/514236.514239>.

SINGH, S. *The Code Book: The Secret History of Codes and Code-breaking*. [S.l.]: Fourth Estate, 2010. ISBN 9780007378302.

SONG, D. X.; BEREZIN, S.; PERRIG, A. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, v. 9, n. 1-2, p. 47–74, 2001. ISSN 1875-8924.

STALLINGS, W. *Cryptography and Network Security: Principles and Practice*. 3rd. ed. [S.l.]: Pearson Education, 2002. ISBN 0130914290.

SYVERSON, P. A taxonomy of replay attacks. In: *In Proceedings of the 7th IEEE Computer Security Foundations Workshop*. [S.l.]: Society Press, 1994. p. 187–191.

TANENBAUM, A. *Computer Networks*. 4th. ed. [S.l.]: Prentice Hall Professional Technical Reference, 2002. ISBN 0130661023.

TANG, Q. et al. A formal study of the privacy concerns in Biometric-Based remote authentication schemes. In: CHEN, L.; MU, Y.; SUSILO, W. (Ed.). *Information Security Practice and Experience*. Springer Berlin / Heidelberg, 2008, (Lecture Notes in Computer Science, v. 4991). p. 56–70. ISBN 978-3-540-79103-4. 10.1007/978-3-540-79104-1_5. Disponível em: <http://dx.doi.org/10.1007/978-3-540-79104-1_5>.

THINK ALIKE. *Indian Smart Card Market Report & Forecast: 2011-2016*. ago. 2012. Acesso em: 19 de Agosto de 2012. Disponível em: <http://syed-thinkalike.blogspot.com.br/2011/12/indian-smart-card-market-report.html>.

TRUSTED COMPUTING GROUP. *Trusted Computing Group - Home*. 2012. Acesso em: 1o de Agosto de 2012. Disponível em: <http://www.trustedcomputinggroup.org/>.

TSE. *Por dentro da urna*. 2. ed. Brasília: [s.n.], 2010. Acesso em: 10 de Julho de 2012. Disponível em: <http://www.tse.jus.br/hotSites-/urnaEletronica/arquivos/por_dentro_da_urna_web.pdf>.

UPMANYU, M. et al. Blind authentication: A secure crypto-biometric verification protocol. *Information Forensics and Security, IEEE Transactions on*, v. 5, n. 2, p. 255 –268, jun. 2010. ISSN 1556-6013.

VERIFINGER. *VeriFinger fingerprint recognition technology, algorithm and SDK for PC and Web*. 2012. Acesso em: 1o de Agosto de 2012. Disponível em: <http://www.neurotechnology.com/verifinger.html>.

WAYMAN, J. L. Fundamentals of biometric authentication technolo-gies. *Int. J. Image Graphics, vol. 1, no*, v. 1, n. 1, p. 93–113, 2001.

WAZLAWICK, R. S. *Metodologia de pesquisa para ciência da computação*. Rio de Janeiro: Elsevier, 2009. ISBN 9788535234107.

WEIDENBACH, C. Towards an automatic analysis of security protocols in first-order logic. In: *Automated Deduction — CADE-16*. Springer Berlin / Heidelberg, 1999, (Lecture Notes in Computer Science, v. 1632). p. 70–70. ISBN 978-3-540-66222-8. 10.1007/3-540-48660-7_29. Disponível em: <http://dx.doi.org/10.1007/3-540-48660-7_29>.

WEIDENBACH, C. *SPASS Input Syntax Version 1.5*. [S.l.: s.n.], 2006.

WEIDENBACH, C. et al. System description: SPASS version 1.0.0. In: GANZINGER, H. (Ed.). *Automated Deduction — CADE-16*. Springer Berlin / Heidelberg, 1999, (Lecture Notes in Computer Science, v. 1632). p. 680–680. ISBN 978-3-540-66222-8. 10.1007/3-540-48660-7_34. Disponível em: <http://dx.doi.org/10.1007/3-540-48660-7_34>.

WEIDENBACH, C. et al. SPASS version 3.5. In: SCHMIDT, R. (Ed.). *Automated Deduction – CADE-22*. [S.l.]: Springer Berlin / Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5663). p. 140–145. ISBN 978-3-642-02958-5.

YANG, D.; YANG, B. A biometric password-based multi-server authentication scheme with smart card. In: *Computer Design and Applications (ICCDA), 2010 International Conference on*. [S.l.: s.n.], 2010. v. 5, p. V5–554 –V5–559.

YANG, D.; YANG, B. A provable security biometric password multi-server authentication scheme with smart card. In: *Data, Privacy and E-Commerce (ISDPE), 2010 Second International Symposium on*. [S.l.: s.n.], 2010. p. 33 –38.

A PROBLEMAS E RESULTADOS

A.1 REPRESENTAÇÃO DE PROBLEMAS

A representação dos problemas lógicos é feita através da linguagem DFG para representação de problemas em Lógica de Primeira Ordem (WEIDENBACH, 2006). Nesta linguagem, os elementos básicos para definição de qualquer problema básico são:

- **Descrição**: espaço destinado para informações gerais sobre o problema modelado. Por exemplo: autor; versão, data, etc;
- **Parte Lógica**: como o próprio nome sugere, contém a descrição lógica do problema. Nesta seção são declarados as listas de símbolos (funções e predicados com respectivas aridades), e as listas de fórmulas (axiomas e conjecturas);
- ***Includes***: reaproveitamento das descrições de outros problemas lógicos contidos em arquivos separados (opcional); e
- **Configurações**: comandos para personalização da execução do problema pelo provador (opcional).

Quantificadores e conectivos lógicos são expressos de modo próprio na linguagem DFG. Comentários são indicados pelo caractere "%". A Tabela A.1 apresenta a tradução dos símbolos matemáticos para seu equivalente na linguagem DFG.

Tabela A.1: Símbolos lógicos na linguagem DFG

Símbolo	Tradução
$\forall$	*forall*
$\exists$	*exists*
$\wedge$	*and*
$\vee$	*or*
$\neg$	*not*
$\rightarrow$	*implies*
$\leftrightarrow$	*equiv*

Como forma de demonstração, o problema clássico do Sócrates-Mortal:

Sócrates é um homem.	$Humano(socrates)$
Todo homem é mortal.	$\forall x Humano(x) \rightarrow Mortal(x)$
Logo, Sócrates é mortal.	$\therefore Mortal(socrates)$

É representado da seguinte forma na linguagem DFG (MAX-PLANCK, 2012b):

```
begin_problem(Sokrates1).

list_of_descriptions.
name({*Sokrates*}).
author({*Christoph Weidenbach*}).
status(unsatisfiable).
description({* Sokrates is mortal and since all humans are
    mortal, he is mortal too. *}).
end_of_list.

list_of_symbols.
  functions[(sokrates,0)].
  predicates[(Human,1),(Mortal,1)].
end_of_list.

list_of_formulae(axioms).

formula(Human(sokrates),1).
formula(forall([x],implies(Human(x),Mortal(x))),2).

end_of_list.

list_of_formulae(conjectures).

formula(Mortal(sokrates),3).

end_of_list.

end_problem.
```

A.2 MODELO LÓGICO PRINCIPAL

$$\Gamma_{principal} = \{ \text{Fórmula 1}, \ldots, \text{Fórmula 31} \}$$

```
begin_problem(MFAP).

list_of_descriptions.
name({* Protocolo de Autenticacao Multifator*}).
author({* Eduardo dos Santos*}).
status(satisfiable).
description({* Modelo Principal *}).
end_of_list.

% Lista de simbolos

list_of_symbols.
functions[
  (pair, 2), (triple, 3), % Funcoes de composicao de
      mensagens
```

```
15
16   (krkey, 2), (kukey, 2), (kp, 2), % Funcoes de Chaves
17
18   (nonce, 2),  % Funcoes de Nonce
19
20   (sent, 3), % Funcoes de Mensagens
21
22   (encr, 2), (sign, 2), % Funcoes de Seguranca
23
24   (a, 0), (c, 0), (s, 0), (d, 0), % Entidades
25
26   (kra, 0), (cera, 0), (krc, 0), (cerc, 0), % Chaves das
         Entidades
27
28   (na, 0), (ns, 0), % Nonces
29
30   (codc, 0), (cpfc, 0), (templatec, 0), % Dados de
         Identificacao do Cliente
31
32   (ida, 0), (idc, 0), (idd, 0), (ids, 0) % Identificadores de
          Roteamento
33
34 ].
35
36 predicates[
37    (M, 1), % Mensagem
38
39   (E, 1), % Entidade
40
41   (Knows, 2), % Conhecimento
42   (Stores, 2) % Armazenamento
43 ].
44 end_of_list.
45
46
47 list_of_formulae(axioms).
48 % Entidade: Administrador
49   formula(E(a), A_is_entity ).
50   formula( Knows( kp( krkey( kra, a ), kukey( cera, a ) ), a
         ), A_knows_A_keypair ).
51   formula( Knows( krkey( kra, a ), a ), A_knows_A_kr ).
52   formula( Knows( kukey( cera, a ), a ), A_knows_A_ku ).
53   formula( Knows(kukey(cerc, c), a ), A_knows_C_ku ).
54   formula( Knows(cpfc, a), A_knows_cpfc).
55   formula( Knows( nonce( na, a ), a ), A_knows_Na ).
56
57 % Entidade: Cliente
58   formula(E(c), C_is_entity ).
59   formula( Knows( kp( krkey( krc, c ), kukey( cerc, c ) ), c
         ), C_knows_C_keypair ).
60   formula( Knows( krkey( krc, c ), c ), C_knows_C_kr ).
61   formula( Knows( kukey( cerc, c ), c ), C_knows_C_ku ).
62   formula( Knows( kukey( cera, a ), c ), C_knows_A_ku ).
```

```
63   formula( Knows( triple ( codc, cpfc, templatec ), c ),
         C_knows_id_data_triple ).
64   formula( Knows( codc, c ), C_knows_codc ).
65   formula( Knows( cpfc, c ), C_knows_cpfc ).
66   formula( Knows( templatec, c ), C_knows_templatec ).
67
68 % Entidade: Servidor
69   formula(E(s), S_is_entity ).
70   formula( Knows( kukey( cera, a ), s ), S_knows_A_ku ).
71   formula( Knows( kukey( cerc, c ), s ), S_knows_C_ku ).
72   formula( Knows( nonce( ns, s), s ), S_knows_Ns ).
73
74 % Entidade: Banco de Dados
75   formula( E( d ), D_is_entity ).
76
77 %
78 % CADASTRO
79 %
80
81 % CADASTRO - PASSO 1
82   formula( implies( and( Knows( kukey( cera, a ), a ), Knows(
          kp( krkey( kra, a ), kukey( cera, a ) ), a ), Knows(
         kukey( cerc, c ), a ), Knows( cpfc, a ), Knows( nonce(
         na, a ), a ) ), and( M( sent( a, s, pair( encr( triple(
          na, cpfc, ida ), cerc ), idc ) ) ), Stores( pair( na,
         cpfc ), a )  )  ), reg_step_1 ).
83
84 % CADASTRO - PASSO 2
85   formula( forall([ xa ], implies( M( sent( xa, s, pair( encr
         ( triple( na, cpfc, ida ), cerc ), idc ) ) ), M( sent(
         s, c, encr( triple( na, cpfc, ida ), cerc ) ) ) ) ) ,
         reg_step_2).
86
87 % CADASTRO - PASSO 3
88   formula( forall([ xs ], implies( and( Knows( kp( krkey( krc
         , c ), kukey( cerc, c ) ), c ), Knows( kukey( cera, a )
         , c ), Knows( triple ( codc, cpfc, templatec ), c ), M(
          sent( xs, c, encr( triple( na, cpfc, ida ), cerc ) )
          ), M( sent( c, s, pair( encr( pair( na, sign( triple(
         codc, cpfc, templatec ), krc ) ), cera ), ida ) ) ) ) )
         , reg_step_3 ).
89
90 % CADASTRO - PASSO 4
91   formula( forall([ xc ], implies( M( sent( xc, s, pair( encr
         ( pair( na, sign( triple( codc, cpfc, templatec ),krc )
          ), cera ), ida ) ) ), M( sent( s, a, encr( pair( na,
         sign( triple( codc, cpfc, templatec ),krc ) ), cera ) )
          ) ) ), reg_step_4 ).
92
93 % CADASTRO - PASSO 5
94   formula( forall([  xs ], implies( and( Knows( nonce( na, a)
         , a ), Knows( cpfc, a ), Knows(kp(krkey(kra, a), kukey(
         cera, a) ), a ), Knows(kukey(cerc, c), a), Stores( pair
```

```
        ( na, cpfc ), a ), M( sent( xs, a, encr( pair( na, sign
        ( triple( codc, cpfc, templatec ), krc ) ), cera ) ) )
        ), M( sent( a, s, pair( sign( sign( triple( codc, cpfc,
         templatec ), krc ), kra ), idd ) ) ) ), reg_step_5 )
        .
95
96 % CADASTRO - PASSO 6
97   formula( forall([ xa ], implies( M( sent( xa, s, pair( sign
        ( sign( triple( codc, cpfc, templatec ), krc ), kra ),
        idd ) ) ), and( Stores( cpfc, d ), Stores( sign( sign(
        triple( codc, cpfc, templatec ), krc ), kra ), d ), M(
        sent( s, d, sign( sign( triple( codc, cpfc, templatec )
        , krc ), kra ) ) ) ) ) ), reg_step_6 ).
98
99 %
100 % AUTENTICACAO
101 %
102
103 % AUTENTICACAO - PASSO 1
104   formula( implies( and( Knows( kukey( cerc, c ), s ), Knows(
         nonce( ns, s ), s ) ), and( M( sent( s, c, encr( ns,
        cerc ) ) ), Stores( pair( ns, cerc ), s ) ) ),
        auth_step_1 ).
105
106 % AUTENTICACAO - PASSO 2
107   formula( forall([xs], implies( and( Knows( cpfc, c ), Knows
        (templatec, c ), Knows( kp( krkey( krc, c ), kukey(
        cerc, c ) ), c ), M( sent( xs, c, encr( ns, cerc ) ) )
        ), and( Stores( pair( ns, xs), c ),M( sent( c, s, encr(
         triple( cpfc, ids, templatec ), ns ) ) ) ) ) ),
        auth_step_2 ).
108
109 % AUTENTICACAO - PASSO 3
110   formula( forall([xc], implies( and( Stores( pair(ns, cerc),
         s ), M( sent( xc, s, encr( triple( cpfc, ids,
        templatec ), ns ) ) ) ), M( sent( s, d, cpfc ) ) ) ),
        auth_step_3 ).
111
112 % AUTENTICACAO - PASSO 4
113   formula( forall([xs], implies( and( Stores( cpfc, d ),
        Stores( sign( sign( triple( codc, cpfc, templatec ),
        krc ), kra ), d ), M( sent( xs, d, cpfc ) ) ), M( sent(
         d, s, sign( sign( triple( codc, cpfc, templatec ),krc
        ),kra ) ) ) ) ), auth_step_4 ).
114
115
116 end_of_list.
117
118 list_of_formulae(conjectures).
119
120 formula( exists( [x],  M( sent( d, s, sign( sign( triple(
       codc, x, templatec ),krc ),kra ) ) ) ), test_auth_step_4
       ).
```

```
121
122 end_of_list.
123
124 list_of_settings(SPASS).
125 {*
126 % 0 = desligado; 1 = ligado
127 set_flag(Auto,1). % Configuracao automatica
128 set_flag(DocProof,1). % Imprimir prova encontrada
129 %set_flag( Loops, -1 ). % Numero maximo de loops
130 set_flag( PGiven, 0 ). % Imprimir clausulas para inferencias
131
132 *}
133 end_of_list.
134
135 end_problem.
```

A.3 MODELO LÓGICO ESTENDIDO

$$\Gamma_{estendido} = \Gamma_{principal} \cup \Gamma_{atacante}$$

```
1  begin_problem(MFAP).
2
3  list_of_descriptions.
4  name({* Protocolo de Autenticacao Multifator*}).
5  author({* Eduardo dos Santos*}).
6  status(satisfiable).
7  description({* Modelo Extendido *}).
8  end_of_list.
9
10 % Lista de simbolos
11
12 list_of_symbols.
13 functions[
14   (pair, 2), (triple, 3), % Funcoes de composicao de
         mensagens
15
16   (krkey, 2), (kukey, 2), (kp, 2), % Funcoes de Chaves
17
18   (nonce, 2),  % Funcoes de Nonce
19
20   (sent, 3), % Funcoes de Mensagens
21
22   (encr, 2), (sign, 2), % Funcoes de Seguranca
23
24   (a, 0), (c, 0), (s, 0), (d, 0), (i, 0), % Entidades
25
26   (kra, 0), (cera, 0), (krc, 0), (cerc, 0), % Chaves das
         Entidades
27
28   (na, 0), (ns, 0), % Nonces
29
```

```
30   (codc, 0), (cpfc, 0), (templatec, 0), % Dados de
         Identificacao do Cliente
31
32   (icodc, 0), (icpfc, 0), (itemplatec, 0), % Dados de
         Identificacao Forjados
33
34   (ida, 0), (idc, 0), (idd, 0), (ids, 0) % Identificadores de
          Roteamento
35
36 ].
37
38 predicates[
39    (M, 1), % Mensagem
40
41   (E, 1), % Entidade
42
43   (Knows, 2), % Conhecimento
44   (Stores, 2), % Armazenamento
45
46   (Im, 1) % Mensagem do Atacante
47 ].
48 end_of_list.
49
50
51 list_of_formulae(axioms).
52 % Entidade: Administrador
53   formula(E(a), A_is_entity ).
54   formula( Knows( kp( krkey( kra, a ), kukey( cera, a ) ), a
         ), A_knows_A_keypair ).
55   formula( Knows( krkey( kra, a ), a ), A_knows_A_kr ).
56   formula( Knows( kukey( cera, a ), a ), A_knows_A_ku ).
57   formula( Knows(kukey(cerc, c), a ), A_knows_C_ku ).
58   formula( Knows(cpfc, a), A_knows_cpfc).
59   formula( Knows( nonce( na, a ), a ), A_knows_Na ).
60
61 % Entidade: Cliente
62   formula(E(c), C_is_entity ).
63   formula( Knows( kp( krkey( krc, c ), kukey( cerc, c ) ), c
         ), C_knows_C_keypair ).
64   formula( Knows( krkey( krc, c ), c ), C_knows_C_kr ).
65   formula( Knows( kukey( cerc, c ), c ), C_knows_C_ku ).
66   formula( Knows( kukey( cera, a ), c ), C_knows_A_ku ).
67   formula( Knows( triple ( codc, cpfc, templatec ), c ),
         C_knows_id_data_triple ).
68   formula( Knows( codc, c ), C_knows_codc ).
69   formula( Knows( cpfc, c ), C_knows_cpfc ).
70   formula( Knows( templatec, c ), C_knows_templatec ).
71
72 % Entidade: Servidor
73   formula(E(s), S_is_entity ).
74   formula( Knows( kukey( cera, a ), s ), S_knows_A_ku ).
75   formula( Knows( kukey( cerc, c ), s ), S_knows_C_ku ).
76   formula( Knows( nonce( ns, s), s ), S_knows_Ns ).
```

```
77
78 % Entidade: Banco de Dados
79   formula( E( d ), D_is_entity ).
80
81 %
82 % CADASTRO
83 %
84
85 % CADASTRO - PASSO 1
86   formula( implies( and( Knows( kukey( cera, a ), a ), Knows(
          kp( krkey( kra, a ), kukey( cera, a ) ), a ), Knows(
         kukey( cerc, c ), a ), Knows( cpfc, a ), Knows( nonce(
         na, a ), a ) ), and( M( sent( a, s, pair( encr( triple(
          na, cpfc, ida ), cerc ), idc ) ) ), Stores( pair( na,
         cpfc ), a )  )  ), reg_step_1 ).
87
88 % CADASTRO - PASSO 2
89   formula( forall([ xa ], implies( M( sent( xa, s, pair( encr
         ( triple( na, cpfc, ida ), cerc ), idc ) ) ), M( sent(
         s, c, encr( triple( na, cpfc, ida ), cerc ) ) ) ) ) ,
         reg_step_2).
90
91 % CADASTRO - PASSO 3
92   formula( forall([ xs ], implies( and( Knows( kp( krkey( krc
         , c ), kukey( cerc, c ) ), c ), Knows( kukey( cera, a )
         , c ), Knows( triple ( codc, cpfc, templatec ), c ), M(
          sent( xs, c, encr( triple( na, cpfc, ida ), cerc ) ) )
          ), M( sent( c, s, pair( encr( pair( na, sign( triple(
         codc, cpfc, templatec ), krc ) ), cera ), ida ) ) ) ) )
         , reg_step_3 ).
93
94 % CADASTRO - PASSO 4
95   formula( forall([ xc ], implies( M( sent( xc, s, pair( encr
         ( pair( na, sign( triple( codc, cpfc, templatec ),krc )
          ), cera ), ida ) ) ), M( sent( s, a, encr( pair( na,
         sign( triple( codc, cpfc, templatec ),krc ) ), cera ) )
          ) ) ), reg_step_4 ).
96
97 % CADASTRO - PASSO 5
98   formula( forall([  xs ], implies( and( Knows( nonce( na, a)
         , a ), Knows( cpfc, a ), Knows(kp(krkey(kra, a), kukey(
         cera, a) ), a ), Knows(kukey(cerc, c), a), Stores( pair
         ( na, cpfc ), a ), M( sent( xs, a, encr( pair( na, sign
         ( triple( codc, cpfc, templatec ), krc ) ), cera ) ) )
         ), M( sent( a, s, pair( sign( sign( triple( codc, cpfc,
          templatec ), krc ), kra ), idd ) ) ) ) ), reg_step_5 )
         .
99
100 % CADASTRO - PASSO 6
101   formula( forall([ xa ], implies( M( sent( xa, s, pair( sign
         ( sign( triple( codc, cpfc, templatec ), krc ), kra ),
         idd ) ) ), and( Stores( cpfc, d ), Stores( sign( sign(
         triple( codc, cpfc, templatec ), krc ), kra ), d ), M(
```

```
        sent( s, d, sign( sign( triple( codc, cpfc, templatec )
        , krc ), kra ) ) ) ) ), reg_step_6 ).
102
103 %
104 % AUTENTICACAO
105 %
106
107 % AUTENTICACAO - PASSO 1
108   formula( implies( and( Knows( kukey( cerc, c ), s ), Knows(
          nonce( ns, s ), s ) ), and( M( sent( s, c, encr( ns,
        cerc ) ) ), Stores( pair( ns, cerc ), s ) ) ),
        auth_step_1 ).
109
110 % AUTENTICACAO - PASSO 2
111   formula( forall([xs], implies( and( Knows( cpfc, c ), Knows
        (templatec, c ), Knows( kp( krkey( krc, c ), kukey(
        cerc, c ) ), c ), M( sent( xs, c, encr( ns, cerc ) ) )
        ), and( Stores( pair( ns, xs), c ),M( sent( c, s, encr(
         triple( cpfc, ids, templatec ), ns ) ) ) ) ) ),
        auth_step_2 ).
112
113 % AUTENTICACAO - PASSO 3
114   formula( forall([xc], implies( and( Stores( pair(ns, cerc),
         s ), M( sent( xc, s, encr( triple( cpfc, ids,
        templatec ), ns ) ) ) ), M( sent( s, d, cpfc ) ) ) ),
        auth_step_3 ).
115
116 % AUTENTICACAO - PASSO 4
117   formula( forall([xs], implies( and( Stores( cpfc, d ),
        Stores( sign( sign( triple( codc, cpfc, templatec ),
        krc ), kra ), d ), M( sent( xs, d, cpfc ) ) ), M( sent(
         d, s, sign( sign( triple( codc, cpfc, templatec ),krc
        ),kra ) ) ) ) ), auth_step_4 ).
118
119 %
120 % FORMALIZACAO DO ATACANTE
121 %
122
123 % (Primeiro bloco):
124
125 % Uma entidade:
126 formula( E( i ), I_is_entity ).
127
128 % Os dados forjados estao no conjunto de mensagens:
129 formula( Im( triple( icodc, icpfc, itemplatec ) ),
        I_has_forged_client_data ).
130
131 % Conhecimento Inicial:
132 formula( Knows( kukey( cerc, c ), i ), I_knows_C_ku ).
133 formula( Knows( kukey( cera, a ), i ), I_knows_A_ku ).
134
135 % Habilidade de gravar mensagens:
```

```
136 formula( forall([xa, xb, xm], implies( M( sent( xa, xb, xm )
        ), Im( xm ) ) ), I_records_all ).
137
138 % Decompor mensagens:
139 formula( forall([u, v], implies( Im( pair( u, v ) ), and( Im(
         u ), Im( v ) ) ) ), I_decomposes_pair ).
140 formula( forall([u, v, w], implies( Im( triple( u, v, w ) ),
        and( Im( u ), Im( v ), Im( w ) ) ) ), I_decomposes_triple
         ).
141
142 % Fabricar mensagens:
143 formula( forall([u, v], implies( and( Im( u ), Im( v ) ), Im(
         pair( u, v ) ) ) ), I_assembles_pair ).
144 formula( forall([u, v, w], implies( and( Im( u ), Im( v ), Im
        ( w ) ), Im( triple( u, v, w ) ) ) ), I_assembles_triple
        ).
145
146 % Enviar mensagens falsas:
147 formula( forall([u, x, y], implies( and( Im( u ), E( x ), E(
        y ) ), M( sent( x, y, u ) ) )  ), I_sends_faked ).
148
149 % (Segundo bloco):
150
151 % Qualquer coisa pode potencialmente ser uma chave:
152 formula( forall( [v, w], implies( and( Im( v ), E( w ) ),
        Knows( krkey(v, w ), i ) ) ), I_tests_for_kr_keys ).
153 formula( forall( [v, w], implies( and( Im( v ), E( w ) ),
        Knows( kukey(v, w ), i ) ) ), I_tests_for_ku_keys ).
154
155 % Qualquer coisa pode potencialmente ser um nonce:
156 formula( forall( [v, w], implies( and( Im( v ), E( w ) ),
        Knows( nonce( v, w ), i ) ) ), I_tests_for_nonces ).
157
158 % Gerar mensagens cifradas:
159 formula( forall([ u, v, x ], implies( and( Im( u ), Knows(
        kukey( v, x ), i ), E( x ) ), Im( encr( u, v ) ) ) ),
        I_generates_encrypted ).
160
161 % Gerar mensagens assinadas:
162 formula( forall([ u, v, x ], implies( and( Im( u ), Knows(
        krkey( v, x ), i ), E( x ) ), Im( sign( u, v ) ) ) ),
        I_generates_signed ).
163
164 % (Terceiro bloco):
165
166 % Decifrar mensagens cifradas com chaves conhecidas:
167 formula( forall( [u, v, w, x], implies( and( Im( encr( u,v )
        ), Knows( kp( krkey( w, x ), kukey( v, x ) ), i ), E( x )
          ), Im( u ) ) ), I_decrypts_with_known_keys ).
168
169 % Decifrar mensagens cifradas com nonces conhecidos (para
        autenticacao):
```

```
170 formula( forall( [u, v, w], implies( and( Im( encr( u, v ) ),
        Knows( nonce( v, w ), i ), E( w ) ), Im( u ) ) ),
      I_decrypts_with_known_nonces ).
171
172 % Acessar conteudo de mensagens assinadas:
173 formula( forall( [u, v], implies( Im( sign(u, v) ), Im( u ) )
      ), I_extracts_signature_contents ).
174
175 % Algumas Hipoteses de Teste:
176
177 % Atacante possui par de chaves do Administrador:
178 %formula( and( Knows( krkey( kra, a ), i ), Knows( kp( krkey(
      kra, a ), kukey( cera, a ) ), i ) ),
      hypothesis_I_has_A_keypair ).
179
180 % Atacante possui par de chaves do Cliente:
181 %formula( and( Knows( krkey( krc, c ), i ), Knows( kp( krkey(
      krc, c ), kukey( cerc, c ) ), i ) ),
      hypothesis_I_has_C_keypair ).
182
183 end_of_list.
184
185 list_of_formulae(conjectures).
186
187 % Testa o modelo do atacante: o atacante deve ser capaz de
      gerar o token duplamente assinado com dados forjados e
      envia-lo ao Banco de Dados. Dependendo das hipoteses
      consideradas, isto pode ou nao ser possivel.
188 %formula( M(sent(i, d, sign( sign( triple( icodc, icpfc,
      itemplatec ), krc ), kra ) ) ), test_attacker_capacity ).
189
190
191
192 end_of_list.
193
194 list_of_settings(SPASS).
195 {*
196 % 0 = desligado; 1 = ligado
197 set_flag(Auto,1). % Configuracao automatica
198 set_flag(DocProof,1). % Imprimir prova encontrada
199 %set_flag( Loops, -1 ). % Numero maximo de loops
200 set_flag( PGiven, 0 ). % Imprimir clausulas para inferencias
201
202 *}
203 end_of_list.
204
205 end_problem.
```

A.4 PROVAS

A.4.1 Fato 1

Saturação do modelo lógico principal (Fórmulas 1 a 31).

```
1
2 ----------------------------SPASS-START
      ------------------------------
3 Input Problem:
4 1[0:Inp] ||  -> E(a)*.
5 2[0:Inp] ||  -> E(c)*.
6 3[0:Inp] ||  -> E(s)*.
7 4[0:Inp] ||  -> E(d)*.
8 5[0:Inp] ||  -> Knows(cpfc,a)*.
9 6[0:Inp] ||  -> Knows(codc,c)*.
10 7[0:Inp] ||  -> Knows(cpfc,c)*.
11 8[0:Inp] ||  -> Knows(templatec,c)*.
12 9[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
13 10[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
14 11[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
15 12[0:Inp] ||  -> Knows(nonce(na,a),a)*.
16 13[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
17 14[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
18 15[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
19 16[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
20 17[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
21 18[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
22 19[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
23 20[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
24 21[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
25 22[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) ->
       Stores(pair(ns,cerc),s)*.
26 23[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> Stores(cpfc,d).
27 24[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) -> M
       (sent(s,c,encr(ns,cerc)))*.
28 25[0:Inp] || M(sent(U,s,encr(triple(cpfc,ids,templatec),ns)))
       * Stores(pair(ns,cerc),s) -> M(sent(s,d,cpfc)).
29 26[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
       idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
30 27[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
       codc,cpfc,templatec),krc),kra),d).
31 28[0:Inp] || M(sent(U,s,pair(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
       triple(codc,cpfc,templatec),krc),kra))).
32 29[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
       ),kukey(cerc,c)),c)* Knows(templatec,c) Knows(cpfc,c) ->
       Stores(pair(ns,U),c).
33 30[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
       cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
       pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
```

```
34 31[0:Inp] || M(sent(U,d,cpfc))* Stores(sign(sign(triple(codc,
       cpfc,templatec),krc),kra),d) Stores(cpfc,d) -> M(sent(d,s
       ,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
35 32[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
       ),kukey(cerc,c)),c) Knows(templatec,c) Knows(cpfc,c) -> M
       (sent(c,s,encr(triple(cpfc,ids,templatec),ns)))*.
36 33[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
       kukey(cera,a)),a)* Knows(kukey(cerc,c),a) Knows(cpfc,a)
       Knows(nonce(na,a),a) -> Stores(pair(na,cpfc),a).
37 34[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
       kukey(cera,a)),a) Knows(kukey(cerc,c),a) Knows(cpfc,a)
       Knows(nonce(na,a),a) -> M(sent(a,s,pair(encr(triple(na,
       cpfc,ida),cerc),idc)))*.
38 35[0:Inp] || M(sent(U,c,encr(triple(na,cpfc,ida),cerc)))*
       Knows(triple(codc,cpfc,templatec),c) Knows(kukey(cera,a),
       c) Knows(kp(krkey(krc,c),kukey(cerc,c)),c) -> M(sent(c,s,
       pair(encr(pair(na,sign(triple(codc,cpfc,templatec),krc)),
       cera),ida)))*.
39 36[0:Inp] || M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,
       templatec),krc)),cera)))* Stores(pair(na,cpfc),a) Knows(
       kukey(cerc,c),a) Knows(kp(krkey(kra,a),kukey(cera,a)),a)
       Knows(cpfc,a) Knows(nonce(na,a),a) -> M(sent(a,s,pair(
       sign(sign(triple(codc,cpfc,templatec),krc),kra),idd)))*.
40  This is a first-order Horn problem without equality.
41  The following monadic predicates have finite extensions: E.
42  Axiom clauses: 36 Conjecture clauses: 0
43  Inferences: IORe=1
44  Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RFSub=1
        RBSub=1 RCon=1
45  Extras    : Input Saturation, Dynamic Selection, No
        Splitting, Full Reduction,  Ratio: 5, FuncWeight: 1,
        VarWeight: 1
46  Precedence: pair > krkey > kukey > kp > nonce > encr > sign
        > triple > sent > Stores > M > E > Knows > a > c > s > d
         > kra > cera > krc > cerc > na > ns > codc > cpfc >
        templatec > ida > idc > idd > ids
47  Ordering  : KBO
48 Processed Problem:
49
50 Worked Off Clauses:
51
52 Usable Clauses:
53 4[0:Inp] ||  -> E(d)*.
54 3[0:Inp] ||  -> E(s)*.
55 2[0:Inp] ||  -> E(c)*.
56 1[0:Inp] ||  -> E(a)*.
57 8[0:Inp] ||  -> Knows(templatec,c)*.
58 7[0:Inp] ||  -> Knows(cpfc,c)*.
59 6[0:Inp] ||  -> Knows(codc,c)*.
60 5[0:Inp] ||  -> Knows(cpfc,a)*.
61 37[0:MRR:22.0,22.1,17.0,18.0] ||  -> Stores(pair(ns,cerc),s)
       *.
```

```
62 42[0:MRR:33.0,33.1,33.2,33.3,33.4,10.0,20.0,11.0,5.0,12.0] ||
       -> Stores(pair(na,cpfc),a)*.
63 17[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
64 16[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
65 15[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
66 11[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
67 18[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
68 14[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
69 13[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
70 12[0:Inp] ||  -> Knows(nonce(na,a),a)*.
71 10[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
72 9[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
73 19[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
74 38[0:MRR:24.0,24.1,17.0,18.0] ||  -> M(sent(s,c,encr(ns,cerc)
      ))*.
75 21[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
76 20[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
77 43[0:MRR:34.0,34.1,34.2,34.3,34.4,10.0,20.0,11.0,5.0,12.0] ||
       -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
      )*.
78 40[0:MRR:29.1,29.2,29.3,21.0,8.0,7.0] || M(sent(U,c,encr(ns,
      cerc)))* -> Stores(pair(ns,U),c).
79 39[0:MRR:25.1,37.0] || M(sent(U,s,encr(triple(cpfc,ids,
      templatec),ns)))* -> M(sent(s,d,cpfc)).
80 23[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(cpfc,d).
81 41[0:MRR:32.1,32.2,32.3,21.0,8.0,7.0] || M(sent(U,c,encr(ns,
      cerc)))* -> M(sent(c,s,encr(triple(cpfc,ids,templatec),ns
      )))*.
82 26[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
      idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
83 27[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
      codc,cpfc,templatec),krc),kra),d).
84 28[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
      triple(codc,cpfc,templatec),krc),kra))).
85 44[0:MRR:35.1,35.2,35.3,19.0,15.0,21.0] || M(sent(U,c,encr(
      triple(na,cpfc,ida),cerc)))* -> M(sent(c,s,pair(encr(pair
      (na,sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
86 45[0:MRR:36.1,36.2,36.3,36.4,36.5,42.0,11.0,20.0,5.0,12.0] ||
       M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,templatec)
      ,krc)),cera)))* -> M(sent(a,s,pair(sign(sign(triple(codc,
      cpfc,templatec),krc),kra),idd)))*.
87 30[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
      cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
      pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
88 31[0:Inp] || M(sent(U,d,cpfc))* Stores(cpfc,d) Stores(sign(
      sign(triple(codc,cpfc,templatec),krc),kra),d) -> M(sent(d
      ,s,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
89 SPASS V 3.5
90 SPASS beiseite: Completion found.
91 Problem: C:/Users/Eduardo/AppData/Local/Tempqt_temp.a01732
```

```
92 SPASS derived 12 clauses, backtracked 0 clauses, performed 0
       splits and kept 48 clauses.
93 SPASS allocated 27052 KBytes.
94 SPASS spent 0:00:00.08 on the problem.
95     0:00:00.00 for the input.
96     0:00:00.02 for the FLOTTER CNF translation.
97     0:00:00.00 for inferences.
98     0:00:00.00 for the backtracking.
99     0:00:00.00 for the reduction.
100
101
102  The saturated set of worked-off clauses is :
103 58[0:Res:48.0,57.0] ||  -> M(sent(d,s,sign(sign(triple(codc,
       cpfc,templatec),krc),kra)))*.
104 53[0:Res:52.0,28.0] ||  -> M(sent(s,d,sign(sign(triple(codc,
       cpfc,templatec),krc),kra)))*.
105 54[0:Res:52.0,27.0] ||  -> Stores(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),d)*.
106 55[0:Res:52.0,23.0] ||  -> Stores(cpfc,d)*.
107 52[0:Res:51.0,45.0] ||  -> M(sent(a,s,pair(sign(sign(triple(
       codc,cpfc,templatec),krc),kra),idd)))*.
108 51[0:Res:50.0,30.0] ||  -> M(sent(s,a,encr(pair(na,sign(
       triple(codc,cpfc,templatec),krc)),cera)))*.
109 50[0:Res:49.0,44.0] ||  -> M(sent(c,s,pair(encr(pair(na,sign(
       triple(codc,cpfc,templatec),krc)),cera),ida)))*.
110 49[0:Res:43.0,26.0] ||  -> M(sent(s,c,encr(triple(na,cpfc,ida
       ),cerc)))*.
111 48[0:Res:47.0,39.0] ||  -> M(sent(s,d,cpfc))*.
112 47[0:Res:38.0,41.0] ||  -> M(sent(c,s,encr(triple(cpfc,ids,
       templatec),ns)))*.
113 46[0:Res:38.0,40.0] ||  -> Stores(pair(ns,s),c)*.
114 40[0:MRR:29.1,29.2,29.3,21.0,8.0,7.0] || M(sent(U,c,encr(ns,
       cerc)))* -> Stores(pair(ns,U),c).
115 43[0:MRR:34.0,34.1,34.2,34.3,34.4,10.0,20.0,11.0,5.0,12.0] ||
         -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
       )*.
116 20[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
117 21[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
118 38[0:MRR:24.0,24.1,17.0,18.0] ||  -> M(sent(s,c,encr(ns,cerc)
       ))*.
119 19[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
120 9[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
121 10[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
122 12[0:Inp] ||  -> Knows(nonce(na,a),a)*.
123 13[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
124 14[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
125 18[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
126 11[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
127 15[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
128 16[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
129 17[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
130 42[0:MRR:33.0,33.1,33.2,33.3,33.4,10.0,20.0,11.0,5.0,12.0] ||
         -> Stores(pair(na,cpfc),a)*.
```

```
37[0:MRR:22.0,22.1,17.0,18.0] ||  -> Stores(pair(ns,cerc),s)
    *.
5[0:Inp] ||  -> Knows(cpfc,a)*.
6[0:Inp] ||  -> Knows(codc,c)*.
7[0:Inp] ||  -> Knows(cpfc,c)*.
8[0:Inp] ||  -> Knows(templatec,c)*.
1[0:Inp] ||  -> E(a)*.
2[0:Inp] ||  -> E(c)*.
3[0:Inp] ||  -> E(s)*.
4[0:Inp] ||  -> E(d)*.
--------------------------SPASS-STOP
    ------------------------------
```

A.4.2 Fato 2

$$\Gamma_{principal} \vdash \exists u[M(sent(d, s, sign(sign(triple(codc, u, templatec), krc), kra)))]$$

```

--------------------------SPASS-START
    ------------------------------
Input Problem:
1[0:Inp] ||  -> E(a)*.
2[0:Inp] ||  -> E(c)*.
3[0:Inp] ||  -> E(s)*.
4[0:Inp] ||  -> E(d)*.
5[0:Inp] ||  -> Knows(cpfc,a)*.
6[0:Inp] ||  -> Knows(codc,c)*.
7[0:Inp] ||  -> Knows(cpfc,c)*.
8[0:Inp] ||  -> Knows(templatec,c)*.
9[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
10[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
11[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
12[0:Inp] ||  -> Knows(nonce(na,a),a)*.
13[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
14[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
15[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
16[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
17[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
18[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
19[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
20[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
21[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
22[0:Inp] || M(sent(d,s,sign(sign(triple(codc,U,templatec),
    krc),kra)))* -> .
23[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) ->
    Stores(pair(ns,cerc),s)*.
24[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(cpfc,d).
25[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) -> M
    (sent(s,c,encr(ns,cerc)))*.
26[0:Inp] || M(sent(U,s,encr(triple(cpfc,ids,templatec),ns)))
    * Stores(pair(ns,cerc),s) -> M(sent(s,d,cpfc)).
```

```
30 27[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
       idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
31 28[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
       codc,cpfc,templatec),krc),kra),d).
32 29[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
       triple(codc,cpfc,templatec),krc),kra))).
33 30[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
       ),kukey(cerc,c)),c)* Knows(templatec,c) Knows(cpfc,c) ->
       Stores(pair(ns,U),c).
34 31[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
       cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
       pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
35 32[0:Inp] || M(sent(U,d,cpfc))* Stores(sign(sign(triple(codc,
       cpfc,templatec),krc),kra),d) Stores(cpfc,d) -> M(sent(d,s
       ,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
36 33[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
       ),kukey(cerc,c)),c) Knows(templatec,c) Knows(cpfc,c) -> M
       (sent(c,s,encr(triple(cpfc,ids,templatec),ns)))*.
37 34[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
       kukey(cera,a)),a)* Knows(kukey(cerc,c),a) Knows(cpfc,a)
       Knows(nonce(na,a),a) -> Stores(pair(na,cpfc),a).
38 35[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
       kukey(cera,a)),a) Knows(kukey(cerc,c),a) Knows(cpfc,a)
       Knows(nonce(na,a),a) -> M(sent(a,s,pair(encr(triple(na,
       cpfc,ida),cerc),idc)))*.
39 36[0:Inp] || M(sent(U,c,encr(triple(na,cpfc,ida),cerc)))*
       Knows(triple(codc,cpfc,templatec),c) Knows(kukey(cera,a),
       c) Knows(kp(krkey(krc,c),kukey(cerc,c)),c) -> M(sent(c,s,
       pair(encr(pair(na,sign(triple(codc,cpfc,templatec),krc)),
       cera),ida)))*.
40 37[0:Inp] || M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,
       templatec),krc)),cera)))* Stores(pair(na,cpfc),a) Knows(
       kukey(cerc,c),a) Knows(kp(krkey(kra,a),kukey(cera,a)),a)
       Knows(cpfc,a) Knows(nonce(na,a),a) -> M(sent(a,s,pair(
       sign(sign(triple(codc,cpfc,templatec),krc),kra),idd)))*.
41  This is a first-order Horn problem without equality.
42  The following monadic predicates have finite extensions: E.
43  Axiom clauses: 36 Conjecture clauses: 1
44  Inferences: IORe=1
45  Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RFSub=1
        RBSub=1 RCon=1
46  Extras    : Input Saturation, Dynamic Selection, No
        Splitting, Full Reduction,  Ratio: 5, FuncWeight: 1,
        VarWeight: 1
47  Precedence: pair > krkey > kukey > kp > nonce > encr > sign
       > triple > sent > Stores > M > E > Knows > a > c > s > d
        > kra > cera > krc > cerc > na > ns > codc > cpfc >
        templatec > ida > idc > idd > ids
48  Ordering  : KBO
49 Processed Problem:
50
```

```
Worked Off Clauses:

Usable Clauses:
4[0:Inp] ||  -> E(d)*.
3[0:Inp] ||  -> E(s)*.
2[0:Inp] ||  -> E(c)*.
1[0:Inp] ||  -> E(a)*.
8[0:Inp] ||  -> Knows(templatec,c)*.
7[0:Inp] ||  -> Knows(cpfc,c)*.
6[0:Inp] ||  -> Knows(codc,c)*.
5[0:Inp] ||  -> Knows(cpfc,a)*.
38[0:MRR:23.0,23.1,17.0,18.0] ||  -> Stores(pair(ns,cerc),s)
    *.
44[0:MRR:34.0,34.1,34.2,34.3,34.4,10.0,20.0,11.0,5.0,12.0] ||
     -> Stores(pair(na,cpfc),a)*.
17[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
16[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
15[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
11[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
18[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
14[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
13[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
12[0:Inp] ||  -> Knows(nonce(na,a),a)*.
10[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
9[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
19[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
39[0:MRR:25.0,25.1,17.0,18.0] ||  -> M(sent(s,c,encr(ns,cerc)
    ))*.
21[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
20[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
45[0:MRR:35.0,35.1,35.2,35.3,35.4,10.0,20.0,11.0,5.0,12.0] ||
     -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
    )*.
22[0:Inp] || M(sent(d,s,sign(sign(triple(codc,U,templatec),
    krc),kra)))* -> .
41[0:MRR:30.1,30.2,30.3,21.0,8.0,7.0] || M(sent(U,c,encr(ns,
    cerc)))* -> Stores(pair(ns,U),c).
40[0:MRR:26.1,38.0] || M(sent(U,s,encr(triple(cpfc,ids,
    templatec),ns)))* -> M(sent(s,d,cpfc)).
24[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(cpfc,d).
43[0:MRR:33.1,33.2,33.3,21.0,8.0,7.0] || M(sent(U,c,encr(ns,
    cerc)))* -> M(sent(c,s,encr(triple(cpfc,ids,templatec),ns
    )))*.
42[0:MRR:32.3,22.0] || M(sent(U,d,cpfc))* Stores(cpfc,d)
    Stores(sign(sign(triple(codc,cpfc,templatec),krc),kra),d)
    * -> .
27[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
    idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
28[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
    codc,cpfc,templatec),krc),kra),d).
```

```
29[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
    triple(codc,cpfc,templatec),krc),kra))).
46[0:MRR:36.1,36.2,36.3,19.0,15.0,21.0] || M(sent(U,c,encr(
    triple(na,cpfc,ida),cerc)))* -> M(sent(c,s,pair(encr(pair
    (na,sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
47[0:MRR:37.1,37.2,37.3,37.4,37.5,44.0,11.0,20.0,5.0,12.0] ||
     M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,templatec)
    ,krc)),cera)))* -> M(sent(a,s,pair(sign(sign(triple(codc,
    cpfc,templatec),krc),kra),idd)))*.
31[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
    cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
    pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
SPASS V 3.5
SPASS beiseite: Proof found.
Problem: C:/Users/Eduardo/AppData/Local/Tempqt_temp.a01732
SPASS derived 11 clauses, backtracked 0 clauses, performed 0
    splits and kept 48 clauses.
SPASS allocated 27054 KBytes.
SPASS spent 0:00:00.08 on the problem.
    0:00:00.02 for the input.
    0:00:00.00 for the FLOTTER CNF translation.
    0:00:00.00 for inferences.
    0:00:00.00 for the backtracking.
    0:00:00.00 for the reduction.


Here is a proof with depth 5, length 46 :
5[0:Inp] ||  -> Knows(cpfc,a)*.
7[0:Inp] ||  -> Knows(cpfc,c)*.
8[0:Inp] ||  -> Knows(templatec,c)*.
10[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
11[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
12[0:Inp] ||  -> Knows(nonce(na,a),a)*.
15[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
17[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
18[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
19[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
20[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
21[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
22[0:Inp] || M(sent(d,s,sign(sign(triple(codc,U,templatec),
    krc),kra)))* -> .
23[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) ->
    Stores(pair(ns,cerc),s)*.
24[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(cpfc,d).
25[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) -> M
    (sent(s,c,encr(ns,cerc)))*.
26[0:Inp] || M(sent(U,s,encr(triple(cpfc,ids,templatec),ns)))
    * Stores(pair(ns,cerc),s) -> M(sent(s,d,cpfc)).
27[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
    idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
```

```
123 28[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
      codc,cpfc,templatec),krc),kra),d).
124 31[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
      cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
      pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
125 32[0:Inp] || M(sent(U,d,cpfc))* Stores(sign(sign(triple(codc,
      cpfc,templatec),krc),kra),d) Stores(cpfc,d) -> M(sent(d,s
      ,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
126 33[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
      ),kukey(cerc,c)),c) Knows(templatec,c) Knows(cpfc,c) -> M
      (sent(c,s,encr(triple(cpfc,ids,templatec),ns)))*.
127 34[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
      kukey(cera,a)),a)* Knows(kukey(cerc,c),a) Knows(cpfc,a)
      Knows(nonce(na,a),a) -> Stores(pair(na,cpfc),a).
128 35[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
      kukey(cera,a)),a) Knows(kukey(cerc,c),a) Knows(cpfc,a)
      Knows(nonce(na,a),a) -> M(sent(a,s,pair(encr(triple(na,
      cpfc,ida),cerc),idc)))*.
129 36[0:Inp] || M(sent(U,c,encr(triple(na,cpfc,ida),cerc)))*
      Knows(triple(codc,cpfc,templatec),c) Knows(kukey(cera,a),
      c) Knows(kp(krkey(krc,c),kukey(cerc,c)),c) -> M(sent(c,s,
      pair(encr(pair(na,sign(triple(codc,cpfc,templatec),krc)),
      cera),ida)))*.
130 37[0:Inp] || M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,
      templatec),krc)),cera)))* Stores(pair(na,cpfc),a) Knows(
      kukey(cerc,c),a) Knows(kp(krkey(kra,a),kukey(cera,a)),a)
      Knows(cpfc,a) Knows(nonce(na,a),a) -> M(sent(a,s,pair(
      sign(sign(triple(codc,cpfc,templatec),krc),kra),idd)))*.
131 38[0:MRR:23.0,23.1,17.0,18.0] ||  -> Stores(pair(ns,cerc),s)
      *.
132 39[0:MRR:25.0,25.1,17.0,18.0] ||  -> M(sent(s,c,encr(ns,cerc)
      ))*.
133 40[0:MRR:26.1,38.0] || M(sent(U,s,encr(triple(cpfc,ids,
      templatec),ns)))* -> M(sent(s,d,cpfc)).
134 42[0:MRR:32.3,22.0] || M(sent(U,d,cpfc))* Stores(cpfc,d)
      Stores(sign(sign(triple(codc,cpfc,templatec),krc),kra),d)
      *+ -> .
135 43[0:MRR:33.1,33.2,33.3,21.0,8.0,7.0] || M(sent(U,c,encr(ns,
      cerc)))*+ -> M(sent(c,s,encr(triple(cpfc,ids,templatec),
      ns)))*.
136 44[0:MRR:34.0,34.1,34.2,34.3,34.4,10.0,20.0,11.0,5.0,12.0] ||
        -> Stores(pair(na,cpfc),a)*.
137 45[0:MRR:35.0,35.1,35.2,35.3,35.4,10.0,20.0,11.0,5.0,12.0] ||
        -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
      )*.
138 46[0:MRR:36.1,36.2,36.3,19.0,15.0,21.0] || M(sent(U,c,encr(
      triple(na,cpfc,ida),cerc)))*+ -> M(sent(c,s,pair(encr(
      pair(na,sign(triple(codc,cpfc,templatec),krc)),cera),ida)
      ))*.
139 47[0:MRR:37.1,37.2,37.3,37.4,37.5,44.0,11.0,20.0,5.0,12.0] ||
       M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,templatec)
      ,krc)),cera)))*+ -> M(sent(a,s,pair(sign(sign(triple(codc
```

```
      ,cpfc,templatec),krc),kra),idd)))*.
140 49[0:Res:39.0,43.0] ||  -> M(sent(c,s,encr(triple(cpfc,ids,
      templatec),ns)))*.
141 50[0:Res:49.0,40.0] ||  -> M(sent(s,d,cpfc))*.
142 51[0:Res:45.0,27.0] ||  -> M(sent(s,c,encr(triple(na,cpfc,ida
      ),cerc)))*.
143 52[0:Res:51.0,46.0] ||  -> M(sent(c,s,pair(encr(pair(na,sign(
      triple(codc,cpfc,templatec),krc)),cera),ida)))*.
144 53[0:Res:52.0,31.0] ||  -> M(sent(s,a,encr(pair(na,sign(
      triple(codc,cpfc,templatec),krc)),cera)))*.
145 54[0:Res:53.0,47.0] ||  -> M(sent(a,s,pair(sign(sign(triple(
      codc,cpfc,templatec),krc),kra),idd)))*.
146 56[0:Res:54.0,28.0] ||  -> Stores(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),d)*.
147 57[0:Res:54.0,24.0] ||  -> Stores(cpfc,d)*.
148 58[0:MRR:42.1,57.0] || M(sent(U,d,cpfc))* Stores(sign(sign(
      triple(codc,cpfc,templatec),krc),kra),d)* -> .
149 59[0:MRR:58.1,56.0] || M(sent(U,d,cpfc))* -> .
150 60[0:UnC:59.0,50.0] ||  -> .
151 Formulae used in the proof : A_knows_cpfc C_knows_cpfc
      C_knows_templatec A_knows_A_ku A_knows_C_ku A_knows_Na
      C_knows_A_ku S_knows_C_ku S_knows_Ns
      C_knows_id_data_triple A_knows_A_keypair
      C_knows_C_keypair test_auth_step_4 auth_step_1 reg_step_6
       auth_step_3 reg_step_2 reg_step_4 auth_step_4
      auth_step_2 reg_step_1 reg_step_3 reg_step_5
152
153 --------------------------SPASS-STOP
      ------------------------------
```

A.4.3 Fato 3

Saturação do modelo lógico estendido (Fórmulas 1 a 49).

```
1
2 --------------------------SPASS-START
      ------------------------------
3 Input Problem:
4 1[0:Inp] ||  -> E(a)*.
5 2[0:Inp] ||  -> E(c)*.
6 3[0:Inp] ||  -> E(s)*.
7 4[0:Inp] ||  -> E(d)*.
8 5[0:Inp] ||  -> E(i)*.
9 6[0:Inp] ||  -> Knows(cpfc,a)*.
10 7[0:Inp] ||  -> Knows(codc,c)*.
11 8[0:Inp] ||  -> Knows(cpfc,c)*.
12 9[0:Inp] ||  -> Knows(templatec,c)*.
13 10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
14 11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
15 12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
16 13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
17 14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
```

```
15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
23[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
24[0:Inp] || Im(pair(U,V))* -> Im(U).
25[0:Inp] || Im(pair(U,V))* -> Im(V).
26[0:Inp] || Im(sign(U,V))* -> Im(U).
27[0:Inp] || M(sent(U,V,W))* -> Im(W).
28[0:Inp] || Im(triple(U,V,W))* -> Im(U).
29[0:Inp] || Im(triple(U,V,W))* -> Im(V).
30[0:Inp] || Im(triple(U,V,W))* -> Im(W).
31[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
32[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
33[0:Inp] || Im(U) Im(V) -> Im(pair(U,V))*.
34[0:Inp] || Im(U) E(V) -> Knows(krkey(U,V),i)*.
35[0:Inp] || Im(U) E(V) -> Knows(kukey(U,V),i)*.
36[0:Inp] || Im(U) E(V) -> Knows(nonce(U,V),i)*.
37[0:Inp] || Im(U) Im(V) Im(W) -> Im(triple(U,V,W))*.
38[0:Inp] || Im(U) E(V) E(W) -> M(sent(V,W,U))*.
39[0:Inp] || Im(U) Knows(kukey(V,W),i)* E(W) -> Im(encr(U,V))
    *.
40[0:Inp] || Im(U) Knows(krkey(V,W),i)* E(W) -> Im(sign(U,V))
    *.
41[0:Inp] || Im(encr(U,V))* Knows(nonce(V,W),i)* E(W) -> Im(U
    ).
42[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) ->
    Stores(pair(ns,cerc),s)*.
43[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(cpfc,d).
44[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) -> M
    (sent(s,c,encr(ns,cerc)))*.
45[0:Inp] || Knows(kp(krkey(U,V),kukey(W,V)),i)* Im(encr(X,W)
    )* E(V) -> Im(X).
46[0:Inp] || M(sent(U,s,encr(triple(cpfc,ids,templatec),ns)))
    * Stores(pair(ns,cerc),s) -> M(sent(s,d,cpfc)).
47[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
    idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
48[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
    codc,cpfc,templatec),krc),kra),d).
49[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
    triple(codc,cpfc,templatec),krc),kra))).
50[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
    ),kukey(cerc,c)),c)* Knows(templatec,c) Knows(cpfc,c) ->
    Stores(pair(ns,U),c).
51[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
    cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
```

```
      pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
55 52[0:Inp] || M(sent(U,d,cpfc))* Stores(sign(sign(triple(codc,
      cpfc,templatec),krc),kra),d) Stores(cpfc,d) -> M(sent(d,s
      ,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
56 53[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
      ),kukey(cerc,c)),c) Knows(templatec,c) Knows(cpfc,c) -> M
      (sent(c,s,encr(triple(cpfc,ids,templatec),ns)))*.
57 54[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
      kukey(cera,a)),a)* Knows(kukey(cerc,c),a) Knows(cpfc,a)
      Knows(nonce(na,a),a) -> Stores(pair(na,cpfc),a).
58 55[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
      kukey(cera,a)),a) Knows(kukey(cerc,c),a) Knows(cpfc,a)
      Knows(nonce(na,a),a) -> M(sent(a,s,pair(encr(triple(na,
      cpfc,ida),cerc),idc)))*.
59 56[0:Inp] || M(sent(U,c,encr(triple(na,cpfc,ida),cerc)))*
      Knows(triple(codc,cpfc,templatec),c) Knows(kukey(cera,a),
      c) Knows(kp(krkey(krc,c),kukey(cerc,c)),c) -> M(sent(c,s,
      pair(encr(pair(na,sign(triple(codc,cpfc,templatec),krc)),
      cera),ida)))*.
60 57[0:Inp] || M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,
      templatec),krc)),cera)))* Stores(pair(na,cpfc),a) Knows(
      kukey(cerc,c),a) Knows(kp(krkey(kra,a),kukey(cera,a)),a)
      Knows(cpfc,a) Knows(nonce(na,a),a) -> M(sent(a,s,pair(
      sign(sign(triple(codc,cpfc,templatec),krc),kra),idd)))*.
61  This is a first-order Horn problem without equality.
62  This is a problem that contains sort information.
63  The following monadic predicates have finite extensions: E.
64  Axiom clauses: 57 Conjecture clauses: 0
65  Inferences: IEmS=1 ISoR=1 IORe=1
66  Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RSST=1
        RSSi=1 RFSub=1 RBSub=1 RCon=1
67  Extras    : Input Saturation, Dynamic Selection, No
        Splitting, Full Reduction,  Ratio: 5, FuncWeight: 1,
        VarWeight: 1
68  Precedence: pair > krkey > kukey > kp > nonce > encr > sign
        > triple > sent > Im > Stores > Knows > M > E > a > c >
        s > d > i > kra > cera > krc > cerc > na > ns > codc >
        cpfc > templatec > icodc > icpfc > itemplatec > ida >
        idc > idd > ids
69  Ordering  : KBO
70 Processed Problem:
71
72 Worked Off Clauses:
73
74 Usable Clauses:
75 5[0:Inp] ||  -> E(i)*.
76 4[0:Inp] ||  -> E(d)*.
77 3[0:Inp] ||  -> E(s)*.
78 2[0:Inp] ||  -> E(c)*.
79 1[0:Inp] ||  -> E(a)*.
80 9[0:Inp] ||  -> Knows(templatec,c)*.
81 8[0:Inp] ||  -> Knows(cpfc,c)*.
82 7[0:Inp] ||  -> Knows(codc,c)*.
```

```
 83 6[0:Inp] ||  -> Knows(cpfc,a)*.
 84 58[0:MRR:42.0,42.1,18.0,19.0] ||  -> Stores(pair(ns,cerc),s)
      *.
 85 63[0:MRR:54.0,54.1,54.2,54.3,54.4,11.0,31.0,12.0,6.0,13.0] ||
       -> Stores(pair(na,cpfc),a)*.
 86 22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
 87 21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
 88 18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
 89 17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
 90 16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
 91 12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
 92 19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
 93 15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
 94 14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
 95 13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
 96 11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
 97 10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
 98 20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
 99 23[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
100 59[0:MRR:44.0,44.1,18.0,19.0] ||  -> M(sent(s,c,encr(ns,cerc)
      ))*.
101 26[0:Inp] || Im(sign(U,V))* -> Im(U).
102 25[0:Inp] || Im(pair(U,V))* -> Im(V).
103 24[0:Inp] || Im(pair(U,V))* -> Im(U).
104 27[0:Inp] || M(sent(U,V,W))* -> Im(W).
105 30[0:Inp] || Im(triple(U,V,W))* -> Im(W).
106 29[0:Inp] || Im(triple(U,V,W))* -> Im(V).
107 28[0:Inp] || Im(triple(U,V,W))* -> Im(U).
108 32[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
109 31[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
110 33[0:Inp] Im(U) Im(V) ||  -> Im(pair(V,U))*.
111 36[0:Inp] E(U) Im(V) ||  -> Knows(nonce(V,U),i)*.
112 35[0:Inp] E(U) Im(V) ||  -> Knows(kukey(V,U),i)*.
113 34[0:Inp] E(U) Im(V) ||  -> Knows(krkey(V,U),i)*.
114 64[0:MRR:55.0,55.1,55.2,55.3,55.4,11.0,31.0,12.0,6.0,13.0] ||
       -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
      )*.
115 61[0:MRR:50.1,50.2,50.3,32.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> Stores(pair(ns,U),c).
116 38[0:Inp] E(U) E(V) Im(W) ||  -> M(sent(V,U,W))*.
117 37[0:Inp] Im(U) Im(V) Im(W) ||  -> Im(triple(W,V,U))*.
118 60[0:MRR:46.1,58.0] || M(sent(U,s,encr(triple(cpfc,ids,
      templatec),ns)))* -> M(sent(s,d,cpfc)).
119 40[0:Inp] E(U) Im(V) || Knows(krkey(W,U),i)* -> Im(sign(V,W))
      *.
120 39[0:Inp] E(U) Im(V) || Knows(kukey(W,U),i)* -> Im(encr(V,W))
      *.
121 41[0:Inp] E(U) || Im(encr(V,W))* Knows(nonce(W,U),i)* -> Im(V
      ).
122 43[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(cpfc,d).
123 62[0:MRR:53.1,53.2,53.3,32.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> M(sent(c,s,encr(triple(cpfc,ids,templatec),ns
```

```
      )))*.
124 45[0:Inp] E(U) || Im(encr(V,W))* Knows(kp(krkey(X,U),kukey(W,
      U)),i)* -> Im(V).
125 47[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
      idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
126 48[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
      codc,cpfc,templatec),krc),kra),d).
127 49[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
      triple(codc,cpfc,templatec),krc),kra))).
128 65[0:MRR:56.1,56.2,56.3,23.0,16.0,32.0] || M(sent(U,c,encr(
      triple(na,cpfc,ida),cerc)))* -> M(sent(c,s,pair(encr(pair
      (na,sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
129 66[0:MRR:57.1,57.2,57.3,57.4,57.5,63.0,12.0,31.0,6.0,13.0] ||
       M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,templatec)
      ,krc)),cera)))* -> M(sent(a,s,pair(sign(sign(triple(codc,
      cpfc,templatec),krc),kra),idd)))*.
130 51[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
      cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
      pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
131 52[0:Inp] || M(sent(U,d,cpfc))* Stores(cpfc,d) Stores(sign(
      sign(triple(codc,cpfc,templatec),krc),kra),d) -> M(sent(d
      ,s,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
132 SPASS V 3.5
133 SPASS beiseite: Completion found.
134 Problem: C:/Users/Eduardo/AppData/Local/Tempqt_temp.b02596
135 SPASS derived 93 clauses, backtracked 0 clauses, performed 0
      splits and kept 117 clauses.
136 SPASS allocated 30203 KBytes.
137 SPASS spent 0:00:00.22 on the problem.
138     0:00:00.02 for the input.
139     0:00:00.00 for the FLOTTER CNF translation.
140     0:00:00.00 for inferences.
141     0:00:00.00 for the backtracking.
142     0:00:00.00 for the reduction.
143
144
145  The saturated set of worked-off clauses is :
146 203[0:Res:179.0,51.0] ||  -> M(sent(s,a,encr(pair(na,sign(
      triple(codc,cpfc,templatec),krc)),cera)))*.
147 196[0:Res:157.0,195.0] ||  -> M(sent(d,s,sign(sign(triple(
      codc,cpfc,templatec),krc),kra)))*.
148 190[0:Res:188.0,49.0] ||  -> M(sent(s,d,sign(sign(triple(codc
      ,cpfc,templatec),krc),kra)))*.
149 208[0:Res:202.0,30.0] ||  -> Im(templatec)*.
150 206[0:Res:202.0,28.0] ||  -> Im(codc)*.
151 202[0:Res:201.0,26.0] ||  -> Im(triple(codc,cpfc,templatec))
      *.
152 201[0:Res:199.0,26.0] ||  -> Im(sign(triple(codc,cpfc,
      templatec),krc))*.
153 199[0:Res:189.0,24.0] ||  -> Im(sign(sign(triple(codc,cpfc,
      templatec),krc),kra))*.
```

```
154 200[0:Res:189.0,25.0] ||  -> Im(idd)*.
155 189[0:Res:188.0,27.0] ||  -> Im(pair(sign(sign(triple(codc,
        cpfc,templatec),krc),kra),idd))*.
156 191[0:Res:188.0,48.0] ||  -> Stores(sign(sign(triple(codc,
        cpfc,templatec),krc),kra),d)*.
157 192[0:Res:188.0,43.0] ||  -> Stores(cpfc,d)*.
158 188[0:SSi:187.1,187.0,183.0,1.0] ||  -> M(sent(a,s,pair(sign(
        sign(triple(codc,cpfc,templatec),krc),kra),idd)))*.
159 185[0:Res:183.0,139.1] Im(cera) ||  -> Im(pair(na,sign(triple
        (codc,cpfc,templatec),krc)))*.
160 183[0:Res:182.0,24.0] ||  -> Im(encr(pair(na,sign(triple(codc
        ,cpfc,templatec),krc)),cera))*.
161 184[0:Res:182.0,25.0] ||  -> Im(ida)*.
162 182[0:Res:179.0,27.0] ||  -> Im(pair(encr(pair(na,sign(triple
        (codc,cpfc,templatec),krc)),cera),ida))*.
163 179[0:Res:163.0,65.0] ||  -> M(sent(c,s,pair(encr(pair(na,
        sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
164 163[0:Res:64.0,47.0] ||  -> M(sent(s,c,encr(triple(na,cpfc,
        ida),cerc)))*.
165 159[0:Res:156.0,139.1] Im(ns) ||  -> Im(triple(cpfc,ids,
        templatec))*.
166 45[0:Inp] E(U) || Im(encr(V,W))* Knows(kp(krkey(X,U),kukey(W,
        U)),i)*+ -> Im(V).
167 156[0:Res:153.0,27.0] ||  -> Im(encr(triple(cpfc,ids,
        templatec),ns))*.
168 158[0:Res:157.0,27.0] ||  -> Im(cpfc)*.
169 157[0:Res:153.0,60.0] ||  -> M(sent(s,d,cpfc))*.
170 153[0:Res:59.0,62.0] ||  -> M(sent(c,s,encr(triple(cpfc,ids,
        templatec),ns)))*.
171 148[0:Res:105.0,139.1] Im(cerc) ||  -> Im(triple(na,cpfc,ida)
        )*.
172 147[0:Res:97.0,139.1] Im(cerc) ||  -> Im(ns)*.
173 139[0:EmS:132.1,1.0] Im(U) || Im(encr(V,U))* -> Im(V).
174 133[0:EmS:130.1,1.0] Im(U) Im(V) ||  -> Im(encr(V,U))*.
175 129[0:SSi:126.0,2.0] Im(U) ||  -> Im(encr(U,cerc))*.
176 41[0:Inp] E(U) || Im(encr(V,W))* Knows(nonce(W,U),i)*+ -> Im(
        V).
177 128[0:SSi:125.0,1.0] Im(U) ||  -> Im(encr(U,cera))*.
178 39[0:Inp] E(U) Im(V) || Knows(kukey(W,U),i)*+ -> Im(encr(V,W)
        )*.
179 118[0:EmS:117.1,1.0] Im(U) Im(V) ||  -> Im(sign(V,U))*.
180 40[0:Inp] E(U) Im(V) || Knows(krkey(W,U),i)*+ -> Im(sign(V,W)
        )*.
181 37[0:Inp] Im(U) Im(V) Im(W) ||  -> Im(triple(W,V,U))*.
182 109[0:SSi:108.2,108.0,97.0,2.0] E(U) ||  -> Stores(pair(ns,U)
        ,c)*.
183 38[0:Inp] E(U) E(V) Im(W) ||  -> M(sent(V,U,W))*.
184 105[0:Res:103.0,24.0] ||  -> Im(encr(triple(na,cpfc,ida),cerc
        ))*.
185 106[0:Res:103.0,25.0] ||  -> Im(idc)*.
186 103[0:Res:64.0,27.0] ||  -> Im(pair(encr(triple(na,cpfc,ida),
        cerc),idc))*.
187 104[0:Res:59.0,61.0] ||  -> Stores(pair(ns,s),c)*.
```

```
188 61[0:MRR:50.1,50.2,50.3,32.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> Stores(pair(ns,U),c).
189 64[0:MRR:55.0,55.1,55.2,55.3,55.4,11.0,31.0,12.0,6.0,13.0] ||
       -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
      )*.
190 34[0:Inp] E(U) Im(V) ||  -> Knows(krkey(V,U),i)*.
191 35[0:Inp] E(U) Im(V) ||  -> Knows(kukey(V,U),i)*.
192 36[0:Inp] E(U) Im(V) ||  -> Knows(nonce(V,U),i)*.
193 33[0:Inp] Im(U) Im(V) ||  -> Im(pair(V,U))*.
194 31[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
195 32[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
196 100[0:Res:20.0,28.0] ||  -> Im(icodc)*.
197 99[0:Res:20.0,29.0] ||  -> Im(icpfc)*.
198 28[0:Inp] || Im(triple(U,V,W))* -> Im(U).
199 29[0:Inp] || Im(triple(U,V,W))* -> Im(V).
200 98[0:Res:20.0,30.0] ||  -> Im(itemplatec)*.
201 30[0:Inp] || Im(triple(U,V,W))* -> Im(W).
202 97[0:Res:59.0,27.0] ||  -> Im(encr(ns,cerc))*.
203 27[0:Inp] || M(sent(U,V,W))* -> Im(W).
204 24[0:Inp] || Im(pair(U,V))* -> Im(U).
205 25[0:Inp] || Im(pair(U,V))* -> Im(V).
206 26[0:Inp] || Im(sign(U,V))* -> Im(U).
207 59[0:MRR:44.0,44.1,18.0,19.0] ||  -> M(sent(s,c,encr(ns,cerc)
      ))*.
208 23[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
209 20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
210 10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
211 11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
212 13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
213 14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
214 15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
215 19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
216 12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
217 16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
218 17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
219 18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
220 21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
221 22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
222 63[0:MRR:54.0,54.1,54.2,54.3,54.4,11.0,31.0,12.0,6.0,13.0] ||
       -> Stores(pair(na,cpfc),a)*.
223 58[0:MRR:42.0,42.1,18.0,19.0] ||  -> Stores(pair(ns,cerc),s)
      *.
224 6[0:Inp] ||  -> Knows(cpfc,a)*.
225 7[0:Inp] ||  -> Knows(codc,c)*.
226 8[0:Inp] ||  -> Knows(cpfc,c)*.
227 9[0:Inp] ||  -> Knows(templatec,c)*.
228 1[0:Inp] ||  -> E(a)*.
229 2[0:Inp] ||  -> E(c)*.
230 3[0:Inp] ||  -> E(s)*.
231 4[0:Inp] ||  -> E(d)*.
232 5[0:Inp] ||  -> E(i)*.
233 -------------------------SPASS-STOP
      ------------------------------
```

A.4.4 Fato 4

$$\Gamma_{estendido} \nvdash M(sent(i, d, sign(sign(triple(icodc, icpfc, itemplatec), krc), kra)))$$

```
1
2 --------------------------SPASS-START
      ------------------------------
3 Input Problem:
4 1[0:Inp] ||  -> E(a)*.
5 2[0:Inp] ||  -> E(c)*.
6 3[0:Inp] ||  -> E(s)*.
7 4[0:Inp] ||  -> E(d)*.
8 5[0:Inp] ||  -> E(i)*.
9 6[0:Inp] ||  -> Knows(cpfc,a)*.
10 7[0:Inp] ||  -> Knows(codc,c)*.
11 8[0:Inp] ||  -> Knows(cpfc,c)*.
12 9[0:Inp] ||  -> Knows(templatec,c)*.
13 10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
14 11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
15 12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
16 13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
17 14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
18 15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
19 16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
20 17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
21 18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
22 19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
23 20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
24 21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
25 22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
26 23[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
27 24[0:Inp] || Im(pair(U,V))* -> Im(U).
28 25[0:Inp] || Im(pair(U,V))* -> Im(V).
29 26[0:Inp] || Im(sign(U,V))* -> Im(U).
30 27[0:Inp] || M(sent(U,V,W))* -> Im(W).
31 28[0:Inp] || Im(triple(U,V,W))* -> Im(U).
32 29[0:Inp] || Im(triple(U,V,W))* -> Im(V).
33 30[0:Inp] || Im(triple(U,V,W))* -> Im(W).
34 31[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
35 32[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
36 33[0:Inp] || Im(U) Im(V) -> Im(pair(U,V))*.
37 34[0:Inp] || Im(U) E(V) -> Knows(krkey(U,V),i)*.
38 35[0:Inp] || Im(U) E(V) -> Knows(kukey(U,V),i)*.
39 36[0:Inp] || Im(U) E(V) -> Knows(nonce(U,V),i)*.
40 37[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)))* -> .
41 38[0:Inp] || Im(U) Im(V) Im(W) -> Im(triple(U,V,W))*.
42 39[0:Inp] || Im(U) E(V) E(W) -> M(sent(V,W,U))*.
43 40[0:Inp] || Im(U) Knows(kukey(V,W),i)* E(W) -> Im(encr(U,V))
      *.
44 41[0:Inp] || Im(U) Knows(krkey(V,W),i)* E(W) -> Im(sign(U,V))
      *.
```

```
45 42[0:Inp] || Im(encr(U,V))* Knows(nonce(V,W),i)* E(W) -> Im(U
       ).
46 43[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) ->
       Stores(pair(ns,cerc),s)*.
47 44[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> Stores(cpfc,d).
48 45[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) -> M
       (sent(s,c,encr(ns,cerc)))*.
49 46[0:Inp] || Knows(kp(krkey(U,V),kukey(W,V)),i)* Im(encr(X,W)
       )* E(V) -> Im(X).
50 47[0:Inp] || M(sent(U,s,encr(triple(cpfc,ids,templatec),ns)))
       * Stores(pair(ns,cerc),s) -> M(sent(s,d,cpfc)).
51 48[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
       idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
52 49[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
       codc,cpfc,templatec),krc),kra),d).
53 50[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
       triple(codc,cpfc,templatec),krc),kra))).
54 51[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
       ),kukey(cerc,c)),c)* Knows(templatec,c) Knows(cpfc,c) ->
       Stores(pair(ns,U),c).
55 52[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
       cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
       pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
56 53[0:Inp] || M(sent(U,d,cpfc))* Stores(sign(sign(triple(codc,
       cpfc,templatec),krc),kra),d) Stores(cpfc,d) -> M(sent(d,s
       ,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
57 54[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
       ),kukey(cerc,c)),c) Knows(templatec,c) Knows(cpfc,c) -> M
       (sent(c,s,encr(triple(cpfc,ids,templatec),ns)))*.
58 55[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
       kukey(cera,a)),a)* Knows(kukey(cerc,c),a) Knows(cpfc,a)
       Knows(nonce(na,a),a) -> Stores(pair(na,cpfc),a).
59 56[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
       kukey(cera,a)),a) Knows(kukey(cerc,c),a) Knows(cpfc,a)
       Knows(nonce(na,a),a) -> M(sent(a,s,pair(encr(triple(na,
       cpfc,ida),cerc),idc)))*.
60 57[0:Inp] || M(sent(U,c,encr(triple(na,cpfc,ida),cerc)))*
       Knows(triple(codc,cpfc,templatec),c) Knows(kukey(cera,a),
       c) Knows(kp(krkey(krc,c),kukey(cerc,c)),c) -> M(sent(c,s,
       pair(encr(pair(na,sign(triple(codc,cpfc,templatec),krc)),
       cera),ida)))*.
61 58[0:Inp] || M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,
       templatec),krc)),cera)))* Stores(pair(na,cpfc),a) Knows(
       kukey(cerc,c),a) Knows(kp(krkey(kra,a),kukey(cera,a)),a)
       Knows(cpfc,a) Knows(nonce(na,a),a) -> M(sent(a,s,pair(
       sign(sign(triple(codc,cpfc,templatec),krc),kra),idd)))*.
62  This is a first-order Horn problem without equality.
63  This is a problem that contains sort information.
64  The conjecture is ground.
65  The following monadic predicates have finite extensions: E.
```

```
66  Axiom clauses: 57 Conjecture clauses: 1
67  Inferences: IEmS=1 ISoR=1 IORe=1
68  Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RSST=1
        RSSi=1 RFSub=1 RBSub=1 RCon=1
69  Extras    : Input Saturation, Dynamic Selection, No
        Splitting, Full Reduction,  Ratio: 5, FuncWeight: 1,
        VarWeight: 1
70  Precedence: pair > krkey > kukey > kp > nonce > encr > sign
        > triple > sent > Im > Stores > Knows > M > E > a > c >
        s > d > i > kra > cera > krc > cerc > na > ns > codc >
        cpfc > templatec > icodc > icpfc > itemplatec > ida >
        idc > idd > ids
71  Ordering  : KBO
72 Processed Problem:
73
74 Worked Off Clauses:
75
76 Usable Clauses:
77 5[0:Inp] ||  -> E(i)*.
78 4[0:Inp] ||  -> E(d)*.
79 3[0:Inp] ||  -> E(s)*.
80 2[0:Inp] ||  -> E(c)*.
81 1[0:Inp] ||  -> E(a)*.
82 9[0:Inp] ||  -> Knows(templatec,c)*.
83 8[0:Inp] ||  -> Knows(cpfc,c)*.
84 7[0:Inp] ||  -> Knows(codc,c)*.
85 6[0:Inp] ||  -> Knows(cpfc,a)*.
86 59[0:MRR:43.0,43.1,18.0,19.0] ||  -> Stores(pair(ns,cerc),s)
       *.
87 64[0:MRR:55.0,55.1,55.2,55.3,55.4,11.0,31.0,12.0,6.0,13.0] ||
         -> Stores(pair(na,cpfc),a)*.
88 22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
89 21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
90 18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
91 17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
92 16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
93 12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
94 19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
95 15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
96 14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
97 13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
98 11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
99 10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
100 20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
101 23[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
102 60[0:MRR:45.0,45.1,18.0,19.0] ||  -> M(sent(s,c,encr(ns,cerc)
       ))*.
103 26[0:Inp] || Im(sign(U,V))* -> Im(U).
104 25[0:Inp] || Im(pair(U,V))* -> Im(V).
105 24[0:Inp] || Im(pair(U,V))* -> Im(U).
106 27[0:Inp] || M(sent(U,V,W))* -> Im(W).
107 30[0:Inp] || Im(triple(U,V,W))* -> Im(W).
108 29[0:Inp] || Im(triple(U,V,W))* -> Im(V).
```

```
109 28[0:Inp] || Im(triple(U,V,W))* -> Im(U).
110 32[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
111 31[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
112 33[0:Inp] Im(U) Im(V) ||  -> Im(pair(V,U))*.
113 69[0:MRR:68.1,68.2,5.0,4.0] Im(sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)) ||  -> .
114 36[0:Inp] E(U) Im(V) ||  -> Knows(nonce(V,U),i)*.
115 35[0:Inp] E(U) Im(V) ||  -> Knows(kukey(V,U),i)*.
116 34[0:Inp] E(U) Im(V) ||  -> Knows(krkey(V,U),i)*.
117 65[0:MRR:56.0,56.1,56.2,56.3,56.4,11.0,31.0,12.0,6.0,13.0] ||
        -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
      )*.
118 37[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)))* -> .
119 62[0:MRR:51.1,51.2,51.3,32.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> Stores(pair(ns,U),c).
120 39[0:Inp] E(U) E(V) Im(W) ||  -> M(sent(V,U,W))*.
121 38[0:Inp] Im(U) Im(V) Im(W) ||  -> Im(triple(W,V,U))*.
122 61[0:MRR:47.1,59.0] || M(sent(U,s,encr(triple(cpfc,ids,
      templatec),ns)))* -> M(sent(s,d,cpfc)).
123 41[0:Inp] E(U) Im(V) || Knows(krkey(W,U),i)* -> Im(sign(V,W))
      *.
124 40[0:Inp] E(U) Im(V) || Knows(kukey(W,U),i)* -> Im(encr(V,W))
      *.
125 42[0:Inp] E(U) || Im(encr(V,W))* Knows(nonce(W,U),i)* -> Im(V
      ).
126 44[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(cpfc,d).
127 63[0:MRR:54.1,54.2,54.3,32.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> M(sent(c,s,encr(triple(cpfc,ids,templatec),ns
      )))*.
128 46[0:Inp] E(U) || Im(encr(V,W))* Knows(kp(krkey(X,U),kukey(W,
      U)),i)* -> Im(V).
129 48[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
      idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
130 49[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
      codc,cpfc,templatec),krc),kra),d).
131 50[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
      triple(codc,cpfc,templatec),krc),kra))).
132 66[0:MRR:57.1,57.2,57.3,23.0,16.0,32.0] || M(sent(U,c,encr(
      triple(na,cpfc,ida),cerc)))* -> M(sent(c,s,pair(encr(pair
      (na,sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
133 67[0:MRR:58.1,58.2,58.3,58.4,58.5,64.0,12.0,31.0,6.0,13.0] ||
       M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,templatec)
      ,krc)),cera)))* -> M(sent(a,s,pair(sign(sign(triple(codc,
      cpfc,templatec),krc),kra),idd)))*.
134 52[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
      cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
      pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
135 53[0:Inp] || M(sent(U,d,cpfc))* Stores(cpfc,d) Stores(sign(
      sign(triple(codc,cpfc,templatec),krc),kra),d) -> M(sent(d
```

```
      ,s,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
SPASS V 3.5
SPASS beiseite: Completion found.
Problem: C:/Users/Eduardo/AppData/Local/Tempqt_temp.b02596
SPASS derived 103 clauses, backtracked 0 clauses, performed 0
      splits and kept 126 clauses.
SPASS allocated 30205 KBytes.
SPASS spent 0:00:00.22 on the problem.
    0:00:00.00 for the input.
    0:00:00.02 for the FLOTTER CNF translation.
    0:00:00.00 for inferences.
    0:00:00.00 for the backtracking.
    0:00:00.00 for the reduction.


 The saturated set of worked-off clauses is :
207[0:Res:196.0,52.0] ||  -> M(sent(s,a,encr(pair(na,sign(
      triple(codc,cpfc,templatec),krc)),cera)))*.
217[0:Res:167.0,216.0] ||  -> M(sent(d,s,sign(sign(triple(
      codc,cpfc,templatec),krc),kra)))*.
227[0:Res:223.0,30.0] ||  -> Im(templatec)*.
225[0:Res:223.0,28.0] ||  -> Im(codc)*.
223[0:Res:222.0,26.0] ||  -> Im(triple(codc,cpfc,templatec))
      *.
211[0:Res:205.0,50.0] ||  -> M(sent(s,d,sign(sign(triple(codc
      ,cpfc,templatec),krc),kra)))*.
222[0:Res:220.0,26.0] ||  -> Im(sign(triple(codc,cpfc,
      templatec),krc))*.
220[0:Res:210.0,24.0] ||  -> Im(sign(sign(triple(codc,cpfc,
      templatec),krc),kra))*.
221[0:Res:210.0,25.0] ||  -> Im(idd)*.
210[0:Res:205.0,27.0] ||  -> Im(pair(sign(sign(triple(codc,
      cpfc,templatec),krc),kra),idd))*.
212[0:Res:205.0,49.0] ||  -> Stores(sign(sign(triple(codc,
      cpfc,templatec),krc),kra),d)*.
213[0:Res:205.0,44.0] ||  -> Stores(cpfc,d)*.
205[0:MRR:201.0,203.0] ||  -> M(sent(a,s,pair(sign(sign(
      triple(codc,cpfc,templatec),krc),kra),idd)))*.
206[0:Res:203.0,149.1] Im(cera) ||  -> Im(pair(na,sign(triple
      (codc,cpfc,templatec),krc)))*.
203[0:Res:202.0,24.0] ||  -> Im(encr(pair(na,sign(triple(codc
      ,cpfc,templatec),krc)),cera))*.
204[0:Res:202.0,25.0] ||  -> Im(ida)*.
202[0:Res:196.0,27.0] ||  -> Im(pair(encr(pair(na,sign(triple
      (codc,cpfc,templatec),krc)),cera),ida))*.
196[0:Res:173.0,66.0] ||  -> M(sent(c,s,pair(encr(pair(na,
      sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
180[0:SSi:178.1,20.0] E(U) E(V) || Knows(krkey(kra,U),i)*+
      Knows(krkey(krc,V),i)* -> .
179[0:SSi:177.1,20.0] E(U) Im(krc) || Knows(krkey(kra,U),i)*
      -> .
120[0:SoR:69.0,41.3] Im(sign(triple(icodc,icpfc,itemplatec),
      krc)) E(U) || Knows(krkey(kra,U),i)* -> .
```

```
171 173[0:Res:65.0,48.0] ||  -> M(sent(s,c,encr(triple(na,cpfc,
        ida),cerc)))*.
172 169[0:Res:166.0,149.1] Im(ns) ||  -> Im(triple(cpfc,ids,
        templatec))*.
173 46[0:Inp] E(U) || Im(encr(V,W))* Knows(kp(krkey(X,U),kukey(W,
        U)),i)*+ -> Im(V).
174 166[0:Res:163.0,27.0] ||  -> Im(encr(triple(cpfc,ids,
        templatec),ns))*.
175 168[0:Res:167.0,27.0] ||  -> Im(cpfc)*.
176 167[0:Res:163.0,61.0] ||  -> M(sent(s,d,cpfc))*.
177 163[0:Res:60.0,63.0] ||  -> M(sent(c,s,encr(triple(cpfc,ids,
        templatec),ns)))*.
178 158[0:Res:107.0,149.1] Im(cerc) ||  -> Im(triple(na,cpfc,ida)
        )*.
179 157[0:Res:100.0,149.1] Im(cerc) ||  -> Im(ns)*.
180 149[0:EmS:142.1,1.0] Im(U) || Im(encr(V,U))* -> Im(V).
181 143[0:EmS:136.1,1.0] Im(U) Im(V) ||  -> Im(encr(V,U))*.
182 140[0:SSi:138.1,20.0] Im(kra) E(U) || Knows(krkey(krc,U),i)*
        -> .
183 42[0:Inp] E(U) || Im(encr(V,W))* Knows(nonce(W,U),i)*+ -> Im(
        V).
184 139[0:SSi:137.1,20.0] Im(kra) Im(krc) ||  -> .
185 128[0:SoR:69.0,123.2] Im(sign(triple(icodc,icpfc,itemplatec),
        krc)) Im(kra) ||  -> .
186 135[0:SSi:132.0,2.0] Im(U) ||  -> Im(encr(U,cerc))*.
187 134[0:SSi:131.0,1.0] Im(U) ||  -> Im(encr(U,cera))*.
188 40[0:Inp] E(U) Im(V) || Knows(kukey(W,U),i)*+ -> Im(encr(V,W)
        )*.
189 123[0:EmS:122.1,1.0] Im(U) Im(V) ||  -> Im(sign(V,U))*.
190 41[0:Inp] E(U) Im(V) || Knows(krkey(W,U),i)*+ -> Im(sign(V,W)
        )*.
191 38[0:Inp] Im(U) Im(V) Im(W) ||  -> Im(triple(W,V,U))*.
192 113[0:SSi:111.2,111.0,100.0,2.0] E(U) ||  -> Stores(pair(ns,U
        ),c)*.
193 39[0:Inp] E(U) E(V) Im(W) ||  -> M(sent(V,U,W))*.
194 109[0:Res:60.0,62.0] ||  -> Stores(pair(ns,s),c)*.
195 62[0:MRR:51.1,51.2,51.3,32.0,9.0,8.0] || M(sent(U,c,encr(ns,
        cerc)))* -> Stores(pair(ns,U),c).
196 37[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
        itemplatec),krc),kra)))* -> .
197 107[0:Res:106.0,24.0] ||  -> Im(encr(triple(na,cpfc,ida),cerc
        ))*.
198 108[0:Res:106.0,25.0] ||  -> Im(idc)*.
199 106[0:Res:65.0,27.0] ||  -> Im(pair(encr(triple(na,cpfc,ida),
        cerc),idc))*.
200 65[0:MRR:56.0,56.1,56.2,56.3,56.4,11.0,31.0,12.0,6.0,13.0] ||
          -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
        )*.
201 34[0:Inp] E(U) Im(V) ||  -> Knows(krkey(V,U),i)*.
202 35[0:Inp] E(U) Im(V) ||  -> Knows(kukey(V,U),i)*.
203 36[0:Inp] E(U) Im(V) ||  -> Knows(nonce(V,U),i)*.
204 69[0:MRR:68.1,68.2,5.0,4.0] Im(sign(sign(triple(icodc,icpfc,
        itemplatec),krc),kra)) ||  -> .
```

```
205 33[0:Inp] Im(U) Im(V) ||  -> Im(pair(V,U))*.
206 31[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
207 32[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
208 103[0:Res:20.0,28.0] ||  -> Im(icodc)*.
209 102[0:Res:20.0,29.0] ||  -> Im(icpfc)*.
210 28[0:Inp] || Im(triple(U,V,W))* -> Im(U).
211 29[0:Inp] || Im(triple(U,V,W))* -> Im(V).
212 101[0:Res:20.0,30.0] ||  -> Im(itemplatec)*.
213 30[0:Inp] || Im(triple(U,V,W))* -> Im(W).
214 100[0:Res:60.0,27.0] ||  -> Im(encr(ns,cerc))*.
215 27[0:Inp] || M(sent(U,V,W))* -> Im(W).
216 24[0:Inp] || Im(pair(U,V))* -> Im(U).
217 25[0:Inp] || Im(pair(U,V))* -> Im(V).
218 26[0:Inp] || Im(sign(U,V))* -> Im(U).
219 60[0:MRR:45.0,45.1,18.0,19.0] ||  -> M(sent(s,c,encr(ns,cerc)
      ))*.
220 23[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
221 20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
222 10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
223 11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
224 13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
225 14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
226 15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
227 19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
228 12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
229 16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
230 17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
231 18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
232 21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
233 22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
234 64[0:MRR:55.0,55.1,55.2,55.3,55.4,11.0,31.0,12.0,6.0,13.0] ||
        -> Stores(pair(na,cpfc),a)*.
235 59[0:MRR:43.0,43.1,18.0,19.0] ||  -> Stores(pair(ns,cerc),s)
      *.
236 6[0:Inp] ||  -> Knows(cpfc,a)*.
237 7[0:Inp] ||  -> Knows(codc,c)*.
238 8[0:Inp] ||  -> Knows(cpfc,c)*.
239 9[0:Inp] ||  -> Knows(templatec,c)*.
240 1[0:Inp] ||  -> E(a)*.
241 2[0:Inp] ||  -> E(c)*.
242 3[0:Inp] ||  -> E(s)*.
243 4[0:Inp] ||  -> E(d)*.
244 5[0:Inp] ||  -> E(i)*.
245 -------------------------SPASS-STOP
      ------------------------------
```

A.4.5 Fato 5

$$\Gamma_{estendido} \cup \{\text{ Fórmula 50 }\} \nvdash M(sent(i, d, sign(sign(triple(icodc, icpfc, itemplatec), krc), kra)))$$

1

```
2 --------------------------SPASS-START
      ----------------------------
3 Input Problem:
4 1[0:Inp] ||  -> E(a)*.
5 2[0:Inp] ||  -> E(c)*.
6 3[0:Inp] ||  -> E(s)*.
7 4[0:Inp] ||  -> E(d)*.
8 5[0:Inp] ||  -> E(i)*.
9 6[0:Inp] ||  -> Knows(cpfc,a)*.
10 7[0:Inp] ||  -> Knows(codc,c)*.
11 8[0:Inp] ||  -> Knows(cpfc,c)*.
12 9[0:Inp] ||  -> Knows(templatec,c)*.
13 10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
14 11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
15 12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
16 13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
17 14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
18 15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
19 16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
20 17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
21 18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
22 19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
23 20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
24 21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
25 22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
26 23[0:Inp] ||  -> Knows(krkey(kra,a),i)*.
27 24[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
28 25[0:Inp] || Im(pair(U,V))* -> Im(U).
29 26[0:Inp] || Im(pair(U,V))* -> Im(V).
30 27[0:Inp] || Im(sign(U,V))* -> Im(U).
31 28[0:Inp] || M(sent(U,V,W))* -> Im(W).
32 29[0:Inp] || Im(triple(U,V,W))* -> Im(U).
33 30[0:Inp] || Im(triple(U,V,W))* -> Im(V).
34 31[0:Inp] || Im(triple(U,V,W))* -> Im(W).
35 32[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
36 33[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
37 34[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),i)*.
38 35[0:Inp] || Im(U) Im(V) -> Im(pair(U,V))*.
39 36[0:Inp] || Im(U) E(V) -> Knows(krkey(U,V),i)*.
40 37[0:Inp] || Im(U) E(V) -> Knows(kukey(U,V),i)*.
41 38[0:Inp] || Im(U) E(V) -> Knows(nonce(U,V),i)*.
42 39[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
       itemplatec),krc),kra)))* -> .
43 40[0:Inp] || Im(U) Im(V) Im(W) -> Im(triple(U,V,W))*.
44 41[0:Inp] || Im(U) E(V) E(W) -> M(sent(V,W,U))*.
45 42[0:Inp] || Im(U) Knows(kukey(V,W),i)* E(W) -> Im(encr(U,V))
       *.
46 43[0:Inp] || Im(U) Knows(krkey(V,W),i)* E(W) -> Im(sign(U,V))
       *.
47 44[0:Inp] || Im(encr(U,V))* Knows(nonce(V,W),i)* E(W) -> Im(U
       ).
48 45[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) ->
       Stores(pair(ns,cerc),s)*.
```

```
46[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(cpfc,d).
47[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) -> M
    (sent(s,c,encr(ns,cerc)))*.
48[0:Inp] || Knows(kp(krkey(U,V),kukey(W,V)),i)* Im(encr(X,W)
    )* E(V) -> Im(X).
49[0:Inp] || M(sent(U,s,encr(triple(cpfc,ids,templatec),ns)))
    * Stores(pair(ns,cerc),s) -> M(sent(s,d,cpfc)).
50[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
    idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
51[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
    codc,cpfc,templatec),krc),kra),d).
52[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
    triple(codc,cpfc,templatec),krc),kra))).
53[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
    ),kukey(cerc,c)),c)* Knows(templatec,c) Knows(cpfc,c) ->
    Stores(pair(ns,U),c).
54[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
    cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
    pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
55[0:Inp] || M(sent(U,d,cpfc))* Stores(sign(sign(triple(codc,
    cpfc,templatec),krc),kra),d) Stores(cpfc,d) -> M(sent(d,s
    ,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
56[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
    ),kukey(cerc,c)),c) Knows(templatec,c) Knows(cpfc,c) -> M
    (sent(c,s,encr(triple(cpfc,ids,templatec),ns)))*.
57[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
    kukey(cera,a)),a)* Knows(kukey(cerc,c),a) Knows(cpfc,a)
    Knows(nonce(na,a),a) -> Stores(pair(na,cpfc),a).
58[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
    kukey(cera,a)),a) Knows(kukey(cerc,c),a) Knows(cpfc,a)
    Knows(nonce(na,a),a) -> M(sent(a,s,pair(encr(triple(na,
    cpfc,ida),cerc),idc)))*.
59[0:Inp] || M(sent(U,c,encr(triple(na,cpfc,ida),cerc)))*
    Knows(triple(codc,cpfc,templatec),c) Knows(kukey(cera,a),
    c) Knows(kp(krkey(krc,c),kukey(cerc,c)),c) -> M(sent(c,s,
    pair(encr(pair(na,sign(triple(codc,cpfc,templatec),krc)),
    cera),ida)))*.
60[0:Inp] || M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,
    templatec),krc)),cera)))* Stores(pair(na,cpfc),a) Knows(
    kukey(cerc,c),a) Knows(kp(krkey(kra,a),kukey(cera,a)),a)
    Knows(cpfc,a) Knows(nonce(na,a),a) -> M(sent(a,s,pair(
    sign(sign(triple(codc,cpfc,templatec),krc),kra),idd)))*.
 This is a first-order Horn problem without equality.
 This is a problem that contains sort information.
 The conjecture is ground.
 The following monadic predicates have finite extensions: E.
 Axiom clauses: 59 Conjecture clauses: 1
 Inferences: IEmS=1 ISoR=1 IORe=1
 Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RSST=1
     RSSi=1 RFSub=1 RBSub=1 RCon=1
```

```
 Extras     : Input Saturation, Dynamic Selection, No
      Splitting, Full Reduction,  Ratio: 5, FuncWeight: 1,
      VarWeight: 1
 Precedence: pair > krkey > kukey > kp > nonce > encr > sign
      > triple > sent > Im > Stores > Knows > M > E > a > c >
      s > d > i > kra > cera > krc > cerc > na > ns > codc >
      cpfc > templatec > icodc > icpfc > itemplatec > ida >
      idc > idd > ids
 Ordering  : KBO
Processed Problem:

Worked Off Clauses:

Usable Clauses:
5[0:Inp] ||  -> E(i)*.
4[0:Inp] ||  -> E(d)*.
3[0:Inp] ||  -> E(s)*.
2[0:Inp] ||  -> E(c)*.
1[0:Inp] ||  -> E(a)*.
9[0:Inp] ||  -> Knows(templatec,c)*.
8[0:Inp] ||  -> Knows(cpfc,c)*.
7[0:Inp] ||  -> Knows(codc,c)*.
6[0:Inp] ||  -> Knows(cpfc,a)*.
61[0:MRR:45.0,45.1,18.0,19.0] ||  -> Stores(pair(ns,cerc),s)
     *.
66[0:MRR:57.0,57.1,57.2,57.3,57.4,11.0,32.0,12.0,6.0,13.0] ||
      -> Stores(pair(na,cpfc),a)*.
23[0:Inp] ||  -> Knows(krkey(kra,a),i)*.
22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
24[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
62[0:MRR:47.0,47.1,18.0,19.0] ||  -> M(sent(s,c,encr(ns,cerc)
     ))*.
27[0:Inp] || Im(sign(U,V))* -> Im(U).
26[0:Inp] || Im(pair(U,V))* -> Im(V).
25[0:Inp] || Im(pair(U,V))* -> Im(U).
28[0:Inp] || M(sent(U,V,W))* -> Im(W).
31[0:Inp] || Im(triple(U,V,W))* -> Im(W).
30[0:Inp] || Im(triple(U,V,W))* -> Im(V).
29[0:Inp] || Im(triple(U,V,W))* -> Im(U).
34[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),i)*.
33[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
```

```
115 32[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
116 35[0:Inp] Im(U) Im(V) ||  -> Im(pair(V,U))*.
117 71[0:MRR:70.1,70.2,5.0,4.0] Im(sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)) ||  -> .
118 38[0:Inp] E(U) Im(V) ||  -> Knows(nonce(V,U),i)*.
119 37[0:Inp] E(U) Im(V) ||  -> Knows(kukey(V,U),i)*.
120 36[0:Inp] E(U) Im(V) ||  -> Knows(krkey(V,U),i)*.
121 67[0:MRR:58.0,58.1,58.2,58.3,58.4,11.0,32.0,12.0,6.0,13.0] ||
       -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
      )*.
122 39[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)))* -> .
123 64[0:MRR:53.1,53.2,53.3,33.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> Stores(pair(ns,U),c).
124 41[0:Inp] E(U) E(V) Im(W) ||  -> M(sent(V,U,W))*.
125 40[0:Inp] Im(U) Im(V) Im(W) ||  -> Im(triple(W,V,U))*.
126 63[0:MRR:49.1,61.0] || M(sent(U,s,encr(triple(cpfc,ids,
      templatec),ns)))* -> M(sent(s,d,cpfc)).
127 43[0:Inp] E(U) Im(V) || Knows(krkey(W,U),i)* -> Im(sign(V,W))
      *.
128 42[0:Inp] E(U) Im(V) || Knows(kukey(W,U),i)* -> Im(encr(V,W))
      *.
129 44[0:Inp] E(U) || Im(encr(V,W))* Knows(nonce(W,U),i)* -> Im(V
      ).
130 46[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(cpfc,d).
131 65[0:MRR:56.1,56.2,56.3,33.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> M(sent(c,s,encr(triple(cpfc,ids,templatec),ns
      )))*.
132 48[0:Inp] E(U) || Im(encr(V,W))* Knows(kp(krkey(X,U),kukey(W,
      U)),i)* -> Im(V).
133 50[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
      idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
134 51[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
      codc,cpfc,templatec),krc),kra),d).
135 52[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
      triple(codc,cpfc,templatec),krc),kra))).
136 68[0:MRR:59.1,59.2,59.3,24.0,16.0,33.0] || M(sent(U,c,encr(
      triple(na,cpfc,ida),cerc)))* -> M(sent(c,s,pair(encr(pair
      (na,sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
137 69[0:MRR:60.1,60.2,60.3,60.4,60.5,66.0,12.0,32.0,6.0,13.0] ||
       M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,templatec)
      ,krc)),cera)))* -> M(sent(a,s,pair(sign(sign(triple(codc,
      cpfc,templatec),krc),kra),idd)))*.
138 54[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
      cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
      pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
139 55[0:Inp] || M(sent(U,d,cpfc))* Stores(cpfc,d) Stores(sign(
      sign(triple(codc,cpfc,templatec),krc),kra),d) -> M(sent(d
      ,s,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
140 SPASS V 3.5
```

```
SPASS beiseite: Completion found.
Problem: C:/Users/Eduardo/AppData/Local/Tempqt_temp.b02596
SPASS derived 115 clauses, backtracked 0 clauses, performed 0
     splits and kept 129 clauses.
SPASS allocated 30209 KBytes.
SPASS spent 0:00:00.23 on the problem.
    0:00:00.02 for the input.
    0:00:00.02 for the FLOTTER CNF translation.
    0:00:00.00 for inferences.
    0:00:00.00 for the backtracking.
    0:00:00.00 for the reduction.


 The saturated set of worked-off clauses is :
215[0:Res:203.0,54.0] ||  -> M(sent(s,a,encr(pair(na,sign(
    triple(codc,cpfc,templatec),krc)),cera)))*.
236[0:Res:169.0,233.0] ||  -> M(sent(d,s,sign(sign(triple(
    codc,cpfc,templatec),krc),kra)))*.
228[0:Res:212.0,52.0] ||  -> M(sent(s,d,sign(sign(triple(codc
    ,cpfc,templatec),krc),kra)))*.
234[0:Res:227.0,25.0] ||  -> Im(sign(sign(triple(codc,cpfc,
    templatec),krc),kra))*.
235[0:Res:227.0,26.0] ||  -> Im(idd)*.
227[0:Res:212.0,28.0] ||  -> Im(pair(sign(sign(triple(codc,
    cpfc,templatec),krc),kra),idd))*.
229[0:Res:212.0,51.0] ||  -> Stores(sign(sign(triple(codc,
    cpfc,templatec),krc),kra),d)*.
230[0:Res:212.0,46.0] ||  -> Stores(cpfc,d)*.
212[0:MRR:208.0,210.0] ||  -> M(sent(a,s,pair(sign(sign(
    triple(codc,cpfc,templatec),krc),kra),idd)))*.
226[0:Res:223.0,31.0] ||  -> Im(templatec)*.
224[0:Res:223.0,29.0] ||  -> Im(codc)*.
223[0:Res:219.0,27.0] ||  -> Im(triple(codc,cpfc,templatec))
    *.
219[0:Res:214.0,26.0] ||  -> Im(sign(triple(codc,cpfc,
    templatec),krc))*.
218[0:Res:214.0,25.0] ||  -> Im(na)*.
214[0:Res:210.0,172.0] ||  -> Im(pair(na,sign(triple(codc,
    cpfc,templatec),krc)))*.
210[0:Res:209.0,25.0] ||  -> Im(encr(pair(na,sign(triple(codc
    ,cpfc,templatec),krc)),cera))*.
211[0:Res:209.0,26.0] ||  -> Im(ida)*.
209[0:Res:203.0,28.0] ||  -> Im(pair(encr(pair(na,sign(triple
    (codc,cpfc,templatec),krc)),cera),ida))*.
203[0:Res:181.0,68.0] ||  -> M(sent(c,s,pair(encr(pair(na,
    sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
181[0:Res:67.0,50.0] ||  -> M(sent(s,c,encr(triple(na,cpfc,
    ida),cerc)))*.
180[0:Res:168.0,160.1] Im(ns) ||  -> Im(triple(cpfc,ids,
    templatec))*.
176[0:Res:109.0,160.1] Im(cerc) ||  -> Im(triple(na,cpfc,ida)
    )*.
175[0:Res:102.0,160.1] Im(cerc) ||  -> Im(ns)*.
```

```
177 160[0:EmS:139.1,1.0] Im(U) || Im(encr(V,U))* -> Im(V).
178 172[0:SSi:171.0,1.0] || Im(encr(U,cera))* -> Im(U).
179 48[0:Inp] E(U) || Im(encr(V,W))* Knows(kp(krkey(X,U),kukey(W,
      U)),i)*+ -> Im(V).
180 168[0:Res:165.0,28.0] ||  -> Im(encr(triple(cpfc,ids,
      templatec),ns))*.
181 170[0:Res:169.0,28.0] ||  -> Im(cpfc)*.
182 169[0:Res:165.0,63.0] ||  -> M(sent(s,d,cpfc))*.
183 165[0:Res:62.0,65.0] ||  -> M(sent(c,s,encr(triple(cpfc,ids,
      templatec),ns)))*.
184 154[0:EmS:133.1,1.0] Im(U) Im(V) ||  -> Im(encr(V,U))*.
185 150[0:SSi:148.0,20.0] Im(krc) ||  -> .
186 142[0:EmS:126.1,1.0] Im(U) Im(V) ||  -> Im(sign(V,U))*.
187 137[0:SSi:136.0,20.0] E(U) || Knows(krkey(krc,U),i)* -> .
188 44[0:Inp] E(U) || Im(encr(V,W))* Knows(nonce(W,U),i)*+ -> Im(
      V).
189 134[0:SoR:71.0,125.1] Im(sign(triple(icodc,icpfc,itemplatec),
      krc)) ||  -> .
190 132[0:SSi:129.0,2.0] Im(U) ||  -> Im(encr(U,cerc))*.
191 131[0:SSi:128.0,1.0] Im(U) ||  -> Im(encr(U,cera))*.
192 125[0:SSi:123.0,1.0] Im(U) ||  -> Im(sign(U,kra))*.
193 42[0:Inp] E(U) Im(V) || Knows(kukey(W,U),i)*+ -> Im(encr(V,W)
      )*.
194 43[0:Inp] E(U) Im(V) || Knows(krkey(W,U),i)*+ -> Im(sign(V,W)
      )*.
195 115[0:SSi:113.2,113.0,102.0,2.0] E(U) ||  -> Stores(pair(ns,U
      ),c)*.
196 40[0:Inp] Im(U) Im(V) Im(W) ||  -> Im(triple(W,V,U))*.
197 41[0:Inp] E(U) E(V) Im(W) ||  -> M(sent(V,U,W))*.
198 111[0:Res:62.0,64.0] ||  -> Stores(pair(ns,s),c)*.
199 64[0:MRR:53.1,53.2,53.3,33.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> Stores(pair(ns,U),c).
200 109[0:Res:108.0,25.0] ||  -> Im(encr(triple(na,cpfc,ida),cerc
      ))*.
201 39[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)))* -> .
202 110[0:Res:108.0,26.0] ||  -> Im(idc)*.
203 108[0:Res:67.0,28.0] ||  -> Im(pair(encr(triple(na,cpfc,ida),
      cerc),idc))*.
204 67[0:MRR:58.0,58.1,58.2,58.3,58.4,11.0,32.0,12.0,6.0,13.0] ||
        -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
      )*.
205 36[0:Inp] E(U) Im(V) ||  -> Knows(krkey(V,U),i)*.
206 37[0:Inp] E(U) Im(V) ||  -> Knows(kukey(V,U),i)*.
207 38[0:Inp] E(U) Im(V) ||  -> Knows(nonce(V,U),i)*.
208 35[0:Inp] Im(U) Im(V) ||  -> Im(pair(V,U))*.
209 71[0:MRR:70.1,70.2,5.0,4.0] Im(sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)) ||  -> .
210 32[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
211 33[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
212 34[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),i)*.
213 105[0:Res:20.0,29.0] ||  -> Im(icodc)*.
214 29[0:Inp] || Im(triple(U,V,W))* -> Im(U).
```

```
104[0:Res:20.0,30.0] ||  -> Im(icpfc)*.
30[0:Inp] || Im(triple(U,V,W))* -> Im(V).
103[0:Res:20.0,31.0] ||  -> Im(itemplatec)*.
31[0:Inp] || Im(triple(U,V,W))* -> Im(W).
102[0:Res:62.0,28.0] ||  -> Im(encr(ns,cerc))*.
28[0:Inp] || M(sent(U,V,W))* -> Im(W).
25[0:Inp] || Im(pair(U,V))* -> Im(U).
26[0:Inp] || Im(pair(U,V))* -> Im(V).
27[0:Inp] || Im(sign(U,V))* -> Im(U).
62[0:MRR:47.0,47.1,18.0,19.0] ||  -> M(sent(s,c,encr(ns,cerc)
    ))*.
24[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
23[0:Inp] ||  -> Knows(krkey(kra,a),i)*.
66[0:MRR:57.0,57.1,57.2,57.3,57.4,11.0,32.0,12.0,6.0,13.0] ||
     -> Stores(pair(na,cpfc),a)*.
61[0:MRR:45.0,45.1,18.0,19.0] ||  -> Stores(pair(ns,cerc),s)
    *.
6[0:Inp] ||  -> Knows(cpfc,a)*.
7[0:Inp] ||  -> Knows(codc,c)*.
8[0:Inp] ||  -> Knows(cpfc,c)*.
9[0:Inp] ||  -> Knows(templatec,c)*.
1[0:Inp] ||  -> E(a)*.
2[0:Inp] ||  -> E(c)*.
3[0:Inp] ||  -> E(s)*.
4[0:Inp] ||  -> E(d)*.
5[0:Inp] ||  -> E(i)*.
-------------------------SPASS-STOP
    ------------------------------
```

A.4.6 Fato 6

$$\Gamma_{estendido} \cup \{\ \text{Fórmula 51}\ \} \nvdash M(sent(i, d, sign(sign(triple(icodc, icpfc, itemplatec), krc), kra)))$$

```
1
2 -------------------------SPASS-START
    ------------------------------
3 Input Problem:
4 1[0:Inp] ||  -> E(a)*.
5 2[0:Inp] ||  -> E(c)*.
```

```
 6 3[0:Inp] ||  -> E(s)*.
 7 4[0:Inp] ||  -> E(d)*.
 8 5[0:Inp] ||  -> E(i)*.
 9 6[0:Inp] ||  -> Knows(cpfc,a)*.
10 7[0:Inp] ||  -> Knows(codc,c)*.
11 8[0:Inp] ||  -> Knows(cpfc,c)*.
12 9[0:Inp] ||  -> Knows(templatec,c)*.
13 10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
14 11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
15 12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
16 13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
17 14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
18 15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
19 16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
20 17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
21 18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
22 19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
23 20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
24 21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
25 22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
26 23[0:Inp] ||  -> Knows(krkey(krc,c),i)*.
27 24[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
28 25[0:Inp] || Im(pair(U,V))* -> Im(U).
29 26[0:Inp] || Im(pair(U,V))* -> Im(V).
30 27[0:Inp] || Im(sign(U,V))* -> Im(U).
31 28[0:Inp] || M(sent(U,V,W))* -> Im(W).
32 29[0:Inp] || Im(triple(U,V,W))* -> Im(U).
33 30[0:Inp] || Im(triple(U,V,W))* -> Im(V).
34 31[0:Inp] || Im(triple(U,V,W))* -> Im(W).
35 32[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
36 33[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
37 34[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),i)*.
38 35[0:Inp] || Im(U) Im(V) -> Im(pair(U,V))*.
39 36[0:Inp] || Im(U) E(V) -> Knows(krkey(U,V),i)*.
40 37[0:Inp] || Im(U) E(V) -> Knows(kukey(U,V),i)*.
41 38[0:Inp] || Im(U) E(V) -> Knows(nonce(U,V),i)*.
42 39[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
       itemplatec),krc),kra)))* -> .
43 40[0:Inp] || Im(U) Im(V) Im(W) -> Im(triple(U,V,W))*.
44 41[0:Inp] || Im(U) E(V) E(W) -> M(sent(V,W,U))*.
45 42[0:Inp] || Im(U) Knows(kukey(V,W),i)* E(W) -> Im(encr(U,V))
       *.
46 43[0:Inp] || Im(U) Knows(krkey(V,W),i)* E(W) -> Im(sign(U,V))
       *.
47 44[0:Inp] || Im(encr(U,V))* Knows(nonce(V,W),i)* E(W) -> Im(U
       ).
48 45[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) ->
       Stores(pair(ns,cerc),s)*.
49 46[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> Stores(cpfc,d).
50 47[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) -> M
       (sent(s,c,encr(ns,cerc)))*.
```

```
48[0:Inp] || Knows(kp(krkey(U,V),kukey(W,V)),i)* Im(encr(X,W)
    )* E(V) -> Im(X).
49[0:Inp] || M(sent(U,s,encr(triple(cpfc,ids,templatec),ns)))
    * Stores(pair(ns,cerc),s) -> M(sent(s,d,cpfc)).
50[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
    idc))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
51[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
    codc,cpfc,templatec),krc),kra),d).
52[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
    triple(codc,cpfc,templatec),krc),kra))).
53[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
    ),kukey(cerc,c)),c)* Knows(templatec,c) Knows(cpfc,c) ->
    Stores(pair(ns,U),c).
54[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
    cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
    pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
55[0:Inp] || M(sent(U,d,cpfc))* Stores(sign(sign(triple(codc,
    cpfc,templatec),krc),kra),d) Stores(cpfc,d) -> M(sent(d,s
    ,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
56[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
    ),kukey(cerc,c)),c) Knows(templatec,c) Knows(cpfc,c) -> M
    (sent(c,s,encr(triple(cpfc,ids,templatec),ns)))*.
57[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
    kukey(cera,a)),a)* Knows(kukey(cerc,c),a) Knows(cpfc,a)
    Knows(nonce(na,a),a) -> Stores(pair(na,cpfc),a).
58[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
    kukey(cera,a)),a) Knows(kukey(cerc,c),a) Knows(cpfc,a)
    Knows(nonce(na,a),a) -> M(sent(a,s,pair(encr(triple(na,
    cpfc,ida),cerc),idc)))*.
59[0:Inp] || M(sent(U,c,encr(triple(na,cpfc,ida),cerc)))*
    Knows(triple(codc,cpfc,templatec),c) Knows(kukey(cera,a),
    c) Knows(kp(krkey(krc,c),kukey(cerc,c)),c) -> M(sent(c,s,
    pair(encr(pair(na,sign(triple(codc,cpfc,templatec),krc)),
    cera),ida)))*.
60[0:Inp] || M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,
    templatec),krc)),cera)))* Stores(pair(na,cpfc),a) Knows(
    kukey(cerc,c),a) Knows(kp(krkey(kra,a),kukey(cera,a)),a)
    Knows(cpfc,a) Knows(nonce(na,a),a) -> M(sent(a,s,pair(
    sign(sign(triple(codc,cpfc,templatec),krc),kra),idd)))*.
 This is a first-order Horn problem without equality.
 This is a problem that contains sort information.
 The conjecture is ground.
 The following monadic predicates have finite extensions: E.
 Axiom clauses: 59 Conjecture clauses: 1
 Inferences: IEmS=1 ISoR=1 IORe=1
 Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RSST=1
     RSSi=1 RFSub=1 RBSub=1 RCon=1
 Extras    : Input Saturation, Dynamic Selection, No
     Splitting, Full Reduction,  Ratio: 5, FuncWeight: 1,
     VarWeight: 1
```

```
 Precedence: pair > krkey > kukey > kp > nonce > encr > sign
      > triple > sent > Im > Stores > Knows > M > E > a > c >
      s > d > i > kra > cera > krc > cerc > na > ns > codc >
      cpfc > templatec > icodc > icpfc > itemplatec > ida >
      idc > idd > ids
 Ordering  : KBO
Processed Problem:

Worked Off Clauses:

Usable Clauses:
5[0:Inp] ||  -> E(i)*.
4[0:Inp] ||  -> E(d)*.
3[0:Inp] ||  -> E(s)*.
2[0:Inp] ||  -> E(c)*.
1[0:Inp] ||  -> E(a)*.
9[0:Inp] ||  -> Knows(templatec,c)*.
8[0:Inp] ||  -> Knows(cpfc,c)*.
7[0:Inp] ||  -> Knows(codc,c)*.
6[0:Inp] ||  -> Knows(cpfc,a)*.
61[0:MRR:45.0,45.1,18.0,19.0] ||  -> Stores(pair(ns,cerc),s)
    *.
66[0:MRR:57.0,57.1,57.2,57.3,57.4,11.0,32.0,12.0,6.0,13.0] ||
     -> Stores(pair(na,cpfc),a)*.
23[0:Inp] ||  -> Knows(krkey(krc,c),i)*.
22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
24[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
62[0:MRR:47.0,47.1,18.0,19.0] ||  -> M(sent(s,c,encr(ns,cerc)
    ))*.
27[0:Inp] || Im(sign(U,V))* -> Im(U).
26[0:Inp] || Im(pair(U,V))* -> Im(V).
25[0:Inp] || Im(pair(U,V))* -> Im(U).
28[0:Inp] || M(sent(U,V,W))* -> Im(W).
31[0:Inp] || Im(triple(U,V,W))* -> Im(W).
30[0:Inp] || Im(triple(U,V,W))* -> Im(V).
29[0:Inp] || Im(triple(U,V,W))* -> Im(U).
34[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),i)*.
33[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
32[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
35[0:Inp] Im(U) Im(V) ||  -> Im(pair(V,U))*.
```

```
117 71[0:MRR:70.1,70.2,5.0,4.0] Im(sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)) ||  -> .
118 38[0:Inp] E(U) Im(V) ||  -> Knows(nonce(V,U),i)*.
119 37[0:Inp] E(U) Im(V) ||  -> Knows(kukey(V,U),i)*.
120 36[0:Inp] E(U) Im(V) ||  -> Knows(krkey(V,U),i)*.
121 67[0:MRR:58.0,58.1,58.2,58.3,58.4,11.0,32.0,12.0,6.0,13.0] ||
       -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
      )*.
122 39[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)))* -> .
123 64[0:MRR:53.1,53.2,53.3,33.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> Stores(pair(ns,U),c).
124 41[0:Inp] E(U) E(V) Im(W) ||  -> M(sent(V,U,W))*.
125 40[0:Inp] Im(U) Im(V) Im(W) ||  -> Im(triple(W,V,U))*.
126 63[0:MRR:49.1,61.0] || M(sent(U,s,encr(triple(cpfc,ids,
      templatec),ns)))* -> M(sent(s,d,cpfc)).
127 43[0:Inp] E(U) Im(V) || Knows(krkey(W,U),i)* -> Im(sign(V,W))
      *.
128 42[0:Inp] E(U) Im(V) || Knows(kukey(W,U),i)* -> Im(encr(V,W))
      *.
129 44[0:Inp] E(U) || Im(encr(V,W))* Knows(nonce(W,U),i)* -> Im(V
      ).
130 46[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(cpfc,d).
131 65[0:MRR:56.1,56.2,56.3,33.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> M(sent(c,s,encr(triple(cpfc,ids,templatec),ns
      )))*.
132 48[0:Inp] E(U) || Im(encr(V,W))* Knows(kp(krkey(X,U),kukey(W,
      U)),i)* -> Im(V).
133 50[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
      idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
134 51[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
      codc,cpfc,templatec),krc),kra),d).
135 52[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
      templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
      triple(codc,cpfc,templatec),krc),kra))).
136 68[0:MRR:59.1,59.2,59.3,24.0,16.0,33.0] || M(sent(U,c,encr(
      triple(na,cpfc,ida),cerc)))* -> M(sent(c,s,pair(encr(pair
      (na,sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
137 69[0:MRR:60.1,60.2,60.3,60.4,60.5,66.0,12.0,32.0,6.0,13.0] ||
       M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,templatec)
      ,krc)),cera)))* -> M(sent(a,s,pair(sign(sign(triple(codc,
      cpfc,templatec),krc),kra),idd)))*.
138 54[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
      cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
      pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
139 55[0:Inp] || M(sent(U,d,cpfc))* Stores(cpfc,d) Stores(sign(
      sign(triple(codc,cpfc,templatec),krc),kra),d) -> M(sent(d
      ,s,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
140 SPASS V 3.5
141 SPASS beiseite: Completion found.
142 Problem: C:/Users/Eduardo/AppData/Local/Tempqt_temp.b02596
```

```
143 SPASS derived 112 clauses, backtracked 0 clauses, performed 0
       splits and kept 125 clauses.
144 SPASS allocated 30209 KBytes.
145 SPASS spent 0:00:00.22 on the problem.
146     0:00:00.00 for the input.
147     0:00:00.02 for the FLOTTER CNF translation.
148     0:00:00.00 for inferences.
149     0:00:00.00 for the backtracking.
150     0:00:00.00 for the reduction.
151
152
153  The saturated set of worked-off clauses is :
154 234[0:Res:207.0,54.0] ||  -> M(sent(s,a,encr(pair(na,sign(
       triple(codc,cpfc,templatec),krc)),cera)))*.
155 226[0:Res:168.0,223.0] ||  -> M(sent(d,s,sign(sign(triple(
       codc,cpfc,templatec),krc),kra)))*.
156 218[0:Res:216.0,52.0] ||  -> M(sent(s,d,sign(sign(triple(codc
       ,cpfc,templatec),krc),kra)))*.
157 231[0:Res:230.0,29.0] ||  -> Im(codc)*.
158 230[0:Res:229.0,27.0] ||  -> Im(triple(codc,cpfc,templatec))
       *.
159 229[0:Res:224.0,27.0] ||  -> Im(sign(triple(codc,cpfc,
       templatec),krc))*.
160 224[0:Res:217.0,25.0] ||  -> Im(sign(sign(triple(codc,cpfc,
       templatec),krc),kra))*.
161 225[0:Res:217.0,26.0] ||  -> Im(idd)*.
162 217[0:Res:216.0,28.0] ||  -> Im(pair(sign(sign(triple(codc,
       cpfc,templatec),krc),kra),idd))*.
163 219[0:Res:216.0,51.0] ||  -> Stores(sign(sign(triple(codc,
       cpfc,templatec),krc),kra),d)*.
164 220[0:Res:216.0,46.0] ||  -> Stores(cpfc,d)*.
165 216[0:SSi:215.1,215.0,211.0,1.0] ||  -> M(sent(a,s,pair(sign(
       sign(triple(codc,cpfc,templatec),krc),kra),idd)))*.
166 213[0:Res:211.0,154.1] Im(cera) ||  -> Im(pair(na,sign(triple
       (codc,cpfc,templatec),krc)))*.
167 211[0:Res:210.0,25.0] ||  -> Im(encr(pair(na,sign(triple(codc
       ,cpfc,templatec),krc)),cera))*.
168 210[0:Res:207.0,28.0] ||  -> Im(pair(encr(pair(na,sign(triple
       (codc,cpfc,templatec),krc)),cera),ida))*.
169 207[0:Res:181.0,68.0] ||  -> M(sent(c,s,pair(encr(pair(na,
       sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
170 197[0:SSi:196.1,20.0] E(U) || Knows(krkey(kra,U),i)* -> .
171 181[0:Res:67.0,50.0] ||  -> M(sent(s,c,encr(triple(na,cpfc,
       ida),cerc)))*.
172 186[0:Res:177.0,31.0] ||  -> Im(templatec)*.
173 185[0:Res:177.0,30.0] ||  -> Im(ids)*.
174 177[0:MRR:170.0,173.0] ||  -> Im(triple(cpfc,ids,templatec))
       *.
175 180[0:Res:174.0,31.0] ||  -> Im(ida)*.
176 178[0:Res:174.0,29.0] ||  -> Im(na)*.
177 174[0:Res:109.0,172.0] ||  -> Im(triple(na,cpfc,ida))*.
178 173[0:Res:102.0,172.0] ||  -> Im(ns)*.
179 172[0:SSi:171.0,2.0] || Im(encr(U,cerc))* -> Im(U).
```

```
180 48[0:Inp] E(U) || Im(encr(V,W))* Knows(kp(krkey(X,U),kukey(W,
      U)),i)*+ -> Im(V).
181 167[0:Res:164.0,28.0] ||  -> Im(encr(triple(cpfc,ids,
      templatec),ns))*.
182 169[0:Res:168.0,28.0] ||  -> Im(cpfc)*.
183 168[0:Res:164.0,63.0] ||  -> M(sent(s,d,cpfc))*.
184 164[0:Res:62.0,65.0] ||  -> M(sent(c,s,encr(triple(cpfc,ids,
      templatec),ns)))*.
185 154[0:EmS:141.1,1.0] Im(U) || Im(encr(V,U))* -> Im(V).
186 145[0:EmS:133.1,1.0] Im(U) Im(V) ||  -> Im(encr(V,U))*.
187 144[0:SSi:142.0,125.0,20.1] Im(kra) ||  -> .
188 135[0:EmS:126.1,1.0] Im(U) Im(V) ||  -> Im(sign(V,U))*.
189 44[0:Inp] E(U) || Im(encr(V,W))* Knows(nonce(W,U),i)*+ -> Im(
      V).
190 132[0:SSi:129.0,2.0] Im(U) ||  -> Im(encr(U,cerc))*.
191 131[0:SSi:128.0,1.0] Im(U) ||  -> Im(encr(U,cera))*.
192 125[0:SSi:123.0,2.0] Im(U) ||  -> Im(sign(U,krc))*.
193 42[0:Inp] E(U) Im(V) || Knows(kukey(W,U),i)*+ -> Im(encr(V,W)
      )*.
194 43[0:Inp] E(U) Im(V) || Knows(krkey(W,U),i)*+ -> Im(sign(V,W)
      )*.
195 115[0:SSi:113.2,113.0,102.0,2.0] E(U) ||  -> Stores(pair(ns,U
      ),c)*.
196 40[0:Inp] Im(U) Im(V) Im(W) ||  -> Im(triple(W,V,U))*.
197 41[0:Inp] E(U) E(V) Im(W) ||  -> M(sent(V,U,W))*.
198 111[0:Res:62.0,64.0] ||  -> Stores(pair(ns,s),c)*.
199 64[0:MRR:53.1,53.2,53.3,33.0,9.0,8.0] || M(sent(U,c,encr(ns,
      cerc)))* -> Stores(pair(ns,U),c).
200 109[0:Res:108.0,25.0] ||  -> Im(encr(triple(na,cpfc,ida),cerc
      ))*.
201 39[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)))* -> .
202 110[0:Res:108.0,26.0] ||  -> Im(idc)*.
203 108[0:Res:67.0,28.0] ||  -> Im(pair(encr(triple(na,cpfc,ida),
      cerc),idc))*.
204 67[0:MRR:58.0,58.1,58.2,58.3,58.4,11.0,32.0,12.0,6.0,13.0] ||
      -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
      )*.
205 36[0:Inp] E(U) Im(V) ||  -> Knows(krkey(V,U),i)*.
206 37[0:Inp] E(U) Im(V) ||  -> Knows(kukey(V,U),i)*.
207 38[0:Inp] E(U) Im(V) ||  -> Knows(nonce(V,U),i)*.
208 35[0:Inp] Im(U) Im(V) ||  -> Im(pair(V,U))*.
209 71[0:MRR:70.1,70.2,5.0,4.0] Im(sign(sign(triple(icodc,icpfc,
      itemplatec),krc),kra)) ||  -> .
210 32[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
211 33[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
212 34[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),i)*.
213 105[0:Res:20.0,29.0] ||  -> Im(icodc)*.
214 29[0:Inp] || Im(triple(U,V,W))* -> Im(U).
215 104[0:Res:20.0,30.0] ||  -> Im(icpfc)*.
216 30[0:Inp] || Im(triple(U,V,W))* -> Im(V).
217 103[0:Res:20.0,31.0] ||  -> Im(itemplatec)*.
218 31[0:Inp] || Im(triple(U,V,W))* -> Im(W).
```

```
102[0:Res:62.0,28.0] ||  -> Im(encr(ns,cerc))*.
28[0:Inp] || M(sent(U,V,W))* -> Im(W).
25[0:Inp] || Im(pair(U,V))* -> Im(U).
26[0:Inp] || Im(pair(U,V))* -> Im(V).
27[0:Inp] || Im(sign(U,V))* -> Im(U).
62[0:MRR:47.0,47.1,18.0,19.0] ||  -> M(sent(s,c,encr(ns,cerc)
    ))*.
24[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
23[0:Inp] ||  -> Knows(krkey(krc,c),i)*.
66[0:MRR:57.0,57.1,57.2,57.3,57.4,11.0,32.0,12.0,6.0,13.0] ||
     -> Stores(pair(na,cpfc),a)*.
61[0:MRR:45.0,45.1,18.0,19.0] ||  -> Stores(pair(ns,cerc),s)
    *.
6[0:Inp] ||  -> Knows(cpfc,a)*.
7[0:Inp] ||  -> Knows(codc,c)*.
8[0:Inp] ||  -> Knows(cpfc,c)*.
9[0:Inp] ||  -> Knows(templatec,c)*.
1[0:Inp] ||  -> E(a)*.
2[0:Inp] ||  -> E(c)*.
3[0:Inp] ||  -> E(s)*.
4[0:Inp] ||  -> E(d)*.
5[0:Inp] ||  -> E(i)*.
-------------------------SPASS-STOP
    ------------------------------
```

A.4.7 Fato 7

$\Gamma_{estendido} \cup \{$ Fórmula 50, Fórmula 51 $\} \vdash M(sent(i, d, sign(sign(triple(icodc, icpfc, itemplatec), krc), kra)))$

```
1
2 -------------------------SPASS-START
    ------------------------------
3 Input Problem:
4 1[0:Inp] ||  -> E(a)*.
5 2[0:Inp] ||  -> E(c)*.
6 3[0:Inp] ||  -> E(s)*.
7 4[0:Inp] ||  -> E(d)*.
8 5[0:Inp] ||  -> E(i)*.
9 6[0:Inp] ||  -> Knows(cpfc,a)*.
```

```
10 7[0:Inp] ||  -> Knows(codc,c)*.
11 8[0:Inp] ||  -> Knows(cpfc,c)*.
12 9[0:Inp] ||  -> Knows(templatec,c)*.
13 10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
14 11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
15 12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
16 13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
17 14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
18 15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
19 16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
20 17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
21 18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
22 19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
23 20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
24 21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
25 22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
26 23[0:Inp] ||  -> Knows(krkey(kra,a),i)*.
27 24[0:Inp] ||  -> Knows(krkey(krc,c),i)*.
28 25[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
29 26[0:Inp] || Im(pair(U,V))* -> Im(U).
30 27[0:Inp] || Im(pair(U,V))* -> Im(V).
31 28[0:Inp] || Im(sign(U,V))* -> Im(U).
32 29[0:Inp] || M(sent(U,V,W))* -> Im(W).
33 30[0:Inp] || Im(triple(U,V,W))* -> Im(U).
34 31[0:Inp] || Im(triple(U,V,W))* -> Im(V).
35 32[0:Inp] || Im(triple(U,V,W))* -> Im(W).
36 33[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
37 34[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
38 35[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),i)*.
39 36[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),i)*.
40 37[0:Inp] || Im(U) Im(V) -> Im(pair(U,V))*.
41 38[0:Inp] || Im(U) E(V) -> Knows(krkey(U,V),i)*.
42 39[0:Inp] || Im(U) E(V) -> Knows(kukey(U,V),i)*.
43 40[0:Inp] || Im(U) E(V) -> Knows(nonce(U,V),i)*.
44 41[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
       itemplatec),krc),kra)))* -> .
45 42[0:Inp] || Im(U) Im(V) Im(W) -> Im(triple(U,V,W))*.
46 43[0:Inp] || Im(U) E(V) E(W) -> M(sent(V,W,U))*.
47 44[0:Inp] || Im(U) Knows(kukey(V,W),i)* E(W) -> Im(encr(U,V))
       *.
48 45[0:Inp] || Im(U) Knows(krkey(V,W),i)* E(W) -> Im(sign(U,V))
       *.
49 46[0:Inp] || Im(encr(U,V))* Knows(nonce(V,W),i)* E(W) -> Im(U
       ).
50 47[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) ->
       Stores(pair(ns,cerc),s)*.
51 48[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> Stores(cpfc,d).
52 49[0:Inp] || Knows(kukey(cerc,c),s) Knows(nonce(ns,s),s) -> M
       (sent(s,c,encr(ns,cerc)))*.
53 50[0:Inp] || Knows(kp(krkey(U,V),kukey(W,V)),i)* Im(encr(X,W)
       )* E(V) -> Im(X).
```

```
54 51[0:Inp] || M(sent(U,s,encr(triple(cpfc,ids,templatec),ns)))
       * Stores(pair(ns,cerc),s) -> M(sent(s,d,cpfc)).
55 52[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
       idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
56 53[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
       codc,cpfc,templatec),krc),kra),d).
57 54[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
       templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
       triple(codc,cpfc,templatec),krc),kra))).
58 55[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
       ),kukey(cerc,c)),c)* Knows(templatec,c) Knows(cpfc,c) ->
       Stores(pair(ns,U),c).
59 56[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
       cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
       pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
60 57[0:Inp] || M(sent(U,d,cpfc))* Stores(sign(sign(triple(codc,
       cpfc,templatec),krc),kra),d) Stores(cpfc,d) -> M(sent(d,s
       ,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
61 58[0:Inp] || M(sent(U,c,encr(ns,cerc)))* Knows(kp(krkey(krc,c
       ),kukey(cerc,c)),c) Knows(templatec,c) Knows(cpfc,c) -> M
       (sent(c,s,encr(triple(cpfc,ids,templatec),ns)))*.
62 59[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
       kukey(cera,a)),a)* Knows(kukey(cerc,c),a) Knows(cpfc,a)
       Knows(nonce(na,a),a) -> Stores(pair(na,cpfc),a).
63 60[0:Inp] || Knows(kukey(cera,a),a) Knows(kp(krkey(kra,a),
       kukey(cera,a)),a) Knows(kukey(cerc,c),a) Knows(cpfc,a)
       Knows(nonce(na,a),a) -> M(sent(a,s,pair(encr(triple(na,
       cpfc,ida),cerc),idc)))*.
64 61[0:Inp] || M(sent(U,c,encr(triple(na,cpfc,ida),cerc)))*
       Knows(triple(codc,cpfc,templatec),c) Knows(kukey(cera,a),
       c) Knows(kp(krkey(krc,c),kukey(cerc,c)),c) -> M(sent(c,s,
       pair(encr(pair(na,sign(triple(codc,cpfc,templatec),krc)),
       cera),ida)))*.
65 62[0:Inp] || M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,
       templatec),krc)),cera)))* Stores(pair(na,cpfc),a) Knows(
       kukey(cerc,c),a) Knows(kp(krkey(kra,a),kukey(cera,a)),a)
       Knows(cpfc,a) Knows(nonce(na,a),a) -> M(sent(a,s,pair(
       sign(sign(triple(codc,cpfc,templatec),krc),kra),idd)))*.
66  This is a first-order Horn problem without equality.
67  This is a problem that contains sort information.
68  The conjecture is ground.
69  The following monadic predicates have finite extensions: E.
70  Axiom clauses: 61 Conjecture clauses: 1
71  Inferences: IEmS=1 ISoR=1 IORe=1
72  Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RSST=1
        RSSi=1 RFSub=1 RBSub=1 RCon=1
73  Extras    : Input Saturation, Dynamic Selection, No
        Splitting, Full Reduction,  Ratio: 5, FuncWeight: 1,
        VarWeight: 1
74  Precedence: pair > krkey > kukey > kp > nonce > encr > sign
        > triple > sent > Im > Stores > Knows > M > E > a > c >
        s > d > i > kra > cera > krc > cerc > na > ns > codc >
```

```
        cpfc > templatec > icodc > icpfc > itemplatec > ida >
        idc > idd > ids
 Ordering  : KBO
Processed Problem:

Worked Off Clauses:

Usable Clauses:
5[0:Inp] ||  -> E(i)*.
4[0:Inp] ||  -> E(d)*.
3[0:Inp] ||  -> E(s)*.
2[0:Inp] ||  -> E(c)*.
1[0:Inp] ||  -> E(a)*.
9[0:Inp] ||  -> Knows(templatec,c)*.
8[0:Inp] ||  -> Knows(cpfc,c)*.
7[0:Inp] ||  -> Knows(codc,c)*.
6[0:Inp] ||  -> Knows(cpfc,a)*.
63[0:MRR:47.0,47.1,18.0,19.0] ||  -> Stores(pair(ns,cerc),s)
    *.
68[0:MRR:59.0,59.1,59.2,59.3,59.4,11.0,33.0,12.0,6.0,13.0] ||
     -> Stores(pair(na,cpfc),a)*.
24[0:Inp] ||  -> Knows(krkey(krc,c),i)*.
23[0:Inp] ||  -> Knows(krkey(kra,a),i)*.
22[0:Inp] ||  -> Knows(kukey(cera,a),i)*.
21[0:Inp] ||  -> Knows(kukey(cerc,c),i)*.
18[0:Inp] ||  -> Knows(kukey(cerc,c),s)*.
17[0:Inp] ||  -> Knows(kukey(cera,a),s)*.
16[0:Inp] ||  -> Knows(kukey(cera,a),c)*.
12[0:Inp] ||  -> Knows(kukey(cerc,c),a)*.
19[0:Inp] ||  -> Knows(nonce(ns,s),s)*.
15[0:Inp] ||  -> Knows(kukey(cerc,c),c)*.
14[0:Inp] ||  -> Knows(krkey(krc,c),c)*.
13[0:Inp] ||  -> Knows(nonce(na,a),a)*.
11[0:Inp] ||  -> Knows(kukey(cera,a),a)*.
10[0:Inp] ||  -> Knows(krkey(kra,a),a)*.
20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
25[0:Inp] ||  -> Knows(triple(codc,cpfc,templatec),c)*.
64[0:MRR:49.0,49.1,18.0,19.0] ||  -> M(sent(s,c,encr(ns,cerc)
    ))*.
28[0:Inp] || Im(sign(U,V))* -> Im(U).
27[0:Inp] || Im(pair(U,V))* -> Im(V).
26[0:Inp] || Im(pair(U,V))* -> Im(U).
29[0:Inp] || M(sent(U,V,W))* -> Im(W).
32[0:Inp] || Im(triple(U,V,W))* -> Im(W).
31[0:Inp] || Im(triple(U,V,W))* -> Im(V).
30[0:Inp] || Im(triple(U,V,W))* -> Im(U).
36[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),i)*.
35[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),i)*.
34[0:Inp] ||  -> Knows(kp(krkey(krc,c),kukey(cerc,c)),c)*.
33[0:Inp] ||  -> Knows(kp(krkey(kra,a),kukey(cera,a)),a)*.
37[0:Inp] Im(U) Im(V) ||  -> Im(pair(V,U))*.
73[0:MRR:72.1,72.2,5.0,4.0] Im(sign(sign(triple(icodc,icpfc,
    itemplatec),krc),kra)) ||  -> .
```

```
122 40[0:Inp] E(U) Im(V) ||  -> Knows(nonce(V,U),i)*.
123 39[0:Inp] E(U) Im(V) ||  -> Knows(kukey(V,U),i)*.
124 38[0:Inp] E(U) Im(V) ||  -> Knows(krkey(V,U),i)*.
125 69[0:MRR:60.0,60.1,60.2,60.3,60.4,11.0,33.0,12.0,6.0,13.0] ||
        -> M(sent(a,s,pair(encr(triple(na,cpfc,ida),cerc),idc))
    )*.
126 41[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
    itemplatec),krc),kra)))* -> .
127 66[0:MRR:55.1,55.2,55.3,34.0,9.0,8.0] || M(sent(U,c,encr(ns,
    cerc)))* -> Stores(pair(ns,U),c).
128 43[0:Inp] E(U) E(V) Im(W) ||  -> M(sent(V,U,W))*.
129 42[0:Inp] Im(U) Im(V) Im(W) ||  -> Im(triple(W,V,U))*.
130 65[0:MRR:51.1,63.0] || M(sent(U,s,encr(triple(cpfc,ids,
    templatec),ns)))* -> M(sent(s,d,cpfc)).
131 45[0:Inp] E(U) Im(V) || Knows(krkey(W,U),i)* -> Im(sign(V,W))
    *.
132 44[0:Inp] E(U) Im(V) || Knows(kukey(W,U),i)* -> Im(encr(V,W))
    *.
133 46[0:Inp] E(U) || Im(encr(V,W))* Knows(nonce(W,U),i)* -> Im(V
    ).
134 48[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(cpfc,d).
135 67[0:MRR:58.1,58.2,58.3,34.0,9.0,8.0] || M(sent(U,c,encr(ns,
    cerc)))* -> M(sent(c,s,encr(triple(cpfc,ids,templatec),ns
    )))*.
136 50[0:Inp] E(U) || Im(encr(V,W))* Knows(kp(krkey(X,U),kukey(W,
    U)),i)* -> Im(V).
137 52[0:Inp] || M(sent(U,s,pair(encr(triple(na,cpfc,ida),cerc),
    idc)))* -> M(sent(s,c,encr(triple(na,cpfc,ida),cerc))).
138 53[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> Stores(sign(sign(triple(
    codc,cpfc,templatec),krc),kra),d).
139 54[0:Inp] || M(sent(U,s,pair(sign(sign(triple(codc,cpfc,
    templatec),krc),kra),idd)))* -> M(sent(s,d,sign(sign(
    triple(codc,cpfc,templatec),krc),kra))).
140 70[0:MRR:61.1,61.2,61.3,25.0,16.0,34.0] || M(sent(U,c,encr(
    triple(na,cpfc,ida),cerc)))* -> M(sent(c,s,pair(encr(pair
    (na,sign(triple(codc,cpfc,templatec),krc)),cera),ida)))*.
141 71[0:MRR:62.1,62.2,62.3,62.4,62.5,68.0,12.0,33.0,6.0,13.0] ||
     M(sent(U,a,encr(pair(na,sign(triple(codc,cpfc,templatec)
    ,krc)),cera)))* -> M(sent(a,s,pair(sign(sign(triple(codc,
    cpfc,templatec),krc),kra),idd)))*.
142 56[0:Inp] || M(sent(U,s,pair(encr(pair(na,sign(triple(codc,
    cpfc,templatec),krc)),cera),ida)))* -> M(sent(s,a,encr(
    pair(na,sign(triple(codc,cpfc,templatec),krc)),cera))).
143 57[0:Inp] || M(sent(U,d,cpfc))* Stores(cpfc,d) Stores(sign(
    sign(triple(codc,cpfc,templatec),krc),kra),d) -> M(sent(d
    ,s,sign(sign(triple(codc,cpfc,templatec),krc),kra)))*.
144 SPASS V 3.5
145 SPASS beiseite: Proof found.
146 Problem: C:/Users/Eduardo/AppData/Local/Tempqt_temp.b02596
147 SPASS derived 25 clauses, backtracked 0 clauses, performed 0
    splits and kept 78 clauses.
```

```
SPASS allocated 30201 KBytes.
SPASS spent 0:00:00.20 on the problem.
    0:00:00.02 for the input.
    0:00:00.02 for the FLOTTER CNF translation.
    0:00:00.00 for inferences.
    0:00:00.00 for the backtracking.
    0:00:00.00 for the reduction.


Here is a proof with depth 2, length 18 :
1[0:Inp] ||  -> E(a)*.
2[0:Inp] ||  -> E(c)*.
4[0:Inp] ||  -> E(d)*.
5[0:Inp] ||  -> E(i)*.
20[0:Inp] ||  -> Im(triple(icodc,icpfc,itemplatec))*.
23[0:Inp] ||  -> Knows(krkey(kra,a),i)*.
24[0:Inp] ||  -> Knows(krkey(krc,c),i)*.
41[0:Inp] || M(sent(i,d,sign(sign(triple(icodc,icpfc,
    itemplatec),krc),kra)))* -> .
43[0:Inp] E(U) E(V) Im(W) ||  -> M(sent(V,U,W))*.
45[0:Inp] E(U) Im(V) || Knows(krkey(W,U),i)*+ -> Im(sign(V,W)
    )*.
72[0:Res:43.3,41.0] Im(sign(sign(triple(icodc,icpfc,
    itemplatec),krc),kra)) E(i) E(d) ||  -> .
73[0:MRR:72.1,72.2,5.0,4.0] Im(sign(sign(triple(icodc,icpfc,
    itemplatec),krc),kra)) ||  -> .
125[0:Res:24.0,45.2] E(c) Im(U) ||  -> Im(sign(U,krc))*.
126[0:Res:23.0,45.2] E(a) Im(U) ||  -> Im(sign(U,kra))*.
128[0:SSi:125.0,2.0] Im(U) ||  -> Im(sign(U,krc))*.
129[0:SSi:126.0,1.0] Im(U) ||  -> Im(sign(U,kra))*.
132[0:SoR:73.0,129.1] Im(sign(triple(icodc,icpfc,itemplatec),
    krc)) ||  -> .
134[0:SSi:132.0,128.0,20.1] ||  -> .
Formulae used in the proof : A_is_entity C_is_entity
    D_is_entity I_is_entity I_has_forged_client_data
    hypothesis_I_has_A_keypair hypothesis_I_has_C_keypair
    test_attacker_capacity I_sends_faked I_generates_signed

-------------------------SPASS-STOP
    ------------------------------
```