UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Utilização de Traços Simbólicos e SMT Solvers para a Verificação de Equivalência Observacional em Segurança e Privacidade de Protocolos

Abraão Aires Urquiza de Carvalho

João Pessoa - PB

Janeiro de 2020

Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Informática

Utilização de Traços Simbólicos e SMT Solvers para a Verificação de Equivalência Observacional em Segurança e Privacidade de Protocolos

Abraão Aires Urquiza de Carvalho

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: SMT solvers | Verificação automática | Segurança de protocolos

Vivek Nigam

João Pessoa, Paraíba, Brasil

Catalogação na publicação
Seção de Catalogação e Classificação

C331u Carvalho, Abraão Aires Urquiza de.
Utilização de Traços Simbólicos e SMT Solvers para a Verificação de Equivalência Observacional em Segurança e Privacidade de Protocolos / Abraão Aires Urquiza de Carvalho. - João Pessoa, 2020.
110 f.

Orientação: Vivek Nigam.
Dissertação (Mestrado) - UFPB/Informática.

1. Equivalência observacional. 2. Verificação Automática de Protocolos de Segurança. 3. Análise de Tráfego. 4. Ataque de Temporização. I. Nigam, Vivek. II. Título.

UFPB/BC

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Abraão Aires Urquiza de Carvalho, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 31 de janeiro de 2020.

Aos trinta e um dias do mês de janeiro, do ano de dois mil e vinte, às dez horas, no Centro de Informática da Universidade Federal da Paraíba, em Mangabeira, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do Sr. Abraão Aires Urquiza de Carvalho, vinculado a esta Universidade sob a matrícula nº 20181001070, candidato ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha de pesquisa "Computação Distribuída", do Programa de Pós-Graduação em Informática, da Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores: Vivek Nigam (PPGI-UFPB), Orientador e Presidente da Banca, Gustavo Henrique Matos Bezerra Motta (PPGI-UFPB), Examinador Interno, Iguatemi Eduardo da Fonseca (PPGI-UFPB), Examinador Interno, Andrei de Araujo Formiga (UFPB), Examinador Externo ao Programa, Carlos Olarte (UFRN), Externo à Instituição. Dando início aos trabalhos, o Presidente da Banca cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse a exposição oral do trabalho de dissertação intitulado: "Utilização de Traços Simbólicos e SMT Solvers para a Verificação de Equivalência Observacional em Segurança e Privacidade de Protocolos". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Ruy Alberto Pisani Altafim, Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai assinada por mim e pelos membros da banca examinadora. João Pessoa, 31 de janeiro de 2020.

Prof. Dr. Ruy Alberto Pisani Altafim

Prof. Dr. Vivek Nigam
Orientador (PPGI-UFPB)

Prof. Dr. Gustavo Henrique Matos Bezerra Motta
Examinador Interno (PPGI-UFPB)

Prof. Dr. Iguatemi Eduardo da Fonseca
Examinador Interno (PPGI-UFPB)

Prof. Dr. Andrei de Araujo Formiga
Examinador Externo ao Programa (UFPB)

Prof. Dr. Carlos Olarte
Examinador Externo à Instituição (UFRN)

Resumo

Atacantes e usuários maliciosos em redes de computadores podem usar uma série de técnicas para obter informação sensível. Dentre elas, a utilização do tempo e da assinatura do tráfego de pacotes como canal lateral abre várias brechas de segurança. A verificação automática desses ataques, no entanto, não consistem apenas nas técnicas convencionais de propriedades baseadas em alcançabilidade. Isso se deve porque o ataque não possui uma configuração intrinsecamente ruim, como um estado em que um intruso possua um segredo. O ataque existe quando uma determinada configuração apresente um comportamento que outra configuração distinta não possua, possibilitando distinguir uma da outra. Esse trabalho oferece uma definição formal de equivalência observacional no âmbito do tempo e da análise de tráfego, baseada em equivalência de traços simbólicos. Essa definição pode ser usada para elaboração de uma ferramenta capaz de identificar falhas de segurança e privacidade em protocolos de segurança de forma automatizada. Elaboramos um protótipo de tal ferramenta, com a implementação de alguns ataques presentes na literatura como prova de conceito.

Palavras-chave: Equivalência Observacional, Verificação Automática de Protocolos de Segurança, Análise de Tráfego, Ataque de Temporização

Abstract

Attackers and malicious users have at their disposal several techniques to acquire sensitive information. Among them, timing attacks and traffic analysis generate a number of security breaches. The automated verification of this attacks is not as straightforward as usual reachability properties, due to the nature of the problem. Instead of having an inherently bad state, *i.e.*, a state where an intruder possess secret intel, the attacker compares two distincts configurations. The attack happens when one of the configurations presents a behavior that distinguish it from the other configuration. This work proposes a formal definition based on trace equivalence for automated verification of observacional equivalence of time and traffic. This definition could be used in the development of a system capable of automated verification of security and privacy breaches in security protocols. Furthermore, we present a prototype of such system and presents the implementation of some examples as proof-of-concept.

Keywords: Observacional Equivalence, Automated Verification of Security Protocols, Traffic Analysis, Timing Attack

Agradecimentos

Eu agradeço primariamente a meu pai e minha mãe. A ele, por ter feito tudo em seu poder pra que seus filhos tivessem uma vida mais fácil do que a dele. A ela, por ainda fazer.

Agradeço aos meus irmãos, Gabriel e Emmanuel, por terem tido uma influência muito positiva no meu amadurecimento acadêmico e intelectual.

Agradeço aos meus tios Luiz, Roberto, Liliana, Fátima, Walmir, Giovanna, Ardigleusa e Maria por sempre estarem presentes.

Aos meus primos, vocês sabem quem são, por todas as risadas.

Ao meu orientador, Vivek Nigam, pela paciência de explicar e pela confiança em mim depositada.

A Carolyn Talcott, por sua influência luminar na minha formação acadêmica.

E a todos os outros, que direta ou indiretamente, tiveram uma participação na minha formação.

"Why do you go away?

So that you can come back.

So that you can see the place you came from with new eyes and extra colors.

And the people there see you differently, too.

Coming back to where you started is not the same as never leaving."

TERRY PRACHETT, A HAT FULL OF SKY

Conteúdo

Lista de Siglas

AES : *Advanced Encryption Standard*

ARP : *Automatic Rover Protection*

BAC : *Basic Access Control*

BMC : *Bounded Model Checking*

DAG : *Directed Acyclic Graph*

DNS : *Domain Name System*

HI : *Hipótese Indutiva*

ILP : *Independent Link Padding*

IoT : *Internet of Things*

MAC : *Message Authentication Code*

MAC : *Media Access Control*

MANET : *Mobile ad hoc Networks*

m.g.u : *Most General Unifier*

PMC : *Probabilistic Model Checking*

RFID : *Radio Frequency Identification*

RPS : *Reactor Protection System*

RSA : *Rivest-Shamir-Adleman*

SAT : *Boolean Satisfiability Problem*

SMT : *Satisfiability Modulo Theories*

Lista de Símbolos

$\mathsf{b}_1, \mathsf{b}_2, \ldots$: Variáveis de tráfego.

$\mathcal{BB}(\mathcal{O})$: Conjunto de termos caixa-preta de $\mathcal{O}$.

$\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2, \ldots$: Configurações.

$\mathcal{DC}$: Conjunto de restrições de derivabilidade.

$\mathcal{DC}(\mathsf{sym})$: Restrição de derivabilidade para sym contida em $\mathcal{DC}$.

$\mathsf{dc}(\mathsf{sym}, \mathcal{S})$: Restrição de derivabilidade do símbolo sym.

$\mathcal{EQ}$: Conjunto de restrições de comparação.

$\mathsf{e}(\mathsf{m}, \mathsf{k})$: Termo m encriptado com chave k.

$\mathcal{F}, \mathcal{F}_1, \mathcal{F}_2, \ldots$: Fluxos.

$\mathcal{FS}$: Conjunto de fluxos.

$\mathcal{IK}, \mathcal{IK}_1, \mathcal{IK}_2, \ldots$: Conhecimento do intruso ; conjunto mínimo de termos simbólicos.

$\mathcal{K}$: Conjunto de de chaves simétricas.

k : Chaves criptográficas.

$\mathcal{L}, \mathcal{L}_1, \mathcal{L}_2, \ldots$: Conjunto de rótulos não-vazios.

$\mathsf{m}, \mathsf{m}_1, \mathsf{m}_2, \ldots$: Termos.

$\mathsf{ms}, \mathsf{ms}_1, \mathsf{ms}_2, \ldots$: Termos simbólicos.

$\mathcal{N}$: Conjunto de *nonces*.

$\mathcal{O}, \mathcal{O}_1, \mathcal{O}_2, \ldots$: Observáveis.

$\mathcal{P}$: Conjunto de nomes de participantes.

$\mathsf{pl}, \mathsf{pl}_1, \mathsf{pl}_2, \ldots$: Função protocolar.

$\mathcal{R}$: Conjunto de regras de reescritura.

$\mathcal{RT}$: Conjunto de restrições de tráfego.

$\mathsf{tt}, \mathsf{tt}_1, \mathsf{tt}_2, \ldots$: Variáveis temporais.

$\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2, ldots$: Conjunto de termos simbólicos.

Syms : Conjunto de símbolos.

$\mathsf{sb}, \mathsf{sb}_1, \mathsf{sb}_2, \ldots$: Substituição de variáveis.

$\mathsf{ssb}, \mathsf{ssb}_1, \mathsf{ssb}_2, \ldots$: Substituição de símbolos.

$\mathsf{sym}, \mathsf{sym}_1, \mathsf{sym}_2, \ldots$: Símbolos.

$\mathcal{TC}, \mathcal{TC}_1, \mathcal{TC}_2, \ldots$: Conjunto de restrições de tempo

$\mathcal{TL}$: Conjunto de funções protocolares.

$\mathcal{T}(\mathcal{C})$: Conjunto de todos os traços que possuem $\mathcal{C}$ como configuração incial.

tc : Restrição de tempo.

tG : Símbolo temporal que representa o tempo global de uma configuração.

$\mathcal{V}$: Conjunto de variáveis.

$\mathsf{v}, \mathsf{v}_1, \mathsf{v}_2, \ldots$: Variáveis.

v_p : Variável do tipo participante.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A tendência conhecida como Internet of Things, ou IoT, tirou a exclusividade do acesso a internet dos PC e notebooks, e expandiu para celulares, impressoras, geladeiras, alarmes, etc. Essa prática expande em muito a efetividade dos eletrônicos, mas abre precedentes perigosos no que diz respeito a segurança e privacidade dos usuários.

O relatório Internet of Evil Things 2017 [Internet of Evil Things 2017], feito com mais de 800 profissionais de TI, afirma que 66% dos entrevistados não sabiam quantos dispositivos com acesso a internet estavam conectados em determinada rede. Gerenciar os pontos de acesso e as vulnerabilidades de uma rede quando existem tantos dispositivos que podem se conectar com a internet não é tarefa fácil, e o fato de muitos desses dispositivos não serem conhecidos pelo indivíduo ou corporação, como os telefones celulares de terceiros, só agrava esse problema.

Manter o controle da rede não é o único problema de ambientes fortemente conectados. O ataque realizado sobre os *e-passports* [Chothia e Smirnov 2010], em que um atacante descobriria se determinado passaporte se encontra nos seus arredores, ou as análises de tráfego realizadas em [Apthorpe Noah Reisman 2017], em que um atacante determinaria quais dispositivos existem dentro de uma dada residência, são ataques contra a privacidade. Muitos deles nem sequer necessitam que o intruso esteja conectado com a rede em questão, bastando ter acesso aos pacotes que são enviados.

Para manter garantias sobre os protocolos e redes utilizadas, é comum a utilização de métodos formais e verificadores automáticos. No entanto, para verificar os ataques discutidos acima, a abordagem difere das usuais propriedades de alcançabilidade. Isso

se dá porque quando se lida com tais propriedades, costuma-se buscar um traço que apresente uma falha, *i.e.* um estado em que o intruso tenha acesso a algum segredo.

Nos ataques descritos acima, o intruso apenas observa o tráfego [Apthorpe Noah Reisman 2017] e monitora o tempo [Chothia e Smirnov 2010]. Através das diferenças do tempo de resposta ou do fluxo de pacotes, o intruso é capaz de determinar algumas características do sistema, como a origem dos pacotes, ou a identidade de determinado passaporte. Isso é feito comparando dois traços distintos, e buscando comportamentos que possam discriminar um do outro. No ataque do passaporte, por exemplo, se o atacante difundir uma mensagem antiga de um determinado passaporte, existe uma diferença no tempo de resposta quando o passaporte em questão responde de quando um passaporte qualquer responde. Perceba que o ataque só se configura ao se comparar um traço com o outro.

Para garantir que ataques como o descritos acima não sejam possíveis, é necessário garantir que os possíveis estados dos sistemas envolvidos sejam observacionalmente equivalentes. Essa é a principal contribuição deste trabalho: fornecer uma definição formal para a verificação de equivalência observacional de tempo e tráfego, que possa ser usada para verificar falhas de segurança de forma automatizada. A segunda contribuição consiste na implementação de alguns exemplos como prova de conceito.

1.1 Objetivos

1.1.1 Objetivos Gerais

Utilização de métodos formais para a verificação automática de problemas de privacidade que se utilizam de canais laterais, como o tempo e o tráfego.

1.1.2 Objetivos Específicos

- Propor definições que formalizem de forma satisfatória os problemas apresentados.
- Estudar as propriedades dessas definições.

- Reduzir as suas soluções para Rewriting Modulo SMT.
- Implementar uma prova de conceito.

1.2 Estrutura da Dissertação

O restante desse texto se organiza como se segue: **Capítulo 2**: Apresentação da fundamentação teórica necessária para o desenvolvimento e compreensão deste trabalho, assim como uma seção de trabalhos relacionados. Conceito básicos de criptografia, modelo de intruso, privacidade, uma visão geral de métodos formais, equivalência observacional, SMT, unificações, Maude e por fim trabalhos relacionados. **Capítulo 3**: Apresentação do trabalho desenvolvido no âmbito da equivalência observacional temporal. O formalismo desenvolvido, assim como os exemplos motivadores e as implementações das provas de conceito da equivalência observacional temporal se encontra nesse capítulo. **Capítulo 4**: Análogo ao capítulo 3, mas com o formalismo e prova de conceito da equivalência observacional de tráfego. **Capítulo 5:** Considerações finais e trabalhos futuros.

Capítulo 2

Fundamentação teórica

Nós propomos uma definição formal de equivalência observacional, que possibilite verificação automática. Esse tipo de problema não consiste em simplesmente encontrar um possível estado que apresente uma falha, e sim a comparação entre comportamentos de diferentes sistemas, de forma a distinguir um sistema do outro. Para isso, nós precisamos elaborar um conceito de equivalência de traços.

Resolver esse tipo de problema também apresentou um desafio ao se lidar com domínios infinitos ou muito grandes, de forma a apresentarem um espaço de busca inviável. Para lidar com esses problemas, nós utilizamos termos simbólicos restritos por fórmulas SMT, uma técnica conhecida como *Rewriting Modulo SMT*, de forma a reduzir consideravelmente o espaço de busca.

Nós também implementamos alguns exemplos como prova de conceito, utilizando uma combinação de Maude integrado com o resolvedor SMT Yices utilizando o *framework* IOP, e o resolvedor SMT Z3.

O restante desse capítulo está organizado como se segue. Na seção 2.1, nós introduzimos conceitos de criptografia necessários para a compreensão plena do trabalho. Na seção 2.2, discorremos sobre o modelo de intruso Dolev-Yao, escolhido para esse trabalho. Na seção 2.3, falamos sobre alguns ataques contra a privacidade. Na seção 2.4, elaboramos sobre a utilização de métodos formais para solução de problemas. Nas seções 2.6 e 2.7, nós abordamos SMT e unificações respectivamente, que são diretamente utilizados nas nossas definições e implementações. Na seção 2.8 nós falamos um pouco sobre a linguagem Maude. Por fim, na seção 2.9, nós apresentamos os trabalhos

relacionados.

2.1 Princípios de Criptografia e Segurança

Apesar desse trabalho não envolver a implementação de algoritmos criptográficos, para entendê-lo é necessário algumas noções básicas em segurança e criptografia, que serão explicadas nessa seção.

Criptografia consiste, de forma intuitiva, em processar uma mensagem de forma que seu conteúdo só possa ser lido por algum participante específico. Para isso, existem os algoritmos criptográficos (cujas implementações serão abstraídas para o propósito desse trabalho) que, *dado um texto a descoberto* (*plaintext*) e uma *chave criptográfica* (*cryptographic key*), geram a *cifra* (*cypher*).

A ideia é que qualquer usuário de posse de uma cifra só possa obter o *plaintext* se ele possuir a chave inversa da que foi usada para encriptar aquele dado. Então, tomando um *plaintext* t, e uma chave criptográfica k, nós dizemos que a cifra de t encriptada com k é

$$\mathsf{e}(\mathsf{t}, \mathsf{k})$$

e a chave inversa necessária para decriptação é k^{-1}. Dessa forma, a equação que representa a decriptação de uma chave seria

$$\mathsf{dec}(\mathsf{e}(\mathsf{t}, \mathsf{k}), \mathsf{k}^{-1}) = \mathsf{t}$$

Existem, no entanto, dois tipos de chaves criptográficas relevantes para esse trabalho: as chaves simétricas, e as chaves assimétricas. As chaves criptográficas simétricas são aquelas utilizadas em algoritmos simétricos, e a principal características delas é a de que $\mathsf{k} = \mathsf{k}^{-1}$, *i.e.*, você utiliza a mesma chave para encriptar e decriptar uma mensagem. Algoritmos como o Twofish [Schneier et al. 1998] e o AES [Processing e The 2001], por exemplo, são algoritmos de criptografia simétrica.

As chaves assimétricas são utilizadas por algoritmos assimétricos, *e.g.*, algoritmo de Diffie-Hellman [Diffie e Hellman 1976] e RSA[Rivest, Shamir e Adleman 1978], e são responsáveis pelo que nós chamamos de criptografia de chave pública. A ideia aqui é que as chaves criptográficas são geradas em pares. Um participante chamado *Alice*,

digamos, gera um par de chaves: uma chave pública $\mathsf{pk}(Alice)$, e uma chave privada $\mathsf{sk}(Alice)$. E a relação que essas chaves têm entre si é a de que $\mathsf{pk}(Alice) = \mathsf{sk}(Alice)^{-1}$ e $\mathsf{sk}(Alice) = \mathsf{pk}(Alice)^{-1}$, *i.e.*, a chave pública é a inversa da chave privada, e vice-versa.

Criptografia simétrica é computacionalmente menos custosa do que a criptografia assimétrica, mas menos flexível. Por exemplo, vamos imaginar um cenário em que um participante *Alice* queira conversar, de forma privada, com um outro participante *Bob*, e ela quer fazer isso utilizando apenas chaves simétricas. Se *Alice* enviar a mensagem $\mathsf{m} = \mathsf{e}(\{Oi, Bob\}, \mathsf{k})$, onde k é simétrica, *Bob* não teria como extrair a mensagem contida na cifra, já que *Bob* não possui a chave k. Por outro lado, se *Alice* envia k para *Bob* pela rede, então nós dizemos que a chave k foi *comprometida*, *i.e.*, não se sabe mais quais são os participantes que possuem aquela chave criptográfica. Qualquer participante pode ter interceptado a mensagem de *Alice* no meio do caminho e feito uma cópia de k para si, de forma que a comunicação entre *Alice* e *Bob* não será segura.

Se *Alice* tivesse um canal seguro de comunicação com *Bob* (mesmo que computacionalmente mais custoso), ela poderia usar esse canal para enviar k, e então se comunicar normalmente. O problema é que obter um canal seguro não é fácil, e nem sempre possível, ainda mais quando se leva em consideração comunicações através da internet. Para resolver esse problema, *Alice* poderia utilizar criptografia assimétrica.

Alice poderia gerar um par de chaves assimétricas, $\mathsf{pk}(Alice)$ e $\mathsf{sk}(Alice)$. Ela disponibilizaria $\mathsf{pk}(Alice)$ para todos (por isso o nome chave pública), e manteria $\mathsf{sk}(Alice)$ em segredo (por isso o nome chave privada, ou chave secreta). *Bob*, por sua vez, faria o mesmo, criando um par de chaves $\mathsf{pk}(Bob)$ e $\mathsf{sk}(Bob)$. Agora, quando *Alice* quisesse falar com *Bob*, bastaria encriptar uma mensagem m qualquer com a chave $\mathsf{pk}(Bob)$. *Bob*, e apenas *Bob*, possuiria a chave inversa $\mathsf{sk}(Bob)$, de forma que apenas ele poderia obter o conteúdo de m.

Essa abordagem tem dois problemas principais. O primeiro deles é o custo computacional. Criptografia assimétrica é uma operação muito custosa computacionalmente, não apenas na geração das chaves mas também na encriptação e decriptação dos dados. O segundo problema dessa abordagem é que, se *Bob* receber uma mensagem m qualquer, encriptada com a própria chave pública, ele não pode ter certeza alguma de que foi Alice que realmente enviou a mensagem. Como todos os participantes têm acesso

a sua chave pública, qualquer um deles poderia ter enviado m.

Idealmente, *Alice* e *Bob* usariam uma troca inicial de mensagens para estabelecer uma chave simétrica que eles usariam para manter o canal de comunicação. Essa troca de mensagens, preferencialmente, também garantiria a identidade dos usuários participantes. Para a chave trocada, nós damos o nome de *chave de sessão* (*session key*), e para a sequência ordenada de mensagens, de *protocolo de segurança*. Dito isso, vamos visualizar a seguinte troca de mensagens, que vamos chamar de Protocolo 1 (Figura 2.1):

$$Alice = (+\mathsf{e}(\langle Alice, \mathsf{e}(Alice, \mathsf{sk}(Alice))\rangle, \mathsf{pk}(Bob))), (-\mathsf{e}(\mathsf{e}(Ok, \mathsf{sk}(Bob)), \mathsf{pk}(Alice))),$$
$$(+\mathsf{e}(\mathsf{e}(\mathsf{k}, \mathsf{sk}(Alice)), \mathsf{pk}(Bob)))$$
$$Bob = (-\mathsf{e}(\langle Alice, \mathsf{e}(Alice, \mathsf{sk}(Alice))\rangle, \mathsf{pk}(Bob))), (+\mathsf{e}(\mathsf{e}(Ok, \mathsf{sk}(Bob)), \mathsf{pk}(Alice))),$$
$$(-\mathsf{e}(\mathsf{e}(\mathsf{k}, \mathsf{sk}(Alice)), \mathsf{pk}(Bob)))$$

A notação utilizada é simples: $(+\mathsf{m})$ quer dizer que o participante envia uma mensagem, e $(-\mathsf{m})$ que o participante espera uma mensagem. $\langle \mathsf{m}, \mathsf{m}_1 \rangle$ é uma concatenação convencional de mensagens.

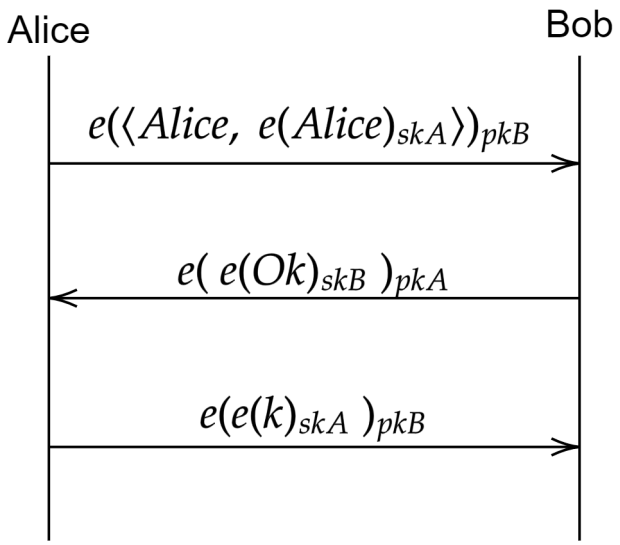

Figura 2.1: Protocolo 1

A participante *Alice* envia $\mathsf{e}(\langle Alice, \mathsf{e}(Alice, \mathsf{sk}(Alice))\rangle, \mathsf{pk}(Bob))$ para *Bob*. A mensagem é encriptada com $\mathsf{pk}(Bob)$ o que garante a *Bob* que aquela mensagem é endereçada para ele. A mensagem de forma $\langle Alice, \mathsf{e}(Alice, \mathsf{sk}(Alice))\rangle$ tem duas fun-

ções: informar para *Bob* qual o participante que quer falar com ele, representado pela parte *Alice*, e garantir a identidade do emissor, com $\mathsf{e}(Alice, \mathsf{sk}(Alice))$.

Perceba que na segunda parte da mensagem, *Alice* encripta o próprio nome com a sua chave secreta. Quando *Bob* receber essa mensagem, ele vai usar a chave pública de *Alice* para decriptar a mensagem encriptada (lembre-se que a chave pública é a inversa da chave privada). Se essa decriptação resultar no nome que o participante assegurou anteriormente, nesse caso *Alice*, é porque de fato foi ela que enviou tal mensagem, já que apenas *Alice* teria acesso a própria chave privada.

Essa técnica, de encriptar uma mensagem com a sua chave privada, é uma forma de autenticação de mensagens, *i.e.*, uma técnica para garantir que uma mensagem veio de quem ela afirma que veio (sua autenticidade) e de que ela não foi alterada no caminho. A utilização de MACs (Message Authentication Code) também presta ao mesmo serviço. Um determinado participante poderia assinar uma mensagem com um MAC, que poderia ser avaliado posteriormente por outrem, como forma de garantir autenticidade [IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications 2007].

Recebida a mensagem de *Alice*, *Bob* então envia uma mensagem de resposta afirmando que está preparado para começar uma troca de mensagens. Por fim, *Alice* envia para *Bob* a mensagem $(\mathsf{e}(\mathsf{e}(\mathsf{k}, \mathsf{sk}(Alice)), \mathsf{pk}(Bob)))$, onde k é a chave de sessão que eles usarão para se comunicar daqui pra frente. Perceba que k também foi encriptada com a chave secreta de *Alice*. Se esse não fosse o caso, algum outro participante malicioso poderia esperar *Bob* responder para *Alice* que está preparado para começar a troca de mensagens, e interceptar tal mensagem. Depois, o atacante simplesmente enviaria uma chave de sessão qualquer k_I encriptada com $\mathsf{pk}(Bob)$ e o resultado seria um *Bob* que estaria se comunicando com um intruso, mas pensaria estar se comunicando com *Alice* [Lowe 1995] .

O problema desse pequeno protocolo é que ele não possui nenhuma prova de temporalidade. Nada indica que *Alice* quer falar com *Bob* agora, e não que *Alice* quis falar com *Bob* meses atrás. Além disso, a geração de chaves é unilateral, no sentido

que *Alice* gera e envia a chave de sessão, e *Bob* apenas recebe. Isso pode gerar alguns problemas.

Vamos revisitar nosso exemplo, mas dessa vez, além de *Alice* e *Bob*, nós teremos uma usuária maliciosa, *i.e.*, um intruso, chamada *Eve*. *Alice* começa uma sessão de protocolo normal com *Eve*, até o momento em que eles trocam a chave de sessão, na mensagem $\mathsf{e}(\mathsf{m}_1, \mathsf{pk}(Eve))$, onde $\mathsf{m}_1 = \mathsf{e}(\mathsf{k}_I, \mathsf{sk}(Alice))$. A partir desse momento, *Eve* só precisa esperar *Alice* querer se comunicar com *Bob*. *Bob* responderia, afirmando disponibilidade para iniciar uma troca de mensagens, e nesse momento o nosso intruso interceptaria essa mensagem. *Eve*, então, enviaria a mensagem m_I encriptada com $\mathsf{pk}(Bob)$. O resultado seria *Bob* estar se comunicando com *Eve* mas achando que está se comunicando com *Alice*.

Para resolver esse tipo de problema, existe na criptografia o conceito de *nonce*. *Nonces* são valores numéricos recém-criados (*fresh*), e que são descartados ao término do protocolo. Eles são utilizados para a geração da chave de sessão e também para garantir que as mensagens que estão sendo recebidas e enviadas são mensagens recentes, e não de sessões anteriores. Vamos observar o protocolo de Needham-Schroeder-Lowe [Lowe 1995] na Figura **??**.

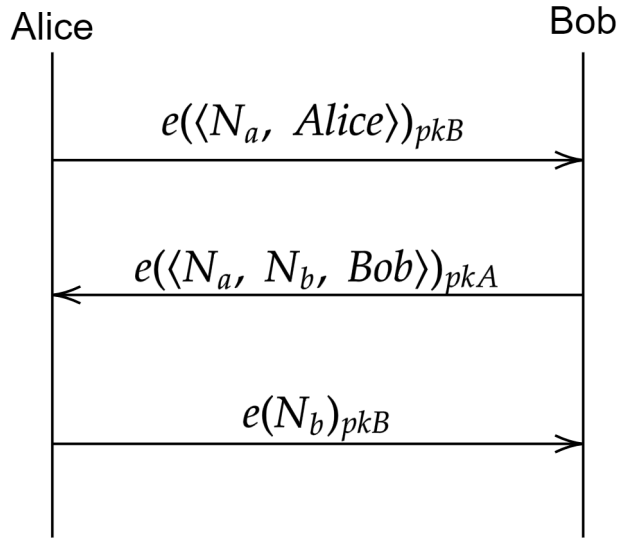

Figura 2.2: Protocolo de Needham-Schroeder-Lowe

Aqui, na, nb são os *nonces* criados durante a execução do protocolo, e os participantes *Alice* e *Bob* utilizam esses valores para a geração de uma chave de sessão. Perceba

que, ao término desse protocolo, ambos os participantes se encontram de posse de ambos os *nonces*, e mais ninguém possui tais valores. Utilizar mensagens antigas também não funciona, já que os valores de *na* e *nb* são re-instanciados a cada nova sessão.

Nesse caso específico, a intenção do protocolo é estabelecer uma chave de sessão que os participantes possam usar para se comunicar entre si. Mas existem outros protocolos de segurança que possuem outros objetivos além desses : garantir a anonimidade de um grupo [Abadi e Fournet 2004], delimitar uma distância máxima que um participante pode estar do outro [Brands e Chaum 1993], etc.

2.2 Intruso Dolev-Yao

Quando se pretende modelar protocolos de segurança, a definição das capacidades e fraquezas do intruso é tão importante quanto a definição dos participantes honestos. Nesse âmbito, um modelo bastante popular na literatura é o definido em [Dolev e Yao 1983], conhecido como intruso de Dolev-Yao.

De um ponto de vista criptográfico, o intruso de Dolev-Yao sabe o nome de todos os participantes do protocolo, assim como suas chaves públicas. Ele é capaz de encriptar mensagens com qualquer chave que esteja contida em seu conhecimento, assim como decriptar mensagens, assumindo que ele possua a chave inversa necessária para tal. Ele é capaz de concatenar e fragmentar mensagens, contanto que estas não estejam protegidas por uma encriptação. Além disso, o intruso é capaz de expandir o seu conhecimento aprendendo novas mensagens e chaves criptográficas.

De um ponto de vista de transporte, esse tipo de modelo representa a própria rede. Toda mensagem enviada por qualquer participante necessariamente passa pelo intruso, que pode manipula-lá, retê-la ou encaminhá-la como lhe convêm. Além disso, restrições físicas não atuam sobre o intruso de Dolev-Yao, que pode exercer a sua influência sobre o protocolo sem levar em consideração distância ou latência, podendo se comunicar com quem deseja instantaneamente.

Uma das grandes vantagens da utilização desse tipo de modelo é a forma como ele simplifica a modelagem. Ao levar em consideração um intruso incrivelmente poderoso, essencialmente se cria um limitante superior para o escopo de atuação de usuários

maliciosos. Se o protocolo se mostrar sólido sob ataque de um intruso tão forte, ele manterá a solidez quando testado por um intruso com capacidades mais limitadas. Além disso, como demonstrado por [Cervesato 2003], qualquer ataque executado por dois ou mais intrusos Dolev-Yao pode ser executado por um único intruso desse tipo, o que simplifica na hora da modelagem formal.

O intruso de Dolev-Yao, no entanto, é considerado forte demais para algumas análises. Ao verificar um protocolo que utiliza esse tipo de modelo para o intruso, pode-se obter ataques que são incrivelmente improváveis ou simplesmente impossíveis de serem realizados por um intruso com capacidades mais realistas.

Esse tipo de problema se torna ainda mais evidente quando se avalia protocolos de segurança que possuem algum elemento físico, como tempo [Ho et al. 2014; Chothia e Smirnov 2010; Kanovich et al. 2014] ou distância [Brands e Chaum 1993]. Como o intruso é a própria rede e pode se comunicar instantaneamente, fazer análises que envolvem restrições físicas não é possível com esse tipo de intruso, a não ser que modificações sejam feitas para adaptá-lo para esse tipo de análise. O artigo [Nigam, Talcott e Urquiza 2016], por exemplo, propõe uma versão do intruso de Dolev-Yao que incorpora elementos de tempo e distância.

2.3 Privacidade

Ataques contra a privacidade (*Privacy attacks*) consistem na obtenção de dados sensíveis. Um invasor conseguir informações sobre a rotina ou a localização de algum participante, por exemplo, consistiria em uma quebra de privacidade.

Em um artigo que analisa alguns protocolos de passaportes eletrônicos europeus [Chothia e Smirnov 2010], os autores demonstraram uma série de ataques sobre algumas implementações do protocolo BAC (Figura 2.3).

Nesse protocolo, o leitor inicia com a mensagem *Get_challenge*, e o passaporte responde com um *nonce* N_T . O leitor então cria o seu próprio *nonce*, N_R, e parte da chave de sessão, K_R, junta os dois com N_T, encripta, e envia de volta para o passaporte, junto com uma assinatura MAC (*Message Authentication Code*) computada usando informações presentes no passaporte. O passaporte checa a assinatura MAC, depois

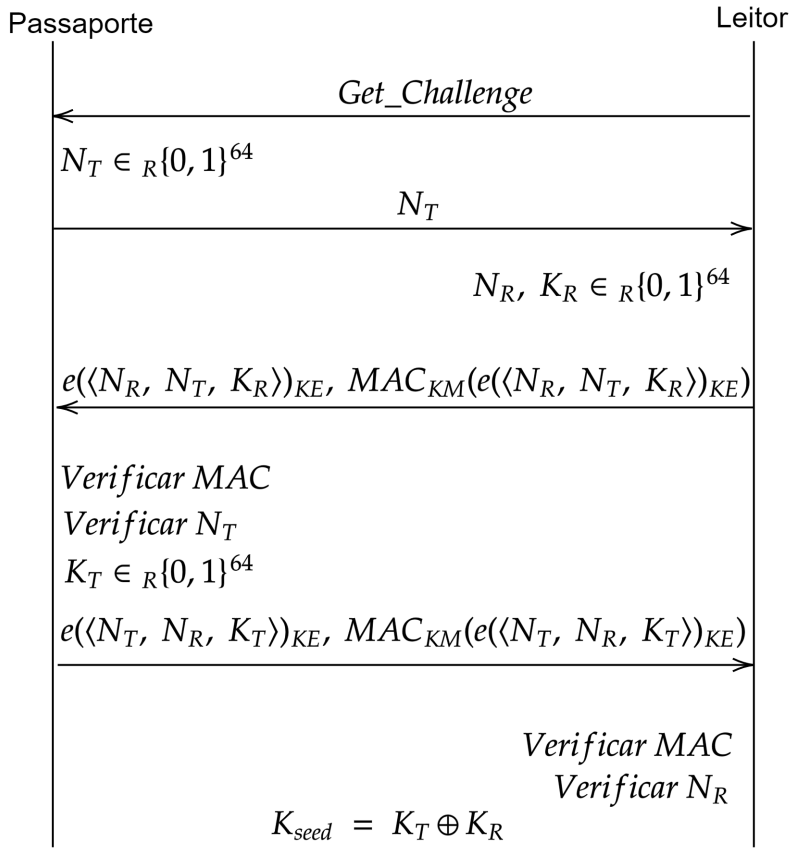

Figura 2.3: Basic Access Control [Chothia e Smirnov 2010]

decripta a mensagem e checa N_T (que garante que a mensagem enviada não é uma mensagem antiga que foi gravada). Se tudo bater, o passaporte cria a sua parte da chave, K_T, e envia da mesma maneira que o leitor, invertendo apenas a ordem dos *nonces* para impedir que a mensagem do leitor não seja simplesmente retransmitida de volta para ele. Se tudo isso bater, o leitor e o passaporte utilizam um xor entre K_T e K_R como material gerador da chave de sessão.

Os autores de [Chothia e Smirnov 2010] perceberam que na implementação francesa do protocolo BAC, o passaporte respondia com uma mensagem (*6300 : No information given*) quando a checagem do MAC falhava, e com outra (*6A80 : Incorrect parameters*) quando o *nonce* verificado não correspondia com o *nonce* enviado. Outro detalhe percebido, que é válido para várias outras implementações do BAC, é a de que existe uma diferença perceptível no tempo de resposta entre a verificação do MAC e a do *nonce*. Isso se deve ao fato de que o passaporte aborta a sessão imediatamente após a

falha, e a diferença de tempo percebida entre os dois retornos equivale ao tempo que demora para o passaporte decriptar a mensagem e checar a validade do *nonce*.

Ambos os cenários consistem em uma falha de privacidade. Se um atacante conseguisse obter uma mensagem antiga de algum determinado passaporte, ele poderia usar essa mensagem para rastrear ou identificar tal passaporte, o que consistiria em mais uma vulnerabilidade que poderia ser explorada. Ao difundir (*broadcast*) a mensagem gravada, ele poderia checar as respostas recebidas. Se alguma delas fosse a 6A80, ou se o tempo de resposta fosse maior do que um determinado limitante superior (obtido facilmente de forma empírica), o atacante saberia que o passaporte que respondeu consiste justamente no passaporte que ele está rastreando.

Outro artigo descreve uma série de ataques realizados sobre dispositivos IoT que funcionavam dentro de uma *smart home* [Apthorpe Noah Reisman 2017]. A ideia desse ataque consistia em avaliar a atividade desses dispositivos, e inferir a sua atividade através da avaliação de tráfego, independente do conteúdo dos pacotes. Esse ataque consiste em duas etapas: identificação e inferência de atividades.

Dispositivos	Requisições DNS
Sense Sleep Monitor	hello-audio.s3.amazonaws.com
	hello-firmware.s3.amazonaws.com
	messeji.hello.is
	ntp.hello.is
	sense-in.hello.is
	time.hello.is
Nest Security Camera	nexus.dropcam.com
	oculus519-vir.dropcam.com
Amcrest Security Camera	amcrestcloud.com
	command-3.amcrestcloud.com
	ftp.amcrestcloud.com
	media-amc-1.hostedcloudvideo.com
	p2p.amcrestview.com
	dh.amcrestsecurity.com
Belkin WeMo Switch	prod1-fs-xbcs-net-1101221371.us-east-1.elb.amazonaws.com
	prod1-api-xbcs-net-889336557.us-east-1.elb.amazonaws.com
TP-Link Smart Plug	devs.tplinkcloud.com
	uk.pool.ntp.org
Orvibo Smart Socket	wiwo.orvibo.com
Amazon Echo	ash2-accesspoint-a92.ap.spotify.com
	device-metrics-us.amazon.com
	ntp.amazon.com
	pindorama.amazon.com
	softwareupdates.amazon.com

Figura 2.4: Requisições DNS de dispositivos IOT utilizados em lares inteligentes [Apthorpe Noah Reisman 2017]

A etapa de identificação, que pode ser considerada uma falha de privacidade por si só, consiste em individualizar cada um dos dispositivos que existem dentro de uma

determinada residência. Ao interceptar os pacotes, o atacante consegue determinar de qual dispositivo aquele pacote provém. Através do endereço MAC (*Media Access Control*, não confundir com o MAC *Message Authentication Code*), o atacante poderia utilizar os primeiros três dígitos, que são usados como um identificador de fabricante, para diminuir bastante o espaço de busca dos possíveis dispositivos emissores de pacotes. Através das requisições DNS, o atacante conseguiria individualizar o dispositivo, já que alguns deles fazem requisições bem específicas (Figura 2.4). E nos casos em que tal requisição não existe, muitos dispositivos fazem requisições para múltiplos domínios, de forma que o conjunto deles poderia ser utilizado como identificador único para um dispositivo específico. Além disso, através da captação de pacotes, em conjunto com os métodos anteriores para diminuir o espaço de busca, é possível fazer uma identificação exata através da assinatura de tráfego. Por exemplo, a fabricante X pode ter apenas um dispositivo que emite pacotes em um formato de impulsos, com alguns picos muito altos de tráfego, seguido por baixa atividade. Logo, se o fabricante for identificado, e esse padrão detectado, pode-se afirmar a origem daquele fluxo de pacotes.

A segunda etapa do ataque consiste na inferência de atividade. Uma vez que o atacante sabe qual a fonte de um determinado fluxo de pacotes, é possível determinar o comportamento do aparelho baseado na assinatura de fluxo dos pacotes ao longo do tempo. Por exemplo, no caso do sensor de movimento Nest Security. Aqui, o sensor aumenta consideravelmente a sua atividade quando detecta movimento (Figura 2.5). Ao perceber esse tipo de flutuação, um atacante é capaz de dizer quando determinado sensor está ativo e, com isso, determinar quando uma criança estivesse no berço, ou o horário em que o dono da casa se deita para dormir. Ao saber quando um determinado dispositivo realiza uma certa atividade, é possível inferir muito sobre o comportamento dos habitantes da residência. Isso é uma falha grave de privacidade por parte desses dispositivos, o que é exacerbado pelo fato de que a maioria deles não consegue funcionar sem acesso a internet.

A distribuição de bancos de dados para pesquisa se tornou muito comum recentemente, com o aumento considerável de interesse nas áreas de *Machine Learning* e *Big Data*. Quando os bancos de dados disponibilizados contêm informações sensíveis sobre alguém, como histórico clínico, o *dataset* normalmente sofre um processo conhecido

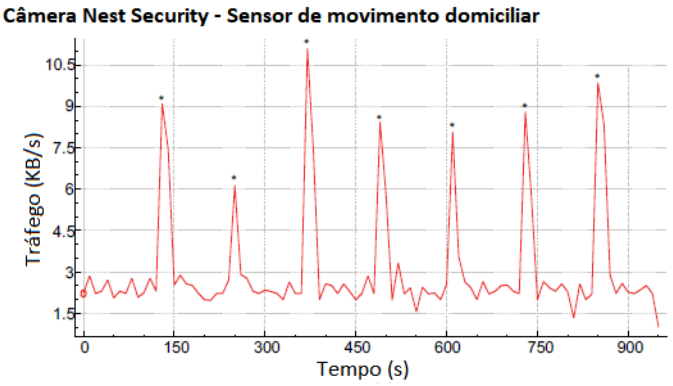

Figura 2.5: Os picos no gráfico consistem nos momentos em que o sensor detectou movimento [Apthorpe Noah Reisman 2017]

como *deidentification*, que consiste na remoção de atributos que possam individualizar uma instância, como o nome, o CPF, o RG, etc. O problema consiste em determinar quais características remover. Quando acumula-se um certo número de características, qualquer subconjunto delas pode servir como um identificador único [Fung et al. 2010]. Por exemplo, existem muitas pessoas que têm 1,69m de altura, mas quantas pessoas tem 1,69m, são do sexo masculino, nasceram em 22/11/1994 e são caucasianas? Em um ataque famoso, um artigo descreve como combinou um banco de dados de alta hospitalar de um hospital em Massachusetts com a lista pública de eleitores, de forma a remontar a identidade de vários registros [Sweeney 1997].

Em um outro artigo similar [Narayanan e Shmatikov 2008], os autores realizam um ataque sobre o *dataset* Netflix Prize, disponibilizado inicialmente pelo Netflix em 2 de Outubro de 2006 como parte de um concurso para melhorar o seu algoritmo de indicação. O *dataset* consistia em aproximadamente 500 mil registros de usuários, contendo os filmes avaliados pelo determinado cliente, a data da avaliação e a nota. Utilizando técnicas parecidas com a discutida anteriormente, e levando em consideração que o atacante poderia descobrir alguma informação sobre o alvo (a nota que ele deu para alguns filmes, dentro de um intervalo de até quinze dias), ele conseguiria descobrir o conjunto de todos os filmes assistidos e avaliados por determinado cliente até o momento de liberação do *dataset*.

Esses exemplos mostram que nem sempre é tarefa fácil determinar o que é ou não uma informação sensível. O atacante, dependendo do cenário em que ele se encontra,

pode usar uma variedade de informações disponíveis para inferir sobre o comportamento de um usuário honesto. Algumas pessoas podem pensar que uma falha de privacidade em um serviço de avaliação de filmes é um problema de menor importância. No entanto, como é assumido que tal informação é sigilosa, atacantes podem levar essa suposição em consideração em um ataque futuro. No caso de uma comunicação pela internet, por exemplo, alguém poderia conseguir obter informações suficientes de alguém protegido por um *nickname* de forma a associar essas informações com um registro do *dataset* da Netflix e potencialmente com uma identidade real.

2.4 Métodos formais

Em 8 de Agosto de 1900, o matemático alemão David Hilbert anunciava pela primeira vez a sua famosa lista de 23 problemas que o século 19 deixava para o século 20 resolver. O décimo problema de Hilbert consistia na elaboração de um processo geral que, para qualquer equação diofantina (uma equação polinomial com um número finito de incógnitas e com coeficientes inteiros), decidisse se aquela equação possuía uma solução na qual todas as incógnitas assumissem valores inteiros.

Na época, a definição clara de algoritmo ainda não havia sido elaborada. Alonzo Church e Alan Turing seriam responsáveis por isso em 1936, com o λ cálculo e as máquinas de Turing. Com uma clara definição do que exatamente Hilbert buscava para as equações diofantinas, Yuri Matiyasevich, baseado nos trabalhos de Hilary Putnam, Martin Davis e Julia Robinson, provou em 1970 que o algoritmo buscado por Hilbert era impossível de ser concebido [Matiyasevich 1993].

Perceba que, com a noção intuitiva de algoritmo que precedia Turing e Church, a elaboração de uma prova de inexistência de um determinado algoritmo seria muito mais difícil de ser concebida. Uma vez de posse da definição formal de um problema ou conceito, elaborar provas sobre ele se torna consideravelmente mais fácil.

Mas ampliar a compreensão sobre um determinado problema não é a única razão para utilização de métodos formais. Para exemplificar o nosso segundo motivo, falaremos um pouco sobre um protocolo clássico da literatura.

O protocolo de Needham-Schroeder(Figura 2.6) foi proposto inicialmente em 1978

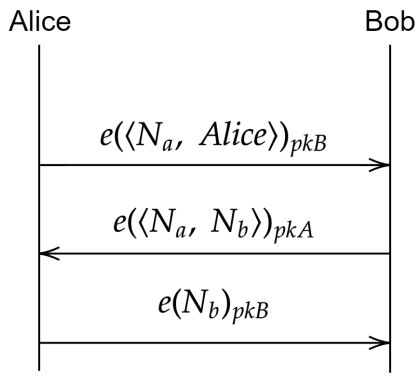

Figura 2.6: Protocolo de Needham-Schroeder

[Needham e Schroeder 1978], e ganhou bastante popularidade. Ele consistia em um protocolo com o intuito de trocar uma chave de sessão, feito para ser usado em uma rede insegura. Em 1995, Gavin Lowe mostrou que existe um ataque de *man-in-the-middle* sobre o protocolo de Needham-Schroeder [Lowe 1995], mostrado na Figura 2.7.

Nesse ataque, um participante *Alice* tenta iniciar uma sessão com o nosso participante malicioso, *Eve*. Nessa instância, se *Eve* retransmitir a mensagem para um segundo participante qualquer, chamado *Bob*, ele pode conceber um cenário que termina com *Eve* e *Bob* tendo trocado uma chave de sessão, e *Bob* achando que se comunica com *Alice*. Lowe propôs uma solução para esse ataque, que consistia na inserção do identificador do participante na segunda mensagem do protocolo. Essa correção viria a ser conhecida como o protocolo de Needham-Schroeder-Lowe, já apresentado nesse trabalho na seção 2.1, Figura 2.2 .

Esse exemplo serve para ilustrar uma segunda propriedade interessante dos métodos formais: as garantias que eles podem fornecer. Entre a concepção do protocolo de Needham-Schroeder (1978) e a descoberta do ataque sobre ele (1995), passaram-se quase 20 anos. Em ambientes críticos, é necessário minimizar ao máximo a possibilidade de erro. Assumir que um software não está errado, ou que está livre de bugs, simplesmente através da intuição se provou muitas vezes problemático, como exemplificado acima. Métodos formais provém formas de gerar garantias mais fortes a respeito do funcionamento de determinada aplicação ou sistema, bastante úteis em áreas como segurança da informação e no setor aerospacial, por exemplo.

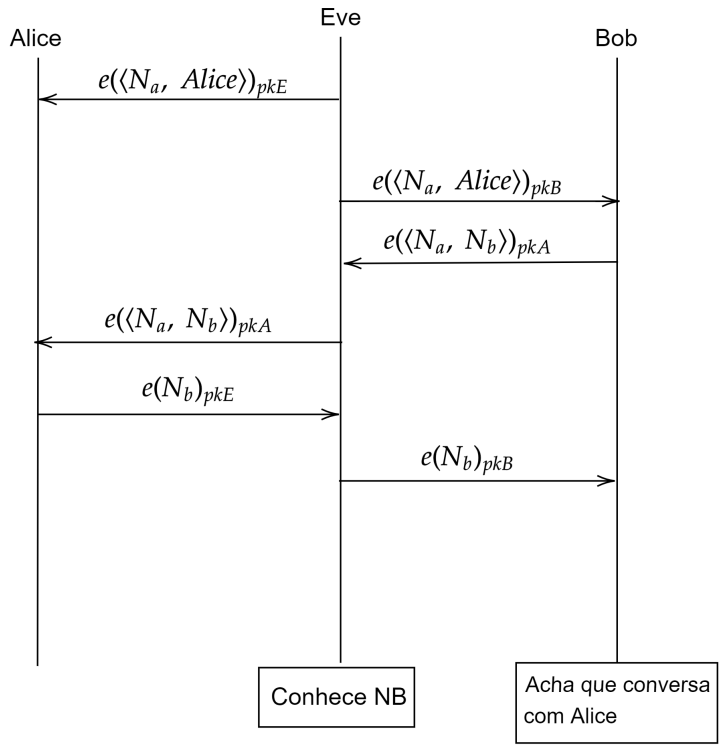

Figura 2.7: Ataque clássico sobre Needham-Schroeder

Na área de avaliação de programas, a análise estática consiste em métodos para avaliar o funcionamento de um programa sem precisar executá-lo, apenas analisando o código (o que ele foi escrito, o código fonte, etc). No artigo [Semple Sean Ponomarev 2013], os autores propõe a utilização de métodos de análise estática como alternativa para a abordagem da análise de assinatura (buscar padrões previamente conhecidos oriundos de softwares maliciosos). Dessa forma, o programa anti-vírus não ficaria completamente a mercê de *malwares* que ainda não sejam conhecidos, ou que utilizem técnicas de ofuscação de código. Um outro artigo [Lee et al. 2013] analisa o uso de métodos estáticos para a garantia de exatidão e segurança na implementação de sistemas RPS (*Reactor Protection System*) em usinas nucleares.

Proposto independentemente por [Clarke e Emerson 1982] e [Queille e Sifakis 1982], o *model checking* consiste em uma busca através dos estados de um modelo até percorrer todos os estados possíveis ou encontrar uma configuração de interesse. Determinado modelo consiste em estados e transições, e uma condição de terminação é definida com base no modelo elaborado, ambos escritos em linguagens matematicamente precisas. É

bem comum as condições serem modais/temporais, cuja validade é checada em função do estado do modelo naquele determinado momento.

Durante a modelagem de algum cenário, muitas vezes a quantidade de estados possíveis é enorme, tornando a busca exaustiva inviável. Nesse contexto, existe uma variante do *model checking* chamada de BMC (*bounded model checking*) [Biere et al. 1999] , em que a profundidade da busca é limitada. Nesses casos, o método de *model checking* garante a ausência de uma configuração crítica até profundidade n, mas não garante a ausência da mesma em uma profundidade n + 1 ou adiante. Apesar disso, esses tipos de métodos são bastante apreciados porque, na presença de uma configuração de interesse, o verificador produz um contraexemplo, construído através do estado inicial e a lista de transições que se seguiram até aquela condição ser atendida.No artigo [Ge et al. 2016] é descrito um método de utilização de BMC para a verificação de código de um sistema anti-colisão (*Automatic Rover Protection*, ou ARP) de um *rover*. O artigo [Kojima, Nagashima e Tsuchiya 2017] disserta sobre técnicas de modelagem que reduzem o espaço de busca em modelos para verificação de MANETs (*Mobile ad hoc Network*)

2.5 Equivalência Observacional

Carl Sagan, no livro O mundo assombrado pelos demônios, de 1995, capítulo 10, fala sobre a discussão do dragão na garagem. Nesse capítulo, Sagan afirma que, se algum conhecido seu dissesse seriamente que existe um dragão cuspidor de fogo dentro da própria garagem, você iria querer averiguar por si mesmo, e iria, junto com ele, para tal garagem. Chegando lá, não haveria nada, e seu amigo afirmaria que é um dragão invisível. Você poderia propor que se jogasse farinha no chão, e assim vocês poderiam ver as marcas das pegadas, onde seu amigo replicaria que o dragão paira no ar, não entrando em contato com o chão. Você proporia que se borrifasse tinta na garagem, que iria então aderir em suas escamas, revelando assim a sua forma, e o seu amigo diria que o dragão é incorpóreo. Você proporia utilizar sensores de calor para determinar a assinatura térmica do dragão, e seu amigo diria que o dragão não emite calor.

No seu livro, Sagan então questiona: qual a diferença entre uma garagem com

um dragão invisível, voador, incorpóreo e atérmico, e uma garagem vazia? Sagan utiliza dessa metáfora para ilustrar os efeitos perniciosos de alegações que não podem ser testadas nem refutadas. Aqui, nós vamos por outro caminho. E se havia um dragão na garagem, que se divertisse com as infrutíferas tentativas de detectá-lo? Nesse caso, um cenário com o dragão mágico e um cenário com uma garagem vazia seriam indistinguíveis, apesar de serem distintos. Para todos os efeitos, esses cenários seriam observacionalmente equivalentes entre si.

Vamos voltar para o nosso exemplo do passaporte, discutido na subseção 2.3. Ali, o passaporte e o leitor enviavam mensagens um para o outro, através do protocolo BAC. Um usuário malicioso poderia, seja através do conteúdo das mensagens de erro enviadas ou o tempo de resposta, determinar o que aconteceu internamente no protocolo, e, através disso, obter informações indesejadas sobre o passaporte que ele está se comunicando. Idealmente, ambos os cenários seriam indistinguíveis para um observador externo, porque nenhuma informação indesejada seria disponibilizada. No caso do passaporte eletrônico, os autores propõe uma mensagem de erro única, e tentar realizar a decriptação para checagem do *nonce* mesmo quando a avaliação da validade do MAC falha. Com isso, discernir um cenário do outro, utilizando a medição do tempo de envio e de retorno, não seria mais viável. Aplicando essas medidas, o cenário em que a checagem do MAC falha e o cenário em que a falha acontece na avaliação do nonce seriam indistinguíveis entre si.

O mesmo vale para os exemplos da falha de privacidade em IoT [Apthorpe Noah Reisman 2017] . Para um observador passivo, monitorando apenas os metadados dos pacotes e o fluxo de envio, haveria uma discrepância entre os diferentes dispositivos IoT. De novo, a solução oferecida pelo artigo consiste em criar um cenário em que, independente de quais dispositivos o usuário possua em casa, a saída é sempre a mesma, deixando todos os cenários equivalentes. No artigo [Apthorpe Noah Reisman 2017] em particular, os autores defendem a utilização de um conceito conhecido como *traffic shaping*, implementado através de um método denominado *independent link padding* ou ILP. No artigo, eles sugerem uma forma de ILP que consiste no envio de pacotes de mesmo tamanho em um intervalo regular de tempo .

Na figura 2.8, nós podemos ver como funcionaria a técnica de ILP, ofuscando a

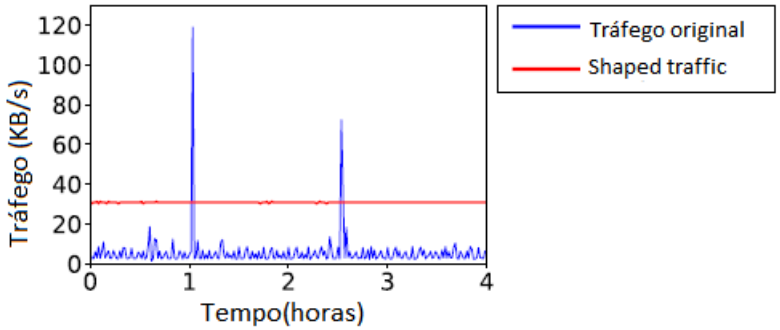

Figura 2.8: Exemplo de implementação de ILP em um tráfego combinado de três dispositivos IoT : Belkin WeMo Switch, Amazon Echo e Sense Sleep Monitor. Todos os pacotes enviados possuíam 512 bytes. [Apthorpe Noah Reisman 2017]

atividade real da rede ao uniformizar a saída . Além de fixar o tamanho dos pacotes, fragmentando ou adicionando bytes dependendo do tamanho original, a solução também fixa a taxa de envios de pacote da rede. Dessa forma, se a rede real possuir uma taxa de envio de pacotes maior do que a taxa determinada, os pacotes seriam inseridos em uma fila até que eles possam ser enviados e ainda assim manter a taxa de saída desejada. Se a taxa de envio dos dispositivos for menor do que a taxa fixada, pacotes falsos são adicionados para suprir a demanda.

Tanto no caso do passaporte quanto no dos dispositivos IoT, o esforço para se obter equivalência observacional entre os estados não vem de graça. No caso do passaporte, você utiliza sempre o maior tempo de resposta, tornando o protocolo mais lento. No caso dos dispositivos IoT, o custo envolve um *trade-off* entre latência e sobrecusto na largura de banda. Se uma taxa lenta de envios de pacotes for escolhida, menos pacotes falsos serão adicionados, mas o tráfego dos dispositivos pode apresentar uma alta latência. Por outro lado, se uma taxa alta de envio de pacotes for escolhida, a latência será baixa, mas bastante largura de banda será consumida para o envio dos pacotes falsos.

Também é importante perceber que equivalência observacional não é universalmente definida. No caso do passaporte, por exemplo, a solução proposta por [Chothia e Smirnov 2010] para mitigar o ataque de rastreamento não funcionaria se o atacante conseguisse quebrar a encriptação das mensagens. Da mesma forma, se o atacante

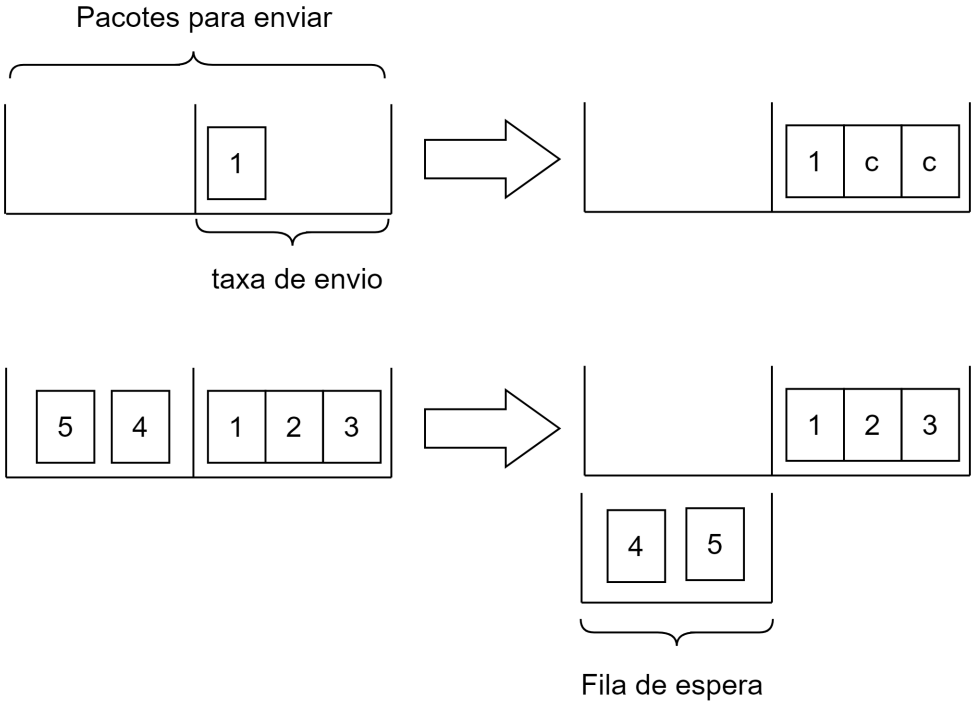

Figura 2.9: Consequências do ILP. Se a taxa de envio selecionada for maior do que os dispositivos exigem, é necessário adicionar pacotes chamariz (representados pelos pacotes rotulados com c), o que exige mais da largura de banda. No caso em que a taxa de envio é menor do que o exigido pelos dispositivos em um dado intervalo de tempo, não há a adição de nenhum pacote adicional. No entanto, os pacotes excedentes serão adicionados em uma fila de espera para envio posterior, o que adiciona latência a conexão.

sobre os dispositivos IoT tivesse capacidade de obter o conteúdo dos pacotes, então a solução proposta não funcionaria. Determinar se dois cenários são equivalentes ou não para um observador externo depende diretamente das capacidades de tal observador. Tais capacidades podem ser mais ou menos prováveis, dependendo do cenário avaliado. Um atacante que consegue medir o tempo entre duas mensagens é bastante realista, enquanto um atacante que é capaz de quebrar a criptografia de mensagens que ele acabou de receber é bem mais improvável.

2.6 SMT

O problema de satisfatibilidade booleana (*Boolean Satisfiability Problem*) consiste em encontrar uma solução para um fórmula de lógica proposicional de forma que o resultado final seja verdadeiro. Nessas fórmulas, os únicos operadores permitidos são: AND, OR, NOT e parênteses.

Por exemplo, vamos visualizar a seguinte fórmula proposicional $\alpha \wedge \beta$ onde α e β são variáveis booleanas. Qual seria uma configuração que tornasse essa fórmula verdadeira ? Nesse caso, uma única configuração, onde α e β são verdadeiros. Quando nós estamos lidando com uma instância tão simples do problema, com apenas duas variáveis, cada uma com dois valores possíveis, nós podemos simplesmente testar todas as possibilidades e verificar se alguma delas resulta em verdadeiro. Quando nós estamos lidando com dezenas, centenas, talvez milhares de variáveis, e fórmulas gigantescas, então resolver o problema através do método da força bruta, seja manualmente ou com auxílio computacional, torna-se inviável. Os programas conhecidos como SAT *solvers* servem justamente para utilizar técnicas mais inteligentes para resolver esse tipo de problema.

SMT, por sua vez, é uma generalização do problema SAT, com uma fórmula em lógica de primeira-ordem, onde alguns símbolos e predicados podem possuir interpretações adicionais, e nós temos que verificar a sua satisfatibilidade. Na prática, imagine uma instância de um problema SAT, onde as variáveis podem ser inteiras, ou reais, por exemplo. O uso de inequalidades, como $<$ ou $>$, e quantificadores da lógica de primeira ordem também são permitidos. Isso dá muito mais expressividade para a lin-

guagem e, como consequência, mais flexibilidade para o usuário, facilitando bastante a modelagem por parte do programador.

Vejamos um exemplo para clarificar o processo de modelagem em SMT, utilizando uma pseudolinguagem para facilitar o entendimento. Nós precisamos gastar exatamente cem reais, e comprar exatamente cem animais. Cachorros custam quinze reais, gatos custam um real e ratos custam vinte e cinco centavos, e precisamos comprar pelo menos um de cada.

Para começar, nós precisamos definir as nossas variáveis. Nós podemos definir três variáveis, uma para cada tipo de animal. Como estamos modelando em SMT, não precisamos usar apenas variáveis booleanas. Os resolvedores SMT aceitam variáveis inteiras. Logo, nós criamos as variáveis

```
1  c, g, r → Int
```

representando o número de cachorros, gatos e ratos, respectivamente . Com as nossas variáveis definidas, começamos a definir as nossas restrições. A primeira delas é a de obter pelo menos um exemplar de cada animal. Utilizando três clausulas, nós modelamos essa característica do problema:

```
1  declara ( c ≥ 1 )
2  declara ( g ≥ 1 )
3  declara ( r ≥ 1 )
```

A função *declara*() aqui funciona como uma pilha de fórmulas. Toda vez que a função *declara*() é chamada, ela empilha o seu argumento em uma pilha de fórmulas que está inicialmente vazia. Na hora de avaliar a validade de uma solução para um problema, é necessário garantir que todas as fórmulas dessa pilha sejam verdadeiras para os valores propostos. Por isso, ao declararmos que c, g e r são maiores ou iguais a um, nós passamos a incluir apenas cenários em que pelo menos um de cada animal é comprado na solução final.

As outras restrições determinam que é preciso comprar exatamente cem animais, e que o valor total da compra seja de exatamente cem reais. Para modelar essa parte do problema, convertemos todos os preços em centavos, e fizemos da seguinte forma:

```
1  declara ( 1500c + 100g + 25r = 10000    )
2  declara (c + g + r = 100)
```

Após definirmos todas as nossas restrições, o nosso resolvedor poderia nos retornar uma solução como $c = 3$, $g = 41$ e $r = 56$, por exemplo. Nesse exemplo em particular, existem mais de um solução, e o resolvedor nos retornaria uma delas.

Alguns SMT *solvers*, como o Z3 [Z3], permitem não apenas que nós encontremos soluções, mas também que otimizemos os resultados. Utilizando o exemplo anterior, nós poderíamos ter feito uma chamada que não apenas retornasse uma solução viável, mas que minimizasse o número de gatos, ou maximizasse o número de cachorros, por exemplo.

2.7 Unificação

Unificação é um processo algorítmico que aborda o problema de transformar um termo em uma dada assinatura, em outro, criando como resposta uma substituição.

Um exemplo simples de unificação pode ser obtido através dos conjuntos de termos $Q_1 = (x, f(x))$ e $Q_2 = (a, f(a))$. Se quisermos que Q_1 seja equivalente a Q_2, teríamos que aplicar em Q_1 uma substituição sb, tal que $\mathsf{sb} = [x \mapsto a]$. Como x, em Q_1, é uma variável, substituímos o seu valor por a, e é fácil perceber que Q_1 se torna Q_2.

Se a substituição estudada permitir variáveis de alta ordem, ou seja, variáveis que representam funções, a unificação é dita como unificação de alta ordem, ou *high-order unification*. Caso contrário, é dito que a unificação é de primeira ordem, ou *first-order unification*. Para o escopo desse trabalho, nós vamos abordar apenas unificações de primeira ordem.

Vamos encontrar a substituição que unifica os termos $Q_1 = (f(x, a), g(y))$ e $Q2 = (f(g(b), a), x)$. Analisando a primeira parte de Q_1 e Q_2, $f(x, a)$ tem que se igualar a $f(g(b), a)$. Para isso, construímos a primeira parte da nossa substituição sb_1, $x \mapsto g(b)$. Analisando a segunda parte, precisamos que $g(y)$ corresponda a x. Perceba que aqui não podemos simplesmente afirmar que $x \mapsto g(y)$, uma vez que x já se encontra instanciado como $g(b)$. Então, nessa segunda parte da nossa substituição, temos que tornar $g(b)$, que equivale a substituição $x \mapsto g(b)$ aplicada em x, equivalente a $g(y)$. Como y é uma variável, então podemos aplicar a substituição $y \mapsto b$ sobre $g(y)$ para transformá-lo em $g(b)$, criando a solução do nosso problema de unificação como a

substituição $\mathsf{sb}_1 = [x \mapsto g(b), y \mapsto b]$.

Alguns termos não são unificáveis entre si, ou seja, não é possível encontrar um conjunto de substituições que os torne equivalentes quando aplicado em ambos os lados. Por exemplo, $Q_1 = g(x)$ e $Q_2 = f(a)$ não são unificáveis.

Agora analisemos os termos $Q = (x, f(y))$ e $Q_2 = (a, f(g(z)))$. Poderíamos unificá-los utilizando as substituições $\mathsf{sb} = [x \mapsto a, y \mapsto g(b), z \mapsto b]$. Outra opção seria $\mathsf{sb}_2 = [x \mapsto a, y \mapsto g(z)]$. Perceba que a segunda opção é completa e menor. Para dois conjuntos de termos, nós chamamos a substituição mínima que unifica os dois conjuntos de unificador mais geral (*most general unifier*), ou m.g.u.

Martelli e Montanari [Martelli e Montanari 1982] propuseram um algoritmo para encontrar o m.g.u de primeira ordem de um conjunto de termos. Dado um conjunto finito $G = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ de equações em potencial, onde s_1 precisa ser unificado com t_1, s_2 com t_2, e assim por diante. O algoritmo de Martelli e Montanari cria uma conjunto equivalente de equações da forma $\{x_1 \doteq u_1, \ldots, x_m \mapsto u_m\}$ onde $x_1, \ldots, x_m$ são variáveis distintas e $u_1, \ldots, u_m$ são termos que não contém nenhuma das variáveis x_i. Esse conjunto equivalente é a substituição necessária para unificar G. Se não existir uma solução para o problema, o algoritmo retorna $\perp$. A operação de substituir todas as variáveis x pelo termo t em G é representada por $G\{x \mapsto t\}$. Abaixo segue as regras do algoritmo mencionado:

- $G \cup \{t \doteq t\} \Rightarrow G$: Essa regra elimina qualquer termo trivial, em que um termo tem que ser unificado com ele mesmo.

- $G \cup \{f(x_0, \ldots, x_k) \doteq f(t_0, \ldots, t_k)\} \Rightarrow G \cup \{x_0 \doteq t_0, \ldots, x_k \doteq t_k\}$: Essa regra, conhecida como decomposição, remove a função mais externa e unifica os seus parâmetros, se possível.

- $G \cup \{f(x_0, \ldots, x_k) \doteq g(t_0, \ldots, t_m)\} \Rightarrow \perp$, se $g \neq f$ ou $k \neq m$: A regra de conflito de símbolos, ou apenas conflito, garante que duas funções distintas não podem ser unificadas entre si.

- $G \cup \{f(t_0, \ldots, t_k) \doteq x\} \Rightarrow G \cup \{x \doteq f(t_0, \ldots, t_k)\}$: Essa regra simplesmente inverte a ordem da unificação, para isolar as variáveis a esquerda.

- $G \cup \{x \doteq t\} \Rightarrow G\{x \mapsto t\} \cup \{x \doteq t\}$ se $x \notin Vars(t)$ e $x \in Vars(G)$: Essa regra, conhecida como eliminação, remove uma variável x de G. Se existir uma ocorrência onde $x \doteq t$, e x não for uma variável pertencente a t, o algoritmo aplica a substituição $x \mapsto t$ sobre o restante de G, eliminando qualquer outra ocorrência de x.

- $G \cup \{x \doteq f(t_0, \ldots, t_k)\} \Rightarrow \perp$ se $x \in Vars(f(t_0, \ldots, t_k))$: Essa regra é conhecida como *occurs check* ou teste de ocorrência. Perceba que, se eu precisar unificar uma variável x com um termo t que possui x como sub-termo, geraria um termo infinito como solução. É precisamente com esse tipo de cenário que essa regra lida.

Utilizando tal algoritmo, o conjunto G basta ser sistematicamente reescrito utilizando as diretrizes descritas acima, até o algoritmo retornar um $\perp$ ou até nenhuma das regras acima puder ser aplicada. Nesse último caso, o conjunto resultante já se encontra em um formato de substituição.

2.8 Maude

Lógica reescritiva (*Rewriting Logic*) [Meseguer 1992] é um sistema de lógica computacional para a descrição de sistemas concorrentes como uma teoria de reescrita, isto é, uma tupla $\langle \Sigma, \mathcal{E}, \mathcal{R} \rangle$, onde $\langle \Sigma, \mathcal{E} \rangle$ representa uma teoria equacional, onde a sintaxe e a estrutura de tipos é determinada por Σ, e as equações (condicionais ou não), definidas por $\mathcal{E}$. $\mathcal{R}$ define as regras de reescritas, que podem ou não ser condicionais.

O sistema Maude [Clavel et al. 2007] oferece uma linguagem de programação pautada sobre tal formalismo. Para o entendimento deste trabalho não é necessário um entendimento profundo do funcionamento interno do Maude, nem da definição formal perfeita da lógica reescritiva, mas alguns fundamentos ajudarão na compreensão do que foi feito.

O primeiro conceito básico que é relevante é o conceito de tipos, ou *Sorts*. Eles declaram qualquer coisa, desde um conjunto numérico até uma classe de objetos. Vejamos um exemplo:

```
1    sort Nat.
```

Nesse exemplo acima, definimos um tipo Natural, referente ao conjunto numérico homônimo. Agora, o que define os naturais, ou qualquer outro tipo, não é apenas a sua existência, mas quais operadores atuam sobre ele. Vamos definir alguns operadores:

```
1    sort Nat.
2    op 0 : -> Nat [ctor] .
3    op sucessor : Nat -> Nat [ctor] .
4    op _+_ : Nat Nat -> Nat [id: 0] .
5    vars M,N : Nat .
6    eq sucessor(M) + N = sucessor(M + N) .
```

Acima nós definimos os naturais, e definimos o operador da adição. Por motivos de simplificação, e por não envolver o escopo deste trabalho, nós modelamos os números naturais utilizando os módulos já estabelecidos do Maude, que possuem as operações aritméticas embutidas. Nós inicialmente declaramos o tipo Nat, depois estabelecemos que 0 pertence aos naturais. A seguir, nós declaramos os operadores de sucessão e adição.

O Maude também oferece suporte para regras de reescritas, que utiliza reconhecimento de padrões para alterar um determinado termo. Pode-se estabelecer uma série de regras, inclusive regras condicionais, que aplicam transições sobre os termos envolvidos.

Dado um termo, é possível que ele unifique da mesma forma com a mesma ou diferentes regras *i.e.*, mais de uma regra pode ser passível de aplicação sobre o mesmo termo. Isso se dá porque quando é realizada uma busca sobre um termo inicial, o Maude cria uma estrutura de árvore, onde cada aresta consiste na aplicação de alguma regra e cada nó é um termo resultante. Dessa forma, adicionar regras distintas que podem ser aplicadas sobre o mesmo termo é possível, mas gera não-determinismo, aumentado a árvore de busca.

Maude permite buscar por determinados estados partindo de um estado inicial especificado, utilizando busca em largura. Para isso, é necessário utilizar o comando *search*, que parte de um estado base inicial (*ground initial state*), corresponde a um determinado padrão de busca e satisfaz uma condição de busca opcional.

```
1 search [ n, m ] in M : t Seta u such that Cond .
```

Nesse comando, (i) n é um parâmetro opcional de busca, estabelecendo uma restrição para o número total de soluções; (ii) m é um parâmetro opcional que limita a profundidade máxima atingida pela busca; (iii) M é o identificador opcional do módulo onde acontece a busca; (iv) t é o termo inicial; (v) u é o padrão de busca que precisa ser satisfeito; (vi) *Seta* é um *token* que representa a forma como a busca será feita de t para u: =>* retorna uma prova consistindo em uma quantidade qualquer de reescritas, =>+ retorna uma prova que tenha pelo menos uma reescrita, =>! retorna uma prova onde os estados finais não podem mais ser reescritos por nenhuma regra; e (vii) *Cond* é uma condição opcional de busca, que precisa ser satisfeita para a solução ser aceita.

Vamos ver um exemplo para facilitar o entendimento. Vamos modelar uma máquina de fliperama, daquelas que possuem uma caixa de vidro cheia de brinquedos, com uma garra metálica controlável que tenta agarrá-los.

Para modelar uma dessas máquinas, nós precisamos pensar em quais estados são necessários. Para os brinquedos, nós precisamos de três construtores:

- o brinquedo está no chão da máquina, e não em cima de outro brinquedo.
- o brinquedo está em cima de um outro brinquedo.
- o brinquedo está livre, isto é, não possui nenhum outro brinquedo sobre ele.

Já para a garra metálica, nós precisamos de dois estados:

- a garra está segurando um brinquedo.
- a garra está vazia.

Qualquer pilha de brinquedos em uma máquina de fliperama pode ser representado através da combinação desses estados. E nós possuímos quatro transições: a garra agarra, solta, empilha e desempilha. As duas últimas transições podem parecer análogas as duas primeiras, mas para a garra agarrar ou soltar um brinquedo, este deve estar no chão da máquina, enquanto empilhar ou desempilhar um brinquedo assume que este esteja sobre ubm outro brinquedo. A implementação em Maude deve clarificar qualquer dúvida.

```
1  mod ARCANE-CRANE is
2      sorts ToyID State.
3
4      op floor : ToyID -> State [ctor] .
5      op on : ToyID ToyID -> State [ctor] .
6      op clear : ToyID -> State [ctor] .
7      op hold : ToyID -> State [ctor] .
8      op empty : -> State [ctor] .
9      op 1 : -> State [ctor] .
10     op _&_ : State State -> State [ctor assoc comm id: 1] .
11     vars X Y : ToyID .
12
13     rl[pickup] : empty & clear(X) & floor(X) => hold(X) .
14     rl[putdown] : hold(X) => empty & clear(X) & floor(X) .
15     rl[unstack] : empty & clear(X) & on(X,Y) => hold(X) & clear(Y) .
16     rl[stack] : hold(X) & clear(Y) => empty & clear(X) & on(X,Y) .
17 endm
```

Vamos visualizar uma modelagem observando as figuras 2.10 e 2.11. Na figura X pode ser representada como *"empty & floor('mothergoose) & on('teddybear, 'mothergoose) & on('soccerball, 'teddybear) & clear('soccerball) & floor('dragondude) & clear('dragondude) "*.

Já a figura 2.11 é representada por *"empty & floor('mothergoose) & clear('mothergoose) & floor('dragondude) & on('soccerball, 'dragondude) & on('teddybear, 'soccerball) & clear('teddybear) "*. O conjunto de transições que levariam o estado A para o estado B seria a regra *unstack* chamada sobre *'soccerball*, *stack* chamada sobre *'soccerball* e *'dragondude*, *unstack* chamada sobre *teddybear* e *stack* chamada sobre *'teddybear* e *'soccerball*. Se quiséssemos colocar *'mothergoose* sobre *'teddybear*, nós teríamos que chamar *pickup* sobre *'mothergoose*, e em seguida *stack* aplicada sobre *'mothergoose* e *'teddybear*.

Perceba que o estado A não é mais simples nem mais complexo que o estado B. Isso quer dizer que não existe uma direção específica para onde a reescrita vai levar, podendo alternar entre estados infinitamente. Perceba também que mais de uma regra pode ser executada ao mesmo tempo (no estado B, poderíamos aplicar *pickup* sobre *'mothergoose*, ou *unstack* sobre *'teddybear*, por exemplo). Quanto mais regras puderem ser aplicadas simultaneamente, maior será o não-determinismo do sistema, e mais extensa a árvore de busca.

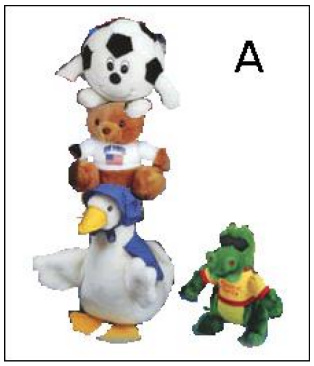

Figura 2.10: Estado A

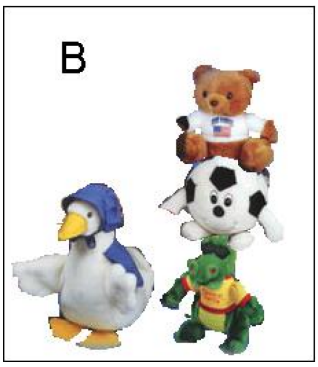

Figura 2.11: Estado B

Os exemplos dessa seção foram tirados do Maude Primer, obtido em http://maude.cs.illinois.edu/w/index.php/The_Maude_System

2.9 Trabalhos Relacionados

O artigo [Kanovich et al. 2014] discute a elaboração de um *framework* para definir protocolos cíber-físicos, assim como elabora na diferença entre avaliar protocolos temporais utilizando modelos com tempo contínuo ou modelos com tempo discreto. [Nigam, Talcott e Urquiza 2016] também aborda o elemento de protocolos temporais, como o *Distance Bounding* [Brands e Chaum 1993], propondo um modelo de intruso temporal. Ambos os artigos diferem desse trabalho por não abordarem mais profundamente a relação de equivalência observacional temporal.

O artigo [Bae e Rocha 2019] aborda a utilização de *Rewriting Modulo SMT* na modelagem de problemas concorrentes, com domínios infinitos. Mais especificamente,

ele sugere a utilização do que ele define como *guarded terms* para reduzir a explosão de estados e lidar melhor com o não-determinismo inerente ao problema, assim como implementa o algoritmo CASH como prova de conceito. Esse artigo, no entanto, não aborda a utilização de *Rewriting Modulo SMT* para lidar com problemas de equivalência observacional, e sim faz uma defesa mais geral da técnica e da adição de *guarded terms*.

O artigo [Künnemann, Esiyok e Backes 2018] segue a mesma estrutura deste trabalho, fornecendo uma definição formal de um problema, junto com a implementação de algumas provas de conceito. Os autores propõem uma definição formal de responsabilidade (*accountability*) para protocolos de segurança, de forma a possibilitar a verificação automática de tal propriedade. Além disso, eles fornecem implementações de alguns exemplos, que foram construídos utilizando a ferramenta SAPiC.

Os artigos [Chothia e Smirnov 2010; Apthorpe Noah Reisman 2017; Sommers 2014; Ying, Makrakis e Mouftah 2014] abordam ataques que utilizam a verificação de propriedades de tempo ou de tráfego para inferir sobre o comportamento de determinado usuário, assim como propõem soluções para tais vazamentos de informação. Os três diferem desse trabalho por apresentarem uma abordagem mais empírica, e soluções feitas sob medida para os problemas tratados.

Os artigos [Marzouki, Radhouani e Rajeb 2015; Bérard et al. 2016] ambos utilizam métodos formais para produzir garantias sobre sistemas críticos, nesse caso a validação de um protocolo de voto eletrônico e as propriedades de protocolos para criação de redes de robôs móveis (*mobile robots networks*).

Capítulo 3

Equivalência Observacional sobre Traços Temporais

Uma forma de ataque conhecida na literatura de segurança são ataques que utilizam o tempo como canal lateral para evitar mecanismos de defesa, inferir propriedades de sistemas, ou rastrear usuários.

Verificar formalmente se um intruso pode inferir tais propriedades difere das usuais propriedades baseadas em alcançabilidade (*reachability based properties*), como sigilo, autenticação e outras propriedades de correspondência. Na verificação de tais propriedades, procura-se um traço que apresente uma falha, *e.g.*, uma configuração em que um intruso possua um segredo. Em ataques como os descritos acima e na Seção 2.3, busca-se por comportamentos que possam distinguir dois sistemas, *e.g.*, um comportamento que possa ser observado quando se interage com um dos sistema, mas não com o outro. Isto é, verificar se os sistemas são *observacionalmente distinguíveis*. Fazer isso requer uma análise mais elaborada sobre um conjunto de traços.

Várias noções de *equivalência de traços* (*trace equivalence*) foram propostas pela comunidade de linguagens de programação, assim como pela de sistemas concorrentes [Cortier e Delaune 2009]. Uma série de propriedades, *e.g.*, anonimidade e inconectabilidade (*unlinkability*), foram reduzidas para o problema de equivalência de traços. Como a verificação de protocolos envolve domínios infinitos, a utilização de métodos simbólicos se provou essencial para o sucesso dessas abordagens.

Esse capítulo abordará a análise desenvolvida a respeito da equivalência de traços

envolvendo elementos temporais. Para isso nós fizemos:

- **Equivalência Simbólica de Traços Temporais:** A fim de raciocinar sobre conjuntos de traços, é necessário uma definição de equivalência temporal sobre instâncias de protocolos que utilizam o tempo como recurso [Nigam, Talcott e Urquiza 2016]. Informações temporais, *e.g.*, duração de processamento e tempo de retorno, são tratadas simbolicamente e podem ser especificadas na forma de restrições temporais que relacionam múltiplos símbolos entre si, *e.g.*, $\mathtt{tt}_1 \geq \mathtt{tt}_2 + 10$

- **Resolvedores SMT para provar a Equivalência de Traços Temporais:** Resolvedores SMT (*SMT solvers*) são usados de duas formas distintas. Nós especificamos a semântica operacional dos protocolos temporais utilizando *Rewriting Modulo SMT* [Rocha 2012], reduzindo consideravelmente o espaço de busca necessário para enumerar todos os traços.

  Ao invés de instanciar símbolos temporais com valores concretos, em *Rewriting Modulo SMT* a configuração de um sistema é simbólica e, portanto, pode representar uma quantidade potencialmente ilimitada de configurações concretas. Reescrever uma configuração simbólica só é permitido se o conjunto de restrições temporais do estado resultante for satisfatível. Resolvedores SMT são utilizados para realizar essa checagem.

  A segunda aplicação de SMT é na prova de equivalência de traços temporais, mais especificamente para verificar se o tempo das observações pode ser equivalente. Essa avaliação envolve checar a satisfatibilidade de fórmulas do tipo $\exists\forall$ [Dutertre 2015].

  A utilização de resolvedores SMT de uso geral significa que a nossa implementação se beneficia de quaisquer melhoramentos feitos a essas ferramentas. Além disso, como existem grandes comunidades que utilizam resolvedores SMT para diversos fins, nós temos confiança na corretude das implementações.

- **Implementação:** Utilizando o suporte que o Maude [Clavel et al. 2007] oferece para *Rewriting Modulo SMT* e usando os resolvedores SMT CVC4 [Barrett et al. 2011] ou Yices [Dutertre 2015], nós implementamos o maquinário necessário para

enumerar traços simbólicos. No entanto, como checar pela satisfatibilidade de fórmulas do tipo $\exists\forall$ [Dutertre 2015] não é suportado pelo Maude, nós integramos o maquinário em Maude com o resolvedor SMT Yices [Dutertre 2015]. Nós realizamos alguns experimentos como prova de conceito demonstrando a viabilidade da nossa abordagem.

Esse trabalho só considera o caso de sessões de protocolo limitadas (*bounded protocol sessions*). O caso de sessões ilimitadas é deixado como trabalho futuro.

3.1 Exemplos

Além do já mencionado ataque aos passaportes eletrônicos [Chothia e Smirnov 2010] da Seção 2.3, nós discutiremos outros exemplos que motivaram este trabalho. Com isso, buscamos ilustrar como os intrusos podem explorar o tempo como canal lateral e, assim, realizar ataques contra protocolos específicos.

Nosso primeiro exemplo é tirado de [Ho et al. 2014]. *Red pills* são pedaços de código que tem como objetivo detectar se eles estão sendo executados dentro de uma máquina virtual ou emulador de CPU. *Honeypots*, por sua vez, são aplicações que simulam usuários reais, com o intuito de atrair comportamentos maliciosos em um ambiente controlado e, assim, descobrir possíveis falhas de segurança. Vários sistemas que avaliam se páginas da *web* são ou não seguras consistem na utilização de *honeypots*, *crawlers* que percorrem a rede em busca de comportamentos maliciosos. Como *honeypots* tentando atrair atacantes normalmente são executados em máquinas virtuais, conseguir determinar se um sistema está sendo executado ou não por uma máquina virtual oferece ao atacante uma oportunidade de evitar *honeypots* [Ho et al. 2014]. O sistema que está sendo executado em uma máquina virtual ou concreta segue exatamente o mesmo protocolo.

Quando uma aplicação se conecta ao servidor malicioso, o servidor inicialmente manda uma requisição de calibragem (*baseline request*) e uma requisição diferencial (*differential request*). O tempo de resposta da requisição de calibragem é o mesmo, independente de estar sendo ou não executada dentro de uma máquina virtual. Ela é utilizada para determinar uma base comparativa, já que existem outros fatores que

podem atrasar o tempo de resposta, como *hardware* antigo ou outras aplicações sendo executadas em outras abas, *e.g.*, um vídeo sendo reproduzido. O tempo de resposta da requisição diferencial é mais longa quando executada em uma máquina virtual. Quando não se está levando em conta a medição do tempo, o conjunto de traços dessa troca de mensagens é igual, independente do ambiente de execução. No entanto, se nós levarmos em consideração o tempo de resposta para ambas as requisições, os traços temporais da aplicação executada em uma máquina virtual distam daqueles executados em uma máquina concreta.

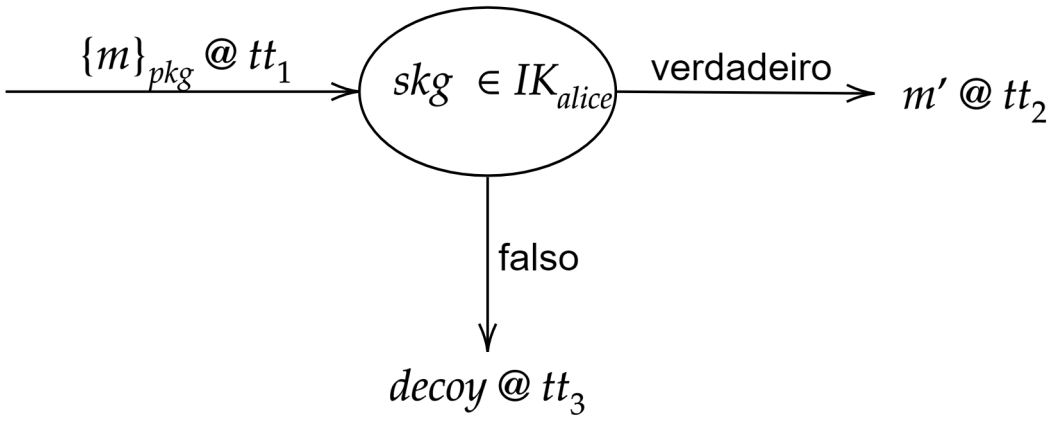

Figura 3.1: Diagrama de fluxo do protocolo de Abadi e Fournet. A figura mostra quando um participante *Alice* recebe uma mensagem m encriptada com com a chave pública do grupo no tempo tt_1. Se *Alice* pertencer ao grupo, ela prepara a mensagem de resposta m' e a envia no tempo tt_2, ou envia a mensagem *decoy* no tempo tt_3 caso contrário. O ataque consiste na diferença entre tt_2 e tt_3.

Abadi e Fournet [Abadi e Fournet 2004] propuseram um protocolo para que membros de grupos anônimos possam se comunicar entre si sem revelar sua afiliação. Um membro de um grupo transmite uma mensagem, m, encriptada com a chave pública de tal grupo. Toda vez que um membro do grupo recebe essa mensagem, ele é capaz de decriptar a mensagem e avaliar o seu conteúdo, determinando assim se o transmissor pertence ao grupo e se a mensagem é endereçada para si. Nesse caso, o receptor transmite uma mensagem m'.

Sempre que um participante que não pertence ao grupo receber uma mensagem m, ele não simplesmente ignora a mensagem, mas envia uma mensagem chamariz (*decoy*)

com o mesmo formato de m'. Dessa forma, outros participantes e usuários externos não conseguiriam determinar se dois participante pertencem ou não ao mesmo grupo.

No entanto, como mostrado em [Corin et al. 2004], ao medir o tempo de resposta das mensagens, um intruso poderia determinar se dois participantes pertencem ao mesmo grupo. Isso se dá porque decriptar e gerar uma mensagem de resposta toma mais tempo do que simplesmente enviar uma mensagem chamariz.

3.2 Protocolos temporais

Nós utilizamos uma linguagem básica de mensagens avaliada como termos, e que contém os operadores criptográficos usuais, como encriptação, *nonces*, e concatenação, ampliada com restrições temporais. Traços e equivalência observacional envolvendo termos mais complexos do que os nossos já foram objetos de estudo de uma miríade de trabalhos. Aqui, nós introduzimos o suficiente para escrever os nossos próprios exemplos. Como a especificação de aspectos temporais é ortogonal a linguagem de termos, extensões aos termos como funções de *hash*, MAC, chaves criptográficas geradas durante a execução do protocolo (*fresh keys*), etc., podem ser adicionadas sem afetar os nossos principais resultados.

A linguagem de termos é definida pela seguinte gramática. Nós assumimos um dado conjunto contável para constantes de texto, $\mathcal{T}$, nomes de participantes, $\mathcal{P}$, *nonces*, $\mathcal{N}$, chaves simétricas, $\mathcal{K}$, símbolos, Syms, e variáveis, $\mathcal{V}$, onde $\mathcal{T}$, $\mathcal{P}$, $\mathcal{N}$, $\mathcal{K}$, Syms e $\mathcal{V}$ são

disjuntos. Abaixo v_p representa uma variável do tipo participante.

Constantes Básicas:

$\mathsf{c} :=$	$\mathsf{t} \in \mathcal{T}$	Constantes de Texto
	$\mid \mathsf{p} \in \mathcal{P}$	Nome dos participantes
	$\mid \mathsf{n} \in \mathcal{N}$	*Nonces*

Chaves:

$\mathsf{k} :=$	$\mathsf{symk} \in \mathcal{K}$	Chaves Simétricas
	$\mid \mathsf{pk(p)} \mid \mathsf{pk}(\mathsf{v}_p)$	Chave pública de p
	$\mid \mathsf{sk(p)} \mid \mathsf{sk}(\mathsf{v}_p)$	Chave privada de p

Símbolos:

$\mathsf{sym} :=$	$\mid \mathsf{sym} \in \mathsf{Syms}$	Símbolo

Termos:

$\mathsf{m} :=$	c	Constantes básicas
	$\mid \mathsf{k}$	Chaves
	$\mid \mathsf{v} \in \mathcal{V}$	Variáveis
	$\mid \mathsf{sym} \in \mathsf{Syms}$	Símbolos
	$\mid \mathsf{e(m, k)}$	Termos encriptados
	$\mid \langle \mathsf{m}_1, \ldots, \mathsf{m}_n \rangle$	Tuplas

Um termo é considerado um *termo base* (*ground term*) se não possuir nenhuma ocorrência de variáveis ou símbolos. Um termo *simbólico* pode conter ocorrências de símbolos, mas não de variáveis. $\mathsf{ms}, \mathsf{ms}_1, \mathsf{ms}_2, \ldots$ representam termos simbólicos.

É possível adicionar outras construções criptográficas, como *hashes* ou assinaturas, mas para manter as coisas mais simples e compreensíveis, nós incluímos apenas a encriptação. Como ambas as construções podem ser definidas utilizando encriptação, isto não é um limitante. Finalmente, é fácil estender esses resultados de forma a incluir chaves recém-criadas (*fresh keys*). Esse tipo de chave é tratado da mesma forma que os *nonces*, mas para manter a simplicidade, nós não as incluímos.

Assuma uma assinatura temporal, Ξ, contendo um conjunto de números $\mathsf{r}_1, \mathsf{r}_2, \ldots$, um conjunto de variáveis temporais $\mathsf{tt}_1, \mathsf{tt}_2, \ldots$, incluindo uma variável especial intitulada cur, e um conjunto pré-definido de símbolos de funções, incluindo $+, -, \times, /, \mathsf{floor}, \mathsf{ceiling}$.

Expressões temporais são construídas indutivamente dos números e variáveis, aplicando símbolos de funções sobre expressões temporais. Por exemplo, $\mathsf{ceiling}((2 + \mathsf{tt} + \mathsf{cur})/10)$ é uma expressão temporal. Os símbolos $\mathsf{tr}_1, \mathsf{tr}_2, \ldots$ representam expressões temporais. A variável temporal cur é uma palavra chave na nossa linguagem de especificação, que denota o tempo global atual. Nós não restringimos o conjunto de símbolos e números em Ξ, mas, na prática, permitimos apenas que assinaturas suportadas por

um resolvedor SMT possam ser usadas. Nós assumimos que as expressões temporais são disjuntas dos termos referentes as mensagens.

Definição 3.2.1 (**Restrições Simbólicas de Tempo**)**.** Considere Ξ uma assinatura temporal. O conjunto de restrições simbólicas de tempo é construído utilizando as expressões temporais da seguinte forma: Considere $\mathsf{tr}_1, \mathsf{tr}_2$ como expressões temporais, então $\mathsf{tr}_1 = \mathsf{tr}_2, \quad \mathsf{tr}_1 \geq \mathsf{tr}_2 \quad \mathsf{tr}_1 > \mathsf{tr}_2, \quad \mathsf{tr}_1 < \mathsf{tr}_2,$ and $\mathsf{tr}_1 \leq \mathsf{tr}_2$
são *restrições simbólicas de tempo* (*symbolic time constraints*)

Por exemplo, $\mathsf{cur} + 10 < \mathsf{floor}(\mathsf{tt} - 5)$ é uma restrição simbólica de tempo. $\mathsf{tc}, \mathsf{tc}_1, \mathsf{tc}_2, \ldots$ representam as restrições de tempo.

Intuitivamente, dado um conjunto de restrições de tempo $\mathcal{TC}$, cada um dos seus modelos com instanciações concretas para as variáveis temporais corresponde a um cenário em particular. Isso significa que um único conjunto de restrições de tempo denota uma quantidade possivelmente infinita de cenários concretos. Por exemplo, o conjunto de restrições $\{\mathsf{tt}_1 \leq 2, \mathsf{tt}_2 \geq 1 + \mathsf{tt}_1\}$ possui uma quantidade infinita de possíveis modelos que a satisfaça, *e.g.*, $[\mathsf{tt}_1 \mapsto 1.9, \mathsf{tt}_2 \mapsto 3.1415]$.

Resolvedores SMT, como o CVC4 [Barrett et al. 2011] e o Yices [Dutertre 2015] conseguem checar a satisfatibilidade de um conjunto de restrições de tempo.

3.2.1 Linguagem para especificação de Protocolos Temporais

A linguagem utilizada para especificar um protocolo criptográfico temporal, introduzida no artigo [Nigam, Talcott e Urquiza 2016], possui as construções padrão, como a criação de novos *nonces*, envio e recebimento de mensagens, e uma cláusula "*if-then-else*", cada uma delas possuindo as suas próprias restrições de tempo.

Definição 3.2.2 (**Protocolos Temporais**)**.** Um protocolo temporal consiste em um conjunto de funções protocolares específicas (*timed protocol roles*). O conjunto de Funções Protocolares, $\mathsf{pl} \in \mathcal{TL}$ é gerado pela seguinte gramática:

nil	Protocolo vazio
$\mid (\mathsf{new\ v} \ \# \ \mathsf{tc})$	Nova Constante
$\mid (+\mathsf{m} \ \# \ \mathsf{tc})$	Envio de mensagem temporal
$\mid (-\mathsf{m} \ \# \ \mathsf{tc})$	Recebimento de mensagem temporal
$\mid (\mathsf{if}\ (\mathsf{m}_1 := \mathsf{m}_2) \ \# \ \mathsf{tc\ then\ pl}_1\ \mathsf{else\ pl}_2)$	Condicional Temporal

Intuitivamente, new gera um novo valor que vai ser vinculado a variável v, $(+\mathsf{m} \ \# \ \mathsf{tc})$ representa o envio do termo m e $(-\mathsf{m} \ \# \ \mathsf{tc})$ denota o recebimento de um termo que corresponda ao termo m. Para o termo condicional $(\mathsf{if}\ \mathsf{m}_1 := \mathsf{m}_2 \ \# \ \mathsf{tc\ then\ pl}_1\ \mathsf{else\ pl}_2)$, nós assumimos que m_1 é um termo base no momento da sua avaliação. Então se m_1 puder corresponder a m_2, isto é, se houver uma possível instanciação das variáveis de m_2 de forma que o termo resultante seja m_1, o protocolo executa pl_1, ou executa pl_2 caso contrário, onde pl_1 e pl_2 são também Funções Protocolares. Além disso, as variáveis em m_2 são instanciadas observando as substituições correspondentes em pl_1. Nós também assumimos que pl_2 não contém nenhuma variável em m_2. Isso se dá porque a vinculação dessas variáveis com seus respectivos termos concretos acontece apenas quando a condição é avaliada como verdadeira. Por último, um comando só é aplicado se a restrição de tempo associada tc for satisfatível.

Nós omitimos as restrições de tempo todas as vezes que tc for uma tautologia. Se limitarmos nossas restrições de tempo apenas a tautologias ($1 = 1$, por exemplo), a linguagem de protocolos temporais pode ser considerada como uma das usuais linguagens de especificação de protocolos de segurança. Os próximos dois exemplos especificam essa característica, suprimindo restrições de tempo trivialmente verdadeiras.

Exemplo 3.2.1. O protocolo de Needham-Schroeder [Needham e Schroeder 1978] é especificado a seguir, onde X, Y, W, Z são variáveis:

$$Alice(Z) := (\mathsf{new}\ N_a), (+\mathsf{e}(\langle N_a, alice\rangle, \mathsf{pk}(Z))), (-\mathsf{e}(\langle N_a, Y\rangle, \mathsf{pk}(alice)\})), (+\mathsf{e}(Y, \mathsf{pk}(Z)))$$
$$Bob := (-\mathsf{e}(\langle X, W\rangle, \mathsf{pk}(bob))), (\mathsf{new}\ N_b), (+\mathsf{e}(\langle X, N_b\rangle, \mathsf{pk}(W)\})), (-\mathsf{e}(N_b, \mathsf{pk}(bob)))$$

Perceba que nós definimos os diferentes participantes. Nesse caso, *Alice* e *Bob* são as funções protocolares necessárias para especificar Needham-Schroeder, consistindo em um iniciador do protocolo (*Alice*) , e um receptor (*Bob*). As mensagens $\mathsf{new}\ N_b$ e

$\mathsf{new}\ N_a$ sinalizam que aquele determinado participante vai gerar uma nova constante que, nesse caso em particular, são os *nonces* trocados durante a sessão.

A construção condicional " *if then else* " e a identificação de padrões nos permite especificar protocolos com caminhos ramificados, como ilustrado no exemplo a seguir:

Exemplo 3.2.2. Considere a seguinte função protocolar, que consiste em uma modificação na função de *Alice* no protocolo de Needham-Schroeder (exemplo 3.2.1):

$$\begin{aligned} Alice(Z) := \quad & (\mathsf{new}\ N_a), (+\mathsf{e}(\langle N_a, alice\rangle, \mathsf{pk}(Z))), (-\mathsf{v}), \\ & \mathsf{if}\ \mathsf{v} := \mathsf{e}(\langle N_a, Y\rangle, \mathsf{pk}(alice)\})\ \mathsf{then}\ (+\mathsf{e}(Y, \mathsf{pk}(Z)))\ \mathsf{else}\ (+error) \end{aligned}$$

Aqui, *Alice* checa se a mensagem recebida v possui o formato esperado antes de prosseguir. Se v não tiver esse formato, então *Alice* envia uma mensagem de erro.

Restrições de tempo podem ser usadas para especificar aspectos temporais de alguns protocolos de segurança. O exemplo a seguir ilustra como as restrições de tempo são capazes de modelar os elementos temporais de uma mensagem do tipo *ping-pong*, que pode ser utilizada para checar a latência de um determinado canal de comunicação com um outro participante. Estratégias similares podem ser usadas para especificar protocolos delimitadores de distância (*Distance Bounding Protocols*), *i.e.*, protocolos cujo objetivo é impedir que participantes que estejam além de uma determinada distância sejam autenticados [Brands e Chaum 1993] .

Exemplo 3.2.3. O exemplo a seguir especifica um participante que envia uma mensagem e só aceita a resposta se ela chegar dentro de quatro unidades de tempo:

$$(\mathsf{new}\ \mathsf{v}), (+\mathsf{v}\ \#\ \mathsf{tt} = \mathsf{cur}), (-\mathsf{v}\ \#\ \mathsf{cur} \leq \mathsf{tt} + 4)$$

O verificador cria uma nova constante v e a envia para o outro participante, lembrando o tempo global atual ao atribuí-lo a variável tt. Ele apenas conclui se a resposta for recebida dentro de quatro unidades de tempo.

O próximo exemplo ilustra como as restrições de tempo podem ser usadas para especificar decisões tomadas pelos participantes com base em aspectos temporais.

Exemplo 3.2.4. A função protocolar a seguir modifica o exemplo 3.2.3:

$$\begin{aligned} & (\mathsf{new}\ \mathsf{v}), (+\mathsf{v}\ \#\ \mathsf{tt} = \mathsf{cur}), (-\mathsf{v}_2\ \#\ \mathsf{tt}' = \mathsf{cur}), \\ & \quad (\mathsf{if}\ (\mathsf{v} := \mathsf{v}_2\ )\#\ \mathsf{tt} + 4 \geq \mathsf{tt}'\ \mathsf{then}\ (+ok\ \#\ \mathsf{cur} < \mathsf{tt}' + 2)\ \mathsf{else}\ (+ko\ \#\ \mathsf{cur} < \mathsf{tt}' + 2)) \end{aligned}$$

Aqui, o participante envia um *nonce* v e recebe uma resposta v_2. O protocolo então checa se a mensagem v_2 coincide com v , e se ela foi recebida em até quatro unidades de tempo. Se sim, ele responde com *ok*, ou com *ko* caso contrário. Além disso, a resposta é enviada em, no máximo, duas unidades de tempo após o recebimento de v_2 .

Como ilustrado pelos exemplos abaixo, descritos na Seção 3.1, as restrições de tempo também podem ser usadas para especificar a duração de determinadas operações, como checar se uma determinada mensagem possui o formato esperado. Na prática, a duração dessas operações podem ser medidas empiricamente para obter uma análise mais acurada do protocolo, como foi feito em [Chothia e Smirnov 2010].

Exemplo 3.2.5 (Passaporte). Considere a seguinte função protocolar, extraída do artigo [Nigam, Talcott e Urquiza 2016], que é a função do passaporte usado para identificação.

$$
\begin{array}{l}
(\mathsf{new}\ \mathsf{v}), (+\mathsf{v}), (-\langle \mathsf{v}_{enc}, \mathsf{v}_{mac}\rangle\ \#\ \mathsf{tt}_0 = \mathsf{cur}) \\
\mathsf{if}\ (\mathsf{v}_{mac} := \mathsf{e}(\mathsf{v}_{enc}, \mathsf{k}_M))\ \#\ \mathsf{tt}_1 = \mathsf{tt}_0 + \mathsf{r}_{mac}\ \mathsf{then} \\
\quad \mathsf{if}\ (\mathsf{v}_{enc} := \mathsf{e}(\mathsf{v}, \mathsf{k}_E))\ \#\ \mathsf{tt}_2 = \mathsf{tt}_1 + \mathsf{r}_{enc})\ \mathsf{then}\ (+done\ \#\ \mathsf{cur} = \mathsf{tt}_2) \\
\quad \mathsf{else}\ (+error\ \#\ \mathsf{cur} = \mathsf{tt}_2) \\
\mathsf{else}\ (+error\ \#\ \mathsf{cur} = \mathsf{tt}_1)
\end{array}
$$

Essa função cria uma nova constante v e a envia. Então ela espera um par de mensagens, v_{mac} e v_{enc}, que são recebidas no tempo tt_0 . Ela então checa se a primeira parte da mensagem, v_{mac}, possui o formato $\mathsf{e}(\mathsf{v}_{enc}, \mathsf{k}_M)$, *i.e.*, possui o MAC correto. Essa operação demora r_{mac} unidades de tempo. A variável de tempo tt_1 é igual a $\mathsf{tt}_0 + \mathsf{r}_{mac}$, *i.e.*, o tempo em que a mensagem foi recebida mais o tempo de duração da validação do MAC. Se o MAC não corresponder ao esperado, uma mensagem de erro (*error*) é enviada no tempo tt_1. Caso contrário, se o componente v_{mac} corresponder as expectativas, a função protocolar checa se o segundo componente v_{enc} é uma encriptação da forma $\mathsf{e}(\mathsf{v}, \mathsf{k}_E)$, que demora r_{enc} unidades de tempo para ser feita. Se a checagem de v_{enc} coincidir com o esperado, envia-se a mensagem de conclusão (*done*), caso contrário envia-se uma mensagem de erro (*error*), ambas enviadas no tempo $\mathsf{tt}_1 + \mathsf{r}_{enc}$. Levando-se em consideração as observações feitas por [Chothia e Smirnov 2010], então $\mathsf{r}_{enc} > \mathsf{r}_{mac}$. Note que o protocolo ilustrado nesse exemplo consiste na versão do BAC que envia a mesma mensagem de erro independente da falha.

Perceba também que ao invés de atribuirmos valores concretos para r_{mac} e r_{enc} durante a verificação do MAC e da encriptação, respectivamente, nós poderíamos ter atribuído intervalos para essas operações. Por exemplo, as restrições de tempo $\mathsf{tt}_0 + \mathsf{r}_{mac-} \leq \mathsf{tt}_1 \leq \mathsf{tt}_0 + \mathsf{r}_{mac+}$ representam que para verificar se o MAC está correto, demora-se um quantidade de tempo que se encontra entre r_{mac-} e r_{mac+} .

Exemplo 3.2.6 (*Redpills*). Nós abstraímos a parte do ataque que consiste no envio da mensagem de calibragem, *e.g.*, a mensagem que estabelece a conexão com o servidor, e a parte que envia as mensagens diferenciais. Nós assumimos que demora dBase para completar a troca das mensagens de calibragem.

$$(-(\mathsf{baseline\_req}) \ \# \ \mathsf{tt}_0 = \mathsf{cur}), (+(\mathsf{baseline\_done}) \ \# \ \mathsf{cur} = \mathsf{tt}_0 + \mathsf{dBase}),$$
$$(-(\mathsf{diff\_req}) \ \# \ \mathsf{tt}_1 = \mathsf{cur}), (+(\mathsf{diff\_done}) \ \# \ \mathsf{cur} = \mathsf{tt}_1 + \mathsf{dAppl})$$

Depois disso, a parte do protocolo que depende do ambiente da aplicação começa. Nós abstraímos essa parte utilizando as mensagens $\mathsf{diff\_req}$ e $\mathsf{diff\_done}$. Se a aplicação estiver sendo executada em uma máquina virtual, então dAppl toma $\mathsf{dVirtual}$ unidades de tempo ; caso contrário, dAppl toma dReal, onde `dVirtual > dReal`.

O intruso consegue discernir quando uma aplicação está sendo executada em uma máquina virtual ou não medindo o tempo que demora para completar uma troca de mensagens $\mathsf{diff\_req}$ e $\mathsf{diff\_done}$.

Exemplo 3.2.7 (Protocolo de grupo anônimo). Nós especificamos uma versão simplificada do protocolo de grupo proposto por Abadi e Fournet para autenticação privada [Abadi e Fournet 2004]. Toda vez que uma mensagem difundida (*broadcasted*) é recebida por algum agente, ele verifica se ela foi encriptada utilizando a sua chave pública KB. Se a verificação for positiva, o agente aprende a chave do grupo k_A e responde com uma mensagem encriptada com a chave do grupo. Caso contrário, o agente envia uma mensagem chamariz encriptada com uma chave recém-criada k_ν, conhecida apenas pelo agente.

$$-(\mathsf{v}), \mathsf{if} \ \mathsf{v} := \langle \mathsf{hello}, \mathsf{e}(\{\mathsf{hello}, \mathsf{v}_n, \mathsf{k}_A\}, \mathsf{k}_G)\rangle \ \ \# \ \mathsf{tt}_1 = \ \mathsf{cur} + \mathsf{dEnc} \ \mathsf{then}$$
$$\mathsf{if} \ (\mathsf{k}_G := KB \ \# \ \mathsf{tt}_2 = \mathsf{tt}_1 + \mathsf{dChk}) \ \mathsf{then} \ + (\langle \mathsf{ack}, \mathsf{e}(\mathsf{rsp}, \mathsf{k}_A)\rangle) \ \# \ \mathsf{cur} = \mathsf{tt}_1 + \mathsf{dCrt}$$
$$\quad \mathsf{else} \ (\mathsf{new} \ \mathsf{k}_\nu), +(\langle \mathsf{ack}, \mathsf{e}(\mathsf{decoy}, \mathsf{k}_\nu)\rangle) \ \# \ \mathsf{cur} = \mathsf{tt}_1$$
$$\mathsf{else} \ (\mathsf{new} \ \mathsf{k}_\nu), +(\langle \mathsf{ack}, \mathsf{e}(\mathsf{decoy}, \mathsf{k}_\nu)\rangle) \ \# \ \mathsf{cur} = \mathsf{tt}_1$$

Aqui, $\mathsf{dEnc}, \mathsf{dChk}$ e dCrt são números que especificam o tempo necessário para, respectivamente, decriptar a mensagem que foi recebida, verificar se a chave recebida é a chave do grupo, e criar e enviar uma mensagem de resposta.

3.3 Semântica Operacional para Protocolos temporais

Essa seção formaliza a semântica operacional para configurações com um número fixo de instâncias de funções protocolares . A semântica operacional utiliza termos simbólicos, onde ao invés de instanciar variáveis com termos concretos, utiliza-se termos simbólicos, onde cada símbolo representa um conjunto potencialmente infinito de termos base que satisfaça as restrições adequadas. Essa ideia possibilitou a verificação de protocolos de segurança, que possuem um espaço de busca infinito quando utiliza-se termos base, mas finito quando utiliza-se termos simbólicos [Basin e Mödersheim 2004; Cortier e Delaune 2009; Cheval e Cortier 2015] .

3.3.1 Restrições de Termos Simbólicos

Nós usaremos dois tipos de substituições. *Substituições de variáveis*, escritas como $\mathsf{sb}, \mathsf{sb}_1, \mathsf{sb}_2, \ldots$, que mapeiam variáveis para termos simbólicos $\mathsf{sb} = [\mathsf{v}_1 \mapsto \mathsf{ms}_1, \mathsf{v}_2 \mapsto \mathsf{ms}_2, \ldots, \mathsf{v}_n \mapsto \mathsf{ms}_n]$. *Substituições de símbolos*, escritas $\mathsf{ssb}, \mathsf{ssb}_1, \mathsf{ssb}_2, \ldots$, que mapeiam símbolos para termos simbólicos $\mathsf{ssb} = [\mathsf{sym}_1 \mapsto \mathsf{ms}_1, \ldots, \mathsf{sym}_n \mapsto \mathsf{ms}_n]$.

A semântica operacional utiliza duas formas de restrições: as restrições de derivabilidade (*derivability constraints*) e as restrições de comparação (*comparison constraints*).

Restrições de derivabilidade são construídas sobre conjuntos mínimos (*minimal sets*), definidos abaixo :

Definição 3.3.1. Um conjunto de termos simbólicos $\mathcal{S}$ é mínimo se ele satisfizer as seguintes condições :

- $\mathcal{S}$ contém todas as constantes inferíveis, como nome dos participantes e chaves públicas;

- $\mathcal{S}$ não contém tuplas;
- $\mathsf{e}(\mathsf{ms}, \mathsf{k}) \in \mathcal{S}$ se e somente se $\mathsf{k}^{-1} \notin \mathcal{S}$ onde k^{-1} é a chave inversa de k;

Formalmente, os termos simbólicos deriváveis de $\mathcal{S}$ são o menor conjunto $\mathcal{R}$ definido indutivamente a seguir:

- Se $\mathsf{ms} \in \mathcal{S}$ então $\mathsf{ms} \in \mathcal{R}$;
- Se $\mathsf{k} \in \mathcal{R}$ e $\mathsf{ms} \in \mathcal{R}$ então $\mathsf{e}(\mathsf{ms}, \mathsf{k}) \in \mathcal{R}$;
- se $\mathsf{ms}', \mathsf{ms}'_1, \mathsf{ms}'_2, \ldots, \mathsf{ms}'_m \in \mathcal{R}$ então $\langle \mathsf{ms}', \mathsf{ms}'_1, \mathsf{ms}'_2, \ldots, \mathsf{ms}'_m \rangle \in \mathcal{R}$;

A partir de dois conjuntos mínimos, $\mathcal{S}_1$ e $\mathcal{S}_2$, nós podemos construir um conjunto mínimo $\mathcal{S}$, que representa a união de $\mathcal{S}_1$ e $\mathcal{S}_2$, aplicando as seguintes operações até que não hajam mais operações possíveis, começando por $\mathcal{S}_0 = \mathcal{S}_1 \cup \mathcal{S}_2$:

- $\mathsf{e}(\mathsf{m}, \mathsf{k}) \in \mathcal{S}_i$ e $\mathsf{k}^{-1} \in \mathcal{S}_i$, então $\mathcal{S}_{i+1} = \mathcal{S}_i \cup \{m\}$;
- $\mathsf{e}(\mathsf{m}, \mathsf{k}), \mathsf{k}, \mathsf{k}^{-1} \in \mathcal{S}_i$ então $\mathcal{S}_{i+1} = \mathcal{S}_i \setminus \{\mathsf{e}(\mathsf{m}, \mathsf{k})\} \cup \{\mathsf{m}\}$;
- $\langle \mathsf{m}_1, \ldots, \mathsf{m}_n \rangle \in \mathcal{S}_i$, então $\mathcal{S}_{i+1} = \mathcal{S}_i \setminus \{\langle \mathsf{m}_1, \ldots, \mathsf{m}_n \rangle\} \cup \{\mathsf{m}_1, \ldots, \mathsf{m}_n\}$;

Por exemplo, dado os seguintes conjuntos mínimos:

$$\mathcal{S}_1 = \{\mathsf{sym}_k, \mathsf{pk}\}, \mathcal{S}_2 = \{\mathsf{e}(\langle \mathsf{e}(\mathsf{t}, \mathsf{sk}), \mathsf{t} \rangle, \mathsf{sym}_k)\}$$

O conjunto mínimo obtido da união de $\mathcal{S}_1$ e $\mathcal{S}_2$ é :

$$\mathcal{S} = \{\mathsf{sym}_k, \mathsf{pk}, \mathsf{t}, \mathsf{e}(\mathsf{t}, \mathsf{sk})\}$$

Definição 3.3.2. Uma restrição de derivabilidade possui a forma $\mathsf{dc}(\mathsf{sym}, \mathcal{S})$ onde $\mathcal{S}$ é um conjunto mínimo e sym é um símbolo. Essa restrição denota que sym pode ser qualquer termo (simbólico) derivável de $\mathcal{S}$.

Por exemplo, a restrição de derivabilidade

$$\mathsf{dc}(\mathsf{sym}, \{alice, \mathsf{n}_1, \mathsf{e}(\mathsf{sym}_2, \mathsf{sk}(alice)), \mathsf{pk}(bob)\})$$

especifica que sym pode ser instanciado por, *e.g.*, $\langle alice, \mathsf{e}(\mathsf{sym}_2, \mathsf{sk}(alice))\rangle$, $\langle alice, \mathsf{n}_1\rangle$, $\mathsf{e}(alice, \mathsf{pk}(bob))$, $\mathsf{e}(\langle alice, bob\rangle, \mathsf{pk}(bob))$ e assim por diante.

Para melhorar a legibilidade (e também para representar melhor a nossa implementação), nós vamos omitir de qualquer restrição $\mathsf{dc}(\mathsf{sym}, \mathcal{S})$ os termos inferíveis (*guessable terms*). Por exemplo, nós escrevemos a restrição de derivabilidade acima apenas como $\mathsf{dc}(\mathsf{sym}, \{\mathsf{n}_1, \mathsf{e}(\mathsf{sym}_2, \mathsf{sk}(alice))\})$ já que *alice* e $\mathsf{pk}(bob)$ são ambos inferíveis.

Perceba que qualquer $\mathsf{dc}(\mathsf{sym}, \mathcal{S})$ representa uma quantidade infinita de termos simbólicos devido ao fecho das tuplas (*tupling closure*). Nós abusaremos da notação e usaremos $\mathsf{ms} \in \mathsf{dc}(\mathsf{sym}, \mathcal{S})$ para representar que o termo simbólico ms está contido no conjunto de termos que podem ser derivados de $\mathcal{S}$. Além disso, nós assumimos que para um conjunto qualquer de restrições de derivabilidade $\mathcal{DC}$, existe no máximo uma restrição de derivabilidade para um dado sym, *i.e.*, se $\mathsf{dc}(\mathsf{sym}, \mathcal{S}_1), \mathsf{dc}(\mathsf{sym}, \mathcal{S}_2) \in \mathcal{DC}$, então $\mathcal{S}_1 = \mathcal{S}_2$. Nós utilizamos a notação $\mathsf{dom}(\mathcal{DC}) = \{\mathsf{sym} \mid \mathsf{dc}(\mathsf{sym}, \mathcal{S}) \in \mathcal{DC}\}$. Nós escrevemos $\mathcal{DC}(\mathsf{sym})$ para representar a restrição de derivabilidade para sym contida em $\mathcal{DC}$, se ela existir. Além disso, nós escrevemos $\mathsf{ms} \in \mathcal{DC}(\mathsf{sym})$ se ms puder ser derivado de $\mathcal{DC}(\mathsf{sym})$.

Nós precisaremos determinar se um dado termo simbólico ms' pode representar um outro termo simbólico ms, escrito $\mathsf{ms} \in \mathcal{DC}(\mathsf{ms}')$. Por cima, isso requer determinar se ms pode ser obtido através de substituições sucessivas de um símbolo sym, presente em ms', por uma de suas possíveis instanciações de acordo com $\mathsf{dc}(\mathsf{sym}, \mathcal{S}) \in \mathcal{DC}$. Por exemplo, assumindo que $\mathcal{DC}$ contenha $\mathsf{dc}(\mathsf{sym}_1, \mathsf{t}_1)$, $\mathsf{dc}(\mathsf{sym}_2, \mathsf{t}_2)$, então $\langle \mathsf{t}_1, \mathsf{t}_2\rangle \in \mathcal{DC}(\langle \mathsf{sym}_1, \mathsf{sym}_2\rangle)$. Para isso, nós vamos precisar de algumas definições adicionais:

Definição 3.3.3. O grafo de dependência simbólica (*symbol dependency graph*) de um dado conjunto de restrições de derivabilidade $\mathcal{DC}$, escrito $\mathcal{G}_{\mathcal{DC}}$, é um grafo direcionado definido a seguir :

- Os nós são os símbolos de $\mathcal{DC}$, isto é, $\mathsf{dom}(\mathcal{DC})$;
- $\mathcal{G}_{\mathcal{DC}}$ contém a aresta $\mathsf{sym}_1 \longmapsto \mathsf{sym}_2$ se e somente se $\mathsf{dc}(\mathsf{sym}_1, \mathcal{S}_1), \mathsf{dc}(\mathsf{sym}_2, \mathcal{S}_2) \in \mathcal{DC}$ e $\mathcal{S}_2$ contém pelo menos uma ocorrência de sym_1;

Apesar do grafo de dependência simbólica de $\mathcal{DC}$ puder ser cíclico, nossa semântica operacional vai garantir que esses grafos se mantenham acíclicos.

Considere o seguinte conjunto de restrições de derivabilidade:

$$\mathcal{DC}_0 = \mathsf{dc}(\mathsf{sym}_1, \{\mathsf{sk}(eve)\}), \mathsf{dc}(\mathsf{sym}_2, \{\mathsf{symk}\}), \mathsf{dc}(\mathsf{sym}_3, \{\mathsf{e}(\langle \mathsf{sym}_2, \mathsf{sym}_1 \rangle, \mathsf{pk}(Alice)\}),$$
$$\mathsf{dc}(\mathsf{sym}_4, \{\mathsf{sym}_3, \mathsf{sym}_2\})$$

O grafo de dependência de $\mathcal{DC}_0$, chamado $\mathcal{G}_{\mathcal{DC}_0}$, é o grafo acíclico direcionado (DAG) representado na Figura 3.2 .

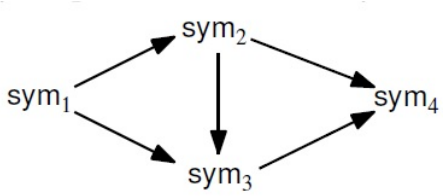

Figura 3.2: Grafo de dependência simbólica de $\mathcal{DC}_0$

Toda vez que o grafo de dependência de um conjunto de restrições é um DAG, nós classificamos o conjunto como acíclico. Nós podemos computar uma ordenação topológica (*topological sort*) de um DAG em tempo linear. Por exemplo, a ordenação topológica de $\mathcal{G}_{\mathcal{DC}_0}$ é $[\mathsf{sym}_1, \mathsf{sym}_2, \mathsf{sym}_3, \mathsf{sym}_4]$.

Dado um conjunto de restrições de derivabilidade, nós agora podemos especificar formalmente o conjunto de termos representado por um termo simbólico .

Definição 3.3.4. Tome um termo simbólico ms, com um conjunto acíclico de restrições de derivabilidade $\mathcal{DC}$. Assumindo que $\mathsf{dc}(\mathsf{sym}, \mathcal{S}) \in \mathcal{DC}$, nós definimos o operador $Sub_{\mathcal{DC}}(\mathsf{sym}, \mathsf{ms})$ como o conjunto de termos simbólicos obtido ao substituir todas as ocorrências de sym em ms por um termo $\mathsf{ms}_1 \in \mathcal{DC}(\mathsf{sym})$. Formalmente, o conjunto:

$$\sigma(\mathsf{ms}) \mid \quad \sigma = \{\mathsf{sym} \mapsto \mathsf{ms}_1\}$$
$$\text{é uma substituição onde } \mathsf{ms}_1 \in \mathcal{DC}(\mathsf{sym}).$$

Sobrecarregando a notação, para um conjunto de termos simbólicos $\mathcal{S}$, chamamos de $Sub_{\mathcal{DC}}(\mathsf{sym}, \mathcal{S})$ o conjunto $\bigcup_{\mathsf{ms} \in \mathcal{S}} Sub_{\mathcal{DC}}(\mathsf{sym}, \mathsf{ms})$.

Considere $\mathcal{T} = [\mathsf{sym}_1, \ldots, \mathsf{sym}_n]$ como uma ordenação topológica qualquer do DAG $\mathcal{G}_{\mathcal{DC}}$. Então o significado de um termo simbólico ms em relação a $\mathcal{DC}$, escrito $\mathcal{DC}(\mathsf{ms})$, é o conjunto obtido ao aplicar $Sub_{\mathcal{DC}}$ consecutivamente como descrito a seguir:

$$Sub_{\mathcal{DC}}(\mathsf{sym}_1, Sub_{\mathcal{DC}}(\mathsf{sym}_2, Sub_{\mathcal{DC}}(\ldots Sub_{\mathcal{DC}}(\mathsf{sym}_n, \mathsf{ms}) \ldots))).$$

Por exemplo, $Sub_{DC}(\mathsf{sym}_4, \mathsf{e}(\mathsf{sym}_4, \mathsf{pk}(bob)))$ é o conjunto:

$$\{\mathsf{e}(\mathsf{ms}_1, \mathsf{pk}(bob)) \mid \mathsf{ms}_1 \in \mathcal{DC}_0(\mathsf{sym}_4)\}$$

Nele, estão contidos os termos $\mathsf{e}(\mathsf{sym}_2, \mathsf{pk}(bob))$, $\mathsf{e}(\mathsf{sym}_3, \mathsf{pk}(bob))$, $\mathsf{e}(\langle \mathsf{sym}_2, \mathsf{sym}_2 \rangle, \mathsf{pk}(bob))$, $\mathsf{e}(\langle \mathsf{sym}_2, \mathsf{sym}_3 \rangle, \mathsf{pk}(bob))$,... O conjunto $\mathcal{DC}_0(\mathsf{e}(\mathsf{sym}_4, \mathsf{pk}(bob)))$ contém $\mathsf{e}(\mathsf{e}(\mathsf{sk}(eve), \mathsf{symk}), \mathsf{pk}(bob))$ ao aplicar sobre o termo $\mathsf{e}(\mathsf{sym}_4, \mathsf{pk}(bob)$ a substituição $[\mathsf{sym}_4 \mapsto \mathsf{sym}_2]$ seguida por $[\mathsf{sym}_2 \mapsto \mathsf{e}(\mathsf{sym}_1, \mathsf{symk})]$ e $\mathsf{sym}_1 \mapsto \mathsf{sk}(eve)$.

Tomemos qualquer conjunto acíclico de restrições de derivabilidade tal que seu símbolo de menor altura (*lowest height symbol*) em relação a $\mathcal{G}_{\mathcal{DC}}$ possua restrições na forma $\mathsf{dc}(\mathsf{sym}, \mathcal{S})$ onde $\mathcal{S}$ possui apenas termos base. Nesse caso, $\mathcal{DC}(\mathsf{ms})$ é um conjunto (infinito) de termos base. Isso se dá porque a aplicação sucessiva de $Sub_{\mathcal{DC}}$ irá eventualmente eliminar todos os símbolos.

Dado os termos $\mathsf{ms}, \mathsf{ms}'$, nós descrevemos agora como avaliar se $\mathsf{ms} \in \mathcal{DC}(\mathsf{ms}')$. Primeiro nós construímos a substituição correspondente $\mathsf{ssb} = \{\mathsf{sym}'_1 \mapsto \mathsf{ms}_1, \ldots, \mathsf{sym}'_n \mapsto \mathsf{ms}_n\}$ dos símbolos de ms' para os (sub)termos de ms. Se tal correspondência não existir, então $\mathsf{ms} \notin \mathcal{DC}(\mathsf{ms}')$. Para cada símbolo $\mathsf{sym}'_i \mapsto \mathsf{ms}_i$, deixe $\mathsf{dc}(\mathsf{sym}'_i, \mathcal{S}_i) \in \mathcal{DC}$. Nós avaliamos se $\mathsf{ms}_i \in \mathcal{DC}(\mathsf{sym}'_i)$ recursivamente, como mostrado a seguir:

- Se $\mathsf{ms}_i \in \mathcal{S}_i$, retorna verdadeiro
- Se $\mathsf{ms}_i = \mathsf{e}(\mathsf{ms}_2, \mathsf{ms}_3)$, então checar se $\langle \mathsf{ms}_2, \mathsf{ms}_3 \rangle \in \mathcal{DC}(\mathsf{sym}'_i)$.
- Se $\mathsf{ms}_i = \langle \mathsf{ms}_i^1, \ldots, \mathsf{ms}_i^m \rangle$, então para cada $1 \leq j \leq m$, nós checamos se $\mathsf{ms}_i^j \in \mathcal{DC}(\mathsf{sym}'_i)$.

Definição 3.3.5. $\mathsf{ssb} \vDash \mathcal{DC}$ se para cada $\mathsf{sym} \mapsto \mathsf{ms} \in \mathsf{ssb}$, $\mathsf{ms} \in \mathcal{DC}(\mathsf{sym})$.

As definições abaixo especificam o outro tipo de restrição, as restrições de comparação.

Definição 3.3.6. Uma restrição de comparação pode ser uma restrição de igualdade, escrita $\mathsf{eq}(\mathsf{ms}_1, \mathsf{ms}_2)$, ou uma restrição de desigualdade, escrita $\mathsf{neq}(\mathsf{ms}_1, \mathsf{ms}_2)$.

Um conjunto $\mathcal{EQ}$ de restrições de comparação deve ser interpretado como uma conjunção de restrições. A definição a seguir especifica quando tal conjunto é satisfatível.

Definição 3.3.7. Seja $\mathcal{DC}$ um conjunto de restrições de derivabilidade qualquer, e $\mathcal{EQ}$ um conjunto de restrições de comparação. O conjunto $\mathcal{EQ}$ é satisfatível em relação a $\mathcal{DC}$, escrito $\mathcal{DC} \vDash \mathcal{EQ}$, se existe uma substituição $\sigma = [\mathsf{sym}_1 \mapsto \mathsf{m}_1, \ldots, \mathsf{sym}_n \mapsto \mathsf{m}_n]$ mapeando todos os símbolos sym_i em $\mathcal{EQ}$ para termos base em $\mathcal{DC}(\mathsf{sym}_i)$, tal que:

- para todas as restrições de igualdade $\mathsf{eq}(\mathsf{ms}, \mathsf{ms}') \in \mathcal{EQ}, \sigma[\mathsf{ms}] = \sigma[\mathsf{ms}']$;
- para todas as restrições de desigualdade $\mathsf{neq}(\mathsf{ms}, \mathsf{ms}') \in \mathcal{EQ}, \sigma[\mathsf{ms}] \neq \sigma[\mathsf{ms}']$.

Nós definimos abaixo o procedimento $\mathsf{eqCheck}$, para verificar se um dado conjunto de restrições de comparação é satisfatível.

Definição 3.3.8. Considere um conjunto (finito) de restrições de comparação $\mathcal{EQ}$, e um conjunto $\mathcal{DC}$ de restrições de derivabilidade. Assumindo que $\mathsf{eq}(\mathsf{ms}_1, \mathsf{ms}'_1), \mathsf{eq}(\mathsf{ms}_2, \mathsf{ms}'_2), \ldots, \mathsf{eq}(\mathsf{ms}_n, \mathsf{ms}'_n)$ são todas as restrições de igualdade contidas em $\mathcal{EQ}$, então $\mathsf{eqCheck}(\mathcal{EQ}, \mathcal{DC})$ é verdadeiro se e somente se:

- Existe um unificador ssb dos termos $\alpha = \langle \mathsf{ms}_1, \ldots, \mathsf{ms}_n \rangle$ e $\beta = \langle \mathsf{ms}'_1, \ldots, \mathsf{ms}'_n \rangle$, mapeando os símbolos para termos simbólicos, *i.e.*, $\mathsf{ssb}(\alpha) = \mathsf{ssb}(\beta)$;
- Para todas as restrições de desigualdade $\mathsf{neq}(\mathsf{ms}, \mathsf{ms}') \in \mathcal{EQ}, \mathsf{ssb}[ms] \neq \mathsf{ssb}[\mathsf{ms}']$;
- ssb é consistente com $\mathcal{DC}$, isto é, não instancia nenhum símbolo $\mathsf{sym} \in \mathsf{dom}(\mathcal{DC})$ para um termo simbólico que ele não possa representar.

Lemma 3.3.1. *$\mathcal{DC} \vDash \mathcal{EQ}$ se e somente se* $\mathsf{eqCheck}(\mathcal{EQ}, \mathcal{DC})$.

Além disso, o significado de um termo simbólico deve levar restrições de comparação $\mathcal{EQ}$ em conta. Isto é, não deve ser possível substituir um símbolo por um termo que falsifica alguma restrição de comparação. Nós estendemos a Definição 3.3.4 de acordo.

Definição 3.3.9. Dado um conjunto acíclico de restrições de derivabilidade $\mathcal{DC}$ e um conjunto de restrições de comparação $\mathcal{EQ}$. O significado do termo simbólico ms respeitando $\mathcal{DC}$ e $\mathcal{EQ}$, escrito $\mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}}$, é o conjunto de termos $\mathsf{ms}' \in \mathcal{DC}(\mathsf{ms})$ tal que existe uma substituição correspondente θ:

- $\theta(\mathsf{ms}) = \mathsf{ms}'$;

- Para todas as restrições de igualdade $\mathsf{eq}(\mathsf{ms}_1, \mathsf{ms}_2) \in \mathcal{EQ},\ \theta(\mathsf{ms}_1) = \theta(\mathsf{ms}_2)$;
- Para todas as restrições de desigualdade $\mathsf{neq}(\mathsf{ms}_1, \mathsf{ms}_2) \in \mathcal{EQ},\ \theta(\mathsf{ms}_1) \neq \theta(\mathsf{ms}_2)$.

Por exemplo, considere $\mathsf{dc}(\mathsf{sym}_1, \{\mathsf{t}_1\}),\ \mathsf{dc}(\mathsf{sym}_2, \{\mathsf{t}_2\}) \in \mathcal{DC}$ e um conjunto com uma única restrição de comparação $\mathcal{EQ} = \{\mathsf{eq}(\mathsf{sym}_1, \mathsf{sym}_2)\}$. O termo $\langle \mathsf{t}_1, \mathsf{t}_2 \rangle \in \mathcal{DC}(\langle \mathsf{sym}_1, \mathsf{sym}_2 \rangle)$, mas $\langle \mathsf{t}_1, \mathsf{t}_2 \rangle \notin \mathcal{DC}(\langle \mathsf{sym}_1, \mathsf{sym}_2 \rangle) \mid_{\mathcal{EQ}}$. Isso se dá porque a substituição $\theta = [\mathsf{sym}_1 \mapsto \mathsf{t}_1, \mathsf{sym}_2 \mapsto \mathsf{t}_2]$ torna a restrição $\mathsf{eq}(\mathsf{sym}_1, \mathsf{sym}_2)$ falsa, já que $\mathsf{t}_1 \neq \mathsf{t}_2$.

3.3.2 Resolução de Restrições Simbólicas

Para a especificação dos protocolos, nós vamos assumir um intruso de Dolev-Yao [Dolev e Yao 1983] tradicional, isto é, um intruso que é capaz de construir mensagens a partir do seu conhecimento através do encadeamento (*tupling*) e encriptação de mensagens. Ele não é capaz de decriptar mensagens que estejam criptografadas com uma chave cuja inversa ele não possua. Essa característica é modelada pela definição de conjuntos mínimos 3.3.1.

Definição 3.3.10. O conhecimento de um intruso $\mathcal{IK}$ é um conjunto mínimo de termos simbólicos.

Durante a execução do protocolo, o intruso envia mensagens para os participantes honestos, construídas a partir da sua base de conhecimento. Suponha que um participante honesto esteja pronto para receber uma mensagem da forma m, possivelmente contendo variáveis. Ao invés de considerar todas as possíveis instâncias de termos base de m, nós consideramos uma representação finita desse conjunto, mais especificamente mensagens simbólicas onde os possíveis valores dos símbolos são definidos pelas restrições de derivabilidade. Para computar essa representação, o intruso substitui todas as variáveis por termos simbólicos, possivelmente contendo símbolos recém-criados (*fresh symbols*), e então restringe os símbolos de forma que as instâncias permitidas coincidam exatamente com os termos que tenham o mesmo formato de m (*terms matching m*) e que o intruso possa derivar do seu conhecimento atual $\mathcal{IK}$.

Por exemplo, considere o termo $\mathsf{m} = \mathsf{e}(\{\mathsf{v}_1, \mathsf{sym}, \mathsf{v}_1, \mathsf{v}_2\}, \mathsf{k})$, que é esperado como entrada por um participante honesto. Aqui v_1 e v_2 são variáveis e sym é restringido por

um conjunto de restrições de derivabilidade $\mathcal{DC}$. Nós criamos dois símbolos novos sym_1 e sym_2 para as variáveis v_1 e v_2, respectivamente. Nós utilizamos sb para representar essas substituições de variáveis para termos simbólicos. Nesse exemplo $\mathsf{sb} = [\mathsf{v}_1 \mapsto \mathsf{sym}_1, \mathsf{v}_2 \mapsto \mathsf{sym}_2]$. Nós então obtemos $\mathsf{ms} = \mathsf{sb}[m] = \mathsf{e}(\{\mathsf{sym}_1, \mathsf{sym}, \mathsf{sym}_1, \mathsf{sym}_2\}, \mathsf{k})$.

Dito isso, nos falta resolver o seguinte problema: dado uma base de conhecimento do intruso $\mathcal{IK}$ e um conjunto de restrições de derivabilidade $\mathcal{DC}$ restringindo os símbolos em $\mathcal{IK}$, precisamos encontrar uma representação de todas as instâncias de um termo simbólico ms, que satisfaça $\mathcal{DC}$, e que possa ser gerada a partir de $\mathcal{IK}$.

Nós implementamos uma função chamada sgen que enumera todas as possíveis instâncias. A sua especificação está descrita no apêndice A. Abaixo, nós descrevemos sgen informalmente, e ilustramos o seu funcionamento com alguns exemplos. Um algoritmo similar foi utilizado em [Cortier e Delaune 2009].

Na prática, $\mathsf{sgen}(\mathsf{m}, \mathcal{IK}, \mathcal{DC})$ recebe como parâmetros um termo m, que representa o termo esperado pelo participante honesto, o conhecimento do intruso $\mathcal{IK}$ e o conjunto de restrições de derivabilidade $\mathcal{DC}$. $\mathsf{sgen}(\mathsf{m}, \mathcal{IK}, \mathcal{DC})$ então gera como saída o par:

$$\{\mathsf{sb}, \{\mathsf{ssb}_1, \mathcal{DC}_1\} \ldots \{\mathsf{ssb}_k, \mathcal{DC}_k\}\}$$

Onde sb mapeia as variáveis de m para símbolos recém-criados, e cada par $\{\mathsf{ssb}_i, \mathcal{DC}_i\}$ é uma solução para o problema descrito acima, para $\mathsf{ms} = \mathsf{sb}[\mathsf{m}]$. Se $k = 0$, então não existe solução, *i.e.*, o intruso não é capaz de gerar um termo que corresponda a m.

Intuitivamente, a função sgen constrói uma solução ao criar uma correspondência entre m com um termo do conhecimento $\mathcal{IK}$ (o caso base) ou construindo m a partir de termos presentes em $\mathcal{IK}$, utilizando concatenação (*tupling*) e encriptação. Os exemplos a seguir ilustram os diferentes casos envolvidos:

Exemplo 3.3.1. Considere para os casos a seguir $\mathsf{m} = \mathsf{e}(\langle \mathsf{v}, \mathsf{sym} \rangle, \mathsf{k})$.

- Caso 1 (criando uma correspondência com um termo pertencente a $\mathcal{IK}$):

$$\mathcal{IK} = \{\mathsf{e}(\langle \mathsf{n}_a, \mathsf{sym}_1 \rangle, \mathsf{k})\} \qquad \mathcal{DC} = \mathsf{dc}(\mathsf{sym}, \{\mathsf{n}_a, \mathsf{n}_c\})\ \mathsf{dc}(\mathsf{sym}_1, \mathcal{S})$$

  Então a solução gerada por sgen é :

$$\{\mathsf{sb}, \{[\mathsf{sym}_\mathsf{v} \mapsto \mathsf{n}_a, \mathsf{sym} \mapsto \mathsf{sym}_1], \mathsf{dc}(\mathsf{sym}_1, \{\mathsf{n}_a, \mathsf{n}_c\} \cap \mathcal{S})\}\}$$

onde $\mathsf{sb} = [\mathsf{v} \mapsto \mathsf{sym}_\mathsf{v}]$ e sym_v é um novo símbolo. Perceba que como sym_v é mapeado para um termo base específico (n_a), nenhuma restrição de derivabilidade é gerada para ele. Além disso, sym possui as mesmas restrições de sym_1. Isso faz com que $\mathsf{dc}(\mathsf{sym}, \mathcal{S})$ seja removida.

- Caso 2 (construir os termos a partir de $\mathcal{IK}$): Assumindo que $\mathsf{k} \in \mathcal{IK}$ e que $\mathcal{IK}$ não possui o termo encriptado. Nesse caso, a solução dada por sgen é:

$$\{[\mathsf{v} \mapsto \mathsf{sym}_\mathsf{v}], \{[], \mathcal{DC} \cup \{\mathsf{dc}(\mathsf{sym}_\mathsf{v}, \mathcal{IK})\}\}\}$$

  que corresponde a gerar o termo $\mathsf{e}(\{\mathsf{sym}_v, \mathsf{sym}\}, \mathsf{k})$. Além disso, sym_v pode ser qualquer termo derivável do conhecimento do intruso.

- Caso 3 (Sem Solução): Assumindo que $\mathcal{IK} = \{\mathsf{e}(\langle \mathsf{n}_a, \mathsf{n}_b \rangle, \mathsf{k})\}$ e que $\mathcal{DC} = \mathsf{dc}(\mathsf{sym}, \{\mathsf{n}_a, \mathsf{n}_b\})$. Como sym não pode ser instanciado para n_b, o intruso não pode utilizar o termo $\mathsf{e}(\langle \mathsf{n}_a, \mathsf{n}_b \rangle, \mathsf{k})$.

3.3.3 Semântica Operacional para Protocolos Temporais

A semântica operacional de protocolos temporais envolve regras que reescrevem configurações. Tais configurações serão descritas abaixo:

Definição 3.3.11. Uma configuração simbólica é uma tupla da forma $\langle \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @\mathsf{tG}$, onde:

- $\mathcal{P}$ é um conjunto finito de participantes da forma $[n \mid \mathsf{pl} \mid \mathsf{keys}]$, onde n é um identificador único, pl representa um protocolo e keys o conjunto de chaves conhecidas por aquele participante;
- $\mathcal{IK}$ é o conhecimento do intruso;
- $\mathcal{DC}$ é um conjunto de restrições de derivabilidade;
- $\mathcal{EQ}$ é um conjunto de restrições de comparação;
- $\mathcal{TC}$ é um conjunto de restrições de tempo;
- tG é um símbolo temporal que representa o tempo global da configuração.

A semântica operacional dos protocolos temporais é ilustrada pela figura 3.3. A regra **New** substitui uma variável v por um novo *nonce* n^{ν}. A regra **Send** envia uma mensagem ms que é então adicionada ao conhecimento do intruso. A regra **Receive** espera um termo de formato m. A função $\mathsf{sgen}(\mathsf{m}, \mathcal{IK}, \mathcal{DC})$ retorna as substituições de variáveis sb e um conjunto de soluções $\{\mathsf{ssb}, \mathcal{DC}_1\}$ css. Cada solução gera traços diferentes. Nós aplicamos sb para o restante de pl e aplicamos a substituição de símbolos ssb para todos os símbolos da configuração resultante. Essa regra também tem uma condição que garante que a mensagem $\mathsf{ms} = \mathsf{ssb}[\mathsf{sb}[m]]$ esteja encriptada com chavens que possam ser decriptadas pelo participante honesto, especificado pela função *isReceivable*. Por fim, a regra **Receive** também adiciona qualquer nova chave que o participante possa ter aprendido com ms ao seu conjunto de chaves conhecidas.

A regra **If-true** avalia se um termo m_1 e um outro ms_1 podem ser correspondidos (*matched*), utilizando o conhecimento do intruso $\mathcal{IK}$. Isso é realizado através da função sgenB, que é definida de uma forma similar a sgen. Primeiramente, ela corresponde a estrutura dos termos com o problema de correspondência $\mathsf{ms}_1 = \mathsf{ms}_2$, onde os símbolos são considerados como variáveis. Se eles não puderem corresponder um com o outro, então o conjunto vazio é retornado como solução. Caso contrário, uma solução $\mathsf{sym}_1 \mapsto \mathsf{ms}_1, \ldots, \mathsf{sym}_n \mapsto \mathsf{ms}_n$ é retornada. Depois, chama-se sgen nos termos $\langle \mathsf{sym}_1, \ldots, \mathsf{sym}_n \rangle$ e $\langle \mathsf{ms}_1, \ldots, \mathsf{ms}_n \rangle$, retornando a saída. A regra então adiciona a restrição de igualdade ao conjunto $\mathcal{EQ}$.

Por último, a regra **If-false** substitui as variáveis em m por novos símbolos, restritos em $\mathcal{DC}'$ com o conhecimento do intruso. Isto é, se sym^{v} é um novo símbolo (*fresh symbol*), então $\mathsf{dc}(\mathsf{sym}^{v}, \mathcal{IK})$. Além disso, ela adiciona a restrição de desigualdade em $\mathcal{EQ}$. A intuição por trás de substituir todas as variáveis em m_1 por novos símbolos é especificar que para qualquer instância dessas variáveis, o termo resultante não pode corresponder com m_2 como especificado pela restrição de desigualdade.

Exemplo 3.3.2. Considere o Exemplo 3.2.1, que representa o protocolo de Needham-Schroeder.

$$Alice(Z) := (\mathsf{new}\ N_a), (+\mathsf{e}(\langle N_a, alice \rangle, \mathsf{pk}(Z))), (-\mathsf{e}(\langle N_a, Y \rangle, \mathsf{pk}(alice)\})), (+\mathsf{e}(Y, \mathsf{pk}(Z)))$$

$$Bob := (-\mathsf{e}(\langle X, Z \rangle, \mathsf{pk}(bob))), (\mathsf{new}\ N_b), (+\mathsf{e}(\langle X, N_b \rangle, \mathsf{pk}(Z)\})), (-\mathsf{e}(N_b, \mathsf{pk}(bob)))$$

Assuma que o intruso, inicialmente, conhece apenas a própria chave secreta além dos

termos inferíveis, $\mathcal{IK}_0 = \{\mathsf{sk}(eve)\}$, e não existe nenhum símbolo em $\mathcal{DC} = \emptyset$. Na execução da função de Alice, o participante cria um novo *nonce* N_a e envia a mensagem $\mathsf{e}(\langle N_a, alice\rangle, \mathsf{pk}(eve))$. Nesse ponto, o conhecimento do intruso passa a ser :

$$\mathcal{IK}_1 = \mathcal{IK}_0 \cup \{N_a\}$$

Ele agora pode mandar a seguinte mensagem para *Bob*, $\mathsf{e}(\langle \mathsf{sym}_1, \mathsf{sym}_2\rangle, \mathsf{pk}(bob))$, onde $\mathsf{sym}_1, \mathsf{sym}_2$ são recém criados, e restritos em $\mathcal{DC}_1 = \{\mathsf{dc}(\mathsf{sym}_1, \mathcal{IK}_1), \mathsf{dc}(\mathsf{sym}_2, \mathcal{IK}_1)\}$. Nesse ponto, Bob cria uma novo *nonce* N_b e envia a mensagem $\mathsf{e}(\langle \mathsf{sym}_1, N_b\rangle, \mathsf{pk}(\mathsf{sym}_2))$. O intruso então aprende essa mensagem (e nada mais), alterando o seu conhecimento para:

$$\mathcal{IK}_2 = \mathcal{IK}_1 \cup \{\mathsf{e}(\langle \mathsf{sym}_1, N_b\rangle, \mathsf{pk}(\mathsf{sym}_2))\}$$

Agora, o intruso pode enganar Alice ao enviar a mensagem na forma $\mathsf{e}(\langle N_a, Y\rangle, \mathsf{pk}(alice)\})$ (já que o intruso possui os inferíveis em $\mathcal{IK}$, sym_2 pode derivar *alice*). Nós criamos um novo símbolo sym_3 para Y, obtendo $\mathsf{e}(\langle N_a, \mathsf{sym}_3\rangle, \mathsf{pk}(alice)\})$ e tentamos gerar essa mensagem a partir de $\mathcal{IK}_2$, utilizando sgen. De fato, essa mensagem pode ser gerada utilizando $\mathsf{e}(\langle \mathsf{sym}_1, N_b\rangle, \mathsf{pk}(\mathsf{sym}_2)) \in \mathcal{IK}_2$. Isso gera a $\mathsf{ssb} = [\mathsf{sym}_1 \mapsto N_a, \mathsf{sym}_2 \mapsto alice, \mathsf{sym}_3 \mapsto N_b]$. Essa substituição é consistente com $\mathcal{DC}_1$. Perceba que sym_3 não contém nenhuma restrição. O protocolo então termina com o intruso simplesmente repassando a mensagem de resposta de Alice para Bob. Bob então pensa que ele está se comunicando com Alice, quando isso não é verdade.

Todas as regra descritas nessa sessão tem que obedecer a duas condições gerais. A primeira delas é que o conjunto resultante de restrições de comparação deve ser consistente. Isso pode ser checado como definido na Definição 3.3.8. A segunda condição diz respeito aos símbolos temporais. Toda vez que uma regra é aplicada, restrições de tempo $\mathcal{TC}_1$ são adicionadas ao conjunto de restrições de tempo da configuração. Essas restrições são obtidas ao se substituir cur em tc com tG_1, junto com a restrição $\mathsf{tG}_1 \geq \mathsf{tG}_0$, especificando que o tempo apenas avança. A regra então é executada apenas se o conjunto resultante $\mathcal{TC} \cup \mathcal{TC}_1$ for consistente, o que pode ser checado utilizando um resolvedor SMT. Essa forma de especificar sistemas foi discutida na seção 2.8, e tem o nome de *Rewriting Modulo SMT* [Rocha 2012].

New: $\langle [n \mid (\mathsf{new\ v} \ \#\ \mathsf{tc}), \mathsf{pl} \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @ \mathsf{tG}_0$
$\longrightarrow \langle [n \mid \mathsf{sb}[\mathsf{pl}] \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC}_1 \rangle @ \mathsf{tG}_1$
onde n^ν é um *nonce* recém-criado e $\mathsf{sb} = [\mathsf{v} \mapsto \mathsf{n}^\nu]$

Send: $\langle [n \mid (+\mathsf{ms} \ \#\ \mathsf{tc}), \mathsf{pl} \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @ \mathsf{tG}_0$
$\longrightarrow \langle [n \mid \mathsf{pl} \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK} \cup \{\mathsf{ms}\}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC}_1 \rangle @ \mathsf{tG}_1$

Receive: $\langle [n \mid (-\mathsf{m} \ \#\ \mathsf{tc}), \mathsf{pl} \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @ \mathsf{tG}_0$
$\longrightarrow \mathsf{ssb}[\langle [n \mid \mathsf{sb}[\mathsf{pl}] \mid \mathsf{addKeys}(\mathsf{ms}, \mathsf{keys})]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}_1, \mathcal{EQ}, \mathcal{TC}_1 \rangle] @ \mathsf{tG}_1$
onde $\{\mathsf{sb}, \{\mathsf{ssb}, \mathcal{DC}_1\}\ \mathsf{css}\} := \mathsf{sgen}(\mathsf{m}, \mathcal{IK}, \mathcal{DC})$ e $\mathsf{ms} = \mathsf{ssb}[\mathsf{sb}[\mathsf{m}]]$ e $isReceivable(\mathsf{ms}, \mathsf{keys})$

If-true: $\langle [n \mid (\mathsf{if}\ (\mathsf{ms}_1 := \mathsf{ms}_2 \ \#\ \mathsf{tc})\ \mathsf{then\ pl}_1\ \mathsf{else\ pl}_2) \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @ \mathsf{tG}_0$
$\longrightarrow \mathsf{ssb}[\langle [n \mid \mathsf{sb}[\mathsf{pl}_1] \mid \mathsf{keys}\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}_1, \mathcal{EQ} \cup \{\mathsf{eq}(\mathsf{sb}[\mathsf{ms}_1], \mathsf{sb}[\mathsf{ms}_2])\}, \mathcal{TC}_1 \rangle] @ \mathsf{tG}_1$
onde $\{\mathsf{sb}, \{\mathsf{ssb}, \mathcal{DC}_1\}\ \mathsf{css}\} := \mathsf{sgenB}(\mathsf{ms}_1 = \mathsf{ms}_2, \mathcal{IK}, \mathcal{DC})$

If-false: $\langle [n \mid (\mathsf{if}\ \mathsf{ms}_1 := \mathsf{ms}_2 \ \#\ \mathsf{tc}\ \mathsf{then\ pl}_1\ \mathsf{else\ pl}_2) \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ} \rangle$
$\longrightarrow \langle [n \mid \mathsf{pl}_2 \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC} \cup \mathcal{DC}', \mathcal{EQ} \cup \{\mathsf{neq}(\mathsf{ms}_1, \mathsf{ms}_2)\} \rangle$

Figura 3.3: Semântica operacional para protocolos básicos. Em toda regra, tc_1 é a restrição de tempo obtida substituindo cur em tc pelo tempo global tG_1 e $\mathcal{TC}_1 = \mathcal{TC} \cup \{\mathsf{tG}_1 \geq \mathsf{tG}_0, \mathsf{tc}_1\}$. A função *isReceivable* avalia se a mensagem $\mathsf{ssb}[\mathsf{sb}[\mathsf{m}]]$ pode ser decriptada com as chaves contidas em keys. Todas as regras têm como condições sine-qua-non para serem aplicáveis que tanto o conjunto de restrições de comparação quanto o conjunto de restrições de tempo devem ser satisfatíveis.

Definição 3.3.12. Considere $\mathcal{R}$ como conjunto de regras definidas na Figura 3.3. Um traço temporal (*timed trace*) é uma sequência rotulada de transições escrita $\mathcal{C}_1 \xrightarrow{l_1} \mathcal{C}_2 \xrightarrow{l_2} \cdots \xrightarrow{l_{n-1}} \mathcal{C}_n$ tal que para todo $1 \leq i \leq n-1$, $\mathcal{C}_i \longrightarrow \mathcal{C}_{i+1}$ é uma instância de alguma regra contida em $\mathcal{R}$ e l_i é $+\mathsf{ms}@\mathsf{tG}_1$ se for uma instância de uma regra *Send* enviando um termo ms no tempo tG_1, ou $-\mathsf{ms}@\mathsf{tG}_1$ se for uma instância de uma regra *Receive* recebendo um termo ms no tempo tG_1, ou $\emptyset$ caso contrário.

Protocolos temporais são sistemas com infinitos estados, já que símbolos de tempo podem ser instanciados por qualquer número real positivo. Com a utilização de *rewri-*

ting modulo SMT, nós temos apenas que acumular as restrições de tempo. Apenas os traços com um conjunto de restrições de tempo satisfatível são permitidos. De fato, como nós descrevemos na Seção 3.5, o número de traços temporais não é apenas finito (como declarado na proposição a seguir), como bem pequeno (menos de 40, em alguns casos). Como a verificação de equivalência observacional envolve a correspondência de traços, essa checagem pode ser automatizada.

Proposição 3.3.1. *O conjunto de traços temporais começando a partir de qualquer configuração $\mathcal{C}_0$ é finito.*

Proposição 3.3.2. *Considere $\tau = \mathcal{C}_1 \xrightarrow{l_1} \mathcal{C}_2 \xrightarrow{l_2} \cdots \xrightarrow{l_{n-1}} \mathcal{C}_n$ um traço temporal. Para qualquer configuração $\mathcal{C}_i = \langle \mathcal{P}_i, \mathcal{IK}_i, \mathcal{DC}_i, \mathcal{EQ}_i, \mathcal{TC}_i \rangle @ \mathsf{tG}_i$, tal que $1 \leq i \leq n$, o que é descrito a seguir vale:*

- *Para qualquer $i \leq j \leq n$, $\mathcal{DC}_n(\mathcal{IK}_i)|_{\mathcal{EQ}_n} \subseteq \mathcal{DC}_n(\mathcal{IK}_j)|_{\mathcal{EQ}_n}$, isto é, o conhecimento do intruso só é capaz de aumentar;*
- *$\mathcal{DC}_i$ é um conjunto acíclico de restrições de derivabilidade;*
- *Considere sym_k como um símbolo criado em algum $\mathcal{C}_k$, $k < i$, e sym_i é um símbolo recém-criado em $\mathcal{C}_i$. Se $\mathsf{dc}(\mathsf{sym}_k, \mathcal{S}_k), \mathsf{dc}(\mathsf{sym}_i, \mathcal{S}_i) \in \mathcal{DC}_i$, então $\mathcal{S}_k \subseteq \mathcal{S}_i$. Isto é, símbolos que são introduzidos em transições posteriores podem ser instanciados para mais termos do que símbolos introduzidos em transições anteriores.*

Intrusos Temporais: De fato, a nossa implementação generaliza o maquinário descrito nessa seção ao considerar mais de um intruso temporal [Kanovich et al. 2014; Nigam, Talcott e Urquiza 2016]. Como descrito em [Kanovich et al. 2014] e abordado na Seção 2.2 , o intruso padrão de Dolev-Yao pode não ser adequado para a verificação de protocolos Cíber-Físicos (*Cyber-Physical Protocols*), onde as propriedades físicas do ambiente são importantes. Diferentemente de um intruso Dolev-Yao , um intruso temporal precisa esperar a mensagem chegar para então aprender o seu conteúdo. [Nigam, Talcott e Urquiza 2016] provou um limitante superior para o número de intrusos temporais necessários. Nossa ferramenta implementa essa estratégia. No entanto, para os exemplos considerados aqui, um intruso Dolev-Yao convencional já bastaria.

3.4 Equivalência Observacional

Nosso objetivo agora é determinar quando duas configurações $\mathcal{C}_I = \langle \mathcal{P}_I, \mathcal{IK}_I, \mathcal{DC}_I, \mathcal{EQ}_I, \mathcal{TC}_I \rangle @\mathsf{tG}$ e $\mathcal{C}'_I = \langle \mathcal{P}'_I, \mathcal{IK}'_I, \mathcal{DC}'_I, \mathcal{EQ}'_I, \mathcal{TC}'_I \rangle @\mathsf{tG}'$ não podem ser diferenciadas por um intruso Dolev-Yao. Isto é, para qualquer traço começando em $\mathcal{C}_I$ existe um *traço equivalente* começando em $\mathcal{C}'_I$. A definição a seguir especifica a noção de observáveis:

Definição 3.4.1. Considere $\tau = \mathcal{C}_1 \xrightarrow{l_1} \mathcal{C}_2 \xrightarrow{l_2} \cdots \xrightarrow{l_{n-1}} \mathcal{C}_n = \langle \mathcal{P}_n, \mathcal{IK}_n, \mathcal{DC}_n, \mathcal{EQ}_n, \mathcal{TC}_n \rangle @\mathsf{tG}_n$ como um traço temporal. Seu observável é a tupla $\langle \mathsf{tt}_I, \mathcal{L}_\tau, \mathcal{IK}_n, \mathcal{DC}_n, \mathcal{EQ}_n, \mathcal{TC}_n \rangle$, onde tt_I é o tempo global da configuração $\mathcal{C}_1$ e $\mathcal{L}_\tau$ é a sequência de rótulos não-vazios em τ. Considere uma configuração $\mathcal{C}$ e $\mathcal{T}(\mathcal{C})$ como o conjunto de todos os traços que possuem $\mathcal{C}$ como sua configuração inicial. Os observáveis de $\mathcal{C}$ é o conjunto $\mathcal{O}(\mathcal{C}) = \{\mathcal{O}_\tau \mid \tau \in \mathcal{T}(\mathcal{C})\}$, isto é, o conjunto contendo os observáveis de todos os traços que possuem $\mathcal{C}$ como sua configuração inicial.

3.4.1 Termos Caixa-Preta

Existem algumas sutilezas envolvidas ao se lidar com *nonces* e termos encriptados cuja chave inversa o intruso ainda não possui. Considere as seguintes sequências :

$$\mathcal{L}_1 = \langle (+\mathsf{n}_1), (+\mathsf{n}_2), (+\langle \mathsf{n}_1, \mathsf{n}_2 \rangle) \rangle$$

$$\mathcal{L}_2 = \langle (+\mathsf{n}_1), (+\mathsf{n}_2), (+\langle \mathsf{n}_2, \mathsf{n}_1 \rangle) \rangle$$

Onde $\mathsf{n}_1, \mathsf{n}_2$ são *nonces*, criados por um participante honesto. O intruso é capaz de distinguir esses observáveis pela ordem do par enviado na última mensagem. No entanto, o intruso seria incapaz de diferenciar as sequências de rótulos a seguir:

$$\mathcal{L}'_1 = \langle (+\mathsf{n}_1), (+\mathsf{n}_2), (+\langle \mathsf{n}_1, \mathsf{n}_2 \rangle) \rangle$$

$$\mathcal{L}'_2 = \langle (+\mathsf{n}'_1), (+\mathsf{n}'_2), (+\langle \mathsf{n}'_1, \mathsf{n}'_2 \rangle) \rangle$$

onde $\mathsf{n}_1, \mathsf{n}'_1, \mathsf{n}_2, \mathsf{n}'_2$ são *nonces*. Isto se dá porque apesar de serem constantes diferentes, elas aparecem na mesma ordem. O mesmo se dá com termos encriptados cuja chave inversa o intruso não possui. Considere as seguintes sequências de rótulos:

$$\mathcal{L}''_1 = \langle (+\mathsf{e}(\mathsf{t}_1, \mathsf{k}_1)), (+\mathsf{e}(\mathsf{t}_2, \mathsf{k}_2)), (+\langle \mathsf{e}(\mathsf{t}_1, \mathsf{k}_1), \mathsf{e}(\mathsf{t}_2, \mathsf{k}_2) \rangle) \rangle$$

$$\mathcal{L}''_2 = \langle (+\mathsf{e}(\mathsf{t}'_1, \mathsf{k}'_1)), (+\mathsf{e}(\mathsf{t}'_2, \mathsf{k}'_2)), (+\langle \mathsf{e}(\mathsf{t}'_2, \mathsf{k}'_2), \mathsf{e}(\mathsf{t}'_1, \mathsf{k}'_1) \rangle) \rangle$$

onde $\mathsf{t}_1, \mathsf{t}_2$ são constantes diferentes e $\mathsf{k}_1, \mathsf{k}_2$ são chaves que o intruso não possui a inversa. Apesar do intruso ser incapaz de decriptar as mensagens que foram trocadas, ele ainda pode distinguir $\mathcal{L}_1''$ de $\mathcal{L}_2''$ por causa da ordem de envio da última tupla.

Definição 3.4.2. Considere $\mathcal{O} = \langle \mathsf{tt}_I, \mathcal{L}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @ \mathsf{tG}$ como um observável. O conjunto de termos caixa-preta (*black-box terms*) de $\mathcal{O}$, escrito $\mathcal{BB}(\mathcal{O})$, é o conjunto de todos os sub-termos de $\mathcal{L}$ que são *nonces*, ou termos encriptados $\mathsf{e}(\mathsf{t}, \mathsf{k})$, cuja chave inversa de k não pertença a $\mathcal{IK}$.

Definição 3.4.3. Considere $\mathcal{O}$ como um observável. Um termo simbólico ms restrito a $\mathcal{BB}(\mathcal{O})$, escrito $\mathsf{ms} \mid_{\mathcal{BB}(\mathcal{O})}$, é o termo obtido ao substituir-se pelo símbolo especial $*$ todos os sub-termos de ms que (1) não contém nenhum termo pertencente a $\mathcal{BB}(\mathcal{O})$ ou (2) não está contido em nenhum termo pertencente a $\mathcal{BB}(\mathcal{O})$.

Por exemplo, considere o termo $\mathsf{ms} = \langle \mathsf{n}_1, \mathsf{t}_1, \mathsf{e}(\langle \mathsf{t}_2, \mathsf{e}(\mathsf{t}_3, \mathsf{k}_2) \rangle, \mathsf{k}_1) \rangle$, onde o intruso conhece a inversa de k_1, mas não de k_2. Então o termo restrito a caixas-pretas é $\langle \mathsf{n}_1, *, \mathsf{e}(\langle *, \mathsf{e}(\mathsf{t}_3, \mathsf{k}_2) \rangle, *) \rangle$, onde $\mathsf{t}_1, \mathsf{t}_2$ e k_1 foram substituídos por $*$.

Definição 3.4.4. Considere $\mathcal{O} = \langle \mathsf{tt}_I, \mathcal{L}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @ \mathsf{tG}$ e $\mathcal{O}' = \langle \mathsf{tt}'_I, \mathcal{L}', \mathcal{IK}', \mathcal{DC}', \mathcal{EQ}', \mathcal{TC}' \rangle @ \mathsf{tG}'$ como dois observáveis, tal que $\mathcal{L} = \langle (\pm_1 \mathsf{ms}_1) \ldots (\pm_n \mathsf{ms}_n) \rangle$ e $\mathcal{L}' = \langle (\pm_1 \mathsf{ms}'_1) \ldots (\pm_n \mathsf{ms}'_n) \rangle$. Uma bijeção de caixa preta, escrita $\mathsf{bij}(\mathcal{O}, \mathcal{O}')$, é qualquer bijeção entre os conjuntos $\mathcal{BB}(\mathcal{O})$ e $\mathcal{BB}(\mathcal{O}')$ tal que $\mathsf{bij}(\mathcal{O}, \mathcal{O}')$ torne $\langle \mathsf{ms}_1, \ldots, \mathsf{ms}_n \rangle \mid_{\mathcal{BB}(\mathcal{O})}$ e $\langle \mathsf{ms}'_1, \ldots, \mathsf{ms}'_n \rangle \mid_{\mathcal{BB}(\mathcal{O}')}$ iguais.

Por exemplo, não existe bijeção entre os termos $\mathcal{L}_1''$ e $\mathcal{L}_2''$ mostrados acima. No entanto, a bijeção $\{\mathsf{e}(\mathsf{t}_1, \mathsf{k}_1) \leftrightarrow \mathsf{e}(\mathsf{t}'_1, \mathsf{k}'_1), \mathsf{e}(\mathsf{t}_2, \mathsf{k}_2) \leftrightarrow \mathsf{e}(\mathsf{t}'_2, \mathsf{k}'_2)\}$ tornam iguais os termos em $\mathcal{L}_1''$ e a sequência de rótulos $\mathcal{L}_2'''$ definida abaixo:

$$\mathcal{L}_2''' = \langle (+\mathsf{e}(\mathsf{t}'_1, \mathsf{k}'_1)), (+\mathsf{e}(\mathsf{t}'_2, \mathsf{k}'_2)), (+\langle \mathsf{e}(\mathsf{t}'_1, \mathsf{k}'_1), \mathsf{e}(\mathsf{t}'_2, \mathsf{k}'_2) \rangle) \rangle.$$

3.4.2 Aproximação de termos

Notação: Para o restante dessa seção, para evitar repetições e facilitar a leitura do texto, nós vamos assumir que $\mathcal{O} = \langle \mathsf{tt}_I, \mathcal{L}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @ \mathsf{tG}$ e $\mathcal{O}' =$

$\langle \mathsf{tt}'_I, \mathcal{L}', \mathcal{IK}', \mathcal{DC}', \mathcal{EQ}', \mathcal{TC}' \rangle @ \mathsf{tG}'$ são dois observáveis com conjuntos disjuntos de termos, símbolos temporais e *nonces*. Além disso, $\mathsf{bij} = \mathsf{bij}(\mathcal{O}, \mathcal{O}')$ é a bijeção entre os termos caixa-preta dos observáveis $\mathcal{O}$ e $\mathcal{O}'$.

Primeiramente, nós lidamos com caixas pretas da mesma forma que com *nonces*: dada uma bijeção com n relações:

$$\mathsf{bij} = \{\mathsf{ms}_1 \leftrightarrow \mathsf{ms}'_1, \ldots, \mathsf{ms}_n \leftrightarrow \mathsf{ms}'_n\}$$

Nós criamos n *nonces* novos $\mathsf{n}^v_1, \ldots, \mathsf{n}^v_n$ e substituímos cada ocorrência de ms_i em $\mathcal{O}$ e de ms'_i em $\mathcal{O}'$ pelo mesmo *nonce* n^v_i, onde $1 \leq i \leq n$. No que se segue, nós assumimos que essas substituições já foram feitas.

Nós definimos agora quando um termo simbólico aproxima outro:

Definição 3.4.5. Considere ms e ms' como dois termos simbólicos contidos em, respectivamente, $\mathcal{O}$ e $\mathcal{O}'$. O termo ms aproxima ms', escrito $\mathsf{ms} \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{ms}'$ se para todos os termo base $\mathsf{m} \in \mathcal{DC} \mid_{\mathcal{EQ}} (\mathsf{ms})$, então $\mathsf{m} \in \mathcal{DC}' \mid_{\mathcal{EQ}'} (\mathsf{ms}')$. Dois termos são considerados equivalentes, representado por $\mathsf{ms} \sim_{\mathcal{O},\mathcal{O}'} \mathsf{ms}'$, se, e somente se, $\mathsf{ms} \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{ms}'$ e $\mathsf{ms}' \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{ms}$.

O problema com a definição acima é a quantificação sobre todos os termos, que é um conjunto infinito. Nós mostramos na definição 3.4.7 um procedimento, chamado $\mathsf{termEqApprox}$, que checa pela equivalência observacional entre termos. Ele utiliza as funções auxiliares $\mathsf{termApprox}$ e canEq.

Nós começamos definindo a função $\mathsf{termApprox}$, que utiliza a função auxiliar symDer.

Definição 3.4.6. Considere sym' como um símbolo pertencente a $\mathcal{O}'$ e $\mathsf{dc}(\mathsf{sym}', \mathcal{S}') \in \mathcal{DC}'$ e ms um termo pertencente a $\mathcal{O}$. Nós dizemos que $\mathsf{symDer}(\mathsf{sym}', \mathsf{ms}, \mathcal{DC}, \mathcal{DC}')$ se uma das alternativas abaixo forem verdadeiras:

- ms é um termo inferível;
- ms é uma chave, então $\mathsf{ms} \in \mathcal{S}'$;
- ms é um *nonce*, então $\mathsf{ms} \in \mathcal{S}'$;
- ms for uma encriptação $\mathsf{e}(\mathsf{ms}_1, \mathsf{k}) \notin \mathcal{BB}(\mathcal{O})$, então $symDer(\mathsf{sym}', \langle \mathsf{ms}_1, \mathsf{k} \rangle, \mathcal{DC}, \mathcal{DC}')$;

- $\mathsf{ms} = \langle \mathsf{ms}_1, \ldots, \mathsf{ms}_n \rangle$, então $\mathsf{symDer}(\mathsf{sym}', \mathsf{ms}_i, \mathcal{DC}, \mathcal{DC}')$ para $1 \leq i \leq n$;
- $\mathsf{ms} = \mathsf{sym}$ é um símbolo tal que $\mathsf{dc}(\mathsf{sym}, \mathcal{S}) \in \mathcal{DC}$, então para cada $\mathsf{ms}_1 \in \mathcal{S}$, $\mathsf{symDer}(\mathsf{sym}', \mathsf{ms}_1, \mathcal{DC}, \mathcal{DC}')$.

Definição 3.4.7. Considere ms e ms' termos pertencentes a, respectivamente, $\mathcal{O}$ e $\mathcal{O}'$. O predicado $\mathsf{termApprox}(\mathsf{ms}, \mathsf{ms}', \mathcal{O}, \mathcal{O}')$ é verdadeiro se, e somente se, ambas as condições descritas abaixo forem satisfeitas:

1. Existir uma substituição correspondente $\theta = \{\mathsf{sym}'_1 \mapsto \mathsf{ms}_1, \ldots, \mathsf{sym}'_n \mapsto \mathsf{ms}_n\}$ que mapeia símbolos em ms' para sub-termos em ms, de forma que $\theta(\mathsf{ms}') = \mathsf{ms}$;
2. Para cada $\mathsf{sym}'_i \mapsto \mathsf{ms}_i \in \theta$, nós temos $\mathsf{symDer}(\mathsf{sym}'_i, \mathsf{ms}_i, \mathcal{DC}, \mathcal{DC}')$ (Definição 3.4.6)

Nós provamos a corretude e completude para as funções symDer e $\mathsf{termApprox}$:

Teorema 3.4.1. *Considere dois termos,* ms *e* ms'*. Então para todo* $\mathsf{m} \in \mathcal{DC}(\mathsf{ms})$*, nós temos que* $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}')$ *se e somente se* $\mathsf{termApprox}(\mathsf{ms}, \mathsf{ms}', \mathcal{DC}, \mathcal{DC}')$ *(demonstração no Apêndice A).*

Exemplo: Nós ilustramos o procedimento $\mathsf{termApprox}$ com um exemplo. Considere

$$\mathsf{ms} = \langle \mathsf{n}, \mathsf{e}(\langle \mathsf{n}, \mathsf{t}, \mathsf{sym}_1, \mathsf{sym}_2 \rangle, \mathsf{k}_1) \rangle$$
$$\mathsf{ms}' - \langle \mathsf{n}', \mathsf{e}(\mathsf{sym}', \mathsf{k}_1) \rangle$$

Além disso, assumimos que o intruso conhece a chave inversa de k_1 em ambos os observáveis, isto é, $\mathsf{k}^{-1} \in \mathcal{IK} \cap \mathcal{IK}'$, e considere a bijeção $\mathsf{bij} = \mathsf{bij}(\mathcal{O}, \mathcal{O}') = \{\mathsf{n} \leftrightarrow \mathsf{n}'\}$. Nós substituímos esses *nonces* pelo *nonce* recém-criado n^v.

Nós checamos pela primeira condição da Definição 3.4.7, isto é, construir uma substituição mapeando os símbolos de ms' para os sub-termos em ms. Nós obtemos a seguinte substituição correspondente:

$$\theta = \{\mathsf{sym}' \mapsto \langle \mathsf{n}^v, \mathsf{t}, \mathsf{sym}_1, \mathsf{sym}_2 \rangle\}$$

, com $\theta[\mathsf{ms}'] = \langle \mathsf{n}^v, \mathsf{e}(\langle \mathsf{n}^v, \mathsf{t}, \mathsf{sym}_1, \mathsf{sym}_2 \rangle, \mathsf{k}_1) \rangle$.

Perceba que se tal substituição não existir, então ms não pode aproximar ms'. De fato, $\mathsf{ms}' \not\leq_{\mathcal{O},\mathcal{O}'} \mathsf{ms}$ já que não existe uma substituição correspondente (*matching substitution*). O termo ms' pode ser instanciado para um termo onde uma não-tupla está encriptada, *e.g.*, $\langle \mathsf{n}_1, \mathsf{e}(\mathsf{t}), \mathsf{k}_1 \rangle$, onde t é um termo inferível, enquanto m só pode ser instanciado por termos em que uma tupla encriptada com pelo menos quatro elementos.

Agora nós checamos pela condição 2. Para fazer isso, nós precisamos saber mais sobre o conteúdo de $\mathcal{DC}$ e $\mathcal{DC}'$. Assuma que o símbolo sym_1 é criado antes do símbolo sym_2 no traço correspondente a $\mathcal{O}$. Então, pela proposição 3.3.2, nós temos que se $\mathsf{dc}(\mathsf{sym}_1, \mathcal{S}_1), \mathsf{dc}(\mathsf{sym}_2, \mathcal{S}_2) \in \mathcal{DC}$, $\mathcal{S}_1 \subseteq \mathcal{S}_2$.

Considere $\mathsf{dc}(\mathsf{sym}_1, \{\mathsf{t}_1, \mathsf{k}_1\}), \mathsf{dc}(\mathsf{sym}_2, \{\mathsf{t}_1, \mathsf{t}_2, \mathsf{k}_1, \mathsf{k}_2\}) \in \mathcal{DC}$, o que satisfaz a condição acima. Além disso, assuma que $\mathsf{dc}(\mathsf{sym}', \mathcal{S}') \in \mathcal{DC}'$ onde $\mathcal{S}' = \langle \mathsf{n}^v, \mathsf{t}_1, \mathsf{t}_2, \mathsf{k}_1, \mathsf{k}_2 \rangle$.

Nós então temos que $\mathsf{symDer}(\mathsf{sym}', \{\mathsf{n}^v, \mathsf{sym}_1, \mathsf{sym}_2\}, \mathcal{DC}, \mathcal{DC}')$ como:

- $\mathsf{symDer}(\mathsf{sym}', \mathsf{n}^v, \mathcal{DC}, \mathcal{DC}')$ já que $\mathsf{n}^v \in \mathcal{S}'$;
- $\mathsf{symDer}(\mathsf{sym}', \mathsf{sym}_1, \mathcal{DC}, \mathcal{DC}')$ já que $\{\mathsf{t}_1, \mathsf{k}_1\} \subseteq \mathcal{S}'$;
- $\mathsf{symDer}(\mathsf{sym}', \mathsf{sym}_2, \mathcal{DC}, \mathcal{DC}')$ já que $\langle \mathsf{n}^v, \mathsf{t}_1, \mathsf{t}_2, \mathsf{k}_1, \mathsf{k}_2 \rangle \subseteq \mathcal{S}'$;

A definição a seguir especifica uma função chamada canEq. Intuitivamente, canEq avalia se é possível instanciar, respeitando $\mathcal{DC}$ e $\mathcal{EQ}$, os símbolos $\mathsf{ms}_1, \mathsf{ms}_2$ de forma a falsear a restrição $\mathsf{neq}(\mathsf{ms}_1, \mathsf{ms}_2)$.

Definição 3.4.8. $\mathsf{canEq}(\mathsf{ms}_1, \mathsf{ms}_2, \mathcal{DC}, \mathcal{EQ})$ resulta em verdadeiro se e somente se as seguintes condições forem satisfeitas :

1. Existir um unificador $\sigma = [\mathsf{sym}_1 \mapsto \mathsf{ms}_1^*, \ldots, \mathsf{sym}_n \mapsto \mathsf{ms}_n^*]$ de ms_1 e ms_2;
2. σ não falsear nenhuma restrição de comparação pertencente a $\mathcal{EQ}$;
3. Para cada $\mathsf{sym}_i \mapsto \mathsf{ms}_i^* \in \sigma$, sym_i poder gerar o termo ms_i^* respeitando $\mathcal{DC}$.

Definição 3.4.9. Considere ms e ms' como termos pertencentes a, respectivamente, $\mathcal{O}$ e $\mathcal{O}'$. O predicado $\mathsf{termEqApprox}(\mathsf{ms}, \mathsf{ms}', \mathcal{O}, \mathcal{O}')$ é verdadeiro se e somente se:

1. Se $\mathsf{eqCheck}(\mathcal{EQ}, \mathcal{DC}) = false$;

2. Caso contrário, se $\mathsf{eqCheck}(\mathcal{EQ}, \mathcal{DC}) = \mathsf{eqCheck}(\mathcal{EQ}', \mathcal{DC}') = \mathit{true}$, considere que ssb e ssb' são as substituições testemunhas correspondentes (*witnessing matching substitution*) resultantes (Definição 3.3.8);

   (a) $\mathsf{termApprox}(\mathsf{ssb}[\mathsf{ms}], \mathsf{ssb}'[\mathsf{ms}'], \mathsf{ssb}[\mathcal{DC}], \mathsf{ssb}'[\mathcal{DC}'])$ é verdadeiro, onde a substituição correspondente resultante (*witnessing matching substitution*) é $\theta = [\mathsf{sym}'_1 \mapsto \mathsf{ms}_1, \ldots, \mathsf{sym}'_n \mapsto \mathsf{ms}_n]$ (Definição 3.4.7);

   (b) $\mathsf{canEq}(\mathsf{ssb}[\theta[\mathsf{ms}'_1]], \mathsf{ssb}[\theta[\mathsf{ms}'_2]], \mathcal{DC}, \mathcal{EQ})$ é falso (Definição 3.4.8) para cada restrição de desigualdade $\mathsf{neq}(\mathsf{ms}'_1, \mathsf{ms}'_2) \in \mathcal{EQ}'$.

Intuitivamente, a primeira condição especificada na Definição 3.4.9 especifica que se $\mathcal{EQ}, \mathcal{DC}$ não são consistentes, então a aproximação é trivial. Caso contrário, se tanto $\mathcal{EQ}, \mathcal{DC}$ quanto $\mathcal{EQ}', \mathcal{DC}'$ são consistentes, então existem substituições correspondentes ssb e ssb'. Nós aplicamos ssb e ssb' sobre, respectivamente, $\mathcal{O}$ e $\mathcal{O}'$ a seguir. Isso resolve todas as restrições de igualdade . A condição 2a avalia se os termos que são gerados são os mesmos. A condição 2b avalia se não é possível falsear alguma restrição de desigualdade pertencente a $\mathcal{EQ}'$ utilizando tanto $\mathcal{DC}$ quanto $\mathcal{EQ}$.

Teorema 3.4.2. *Considerando* ms *e* ms' *como dois termos pertencentes a, respectivamente,* $\mathcal{O}$ *e* $\mathcal{O}'$*. Então* $\mathsf{ms} \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{ms}'$ *e e somente se* $\mathsf{termEqApprox}(\mathsf{ms}, \mathsf{ms}', \mathcal{O}, \mathcal{O}')$ *(demonstração no Apêndice A)*

Exemplo Considere a função protocolar de Alice no Exemplo 3.3.2. Nós omitimos a restrição temporal já que elas não são importantes para este exemplo em particular. A partir de uma configuração inicial com Alice e um intruso chamado *eve*, nós podemos construir um observável $\mathcal{O}'$:

$$\langle +\mathsf{e}(\langle \mathsf{n}'_1, \mathit{alice}\rangle, \mathsf{pk}(\mathit{eve})), -\mathsf{sym}', +\mathit{error}\rangle,$$
$$\mathcal{IK}', \mathcal{DC}', \{\mathsf{neq}(\mathsf{sym}', \mathsf{e}(\langle n_1, \mathsf{v}\rangle, \mathsf{pk}(\mathit{alice})))\}$$

para um conhecimento do intruso $\mathcal{IK}'$ e $\mathcal{DC}'$.

Considere a função protocolar que retorna o seguinte erro:

$$\mathit{Alice}' \coloneqq (\mathsf{new}\ N_a), (+\mathsf{e}(\langle N_a, \mathit{alice}\rangle, \mathsf{pk}(Z))), (-\mathsf{v}), (+\mathit{error})$$

Considere o observável $\mathcal{O}$ para esse protocolo:

$$\langle +\mathsf{e}(\langle \mathsf{n}_1, alice\rangle, \mathsf{pk}(eve)), -\mathsf{sym}, +error\rangle, \mathcal{IK}, \mathcal{DC}, \emptyset$$

que é bem similar ao observável $\mathcal{O}'$ mostrado acima, mas sem as restrições de comparação. Existe a bijeção $\mathsf{n}_1 \leftrightarrow \mathsf{n}'_1$. Além disso, como a mensagem é enviada para $eve, \mathsf{n}_1 \in \mathcal{IK}$ e $\mathsf{n}'_1 \in \mathcal{IK}'$.

Perceba que:

$$\begin{aligned}\mathsf{termApprox}(&\langle \mathsf{e}(\langle \mathsf{n}_1, alice\rangle, \mathsf{pk}(eve)), \mathsf{sym}, error\rangle,\\ &\langle \mathsf{e}(\langle \mathsf{n}'_1, alice\rangle, \mathsf{pk}(eve)), \mathsf{sym}', error\rangle, \mathcal{DC}, \mathcal{DC}')\end{aligned}$$

já que as mensagens são iguais. No entanto, devido as restrições de comparação presentes em $\mathcal{O}$, não e o caso que

$$\begin{aligned}\mathsf{termEqApprox}(&\langle \mathsf{e}(\langle \mathsf{n}_1, alice\rangle, \mathsf{pk}(eve)), \mathsf{sym}, error\rangle,\\ &\langle \mathsf{e}(\langle \mathsf{n}'_1, alice\rangle, \mathsf{pk}(eve)), \mathsf{sym}', error\rangle, \mathcal{O}, \mathcal{O}')\end{aligned}$$

De fato, a função $\mathsf{canEq}(\mathsf{sym}, \mathsf{e}(n_1, \mathsf{t}), \mathcal{DC}, \emptyset)$ é verdadeira. (Relembre que as variáveis são substituídas por símbolos recém-criados.) Isso acontece porque o intruso conhece a chave pública de Alice e conhece o valor de n_1.

3.4.3 Tempo das Mensagens

Nós reduzimos as fórmulas para fórmulas que resolvedores SMT existentes conseguem lidar [Dutertre 2015], mais especificamente fórmulas da forma $\exists\forall$:

$$\begin{aligned}&\forall\widetilde{\mathsf{tt}}.\left[\mathcal{TC} \Rightarrow \exists\widetilde{\mathsf{tt}'}.\left[\mathcal{TC}' \wedge \mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \cdots \wedge \mathsf{tG}_N = \mathsf{tG}'_N\right]\right] \text{ é uma tautologia}\\ &\Leftrightarrow \neg\forall\widetilde{\mathsf{tt}}.\left[\mathcal{TC} \Rightarrow \exists\widetilde{\mathsf{tt}'}.\left[\mathcal{TC}' \wedge \mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \cdots \wedge \mathsf{tG}_N = \mathsf{tG}'_N\right]\right] \text{ é unsat}\\ &\Leftrightarrow \exists\widetilde{\mathsf{tt}}.\left[\mathcal{TC} \wedge \forall\widetilde{\mathsf{tt}'}.\left[\mathcal{TC}' \Rightarrow \neg\left[\mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \cdots \wedge \mathsf{tG}_N = \mathsf{tG}'_N\right]\right]\right] \text{ é unsat}\end{aligned}$$

Na nossa implementação, nós utilizamos o resolvedor SMT Yices para resolver essa fórmula(que é decidível [Dutertre 2015]), onde todas as variáveis de tempo possuem o tipo Reals.

Lemma 3.4.1. *Determinar se dois observáveis temporais são equivalentes é decidível.*

Junto com a proposição 3.3.1, nós obtemos a decidabilidade da verificação da equivalência observacional temporal.

Lemma 3.4.2. *Determinar se duas configurações são observacional-temporalmente equivalentes é decidível.*

3.4.4 Definição de equivalência observacional

Duas configurações são observacionalmente equivalentes se os seus observáveis forem equivalentes.

Definição 3.4.10. Considere $\mathcal{O} = \langle \mathsf{tt}_I, \mathcal{L}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @ \mathsf{tG}$ e $\mathcal{O}' = \langle \mathsf{tt}'_I, \mathcal{L}', \mathcal{IK}', \mathcal{DC}', \mathcal{EQ}', \mathcal{TC}' \rangle @ \mathsf{tG}'$ como dois observáveis, tal que

$$\mathcal{L} = \langle (\pm_1 \mathsf{ms}_1 @ \mathsf{tG}_1) \dots (\pm_p \mathsf{ms}_p @ \mathsf{tG}_p) \rangle, \mathcal{L}' = \langle (\pm'_1 \mathsf{ms}'_1 @ \mathsf{tG}'_1) \dots (\pm'_n \mathsf{ms}'_n @ \mathsf{tG}'_n) \rangle$$

A observação $\mathcal{O}$ é equivalente a $\mathcal{O}'$, escrito $\mathcal{O} \sim \mathcal{O}'$ se as seguintes condições forem verdadeiras:

1. $p = n = N$, isto é, ambas as observações possuem o mesmo tamanho N.
2. $\pm_i = \pm'_i$, para todo $1 \leq i \leq N$, isto é, possuem o mesmo tipo de rótulo;
3. Existe uma bijeção de caixa-preta (*black-box bijection*) $\mathsf{bij}(\mathcal{O}, \mathcal{O}')$ (Definição 3.4.4) entre $\mathcal{O}$ e $\mathcal{O}'$;
4. As mensagens observadas são equivalentes, isto é $\langle \mathsf{ms}_1, \dots, \mathsf{ms}_N \rangle \sim_{\mathcal{O},\mathcal{O}'} \langle \mathsf{ms}'_1, \dots, \mathsf{ms}'_N \rangle$ (Definição 3.4.5);
5. Assuma que $\widetilde{\mathsf{tt}}$ e $\widetilde{\mathsf{tt}'}$ são o conjunto de símbolos temporais em $\mathcal{TC}$ e $\mathcal{TC}'$, respectivamente. Esses conjuntos de símbolos temporais são considerados disjuntos sem perda de generalidade. As fórmulas a seguir são tautologias:

$$\forall \widetilde{\mathsf{tt}}. \left[\mathcal{TC} \Rightarrow \exists \widetilde{\mathsf{tt}'}. \left[\mathcal{TC}' \wedge \mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \dots \wedge \mathsf{tG}_N = \mathsf{tG}'_N \right] \right]$$
$$\forall \widetilde{\mathsf{tt}'}. \left[\mathcal{TC}' \Rightarrow \exists \widetilde{\mathsf{tt}}. \left[\mathcal{TC} \wedge \mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \dots \wedge \mathsf{tG}_N = \mathsf{tG}'_N \right] \right]$$

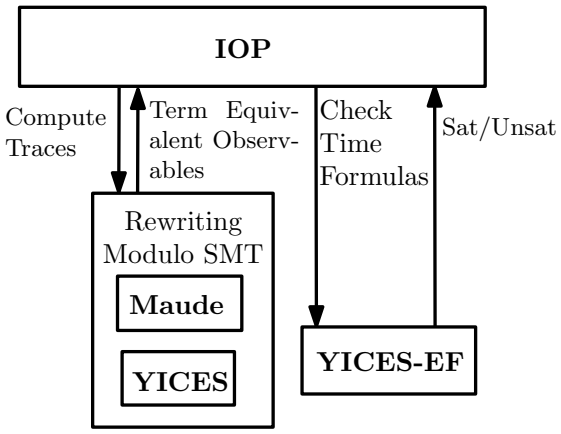

Figura 3.4: Arquitetura do Verificador de Equivalência Observacional Temporal.

especificando que para qualquer tempo $\mathsf{tG}_1, \ldots, \mathsf{tG}_N$ que satisfaça $\mathcal{TC}$ quando as mensagens $\mathsf{ms}_1, \ldots, \mathsf{ms}_N$ foram observadas em $\mathcal{O}$, nós podemos encontrar $\mathsf{tG}'_1, \ldots, \mathsf{tG}'_N$ que satisfaça $\mathcal{TC}'$ e que torne o tempo da observação igual.
As primeiras duas condições são claras. Se dois observáveis diferem no número de observações ou em seus tipos, então eles podem ser diferenciados. Nós definimos a seguir as condições 3 e 4, que governam a bijeção de caixa-preta e aproximação de termos. Finalmente, a condição 5 especifica que o intruso não dever ser capaz de distinguir as mensagens observadas medindo o tempo da observação.

Definição 3.4.11. Uma configuração $\mathcal{C}$ aproxima uma configuração $\mathcal{C}'$, representado por $\mathcal{C} \preceq \mathcal{C}'$, se para cada $\mathcal{O} \in \mathcal{O}(\mathcal{C})$ existe uma observação equivalente $\mathcal{O}' \in \mathcal{O}(\mathcal{C}')$, isto é, $\mathcal{O} \sim \mathcal{O}'$ (Definição 3.4.10). As configurações são observacionalmente equivalentes, representado por $\mathcal{C} \sim \mathcal{C}'$, se e apenas se $\mathcal{C} \preceq \mathcal{C}'$ e $\mathcal{C}' \preceq \mathcal{C}$.

3.5 Resultados Experimentais

Nós implementamos uma ferramenta que verifica equivalência observacional temporal. A sua arquitetura é representada pela Figura 3.4. É construída sobre as ferramentas:

- **Maude:** Nós implementamos em Maude todo o maquinário necessário para a especificação de procolos temporais assim como a verificação da equivalência de termos entre os observáveis. Além disso, desde a versão alpha 111, Maude permite

chamar o resolvedor SMT CVC4 [Barrett et al. 2011] de dentro de programas Maude. Isso permite a implementação de *Rewriting Modulo SMT* ao utilizar regras de reescrita condicionais cuja execução é permitida se e somente se o conjunto resultante de restrições for satisfatível. Para a nossa aplicação, isso quer dizer o conjunto de restrições de tempo;

- (YICES-EF:) Já que o CVC4 não provém uma API para a verificação de fórmulas $\exists\forall$ necessárias para provar equivalência temporal, nós integramos o nosso maquinário em Maude com o resolvedor SMT YICES-EF que é capaz de verificar a satisfatibilidade de tais fórmulas. Essa integração aconteceu utilizando o *framework* IOP [Mason e Talcott 2004].

A verificação é coordenada pela implementação em IOP. Dadas duas configurações iniciais $\mathcal{C}$ e $\mathcal{C}'$ para serem avaliadas por equivalência observacional temporal, IOP envia um comando para o Maude+CVC4 para enumerar todos os observáveis de $\mathcal{C}$ e $\mathcal{C}'$, computar para cada observável $\mathcal{O}$ de $\mathcal{C}$ o conjunto de observáveis termo-equivalentes (*term equivalents*) $\{\mathcal{O}'_1, \ldots, \mathcal{O}'_n\}$ de $\mathcal{C}'$ e vice versa. Se o Maude encontrar algum observável de $\mathcal{C}$ que não possuem pelo menos um par correspondente, isto é, $n = 0$, então $\mathcal{C}$ e $\mathcal{C}'$ não são equivalentes. O mesmo acontece para algum par de $\mathcal{C}'$. Caso contrário, nós continuamos para a verificação da condição de equivalência temporal. IOP tenta encontrar pelo menos um observável temporalmente equivalente $\mathcal{O}'_i$ para $\mathcal{O}$. Se ele não for capaz disso, então $\mathcal{C}$ e $\mathcal{C}'$ não são equivalentes. O maquinário realiza essa checagem construindo a fórmula como foi descrito na seção 3.4.3 para checar por equivalência temporal de $\mathcal{O}$ com $\mathcal{O}'_i$. Se o YICES-EF retornar Unsat, eles são equivalentes. Caso contrário, eles não são equivalentes e o IOP tenta com outro observável.

Resultados Experimentais Nós realizamos os seguintes experimentos:

- **Exemplo do Redpill:** Considere o protocolo especificado no exemplo 3.2.6. Nós checamos se é possível para um intruso distinguir se uma aplicação está sendo executada ou não em uma máquina virtual. Isto é, nós checamos se uma

Cenário	Resultados	Observáveis	Estados
Red-Pill	Não Equiv	19/19	74/74
Passporte	Não Equiv	36/27	138/112
Passport-Corrigido	Equiv	36/27	138/112
Anônimo	Não Equiv	2/3	7/9

Tabela 3.1: Resultados Experimentais. Cada experimento envolve provar a equivalência observacional temporal de duas configurações. A tabela acima representa o resultado da análise, o número de observáveis(traços) para cada configuração e o número total de estados na árvore de busca que é preciso percorrer para enumerar todos os observáveis.

configuração inicial com um participante executando uma aplicação em uma máquina virtual é temporalmente equivalente a uma configuração inicial em que um participante executa a mesma aplicação em uma máquina concreta.

- **Exemplo do Passaporte:** Considere o protocolo BAC especificado no exemplo 3.2.5. Nós avaliamos se o intruso era capaz de distinguir as seguintes configurações, ambas com duas sessões de protocolo: Na primeira configuração, ambas as sessões são executadas com o mesmo passaporte, onde na segunda eles são distintos.

- **Exemplo do Passaporte Corrigido:** Adicionalmente, nós consideramos uma modificação no protocolo do exemplo do Passaporte, onde ele necessariamente manda as mensagens de erro no mesmo momento.

- **Protocolo de Grupo Anônimo:** Considere o protocolo especificado no exemplo 3.2.7. Nós avaliamos se era possível para um intruso distinguir se dois participantes pertencem ou não ao mesmo grupo. Isto é, checamos se dada uma configuração inicial em que o participante recebe uma mensagem de um membro do mesmo grupo é temporalmente equivalente a uma configuração em que o participante recebe uma mensagem de um participante de um grupo distinto.

A Tabela 3.1 sumariza os resultados dos nossos experimentos. Nossa ferramenta

foi capaz de (corretamente) identificar os casos em que as configurações dadas eram observacionalmente equivalentes. Mais importante, no entanto, é o número de estados e observáveis percorridos para fazê-lo. Em todos os nossos experimentos, o número de estados em toda a árvore de busca foi menor do que 140, e o número de observáveis menor do que 40. Esse é um número muito pequeno, se comparado a aplicações convencionais em Maude (que pode lidar com milhares de estados, mesmo utilizando *Rewriting Modulo SMT* [Nigam, Talcott e Urquiza 2016]). Isso demonstra a vantagem de representar o tempo simbolicamente. Como esperado, o número de observáveis do exemplo do passaporte foi maior já que a sua configuração possuía duas sessões de protocolo, enquanto os exemplos restantes possuíam apenas uma sessão cada. Finalmente, já que o número de observáveis foi pequena, o número de chamadas para o Yices também foi pequena, de forma que a verificação de todos experimentos demorou menos do que alguns segundos. A implementação pode ser encontrada em https://github.com/SRICSL/VCPublic/obseq.git.

Capítulo 4

Equivalência Observacional de Tráfego de Pacotes

Ataques envolvendo análise de tráfego como canal lateral para um intruso obter conhecimento que devia ser privado não é novidade na literatura, mas tem ganhado mais relevância recentemente com o crescimento de IoT e da miniaturização e barateamento do *hardware*.

Por exemplo, nós temos o caso descrito em [Apthorpe Noah Reisman 2017], onde o intruso utiliza uma análise de tráfego para discernir não apenas quais dispositivos se encontram dentro de uma *smart home*, como inferir o comportamento de tais dispositivos. Outro exemplo é o descrito por [Ying, Makrakis e Mouftah 2014], em que os autores mencionam a possibilidade de utilização de análise de tráfego em redes WSN (*Wireless Sensor Network*) para conseguir determinar a localização, identidade e funcionalidade do nó sumidouro (*sink node*).

Perceber esse tipo de problema consiste na mesma lógica utilizada sobre ataques envolvendo o tempo, descritas no Capítulo 3. Simplesmente procurar por um estado em que uma determinada condição seja satisfeita não costuma funcionar nesse cenário, já que muitas vezes não existe nada intrinsecamente errado com um estado, quando analisado isoladamente. É a comparação entre uma determinada configuração e outra, e a presença de um comportamento restrito a uma delas, que resulta em uma falha de segurança.

Nesse capítulo, nós abordamos:

- (Exemplos:) buscamos falar dos exemplos motivadores para o nosso trabalho, junto com as soluções fornecidas. Tentamos também relacionar tais exemplos com a nossa premissa de equivalência observacional.

- **Semântica Operacional:** buscamos definir a nossa equivalência observacional de tráfego, baseado em um sistema de blocos temporais. Intuitivamente, nós adicionamos restrições de tráfego a cada um dos blocos temporais, na forma de fórmulas SMT. Por fim, analisamos a equivalência observacional verificando a satisfatibilidade de fórmulas do tipo $\exists\forall$.

- **Implementação e Resultados**: comentamos sobre a implementação de alguns dos exemplos comentados como prova de conceito, utilizando o resolvedor SMT Z3 ([Z3]) como ferramenta principal, tanto em sua forma Web quanto na sua integração com Python por meio da API Z3Py.

4.1 Exemplos

Nessa seção, vamos discutir alguns exemplos que motivaram esse trabalho.

O principal motivador foi o artigo de [Apthorpe Noah Reisman 2017]. Nele, como já falamos um pouco no Capítulo 2, um intruso realiza um ataque sobre dispositivos IoT em duas partes: isolar os fluxos de pacotes e identificar os dispositivos que produzem tais fluxos, e inferir a atividade dos ditos dispositivos realizando análise de tráfego. A identificação dos dispositivos pode ser feita utilizando outros fatores que não se relacionam com a análise de tráfego, através de requisições DNS e verificação do endereço MAC, por exemplo. No entanto, a utilização desses outros fatores pode ser usado apenas para a redução do espaço de busca, restringindo os possíveis dispositivos emissores. Então, a análise de tráfego é utilizado para terminar de identificar o dispositivo em questão. Uma vez descoberta a identidade dos dispositivos, o atacante pode continuar analisando o fluxo de pacotes para determinar o comportamento de tal dispositivo. Uma câmera de detecção de movimento, por exemplo, exibia um pico no envio de pacotes quando detectava alguma coisa. Perceba que a câmera enviar uma grande quantidade de pacotes quando detecta movimento, ou não enviar tantos dados

quando ociosa, não configura uma falha de segurança por si só. A disparidade entre os dois estados, no entanto, permite distinguir um estado do outro e então inferir a atividade do dispositivo.

Uma situação similar é descrita em [Ying, Makrakis e Mouftah 2014], onde os autores relatam um método para frustrar análises de tráfego para descobrir a localização do nó sumidouro em redes WSN. Quando se trata de redes de sensores, os nós precisam enviar os dados coletados para o sumidouro, onde eles serão processados e armazenados. Analisando o padrão de tráfego nos nós, a medida que eles se aproximam do sumidouro, o tráfego de pacotes que passa por aquela nó aumenta. Seguindo o rastro de aumento de tráfego, é possível localizar o sumidouro.

As soluções propostas para ambos os problemas envolvem um *trade-off*: em ambos os casos pacotes chamariz (*decoy packages*) são adicionados a rede. No caso do IoT [Apthorpe Noah Reisman 2017], os autores propõem a utilização de um modelador de tráfego (*traffic shaper*), em que a saída de pacotes da rede é fixada, adicionando pacotes falsos ou inserindo pacotes em uma fila de acordo com a saída dos dispositivos IoT. Nesse caso, se o modelador estivesse fixando uma vazão muito baixa de pacotes, teria pouco impacto sobre a largura de banda, mas aumentaria consideravelmente a latência. Por outro lado, uma taxa de vazão muito alta praticamente não impactaria na latência da rede, mas exigiria muito mais largura de banda. Além disso, o tamanho dos pacotes emitidos também é padronizado.

Uma escolha similar é proposta em [Ying, Makrakis e Mouftah 2014]. O artigo propõe uma versão melhorada de uma ferramenta de ATA (*anti-traffic analysis*) que utiliza uma heurística para que alguns outros nós da rede emulem o comportamento do nó sumidouro. Eles fazem isso através da emissão de pacotes chamariz. Em outro algoritmo mencionado no artigo, os nós da rede geram uma quantidade de pacotes falsos relacionada com a quantidade de nós-filho daquele nó em particular. Nesse caso, processamento e envio de pacotes adicionais aumenta o consumo de energia, o que pode ser um problema em uma rede de sensores.

Em ambos os problemas, a adição de tráfego adicional é necessária para ofuscar o intruso, e tornar os dois estados observacionalmente equivalentes. Na elaboração da nossa semântica operacional, nós propomos uma modelagem para determinar a

eficiência da inserção de tráfego para a aquisição de equivalência observacional.

4.2 Semântica Operacional

Nós simplificamos os nossos participantes na hora de avaliar a equivalência de tráfego. Nosso intruso não é capaz de visualizar o conteúdo das mensagens, capaz apenas de analisar quantos pacotes foram enviados em um dado instante.

Dado uma assinatura de tráfego Ξ, contendo um conjunto de números naturais $\mathsf{r}_1, \mathsf{r}_2, \ldots$, um conjunto de variáveis de tráfego $\mathsf{b}_1, \mathsf{b}_2, \ldots$, e um conjunto pré-definido de símbolos de funções, incluindo $+, -, =, \leq, \geq$.

Definição 4.2.1. *Restrições de Tráfego* são construídas indutivamente a partir de $\mathsf{r}_1, \mathsf{r}_2, \ldots$ e de $\mathsf{b}_1, \mathsf{b}_2, \ldots$, aplicando símbolos de funções sobre restrições de tráfego. Então, $\mathsf{b} \leq \mathsf{b}_2 - 5$, $\mathsf{b} \geq 8 + \mathsf{b}_2$, $\mathsf{b}_1 = \mathsf{b}_2$ são exemplos de restrições de tráfego.

Definição 4.2.2. Considere $\mathsf{b}_1, \ldots, \mathsf{b}_n$ variáveis de tráfego e $\mathsf{r}_1, \ldots, \mathsf{r}_{1n}$ números naturais. Dizemos que uma substituição $\theta = [\mathsf{b}_1 \mapsto \mathsf{r}_1, \ldots, \mathsf{b}_n \mapsto \mathsf{r}_n]$ respeita um conjunto de restrições de tráfego $\mathcal{RT}$, escrito $\theta \mid_{\mathcal{RT}}$, se e somente se $\theta[\mathcal{RT}]$ não falseia nenhuma restrição de tráfego $\mathsf{rt} \in \mathcal{RT}$.

Por exemplo, tomemos o seguinte $\mathcal{RT}_1$:

$$\mathcal{RT}_1 = \{\mathsf{b}_1 \leq 5, \quad \mathsf{b}_1 \geq 0, \quad \mathsf{b}_2 \leq 8, \quad \mathsf{b}_2 \geq 4\}$$

Nesse caso, a substituição $\theta_1 = [\mathsf{b}_1 \mapsto 3, \ \mathsf{b}_2 \mapsto 6]$ respeita $\mathcal{RT}_1$ já que $\theta[\mathcal{RT}_1] = \{3 \leq 5, \quad 3 \geq 0, \quad 6 \leq 8, \quad 6 \geq 4\}$ não contém nenhuma restrição de tráfego falsa. Por outro lado, a substituição $\theta_2 = [\mathsf{b}_1 \mapsto 6]$ falseia a restrição $\mathsf{b}_1 \leq 5$ e, por isso, não respeita $\mathcal{RT}_1$.

Definição 4.2.3. Um fluxo $\mathcal{F}$ é uma tupla $\langle \mathsf{b}_1, \ldots, \mathsf{b}_n \rangle$, onde $\mathsf{b}_1, \ldots, \mathsf{b}_n$ são variáveis de tráfego.

Definição 4.2.4. Dado um conjunto de restrições de tráfego $\mathcal{RT}$, dizemos que $Var(\mathsf{b}) \in \mathcal{RT}$ se existe uma restrição de tráfego rt que contém a variável b, tal que $\mathsf{rt} \in \mathcal{RT}$. Da mesma forma, dado um conjunto de fluxos $\mathcal{FS}$, dizemos que $Var(\mathsf{b}) \in \mathcal{FS}$

se $\mathcal{FS} = \mathcal{FS}_0 \cup \{\mathcal{F}\}$, tal que $\mathsf{b} \in \mathcal{F}$, *i.e.*, b é uma variável de tráfego presente em algum fluxo contido em $\mathcal{FS}$.

Definição 4.2.5. Uma configuração $\mathcal{C}$ é uma tupla $\langle \mathcal{FS}, \mathcal{RT} \rangle$, onde $\mathcal{FS}$ é um conjunto de fluxos e $\mathcal{RT}$ um conjunto de restrições de tráfego. Além disso, consideramos que $Var(\mathsf{b}) \in \mathcal{RT}$ se e somente se $Var(\mathsf{b}) \in \mathcal{FS}$. *i.e.*, as variáveis contidas em $\mathcal{RT}$ provém de $\mathcal{FS}$.

Definição 4.2.6. Um traço $\mathcal{T}$ é uma tupla $\langle \mathsf{r}_1, \ldots, \mathsf{r}_n \rangle$ de números naturais.

Definição 4.2.7. Dado um fluxo $\mathcal{F}$ e um conjunto de restrições $\mathcal{RT}$, nós dizemos que um traço $\mathcal{T} \in \mathcal{F}$ respeitando $\mathcal{RT}$, se existe uma substituição θ que $\theta[\mathcal{F}] = \mathcal{T}$ e $\theta \mid_{\mathcal{RT}}$.

Definição 4.2.8. O conjunto $\mathcal{T}(\mathcal{F}) \mid_{\mathcal{RT}}$ é o conjunto de todos os possíveis traços $\mathcal{T} \in \mathcal{F}$ que respeitam $\mathcal{RT}$.

Intuitivamente, cada elemento de um fluxo é um intervalo fixo de tempo. Quando dois fluxos estão na mesma configuração, assumimos que a duração dos intervalos é o mesmo em ambos os fluxos. As variáveis de tráfego associadas a cada intervalo de tempo, restritas por $\mathcal{RT}$, representam os possíveis valores de tráfego que aquele determinado intervalo de tempo pode assumir. Um traço representa uma instância específica de um fluxo definido por um conjunto $\mathcal{RT}$.

Exemplo Vamos tomar como exemplo a Figura 4.1, que ilustra a atividade da câmera Nest Security. Perceba que sempre que a câmera entra no modo de *streaming*, ocorre um pico de pacotes, e depois um aumento na taxa de envio até esse modo ser alterado. Nós poderíamos utilizar um fluxo $\mathcal{F}_1$ restrito por $\mathcal{RT}$ que representa o modo *live stream*,

$$\mathcal{F}_1 = \langle \mathsf{b}_1, \overbrace{\mathsf{b}_2, \mathsf{b}_2, \ldots, \mathsf{b}_2}^{n} \rangle$$
$$\mathcal{RT} = \{(, \mathsf{b}_1 \leq 48), \quad (\mathsf{b}_1 \geq 28), \quad (\mathsf{b}_2 \leq 20), \quad (\mathsf{b}_2 \geq 8)\}$$

onde a variável de tráfego b_1 representaria o pico de tráfego, o momento em que a câmera muda de um modo para o outro, e o b_2 representaria o tráfego normal do modo *live stream*. Perceba também que o fluxo $\mathcal{F}_1$ possui mais de uma variável b_2. Isso se dá porque como cada elemento do fluxo representa um intervalo de tempo, e nós estamos

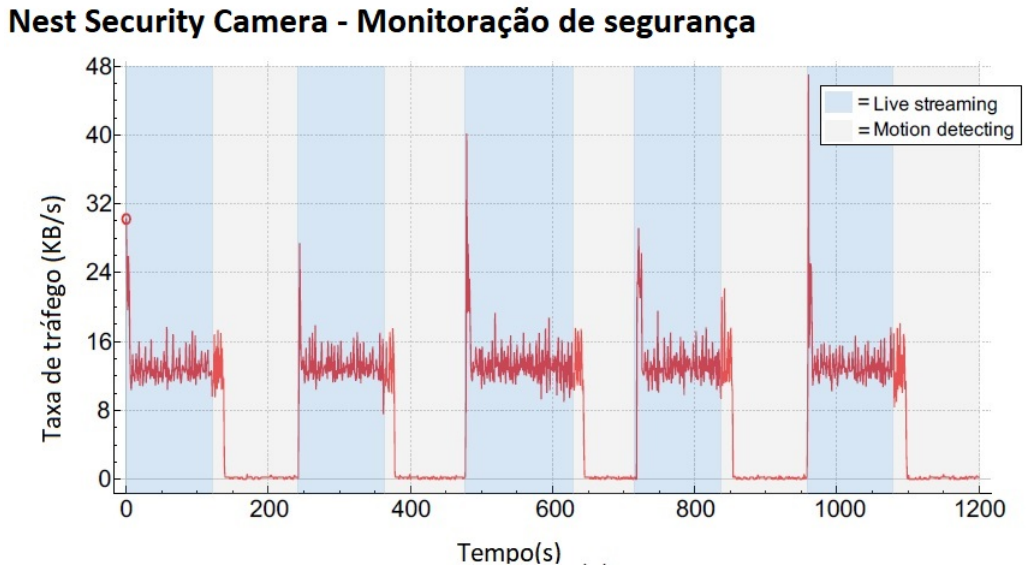

Figura 4.1: Gráfico do envio de pacotes da câmera Nest Security [Apthorpe Noah Reisman 2017]

modelando um cenário em que a câmera passa mais tempo funcionando em modo *live stream* do que trocando de modos, então é natural que ela possua a mesma variável de tráfego repetidas múltiplas vezes. Como é uma lista de variáveis, e não um conjunto, ter variáveis repetidas faz diferença. De forma similar, pode-se modelar o estado em que a câmera está em modo de detecção de movimento.

4.2.1 Equivalência Observacional

Agora, dados dois fluxos $\mathcal{F}_1, \mathcal{F}_2$ restritos pelo conjunto $\mathcal{RT}$, nós definimos quando $\mathcal{F}_1$ e $\mathcal{F}_2$ são equivalentes entre si.

Definição 4.2.9. Dado dois traços, $\mathcal{T} = \langle \mathsf{r}_1, \ldots, \mathsf{r}_n \rangle$ e $\mathcal{T}' = \langle \mathsf{r}'_1, \ldots, \mathsf{r}'_m \rangle$, nós dizemos que $\mathcal{T} \equiv \mathcal{T}'$ se e somente se:

- $n = m$;
- $\mathsf{r}' - \varepsilon_1 \leq \mathsf{r} \leq \mathsf{r}' + \varepsilon_1 \wedge \ldots \wedge \mathsf{r}'_m - \varepsilon_n \leq \mathsf{r}_n \geq \mathsf{r}'_m + \varepsilon_n$, onde $\varepsilon_1, \ldots \varepsilon_n \geq 0$.

Definição 4.2.10. Dado dois conjuntos $\mathcal{S}$ e $\mathcal{S}'$, tal que $\mathcal{S} = \mathcal{T}(\mathcal{F}) \mid_{\mathcal{RT}}$ e $\mathcal{S}' = \mathcal{T}(\mathcal{F}') \mid_{\mathcal{RT}}$, nós dizemos que $\mathcal{F}$ aproxima $\mathcal{F}'$, escrito $\mathcal{F} \preceq \mathcal{F}'$, se e somente se

$\forall \mathcal{T} \in \mathcal{S} \;\; \exists \mathcal{T}' \in \mathcal{S}' \mid \mathcal{T} \equiv \mathcal{T}'$, *i.e.*, para todo traço contido em $\mathcal{S}$, existe um traço equivalente contido em $\mathcal{S}'$.

Definição 4.2.11. Dados dois fluxos $\mathcal{F}$ e $\mathcal{F}'$ e um conjunto de restrições de tráfego $\mathcal{RT}$, nós dizemos que $\mathcal{F}$ é observacionalmente equivalente a $\mathcal{F}'$ se $\mathcal{F} \preceq \mathcal{F}'$ e $\mathcal{F}' \preceq \mathcal{F}$.

Intuitivamente, nós avaliamos todos as possíveis instanciações de um fluxo, *i.e.*, todos os traços daquele fluxo, e vemos se é possível o outro fluxo produzir aquela mesma assinatura. Senão, quer dizer que um intruso poderia utilizar aquele evento para descriminar um fluxo do outro, obtendo assim informação indesejada.

Nós definimos também uma forma para determinar se é possível se defender de um atacante ao adicionar tráfego, como mensagens chamariz, por exemplo.

Definição 4.2.12. Dado um traço $\mathcal{T} = \langle \mathsf{r}_1 \ldots, \mathsf{r}_n \rangle$ e uma lista $E = \langle \varepsilon_1, \ldots, \varepsilon_n \rangle$ de naturais, dizemos que $\mathcal{T} + E = \langle \mathsf{r}_1 + \varepsilon_1, \ldots, \mathsf{r}_n + \varepsilon_n \rangle$.

Definição 4.2.13. Dados dois conjuntos $\mathcal{S}$ e $\mathcal{S}'$, tal que $\mathcal{S} = \mathcal{T}(\mathcal{F}) \mid_{\mathcal{RT}}$ e $\mathcal{S}' = \mathcal{T}(\mathcal{F}') \mid_{\mathcal{RT}}$, existe uma forma de fazer $\mathcal{F} \preceq \mathcal{F}'$ adicionando tráfego se $\exists (E, E') \; \forall (\mathcal{T} \in \mathcal{S}) \; \exists (\mathcal{T}' \in \mathcal{S}') \mid \mathcal{T} + E \equiv \mathcal{T}'_1 + E'$, onde $E = \langle \varepsilon_1, \ldots, \varepsilon_n \rangle$ e $E' = \langle \varepsilon'_1, \ldots, \varepsilon'_n \rangle$ são listas de naturais e possuem o mesmo tamanho de $\mathcal{F}$ e $\mathcal{F}'$

Nesse caso, nós consideramos adição de tráfego por ambas as partes, de forma a tornar as observações equivalentes. Na nossa implementação no entanto, a presença de uma fórmula de formato $\exists\forall\exists$ é problemática porque até o momento de escrita desse trabalho, os resolvedores SMT tem problemas em resolver fórmulas desse tipo. Por isso, nós criamos uma definição alternativa para verificar equivalência adicionando tráfego, mais simples mas incompleta.

Definição 4.2.14. Dados dois conjuntos $\mathcal{S}$ e $\mathcal{S}'$, tal que $\mathcal{S} = \mathcal{T}(\mathcal{F}) \mid_{\mathcal{RT}}$ e $\mathcal{S}' = \mathcal{T}(\mathcal{F}') \mid_{\mathcal{RT}}$, $\forall (\mathcal{T} \in \mathcal{S}) \; \exists E, (\mathcal{T}' \in \mathcal{S}') \mid \mathcal{T}' + E \equiv \mathcal{T}$, onde $E = \langle \varepsilon_1, \ldots, \varepsilon_n \rangle$ é uma lista de naturais e possui o mesmo tamanho de $\mathcal{F}$ e $\mathcal{F}'$.

A ideia dessa última definição é a mesma, mas nós avaliamos a adição de tráfego em apenas um dos lados, ao invés de permitir adicionar tráfego em ambos os traços.

Exemplo: Tomemos dois fluxos $\mathcal{F}$ e $\mathcal{F}'$, definidos abaixo,

$$\mathcal{F} = \langle \mathsf{b}_1, \ldots, \mathsf{b}_n \rangle$$

$$\mathcal{F}' = \langle \mathsf{b}'_1, \ldots, \mathsf{b}'_n \rangle$$

e queremos comparar se $\mathcal{F}$ pode ser tornado equivalente ao fluxo $\mathcal{F}'$ através da adição de tráfego. Tomemos $\mathsf{b}_1 = 7$ e $\mathsf{b}'_1 = 3$. Levando-se em consideração ambas as definições anteriores, tanto a 4.2.13 quanto a 4.2.14, nós podemos tomar um $\varepsilon = 4$, de forma que $\mathsf{b}'_1 + \varepsilon = \mathsf{b}_1$. Se, por outro lado, nós tomarmos $\mathsf{b}_1 = 3$ e $\mathsf{b}'_1 = 7$, então $\mathcal{F}$ só pode se tornar equivalente a $\mathcal{F}'$ através da adição de tráfego pela definição 4.2.13, já que envolveria a adição de um $\varepsilon \geq 0$ com b_1.

4.3 Implementação e resultados experimentais

Nessa seção, nós vamos abordar um pouco a nossa implementação utilizando exemplos, e comentar os resultados obtidos durante os nossos experimentos.

Nós implementamos a nossa prova de conceito inteiramente no resolvedor SMT Z3 [Z3], alguns exemplos inclusive diretamente no resolvedor web (https://rise4fun.com/z3/). Um ou outro exemplo exigia mais processamento do que o resolvedor web era capaz de fornecer, então nós utilizamos a API Z3Py, que integra o Z3 com o python.

Para facilitar o entendimento, nós não utilizaremos aqui a sintaxe do Z3, mas sim uma pseudolinguagem mais parecida com as definições anteriores. No entanto, a lógica de implementação é exatamente a mesma, bastando apenas traduzir para o Z3 para replicar o funcionamento.

Exemplos: Considere dois fluxos simples, $\mathcal{F}$ e $\mathcal{F}'$, com apenas uma variável de tráfego cada, r e r'. Considere o nosso conjunto $\mathcal{RT} = \{\mathsf{r} \geq 0,\ , \mathsf{r} \leq 10,\ \mathsf{r}' \geq 0,\ \mathsf{r}' \leq 7\}$. Agora, nós queremos comparar a equivalência observacional utilizando nossa definição. Para isso, basta inserir no Z3 a fórmula:

$$\textbf{assert}\ \ \forall[\mathsf{r}\ \ Int].(0 \leq \mathsf{r} \leq 10) \Rightarrow \exists[\mathsf{r}'\ \ Int].(0 \leq \mathsf{r}' \leq 7) \wedge (\mathsf{r} = \mathsf{r}'))$$

Na hora da implementação, nós tivemos alguns problemas na forma como o Z3 instancia as suas variáveis. Por isso, nós definimos as restrições das variáveis do quantificador $\forall$

fora do quantificador $\exists$. Se o resultado retornado pelo Z3 quer dizer que o $\mathcal{F} \preceq \mathcal{F}'$, ou $\mathcal{F} \npreceq \mathcal{F}'$ caso contrário. Nesse caso, a o Z3 retorna Unsat.

No caso de dois fluxos $\mathcal{F} = \langle \mathsf{r}_1, \mathsf{r}_2 \rangle$ e $\mathcal{F}' = \langle \mathsf{r}'_1, \mathsf{r}'_2 \rangle$, com um conjunto de restrições $\mathcal{RT} = \{\mathsf{r}_1 \geq 0,\ \mathsf{r}_1 \leq 7,\ \mathsf{r}_2 \geq 0,\ \mathsf{r}_2 \leq 5,\ \mathsf{r}'_1 \geq 0,\ \mathsf{r}'_1 \leq 10,\ \mathsf{r}'_2 \geq 0,\ \mathsf{r}'_2 \leq 5\}$. A ideia aqui é muito parecida com o primeiro exemplo, como mostrado abaixo :

$$\begin{aligned}\textbf{assert}\ \ &\forall[\mathsf{r}_1\ \ Int,\ \ \mathsf{r}_2\ \ Int].((0 \leq \mathsf{r}_1 \leq 7) \wedge (0 \leq \mathsf{r}_2 \leq 5)) \Rightarrow \\ &\exists[\mathsf{r}'_1\ \ Int, \mathsf{r}'_2\ \ Int].(((0 \leq \mathsf{r}'_1 \leq 10) \wedge (0 \leq \mathsf{r}'_2 \leq 5)) \wedge ((\mathsf{r}_1 = \mathsf{r}'_1) \wedge (\mathsf{r}_2 = \mathsf{r}'_2)))\end{aligned}$$

Se nós quisermos avaliar se ao adicionar tráfego nós podemos tornar fluxos distintos equivalentes, existem algumas abordagens que podem ser feitas. Vamos tomar o primeiro exemplo, mais simples, e mostrar a intuição.

Antes de avaliarmos a fórmula de equivalência, nós podemos declarar uma constante ε:

$$\begin{aligned}&\textbf{declare-const}\ \ \varepsilon\ \ Int \\ &\textbf{assert}\ \ 0 \leq \epsilon \leq 3 \\ &\textbf{assert}\ \ \forall[\mathsf{r}\ \ Int].(0 \leq \mathsf{r} \leq 10) \Rightarrow \exists[\mathsf{r}'\ \ Int].(0 \leq \mathsf{r}' \leq 7) \wedge (\mathsf{r} = \mathsf{r}' + \varepsilon))\end{aligned}$$

Nesse caso, estamos definindo um único valor para ε, e esse valor tem que estar contido no intervalo entre 0 e 3. Ou seja, quando o resolvedor for avaliar a fórmula $\forall\exists$, ele vai instanciar um valor para ε, e vai adicionar sempre o mesmo valor a r', independente de r. Uma outra forma de avaliar adição de tráfego seria :

$$\textbf{assert}\ \forall[\mathsf{r}\ \ Int].(0 \leq \mathsf{r} \leq 10) \Rightarrow \exists[\mathsf{r}'\ \ Int, \varepsilon\ \ Int].((0 \leq \mathsf{r}' \leq 7) \wedge (0 \leq \varepsilon \leq 3)) \wedge (\mathsf{r} = \mathsf{r}' + \varepsilon))$$

Nesse caso, para cada valor testado de r, o resolvedor adicionaria um ε que varia entre 0 e 3 para a variável r'. Nesse exemplo, o Z3 retorna Sat.

Os dispositivos citados no artigo [Apthorpe Noah Reisman 2017] foram adaptados utilizando as estratégias mencionadas acima, buscando a comparação entre diferentes dispositivos ou entre diferentes estados do mesmo dispositivo. O tempo de resposta para todos foi de poucos segundos. As implementações em Z3 se encontram em https://github.com/asbeno/TrafficEquivalence.git.

Capítulo 5

Considerações finais e trabalhos futuros

Neste trabalho foram propostas definições formais para a elaboração de uma ferramenta de verificação automática para equivalência observacional levando em consideração aspectos temporais e de tráfego de pacotes. Com o aumento da relevância de IoT e de WSN, com o barateamento de *hardware*. A tendência é que cenários como o abordado aqui fiquem cada vez mais frequentes, e possuir uma forma de avaliá-los pode se provar útil.

Utilizando métodos de computação simbólica, como *Rewriting Modulo SMT* e utilização de termos simbólicos, nós conseguimos lidar com a explosão de estados quando se busca por uma solução em um sistema com termos que possuem infinitas instâncias possíveis.

Com o formalismo definido, os experimentos realizados se mostraram promissores, com um tempo de resposta muito baixo, e uma contagem de estados também muito pequena (abaixo de 40). Isso sugere uma possível utilização dessa metodologia para outros tipo de modelagem, em cenários que se assemelhem aos aqui discutidos.

Apesar disso, nós tivemos problemas com a escalabilidade da ferramenta. Protocolos mais longos, ou verificações com mais de uma sessão, demoravam consideravelmente mais tempo para terminarem, isso quando terminavam. Isso se deve principalmente a natureza não-determinística e exponencial do problema, aumentando demais o espaço de busca. Ademais, os resolvedores SMTs também representaram um gargalo de

processamento em certas situações, em parte porque a integração entre o Maude e os resolvedores ainda apresenta um certo sobrecusto sobre a execução do programa.

No entanto, isso apresenta boas oportunidades para trabalhos futuros. A ferramenta vai se beneficiar de qualquer melhora que venha a ocorrer tanto nos resolvedores SMT quanto na integração destes com o Maude. Além disso, existe sempre a possibilidade de incorporação da ferramenta em análises mais pontuais de um sistema, em que a parte temporal ou de tráfego é crítica mas reduzida, ao invés de utilizá-lo para modelar sistemas inteiros. Adicionar teorias equacionais também é uma possibilidade, apesar de que agravaria a questão da escalabilidade, se deixada como está.

Poderíamos, também, explorar o conceito a utilização de bissimulação para avaliar equivalência de tráfego, visualizando os fluxos como máquinas de estado. A integração das duas propostas do trabalho também é uma possibilidade, incorporando elementos de tráfego e tempo dentro de uma mesma ferramenta, assim como teorias equacionais. Um maquinário que já incorpora algumas ideias apresentados nesse trabalho, como a utilização de resolvedores SMT para lidar com os domínios do tempo e do tráfego, foi apresentado em [Urquiza et al. 2019], mas sem o elemento de equivalência observacional e com ênfase em ataques de negação de serviço realizados por atacantes com recursos limitados.

Bibliografia

[Abadi e Fournet 2004]ABADI, M.; FOURNET, C. Private authentication. *Theor. Comput. Sci.*, v. 322, n. 3, p. 427–476, 2004.

[Apthorpe Noah Reisman 2017]APTHORPE NOAH REISMAN, D. . S. S. . N. A. . F. N. Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic. 2017.

[Bae e Rocha 2019]BAE, K.; ROCHA, C. Symbolic state space reduction with guarded terms for rewriting modulo smt. *Science of Computer Programming*, v. 178, p. 20 – 42, 2019. ISSN 0167-6423. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167642318301187>.

[Barrett et al. 2011]BARRETT, C. et al. CVC4. In: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. [S.l.: s.n.], 2011. p. 171–177.

[Basin e Mödersheim 2004]BASIN, D.; MÖDERSHEIM, L. V. S. OFMC: A symbolic model checker for security protocols. *Interational Journal of Information Security*, 2004.

[Bérard et al. 2016]BÉRARD, B. et al. Formal verification of mobile robot protocols. *Distributed Computing*, Springer Verlag, v. 29, n. 6, p. 459–487, 2016. Disponível em: <https://hal.sorbonne-universite.fr/hal-01344903>.

[Biere et al. 1999]BIERE, A. et al. Symbolic model checking without bdds. In: CLEAVELAND, W. R. (Ed.). *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. p. 193–207.

[Brands e Chaum 1993]BRANDS, S.; CHAUM, D. Distance-bounding protocols (extended abstract). In: *EUROCRYPT*. [S.l.: s.n.], 1993. p. 344–359.

[Cervesato 2003]CERVESATO, I. Data access specification and the most powerful symbolic attacker in msr. *Lecture Notes in Computer Science*, v. 2609, 05 2003.

[Cheval e Cortier 2015]CHEVAL, V.; CORTIER, V. Timing attacks: symbolic framework and proof techniques. In: FOCARDI, R.; MYERS, A. (Ed.). *Proceedings of the 4th International Conference on Principles of Security and Trust (POST'15)*. London, UK: Springer Berlin Heidelberg, 2015. (Lecture Notes in Computer Science). To appear.

[Chothia e Smirnov 2010]CHOTHIA, T.; SMIRNOV, V. A traceability attack against e-passports. In: SION, R. (Ed.). *Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 20–34. ISBN 978-3-642-14577-3.

[Clarke e Emerson 1982]CLARKE, E. M.; EMERSON, E. A. Design and synthesis of synchronization skeletons using branching-time temporal logic. In: *Logic of Programs, Workshop*. Berlin, Heidelberg: Springer-Verlag, 1982. p. 52–71. ISBN 3-540-11212-X. Disponível em: <http://dl.acm.org/citation.cfm?id=648063.747438>.

[Clavel et al. 2007]CLAVEL, M. et al. *All About Maude: A High-Performance Logical Framework*. [S.l.]: Springer, 2007. (LNCS, v. 4350).

[Corin et al. 2004]CORIN, R. et al. Timed model checking of security protocols. In: *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering*. New York, NY, USA: ACM, 2004. (FMSE '04), p. 23–32.

[Cortier e Delaune 2009]CORTIER, V.; DELAUNE, S. A method for proving observational equivalence. In: *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*. [S.l.: s.n.], 2009. p. 266–276.

[Diffie e Hellman 1976]DIFFIE, W.; HELLMAN, M. E. *New Directions in Cryptography*. 1976.

[Dolev e Yao 1983]DOLEV, D.; YAO, A. On the security of public key protocols. *IEEE Transactions on Information Theory*, v. 29, n. 2, p. 198–208, 1983.

[Dutertre 2015]DUTERTRE, B. Solving exists/forall problems with yices. In: *SMT*. [S.l.: s.n.], 2015.

[Fung et al. 2010]FUNG, B. C. M. et al. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 42, n. 4, jun. 2010. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/1749603.1749605>.

[Ge et al. 2016]GE, N. et al. Formal verification of a rover anti-collision system. In: BEEK, M. H. ter; GNESI, S.; KNAPP, A. (Ed.). *Critical Systems: Formal Methods and Automated Verification*. Cham: Springer International Publishing, 2016. p. 171–188.

[Ho et al. 2014]HO, G. et al. Tick tock: Building browser red pills from timing side channels. In: *8th USENIX Workshop on Offensive Technologies (WOOT 14)*. San Diego, CA: USENIX Association, 2014. Disponível em: <https://www.usenix.org/conference/woot14/workshop-program/presentation/ho>.

[IEEE Standard for Information Technology - Telecommunications and Information Exchange Betw IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, p. 1–1076, June 2007. ISSN null.

[Internet of Evil Things 2017]INTERNET of Evil Things 2017. `https://www.pwnieexpress.com/hubfs/2017InternetOfEvilThings.pdf`. Accessed: 2018-11-19.

[Kanovich et al. 2014]KANOVICH, M. I. et al. Towards timed models for cyber-physical security protocols. Available in Nigam's homepage. 2014.

[Kojima, Nagashima e Tsuchiya 2017]KOJIMA, H.; NAGASHIMA, Y.; TSUCHIYA, T. State space reduction techniques for model checking of manet protocols. *International Journal of Networking and Computing*, v. 7, p. 29–49, 02 2017.

[Künnemann, Esiyok e Backes 2018]KÜNNEMANN, R.; ESIYOK, I.; BACKES, M. Automated verification of accountability in security protocols. *CoRR*, abs/1805.10891, 2018. Disponível em: <http://arxiv.org/abs/1805.10891>.

[Lee et al. 2013]LEE, J.-H. et al. A preliminary report on static analysis of c code for nuclear reactor protection system. *IFAC Proceedings Volumes*, v. 46, n. 9, p. 2134 – 2139, 2013. ISSN 1474-6670. 7th IFAC Conference on Manufacturing Modelling, Management, and Control. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1474667016346122>.

[Lowe 1995]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>.

[Martelli e Montanari 1982]MARTELLI, A.; MONTANARI, U. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, ACM, New York, NY, USA, v. 4, n. 2, p. 258–282, abr. 1982. ISSN 0164-0925. Disponível em: <http://doi.acm.org/10.1145/357162.357169>.

[Marzouki, Radhouani e Rajeb 2015]MARZOUKI, K.; RADHOUANI, A.; RAJEB, N. B. Formal verification of secrecy, coercion resistance and verifiability properties for a remote electronic voting protocol. *International Journal of Information Security and Privacy*, v. 7, p. 57–85, 08 2015.

[Mason e Talcott 2004]MASON, I. A.; TALCOTT, C. L. IOP: The InterOperability Platform & IMaude: An interactive extension of Maude. In: *Fifth International Workshop on Rewriting Logic and Its Applications (WRLA'2004)*. [S.l.]: Elsevier, 2004. (Electronic Notes in Theoretical Computer Science).

[Matiyasevich 1993]MATIYASEVICH, Y. *Hilbert's Tenth Problem*. [S.l.]: MIT Press, 1993.

[Meseguer 1992]MESEGUER, J. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, v. 96, n. 1, p. 73 – 155, 1992. ISSN 0304-3975. Disponível em: <http://www.sciencedirect.com/science/article/pii/030439759290182F>.

[Narayanan e Shmatikov 2008]NARAYANAN, A.; SHMATIKOV, V. Robust de-anonymization of large sparse datasets. In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. [S.l.: s.n.], 2008. p. 111–125. ISSN 1081-6011.

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

[Nigam, Talcott e Urquiza 2016]NIGAM, V.; TALCOTT, C. L.; URQUIZA, A. A. Towards the automated verification of cyber-physical security protocols: Bounding the number of timed intruders. In: *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II*. [S.l.: s.n.], 2016. p. 450–470.

[Processing e The 2001]PROCESSING, F. I.; THE, A. *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. 2001.

[Queille e Sifakis 1982]QUEILLE, J. P.; SIFAKIS, J. Specification and verification of concurrent systems in cesar. In: DEZANI-CIANCAGLINI, M.; MONTANARI, U. (Ed.). *International Symposium on Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982. p. 337–351. ISBN 978-3-540-39184-5.

[Rivest, Shamir e Adleman 1978]RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1978.

[Rocha 2012]ROCHA, C. *Symbolic Reachability Analysis for Rewrite Theories*. Tese (Doutorado) — University of Illinois at Urbana-Champagne, 2012.

[Schneier et al. 1998]SCHNEIER, B. et al. Twofish: A 128-bit block cipher. In: *in First Advanced Encryption Standard (AES) Conference*. [S.l.: s.n.], 1998.

[Semple Sean Ponomarev 2013]SEMPLE SEAN PONOMAREV, S. . D. J. . A. T. Applying static analysis to high-dimensional malicious application detection. In: *Proceedings of the Annual Southeast Conference*. [S.l.: s.n.], 2013.

[Sommers 2014]SOMMERS, J. Public review for traffic analysis of encrypted messaging services: Apple imessage and beyond. *ACM SIGCOMM Computer Communication Review*, v. 44, p. 5–5, 10 2014.

[Sweeney 1997]SWEENEY, L. Weaving technology and policy together to maintain confidentiality. *The Journal of Law, Medicine & Ethics*, v. 25, n. 2-3, p. 98–110, 1997. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1748-720X.1997.tb01885.x>.

[Urquiza et al. 2019]URQUIZA, A. A. et al. Resource-bounded intruders in denial of service attacks. In: *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019. p. 382–396. Disponível em: <https://doi.org/10.1109/CSF.2019.00033>.

[Ying, Makrakis e Mouftah 2014]YING, B. D.; MAKRAKIS, D.; MOUFTAH, H. T. Anti-traffic analysis attack for location privacy in wsns. *EURASIP Journal on Wireless Communications and Networking*, v. 2014, n. 1, p. 131, Aug 2014. ISSN 1687-1499. Disponível em: <https://doi.org/10.1186/1687-1499-2014-131>.

[Z3]Z3. `https://rise4fun.com/z3/`. Accessed: 2019-05-07.

Apêndice A

Provas e Especificações Adicionais

A.1 Especificação da função sGen

Nós definimos a função sgen, abordada na subseção 3.3.2. A função sgen recebe como entrada:

- ms - o termo simbólico que precisa ser gerado;
- $\mathcal{IK}$ - o conhecimento do intruso;
- ssb - uma substituição que mapeia símbolos para termos simbólicos, que começa a execução vazia ($\emptyset$). Intuitivamente, essa substituição representa os símbolos que não mais se encontram restritos no conjunto de restrições de derivabilidade, mas que precisam ser substituídos por um termo simbólico específico;
- $\mathcal{DC}$ - o conjunto de restrições de derivabilidade que especifica os símbolos contidos em ms e $\mathcal{IK}$ e o alcance de ssb.

$\mathsf{sgen}(\mathsf{ms}, \mathcal{IK}, \mathsf{ssb}, \mathcal{DC})$ retorna um conjunto de pares

$$\langle \mathsf{ssb}_1, \mathcal{DC}_1 \rangle, \ldots, \langle \mathsf{ssb}_n, \mathcal{DC}_n \rangle$$

onde para $1 \leq i \leq n$, ssb_i é a substituição de símbolos que foi obtida e $\mathcal{DC}_i$ o conjunto de restrições de derivabilidade que refina $\mathcal{DC}$.

$\mathsf{sgen}(\mathsf{ms}, \mathcal{IK}, \mathsf{ssb}, \mathcal{DC})$ é definido a seguir:

- ms é um símbolo : se $\mathsf{ms} \in \mathcal{DC}$ ou ms for um tipo inferível, retorna $\langle \mathsf{ssb}, \mathcal{DC} \rangle$, caso contrário o símbolo precisa ser restrito, então retorne $\langle \mathsf{ssb}, \mathsf{dc}(\mathsf{ms}, \mathcal{IK})\ \mathcal{DC} \rangle$; Por exemplo, no caso $\mathsf{sgen}(\mathsf{p}(\mathit{Alice}), \mathcal{IK}, \mathsf{ssb}, \mathcal{DC})$, onde $\mathsf{p}(\mathit{Alice})$ é o nome de um participante, retorna $\langle \mathsf{ssb}, \mathcal{DC} \rangle$. No caso de $\mathsf{sgen}(\mathsf{sym}, \mathcal{IK}, \mathsf{ssb}, \mathcal{DC})$, onde sym é um símbolo novo (não pertencente a $\mathcal{DC}$), retorna $\langle \mathsf{ssb}, \mathsf{dc}(\mathsf{sym}, \mathcal{IK})\ \mathcal{DC} \rangle$. Isso se dá porque o simbolo precisa ser restrito ao conhecimento atual do intruso, por isso a adição de $\mathsf{dc}(\mathsf{sym}, \mathcal{IK})$, já que sym só poderia representar o que pode ser gerado por $\mathcal{IK}$.

- ms é um *nonce*: se $\mathsf{ms} \in \mathcal{IK}$, retorna $\langle \mathsf{ssb}, \mathcal{DC} \rangle$, caso contrário retorna $\emptyset$ (não existe solução); Aqui, o intruso só pode gerar um *nonce* específico se já o possuir no seu conhecimento.

- ms é uma constante inferível, tal qual o nome de um participante, chaves públicas, etc: retorna $\langle \mathsf{ssb}, \mathcal{DC} \rangle$; Se o termo a ser gerado for inferível, o intruso é capaz de fazê-lo sem problemas.

- $\mathsf{ms} = \langle \mathsf{ms}_1, \ldots, \mathsf{ms}_n \rangle$: itera sobre os elementos da tupla, acumulando substituições e restrições de símbolos; Para gerar uma tupla, o intruso deve ser capaz de gerar cada termo individual presente na tupla, e é isso que esse caso modela.

- $\mathsf{ms} = \mathsf{e}(\mathsf{ms}_1, \mathsf{ms}_2)$: Existem duas possibilidades

  – O intruso é capaz de derivar ms. Isto é feito simplesmente chamando sgen sob a tupla $\langle \mathsf{ms}_1, \mathsf{ms}_2 \rangle$; Por exemplo, em um caso em que $\mathsf{ms} = \mathsf{e}(\mathsf{k}, \mathsf{ms}_1)$ e $\mathsf{k} \in \mathcal{IK}$. Aqui, o intruso pode considerar a encriptação como uma tupla, e simplesmente verificar se consegue gerar cada parte (a chave k e o termo simbólico ms_1) individualmente.

  – O intruso possui um termo ms_1 que é unificável com ms. Para isso, nós realizamos dois passos. Primeiro, nós tratamos símbolos como variáveis e tentamos unificar ms com os termos em $\mathcal{IK}$, checando por ciclos (checagens de ocorrência ou *occurs check*). Uma vez que todas as unificações (substituições de símbolos) são encontradas, nós mantemos apenas as unificações que são consistentes com $\mathcal{DC}$. Isto é feito ao se acumular as restrições retornadas

pela função $\mathsf{checkSubst}(\mathsf{ssb}_i, \mathsf{ssb}, \mathsf{ds})$, definida abaixo, para cada unificação ssb_i. Perceba que os domínios de ssb_i e ssb são disjuntos. Por exemplo, vamos assumir que $\mathsf{ms} = \mathsf{e}(\mathsf{v}, \mathsf{k})$ e que $\mathsf{k} \notin \mathcal{IK}$, e que $\{\mathsf{ms}_1 = \mathsf{e}(\mathsf{t}, \mathsf{k})\} \in \mathcal{IK}$. Aqui, apesar do intruso não conseguir gerar $\langle \mathsf{k} \rangle$, ele é capaz de enviar uma mensagem de mesmo formato que ms, já que ele possui uma mensagem em seu conhecimento de mesmo formato, ms_1.

Intuitivamente, após avaliarmos a existência e um termo no conhecimento do nosso intruso que pode ser unificado, nós precisamos verificar a validade da substituição de símbolos gerada. Por exemplo, tomemos o termo $\mathsf{ms} = \mathsf{e}(\langle \mathsf{v}, \mathsf{sym} \rangle, \mathsf{k})$, e $\mathcal{IK} = \{\mathsf{e}(\langle \mathsf{n}_a, \mathsf{sym}_1 \rangle, \mathsf{k})\}$ e $\mathcal{DC} = \mathsf{dc}(\mathsf{sym}, \{\mathsf{n}_a, \mathsf{n}_c\})\ \mathsf{dc}(\mathsf{sym}_1, \mathcal{S})$. Nesse caso, a minha substituição que eliminaria as variáveis e unificaria ms com $\mathsf{e}(\langle \mathsf{n}_a, \mathsf{sym}_1 \rangle, \mathsf{k})$ seria $\mathsf{sb} = [\mathsf{v} \mapsto \mathsf{sym}_\mathsf{v}]$. Mas apenas isso não é o bastante.

Agora nós precisamos verificar se o vínculo gerado, a substituição em si, é válida, levando-se em consideração as restrições de derivabilidade. Cada substituição precisa ser avaliada por consistência, além de que elas não apenas precisam funcionar isoladamente, mas em conjunto. Por isso, elas precisam ser avaliadas de forma cumulativa, em que cada substituição checada é aplicada sobre as substituições restantes e sobre as restrições de derivabilidade.

A função $\mathsf{checkSubst}(\mathsf{ssb}_0, \mathsf{ssb}, \mathcal{DC})$ retorna esse conjunto de refinamentos consistentes de $\langle \mathsf{ssb}_0, \mathsf{ssb}, \mathcal{DC} \rangle$. Ela processa cada vínculo, $\mathsf{sym}_i \mapsto \mathsf{ms}_i \in \mathsf{ssb}_0$ no contexto do par $\langle \mathsf{ssb}, \mathcal{DC} \rangle$ produzindo o conjunto de soluções $\mathsf{css} = \{\mathsf{ssb}_j, \mathcal{DC}_j\}, 1 \leq j \leq k$ que refina $\{\mathsf{ssb}, \mathcal{DC}\}$, e vincula sym_i com uma instância de ms_i utilizando $\mathsf{chkBnd}(\mathsf{sym}_i, \mathsf{ms}_i, \mathsf{ssb}, \mathcal{DC})$. O próximo vínculo é processado no contexto de cada elemento de css produzidos ao processar os vínculos anteriores.

A função $\mathsf{chkBnd}(\mathsf{sym}, \mathsf{ms}, \mathsf{ssb}, \mathcal{DC})$ está definida abaixo:

- se sym não estiver restrito em $\mathcal{DC}$, então chkBnd retorna $\{(\mathsf{sym} \mapsto \mathsf{ms})\mathsf{ssb}', \mathcal{DC}'\}$ onde ssb' é o resultado de aplicar $\mathsf{sym} \mapsto \mathsf{ms}$ sobre o alcance de ssb, e $\mathcal{DC}'$ o resultado de aplicar $\mathsf{sym} \mapsto \mathsf{ms}$ sobre os termos em $\mathcal{DC}$;

- se sym for restrito por $\mathcal{DC}$ e ms não for um símbolo, então chkBnd tem que garantir que ms é derivável sob as restrições acumuladas. Isto pode ser feito ao adicionar

o vínculo $\mathsf{sym} \mapsto \mathsf{ms}$ para cada solução retornada por $\mathsf{sgen}(\mathsf{ms}, \mathcal{IK}, \mathsf{ssb}_1, \mathcal{DC}_1)$. Aqui, $\mathcal{DC} = \mathcal{DC}_0\ \mathsf{dc}(\mathsf{sym}, \mathcal{IK})$, ssb_1 é o resultado da aplicação de $\mathsf{sym} \mapsto \mathsf{ms}$ sobre ssb e $\mathcal{DC}_1$ é o resultado da aplicação de $\mathsf{sym} \mapsto \mathsf{ms}$ sobre os termos em $\mathcal{DC}_0$;

- se sym está restrito em $\mathcal{DC}$, e ms é um símbolo que não está restrito em $\mathcal{DC}$, então retorne $\{(\mathsf{sym} \mapsto \mathsf{ms})\ \mathsf{ssb}_1, \mathcal{DC}_1\ \mathsf{dc}(\mathsf{ms}, \mathcal{IK})\}$ onde $\mathsf{ssb}_1, \mathcal{DC}_1, \mathcal{IK}$ são como definidos acima;

- se sym é restrito por $\mathcal{DC}$, e ms for um símbolo também restrito por $\mathcal{DC}$, então retorne $\{(\mathsf{sym} \mapsto \mathsf{ms})\ \mathsf{ssb}_1, \mathcal{DC}_1\ \mathsf{dc}(\mathsf{ms}, \mathcal{IK}')\}$ onde $\mathsf{ssb}_1, \mathcal{DC}_1$ são como definidos acima e $\mathcal{IK}'$ é a restrição do anterior entre $\mathsf{sym}, \mathsf{ms}$. Um símbolo é anterior a outro se ele foi gerado em um passo anterior na execução do protocolo em relação ao outro. Como o conhecimento do intruso aumenta com o tempo, nós estamos efetivamente restringindo os termos ao menor dos conjuntos de conhecimento.

A.2 Prova do Teorema 3.4.1

Lemma A.2.1. *Considere* ms *um termo pertencente ao observável (definição 3.4.1)* $\mathcal{O}$ *e* sym' *um símbolo pertencente ao observável* $\mathcal{O}'$. $\mathsf{ms} \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{sym}'$ *se e somente se* $\mathsf{symDer}(\mathsf{sym}', \mathsf{ms}, \mathcal{DC}, \mathcal{DC}')$.

Prova: Nós provamos por indução sobre a maior altura, h, dos símbolos em ms no grafo de dependência (acíclico, como mostrado na definição 3.3.3) de $\mathcal{DC}$. A seguir nós assumimos $\mathsf{dc}(\mathsf{sym}', \mathcal{S}') \in \mathcal{DC}$.

- **Caso Base:** Se $h = 0$, isto é, não existir nenhum símbolo em ms, ou seja, ms é um termo base. Nós prosseguimos para a indução sobre ms.

  - **Caso Base 1:** se ms for um termo inferível, então $\mathsf{ms} \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{sym}'$ e $\mathsf{symDer}(\mathsf{sym}', \mathsf{ms}, \mathcal{DC}, \mathcal{DC}')$ são ambos verdadeiros;
  - **Caso Base 2:** se ms for um *nonce*, então existem dois casos. Ou este *nonce* provém de uma bijeção, o que significa que $\mathsf{ms} \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{sym}'$ se e somente se $\mathsf{n} \in \mathcal{DC}(\mathsf{sym}')$ se e somente se $\mathsf{n} \in \mathcal{S}'$ se e somente se $\mathsf{symDer}(\mathsf{sym}', \mathsf{ms}, \mathcal{DC}, \mathcal{DC}')$. Caso contrário, $\mathsf{n} \in \mathcal{DC}(\mathsf{sym}')$ se e somente se $\mathsf{symDer}(\mathsf{sym}', \mathsf{ms}, \mathcal{DC}, \mathcal{DC}')$;

– **Caso Indutivo:** Se ms for uma tupla $\langle \mathsf{ms}_1, \ldots, \mathsf{ms}_n \rangle$, então $\mathsf{ms} \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{sym}'$ se e somente se $\mathsf{ms}_i \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{sym}'$ se e somente se (pela HI) $\mathsf{symDer}(\mathsf{sym}', \mathsf{ms}, \mathcal{DC}, \mathcal{DC}')$ para todo $1 \leq i \leq n$.

– **Caso Indutivo:** Se ms for uma encriptação $\mathsf{e}(\mathsf{m}, \mathsf{k})$, então ou ms é uma caixa-preta, e nesse caso segue a mesma lógica do caso do *nonce*. Caso contrário nós apelamos para a hipótese indutiva, como no caso da tupla.

• **Caso Indutivo:** Se $h = n + 1$ com $n \geq 0$, então ms contém algum símbolo. Nós prosseguimos por indução no tamanho de ms. A maioria dos casos seguem a mesma lógica na prova do caso base (Caso 1), com a exceção do seguinte caso base:

– **Caso Base:** se ms é um símbolo sym. Considere $\mathsf{dc}(\mathsf{sym}, \mathcal{S})$. Todos os termos base em $\mathcal{DC}(\mathsf{sym}, \mathcal{S})$ são construídos utilizando os termos em $\mathcal{S}$. Como $\mathcal{DC}$ é acíclico, todos os símbolos em $\mathcal{S}$ tem uma altura de no máximo n. Logo, nós apelamos para a HI para provar esse caso.

Prova :Continuando a prova do teorema 3.4.1, prosseguimos com a indução do tamanho de ms'.

• **Caso Base:** ms' é um símbolo sym'. A substituição correspondente é $\mathsf{sym}' \mapsto \mathsf{ms}$, que é tratado pelo Lemma $A.2.1$.

• **Outros Casos Base:** $\mathsf{ms}' = \mathsf{n}'$ é um *nonce*. Então $\mathsf{ms} \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{n}'$ se e somente se $\mathsf{ms} = \mathsf{bij}[\mathsf{n}']$ se e somente se $\mathsf{termApprox}(\mathsf{ms}, \mathsf{ms}', \mathcal{O}, \mathcal{O}')$. Se for um inferível, então ms tem que ser o mesmo inferível e $\mathsf{termApprox}(\mathsf{ms}, \mathsf{ms}', \mathcal{O}, \mathcal{O}')$. O mesmo acontece se ms' for uma chave.

• **Caso Indutivo:** $\mathsf{ms}' = \mathsf{e}(\mathsf{ms}'_1, \mathsf{k}')$ é um termo encriptado. Então ou é um termo caixa-preta, e nesse caso a prova tem uma lógica similar ao caso do *nonce*. Caso contrário, nós apelamos para a HI nos termos menores ms'_1 e k';

• **Caso Indutivo:** $\mathsf{ms} = \{\mathsf{ms}'_1, \ldots, \mathsf{ms}'_n\}$ então nós apelamos para a HI nos termos menores ms'_i.

A.3 Prova do Teorema 3.4.2

Lemma A.3.1. *Considere* $\mathcal{EQ} = \mathcal{EQ}_1 \cup \mathcal{EQ}_2$ *como um conjunto de restrições de comparação (definição 3.3.6), onde* $\mathcal{EQ}_1$ *contém apenas restrições de igualdade e* $\mathcal{EQ}_2$ *apenas restrições de desigualdade. Considere* θ *como o unificador mais geral (ou m.g.u) de todas as restrições em* $\mathcal{EQ}_1$*. Para todos os termos base* $\mathsf{m}, \mathsf{ms} \in \mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}}$ *se e somente se* $\mathsf{m} \in \mathcal{DC}(\theta[\mathsf{ms}]) \mid_{\mathcal{EQ}_2}$

Prova: $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}}$ se e somente se o padrão de correspondência ssb de m e ms não falsificar nenhuma restrição em $\mathcal{EQ}$ se e somente se ssb for uma instância de θ (já que ele é o m.g.u) e ssb não falsificar nenhuma restrição em $\mathcal{EQ}_2$ se e somente se $\mathsf{m} \in \mathcal{DC}(\theta[\mathsf{ms}]) \mid_{\mathcal{EQ}_2}$.

Lemma A.3.2. *Existe um termo base* $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}_1) \mid_{\mathcal{EQ}}$ *e* $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}_2) \mid_{\mathcal{EQ}}$ *com a mesma substituição solução* θ *se e somente se* $\mathsf{canEq}(\mathsf{ms}_1, \mathsf{ms}_2, \mathcal{DC}, \mathcal{EQ})$.

Prova: $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}_1) \mid_{\mathcal{EQ}}$ e $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}_2) \mid_{\mathcal{EQ}}$ com a mesma substituição θ se e somente se $\theta(\mathsf{ms}_1) = \theta(\mathsf{ms}_2) = \mathsf{m}$ se e somente se ms_1 e ms_2 puderem ser unificados por θ e θ satisfaça todas as restrições de $\mathcal{EQ}$ e para todo $\mathsf{sym}_i \mapsto \mathsf{m}_i \in \theta, \mathsf{m}_i \in \mathcal{DC}(\mathsf{sym}_i)$ (Definição de pertencimento a $\mathcal{DC}(\mathsf{sym}_i)$) se e somente se $\mathsf{canEq}(\mathsf{ms}_1, \mathsf{ms}_2, \mathcal{DC}, \mathcal{EQ})$.

Lemma A.3.3. *Considere* $\mathcal{EQ}$ *como um conjunto finito de restrições de desigualdade apenas, e* $\mathcal{DC}$ *e* $\mathcal{DC}'$ *conjuntos de restrições de derivabilidade. Assuma que* $\mathcal{DC} \vDash \mathcal{EQ}$*. Para todos os termos base* m *nós temos que* $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}} \Rightarrow \mathsf{m} \in \mathcal{DC}'(\mathsf{ms}')$ *se e somente se* $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}) \Rightarrow \mathsf{m} \in \mathcal{DC}'(\mathsf{ms})$

Prova: A direção reversa é imediata. Nós provamos a direção para direita. Assuma (1) $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}} \Rightarrow \mathsf{m} \in \mathcal{DC}'(\mathsf{ms}')$ para todos os termos base m e assuma também que (2) $\mathsf{m}_1 \in \mathcal{DC}(\mathsf{ms})$. Nós mostramos que $\mathsf{m}_1 \in \mathcal{DC}'(\mathsf{ms})$.

Nós prosseguimos com a indução do tamanho de ms. O caso interessante acontece quando $\mathsf{ms} = \mathsf{sym}$ é um símbolo o que significa que $\mathsf{ms}' = \mathsf{sym}'$ tem que ser um símbolo. Caso contrário, é fácil construir um termo $\mathsf{m}_2 \in \mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}}$ tal que $\mathsf{m}_2 \notin \mathcal{DC}'(\mathsf{ms}')$. (Por exemplo, uma tupla muito grande de inferíveis)

Considere a substituição correspondente de símbolos $\mathsf{ssb} = [\mathsf{sym} \mapsto \mathsf{m}_1]$. Existem dois casos:

- ssb não falsifica nenhuma restrição em $\mathcal{EQ}$, então $\mathsf{m}_1 \in \mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}}$ e $\mathsf{m}_1 \in \mathcal{DC}(\mathsf{ms})$ por (1).

- ssb torna falsa alguma restrição em $\mathcal{EQ}$. Assuma por contradição que $\mathsf{m}_1 \notin \mathcal{DC}'(\mathsf{ms})$. Já que $\mathsf{m}_1 \in \mathcal{DC}(\mathsf{ms})$, uma tupla arbitrariamente grande $\langle \mathsf{m}_1, \ldots, \mathsf{m}_1 \rangle \in \mathcal{DC}(\mathsf{ms})$. No entanto, como $\mathsf{m}_1 \notin \mathcal{DC}'(\mathsf{ms})$, então (3) $\langle \mathsf{m}_1, \ldots, \mathsf{m}_1 \rangle \notin \mathcal{DC}'(\mathsf{ms})$. Escolha uma tupla grande tal que nenhuma restrição em $\mathcal{EQ}$ seja tornada falsa. (Lembre-se que todas as restrições contidas em $\mathcal{EQ}$ são de desigualdade.) Então $\langle \mathsf{m}_1, \ldots, \mathsf{m}_1 \rangle \in \mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}}$. Por (1) nós temos que $\langle \mathsf{m}_1, \ldots, \mathsf{m}_1 \rangle \in \mathcal{DC}'(\mathsf{ms})$, resultando em uma contradição com (3). Logo $\mathsf{m}_1 \in \mathcal{DC}'(\mathsf{ms})$.

Prova Provando o teorema 3.4.2, $\mathsf{ms} \preceq_{\mathcal{O},\mathcal{O}'} \mathsf{ms}'$ se e somente se $\mathsf{termEqApprox}(\mathsf{ms}, \mathsf{ms}', \mathcal{O}, \mathcal{O}')$:

- $\mathcal{DC} \nvDash \mathcal{EQ}$ se e somente se $\mathsf{termEqApprox}(\mathsf{ms}, \mathsf{ms}', \mathcal{O}, \mathcal{O}')$ pelo Lemma 3.3.1;

- ou $\mathcal{DC} \vDash \mathcal{EQ}$ e $\mathcal{DC}' \vDash \mathcal{EQ}'$ se para todo $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}}$, nós temos que $\mathsf{m} \in \mathcal{DC}'(\mathsf{ms}') \mid_{\mathcal{EQ}'}$. Pelo Lemma A.3.1, nós temos que $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}}$ se e somente se $\mathsf{m} \in \mathcal{DC}(\theta[\mathsf{ms}]) \mid_{\mathcal{EQ}_2}$, onde θ é o m.g.u das restrições de igualdade em $\mathcal{EQ}$ e $\mathcal{EQ}_2$ são as restrições de desigualdade. De forma similar, $\mathsf{m} \in \mathcal{DC}'(\mathsf{ms}') \mid_{\mathcal{EQ}'}$ se e somente se $\mathsf{m} \in \mathcal{DC}'(\theta'[\mathsf{ms}']) \mid_{\mathcal{EQ}'_2}$. Logo, nós só precisamos considerar as restrições de desigualdade $\mathcal{EQ}_2$ e $\mathcal{EQ}'_2$.

  Por contraposição, nós tentamos encontrar um termo $\mathsf{m} \in \mathcal{DC}(\mathsf{ms}) \mid_{\mathcal{EQ}_2}$ tal que $\mathsf{m} \notin \mathcal{DC}'(\mathsf{ms}') \mid_{\mathcal{EQ}'_2}$. Isso nos leva a duas possibilidades, onde $\mathsf{ssb}, \mathsf{ssb}'$ são substituições de símbolos tal que $\mathsf{ssb}[\mathsf{ms}] = \mathsf{ssb}'[\mathsf{ms}'] = \mathsf{m}$. Isso só pode existir se $\mathsf{ssb}[\theta[\mathsf{ms}']] = \mathsf{m}$ onde θ é a substituição correspondente dos símbolos em ms' para os termos em ms . Caso contrário, os termos ms e ms' não podem derivar os mesmos termos, já que para dois termos simbólicos derivarem os mesmos termos, o predicado $\mathsf{termApprox}(\mathsf{ms}, \mathsf{ms}', \mathcal{O}, \mathcal{O}')$ precisa ser verdadeiro (como mostrado na Definição 3.4.7 e corroborado pelo Teorema 3.4.1), o que exige uma substituição θ tal que $\theta[\mathsf{ms}'] = \mathsf{ms}$, e como $\mathsf{ssb}[\mathsf{ms}] = \mathsf{m}$, então $\mathsf{ssb}[\theta[\mathsf{ms}']] = \mathsf{m}$.

- ssb' não torna falsa nenhuma restrição em $\mathcal{EQ}'_2$ se e somente se $\mathsf{m} \notin \mathcal{DC}'(\mathsf{ms}')$ se e somente se pelo Lemma A.3.3 e pelo Teorema 3.4.1 $\mathsf{termApprox}(\mathsf{ms}, \mathsf{ms}', \mathcal{O}, \mathcal{O}')$ é falso;

- ssb'_1 torna falsa uma restrição $\mathsf{neq}(\mathsf{ms}_1, \mathsf{ms}_2) \in \mathcal{EQ}'_2$ se e somente se $\mathsf{ssb}_1[\theta(\mathsf{ms}_1)] = \mathsf{ssb}_1[\theta(\mathsf{ms}_2)]$ se e apenas se $\mathsf{canEq}(\theta(\mathsf{ms}_1), \theta(\mathsf{ms}_2), \mathcal{DC}, \mathcal{EQ})$ é verdadeiro (Lemma A.3.2).