Instituto de Computação, Universidade Estadual de Campinas

Verificação Formal de Protocolos de Trocas Justas Utilizando o Método de Espaços de Fitas

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Fabio Rogério Piva e aprovada pela Banca Examinadora.

Campinas, 9 de junho de 2009.

Prof. Ricardo Dahab (Orientador)
Instituto de Computação, Universidade Estadual de Campinas

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Crisllene Queiroz Custódio – CRB8 / 7966

Piva, Fabio Rogério
P688v Verificação formal de protocolos de trocas justas utilizando o método de espaços de fitas / Fabio Rogério Piva -- Campinas, [S.P. : s.n.], 2009.

Orientador : Ricardo Dahab
Dissertação (Mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Criptografia. 2. Espaços de fitas (Ciência da computação). 3. Metodos formais (Ciência da computação). 4. Redes de computação - Protocolos. 5. Redes de computação - Medidas de segurança. I. Dahab, Ricardo. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

(cqc/imecc)

Título em inglês: Formal verification of fair exchange protocols using the strand spaces method.

Palavras-chave em inglês (Keywords): 1. Cryptography. 2. Strand Spaces (Computer science). 3. Formal methods (Computer science). 4. Computer network protocols. 5. Computer network security.

Área de concentração: Ciência de Computação

Titulação: Mestre em Ciência de Computação

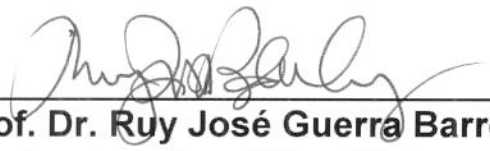
Banca examinadora: Prof. Dr. Ricardo Dahab (IC-Unicamp)
Prof. Dr. Ruy de Queiroz (CIN/UFPE)
Prof. Dr. Julio López (IC-Unicamp)

Data da defesa: 25/03/2009

Programa de Pós-Graduação: Mestrado em Ciência de Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 25 de março de 2009, pela Banca examinadora composta pelos Professores Doutores:



Prof. Dr. Ruy José Guerra Barreto de Queiroz
Centro de Informática / Universidade Federal de Pernambuco.

Prof. Dr. Julio César López Hernández
IC / UNICAMP.

Prof. Dr. Ricardo Dahab
IC / UNICAMP.

Instituto de Computação
Universidade Estadual de Campinas

Verificação Formal de Protocolos de Trocas Justas Utilizando o Método de Espaços de Fitas

Fabio Rogério Piva[1]

10 de fevereiro de 2009

Banca Examinadora:

- Prof. Ricardo Dahab (Orientador)
  Instituto de Computação, Universidade Estadual de Campinas
- Prof. Ruy de Queiroz
  Centro de Informática, Universidade Federal de Pernambuco
- Prof. Julio López
  Instituto de Computação, Universidade Estadual de Campinas
- Prof. Anderson C. Nascimento (Suplente)
  Departamento de Engenharia Elétrica, Universidade de Brasília
- Prof. Arnaldo V. Moura (Suplente)
  Instituto de Computação, Universidade Estadual de Campinas

[1]Suporte financeiro de: Bolsa CNPq (processo 2006/05219-7) 2007-2008.

Resumo

Os protocolos de trocas justas foram propostos como solução para o problema da troca de itens virtuais, entre duas ou mais entidades, sem que haja a necessidade de confiança entre elas. A popularização da internet criou uma crescente classe de usuários leigos que diariamente participam de transações de troca, como comércio eletrônico (*ecommerce*), *internet banking*, redes ponto-a-ponto (P2P), etc. Com tal demanda por justiça, é preciso garantir que os protocolos de trocas justas recebam a mesma atenção acadêmica dedicada aos protocolos clássicos. Neste contexto, fazem-se necessárias diretrizes de projeto, ferramentas de verificação, taxonomias de ataques e quaisquer outros artefatos que possam auxiliar na composição de protocolos sem falhas. Neste trabalho, apresentamos um estudo sobre o problema de trocas justas e o atual estado da arte das soluções propostas, bem como a possibilidade de criar, a partir de técnicas para a verificação formal e detecção de falhas em protocolos clássicos, metodologias para projeto e correção de protocolos de trocas justas.

Abstract

Fair exchange protocols were first proposed as a solution to the problem of exchanging digital items, between two or more entities, without forcing them to trust each other. The popularization of the internet resulted in an increasing amount of lay users, which constantly participate in exchange transactions, such as electronic commerce (ecommerce), internet banking, peer-to-peer networks (P2P), etc. With such demand for fairness, we need to ensure that fair exchange protocols receive the same amount of attention, from academia, as classic protocols do. Within this context, project guideliness are needed, and so are verification tools, taxonomies of attack, and whatever other artifacts that may help correct protocol design. In this work we present a study on the fair exchange problem and the current state-of-the-art of proposed solutions, as well as a discussion on the possibility of building, from currently available formal verification and attack detection techniques for classic protocols, methods for fair exchange protocols design and correction.

Agradecimentos

Esta dissertação é dedicada às seguintes pessoas, sem as quais não teria sido possível:

Ao professor Ricardo Dahab, pelos ensinamentos, amizade e paciência durante estes cinco anos de parceria.

À minha namorada, Karina, por ter participado de todos os altos e baixos deste início de carreira acadêmica, e por ter sido sempre não apenas namorada, mas também amiga, colega, revisora, psicóloga e muito mais.

Ao meu pai, Francisco, por sua honestidade, hombridade e caráter, que sempre me servirão como exemplo em qualquer exercício de minha vida.

À minha mãe, Vânia, por sua perseverança, energia e por apontar-me sempre o caminho correto a seguir – e encorajar-me a percorrê-lo mesmo quando eu hesitava.

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

Quando comparados a outros artefatos da Criptografia – como os algoritmos criptográficos, ou hardware seguro, por exemplo – os protocolos apresentam-se como componentes enganosamente simples; o desafio de estudar e projetar protocolos não está na necessidade de análise de inúmeras operações matemáticas sobre grupos numéricos, e tampouco em uma seqüência de passos suficientemente complexa ao ponto de tornar-se virtualmente impossível de revertê-la sem equipamento computacional específico. Na verdade, como aponta o pesquisador Roger Needham, protocolos criptográficos são "*programas de três linhas que as pessoas insistem em errar*", o que os torna ainda mais fascinantes.

Inicialmente, a aplicabilidade de protocolos criptográficos era restrita essencialmente a ambientes institucionais. Os usuários do sistema de uma organização tinham a necessidade de comunicarem-se à distância, ou conectarem-se a algum servidor para recuperarem arquivos pessoais, por exemplo. Assim, esta classe de protocolos – chamada de **protocolos clássicos** – visa resolver problemas de consenso, e podem ser classificados em duas categorias: **Protocolos de autenticação**, em que cada uma das entidades pretende certificar-se de que a outra é de fato quem diz ser; e **protocolos de estabelecimento de chaves**, em que as entidades pretendem estabelecer um canal de comunicação seguro, livre de interceptações. Protocolos clássicos mais robustos costumam buscar ambos os objetivos.

Todavia, a popularização da internet transformou um sólido grupo de usuários leigos em participantes rotineiros de transações eletrônicas. O comércio eletrônico (*ecommerce*), aplicações como *internet banking* e sites especializados em leilões dependem da interação entre usuários que, além de não serem especialistas em informática, podem ter interesses conflitantes quanto ao resultado da transação. Surgiram assim os chamados

protocolos diplomáticos, ou **protocolos de trocas justas** [4], que diferenciam-se dos protocolos clássicos pela natureza das entitades participantes: enquanto nos clássicos os participantes compartilham intenções quanto ao resultado da transação, preocupando-se com a possibilidade de que adversários externos ajam sobre o sistema, nos protocolos diplomáticos o adversário pode ser o próprio interlocutor. Assim, o protocolo deve ser suficientemente robusto para proteger as entidades honestas de adversários não só externos, mas possivelmentes internos. Em um contexto em que a confiança entre as entidades é limitada, novas técnicas de projeto fazem-se necessárias.

No estudo de protocolos clássicos, é costumeira a utilização de métodos formais de verificação para fornecer garantias de que um protocolo é de fato correto. Tais ferramentas podem, além de provar a correção de um protocolo, auxiliar na delicada tarefa de detectar e corrigir falhas de segurança. Como mesmo estas ferramentas formais não são perfeitas, os protocolos criptográficos sempre são rodeados por uma indissipável nuvem de dúvida – mesmo quando contam com mais de uma prova de correção, obtidas com diferentes técnicas.

Mesmo não fornecendo garantias definitivas, o uso de técnicas formais é de inestimável ajuda para projetistas e analistas de protocolos, auxiliando-os a ao menos minimizar a susceptibilidade dos protocolos a ataques. Por isso, é fundamental que também os protocolos diplomáticos – recentes, em comparação aos clássicos – possam contar com ferramentas de verificação. As diferenças estruturais das soluções de trocas justas são suficientes para inviabilizar a utilização da maioria dos métodos desenvolvidos para protocolos clássicos, e portanto novas técnicas, ou adaptações das antigas, devem ser propostas.

1.2 Objetivos

Este trabalho tem como objetivo principal o estudo de protocolos de trocas justas – projeto e propostas existentes – bem como a identificação de técnicas de verificação formal de protocolos clássicos que possam ser adaptadas para esta classe de protocolos, englobando os seguintes objetivos específicos:

1. Contribuição para o estado da arte de verificação de protocolos de trocas justas, investigando o método de espaços de fitas como candidato para adaptação para verificação de protocolos de trocas justas;

2. estudo da interação entre o método de espaços de fitas e os protocolos de trocas justas;

3. sugestão de novas diretrizes para o projeto de protocolos de trocas justas, resultantes

do aprofundamento do estudo da interação entre o método de espaços de fitas e tais protocolos;

4. identificação e correção de falhas de segurança e ataques em protocolos propostos na literatura.

1.3 Resultados obtidos

1. Aplicação do método de espaços de fitas a protocolos de trocas justas otimistas, resultando na detecção de novos ataques;
2. estudo qualitativo da propriedade de Tempestividade, fundamental para a obtenção de justiça em trocas, detalhado no Capítulo 3;
3. identificação do ataque do termo não-testável, efetivo em diversos protocolos otimistas conhecidos e apresentado na Seção 3.2.3;
4. sugestão de diretrizes para projeto de protocolos de trocas justas, detalhadas na Seção 3.3;
5. proposta de um protocolo otimista para a troca justa e tempestiva de itens genéricos entre duas entidades, apresentado na Seção 3.4;
6. artigos publicados em conferência [48], [47];
7. relatório técnico publicado no Instituto de Computação da UNICAMP [46].

1.4 Organização do documento

Este documento é organizado da seguinte forma: O Capítulo 2 é dedicado à apresentação dos conceitos estudados durante o mestrado. O Capítulo 3 dedica-se ao estudo qualitativo de uma das propriedades originais de trocas justas, a Tempestividade; neste capítulo, propomos um protocolo otimista e eficiente para a troca justa e tempestiva de dois itens genéricos. A seguir, o Capítulo 4 apresenta nossa proposta para modelagem de protocolos e propriedades de trocas justas com o método de espaços de fitas, e traz alguns novos ataques sobre protocolos de trocas justas otimistas. Por fim, revisitamos brevemente os problemas tratados e as atividades realizadas, bem como temas que consideramos interessantes para estudo futuro, no Capítulo 5. O Apêndice A traz as descrições dos protocolos criptográficos analisados no decorrer deste texto.

Capítulo 2

Revisão bibliográfica

Neste capítulo apresentaremos alguns dos trabalhos seminais que nos serviram como base, bem como uma descrição dos problemas abordados.

2.1 Protocolos criptográficos clássicos

Os primeiros protocolos de segurança, denominados **protocolos clássicos**, foram propostos para resolver problemas de consenso através de canais de comunicação inseguros. A aurora dos sistemas distribuídos previa cenários em que duas ou mais entidades pretendiam estabelecer chaves criptográficas seguras para comunicação posterior (**protocolos de estabelecimento de chaves**), ou mesmo certificar-se da verdadeira identidade de seus interlocutores (**protocolos de autenticação**).

Nesta seção, apresentaremos alguns dos principais focos de estudo sobre protocolos clássicos. Embora diferentes infraestruturas criptográficas permitam a implementação de diferentes famílias de protocolos, manteremos nossa apresentação no âmbito da criptografia simétrica, de forma a concentrar nossa atenção nas propriedades relevantes à nossa discussão. Uma abordagem mais completa sobre protocolos clássicos é fornecida por Boyd et al. [9].

2.1.1 Notação

As seguintes notações se fazem necessárias para o estudo de protocolos criptográficos:

i. A: identidade do iniciador, entidade que inicia a execução do protocolo;

ii. B: identidade do respondente, entidade que é contactada pelo iniciador;

iii. T, TTP, S: identidade da terceira parte confiável (TTP), entidade em que os demais participantes do protocolo confiam mutuamente;

iv. I: adversário que pretende burlar os objetivos de segurança do protocolo. No decorrer deste texto, os termos adversário, intruso, entidade desonesta e entidade maliciosa são intercambiáveis;

v. I_X: intruso I personificando uma outra entidade X;

vi. $A \rightarrow B : M$: transmição da mensagem M, enviada por A e recebida por B;

vii. N_X: *nonce* gerado pela entidade X. Um *nonce* é um número escolhido aleatoriamente pela entidade geradora, e é comumente utilizado em mecanismos de desafio-resposta para garantir autenticidade da execução do protocolo. A cada execução do protocolo, X deve gerar um *nonce* diferente;

viii. M, N: concatenação de duas mensagens M e N. No decorrer deste texto, as vírgulas podem ser omitidas por questões de clareza de representação;

ix. K_{XY}: chave simétrica compartilhada entre duas entidades X e Y;

x. $\{M\}_K$: texto cifrado obtido a partir do ciframento de M, utilizando um algoritmo simétrico e uma chave secreta K;

xi. $PU_T(M)$: texto cifrado obtido a partir do ciframento da mensagem M, utilizando um algoritmo assimétrico e a chave pública PU_T da TTP;

xii. $SIG_X(M)$: assinatura digital da entidade X sobre o texto M, verificável por qualquer outro participante do protocolo. Assume-se que M é recuperável a partir de $SIG_X(M)$, a não ser quando explicitado o contrário;

xiii. $H(M)$: resultado da aplicação de uma função H de hash criptográfico, não-inversível e resistente a colisões, à mensagem M;

2.1.2 Protocolos de autenticação

Grande parte das transações eletrônicas realizadas diariamente ocorre através de canais inseguros. Dessa forma, ao enviar informações pela rede, uma entidade não sabe se de fato alcançará o real destinatário, se suas mensagens serão interceptadas por algum adversário tentando passar-se por ele, ou mesmo se a comunicação será interrompida devido a alguma falha física na rede. Para viabilizar a simulação de canais seguros sobre uma infraestrutura sujeita a tais intermitências, os protocolos de autenticação visam permitir que duas ou mais entidades se certifiquem das identidades de seus interlocutores. Aplicações que envolvem assinatura de contratos, por exemplo, podem utilizar-se de uma fase de *setup* que envolva um protocolo de autenticação; de posse das verdadeiras identidades de seus pares,

as entidades podem então certificar-se de que o contrato será assinado pelas entidades corretas.

Talvez a principal aplicação de protocolos de autenticação seja em ambientes de autenticação de usuários. Um sem-número de serviços *online*, tais como *webmail*, *internet banking*, redes sociais, etc, possuem acesso baseado em pares *login* e senha, definidos previamente pelos usuários. Ao acessar um ambiente protegido, um usuário deve informar estes dois dados; o protocolo de autenticação deve assegurar-se de transmiti-los ao servidor do serviço sem revelá-los a eventuais adversários.

1. $A \rightarrow B : A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : \{N_B\}_{K_{AS}}$
4. $B \rightarrow S : \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$
5. $S \rightarrow B : \{N_B\}_{K_{BS}}$

Figura 2.1: Protocolo Woo-Lam Π para autenticação unilateral

A Figura 2.1 ilustra o protocolo de autenticação unilateral de Woo-Lam Π [60]. O objetivo deste protocolo é garantir a uma entidade B a identidade do interlocutor A, através de uma TTP S, com quem as entidades pré-compartilham chaves simétricas secretas; tais chaves podem representar senhas pré-definidas pelos usuários durante o cadastramento junto ao servidor S.

Inicialmente, a entidade A informa sua identidade a B. Como a primeira mensagem não é cifrada, ou assinada de qualquer forma, ela tem o objetivo de apenas iniciar a transação; não é possível para B, a partir desta informação, assumir que a entidade iniciadora do protocolo é de fato A (poderia ser um adversário tentando passar-se por A). B responde ao hipotético A com um desafio, um *nonce* N_B gerado aleatoriamente para aquela execução. A seguir, o hipotético A cifra o desafio de B e o responde, utilizando para isso sua chave compartilhada com o servidor S. Ao receber a mensagem 3, B não consegue verificar se o conteúdo é de fato N_B, pois não conhece a chave K_{AS} necessária para o deciframento. Assim, B recorre ao servidor S como oráculo[1], enviando na mensagem 4 o pacote a ser decifrado e a identidade esperada do gerador do pacote. O servidor S decifra o pacote recebido e reenvia seu conteúdo a B; se o valor for igual ao *nonce* gerado, B sabe que foi A de fato quem respondeu ao seu desafio.

[1]Um **oráculo** consiste em uma entidade do protocolo responsável por realizar ciframentos ou deciframentos para terceiros.

Embora aparentemente simples, o protocolo de Woo-Lam Π apresenta uma falha crucial, que impede o estabelecimento da autenticação unilateral para B. O fato de o termo enviado na mensagem 3 ser não-testável[2] para B, e não conter nenhuma informação sobre o respondente B (a não ser o *nonce* gerado por ele, que não pode ser diretamente vinculado à sua identidade), permite que um adversário I seja bem sucedido em se passar por A. Esta falha viabiliza os dois ataques registrados na literatura [18, 61] sobre este protocolo. A Figura 2.2 ilustra o ataque de Abadi [61].

1. $I_A \rightarrow B : A$
1'. $I \rightarrow B : I$
2. $B \rightarrow I_A : N_B$
2'. $B \rightarrow I : N'_B$
3. $I_A \rightarrow B : R$
3'. $I \rightarrow B : \{N_B\}_{K_{IS}}$
4. $B \rightarrow S : \{A, R\}_{K_{BS}}$
4'. $B \rightarrow S : \{I, \{N_B\}_{K_{IS}}\}_{K_{BS}}$
5. $S \rightarrow B : \{N_B\}_{K_{BS}}$

Figura 2.2: Ataque de Abadi ao protocolo Woo-Lam Π

Neste ataque, duas execuções do protocolo ocorrem em paralelo. A primeira (passos $1, 2, 3, 4$ e 5) é a execução atacada, ao final da qual o intruso consegue se autenticar como A para B. A segunda (passos $1', 2', 3'$ e $4'$) é uma execução legítima, em que o intruso se apresenta a B com sua real identidade, mas que não precisa ser concluída para que I atinja seus objetivos maliciosos. A estratégia do intruso é fazer com que B envie dois pacotes – recebidos nas mensagens 3 e $3'$ – a serem decifrados por S, mas receba apenas uma resposta do servidor. Dessa forma, B aceita a transação atacada – em que ele enviou N_B, que é o *nonce* retornado por S na mensagem 5 – e recusa a transação legítima, em que I intencionalmente enviou um termo-lixo R como resposta ao desafio N'_B de B. Por fim, B aceita I como sendo A, o que viola os objetivos do protocolo.

[2]A não-testabilidade de termos e seus efeitos sobre protocolos criptográficos serão discutidos no Capítulo 3.

2.1.3 Protocolos de estabelecimento de chaves

Muitas vezes, duas ou mais entidades desejam comunicar-se através de um canal físico inseguro durante um certo período de tempo, denominado **sessão**. Para manter as informações trocadas fora do alcance de observadores externos, as entidades podem executar um protocolo de estabelecimento de chaves, com o objetivo de obterem uma chave simétrica, mutuamente conhecida, para o ciframento das mensagens a serem trocadas posteriormente. Esta chave é denominada **chave de sessão** e tem validade apenas pela duração da comunicação; no evento de um novo contato, as entidades devem estabelecer uma nova chave.

Naturalmente, uma entidade não pode simplesmente gerar uma chave aleatória e transmiti-la em texto claro para seu interlocutor, já que o canal pode estar sendo observado por adversários; é fundamental que a chave de sessão estabelecida ao final do protocolo seja **boa para a comunicação entre** A **e** B (i.e., gerada por uma entidade confiável para esta tarefa, e conhecida apenas por A, B e possivelmente o centro gerador de chaves S). Assim, existem duas estratégias para o estabelecimento da chave de sessão entre duas entidades[3]: **transporte de chaves**, em que uma das entidades faz o papel de centro gerador de chaves e transmite a chave gerada à outra; e **acordo de chaves**, em que as duas entidades contribuem para a geração da chave de sessão. No restante desta seção, discutiremos exemplos para cada abordagem.

Transporte de chaves

Um protocolo de transporte de chaves atribui a uma entidade, em particular, a responsabilidade de gerar uma chave boa para a comunicação entre iniciador e respondente. Tal entidade pode tanto ser uma das interessadas na chave, especificamente A ou B, como uma TTP que faça as vezes de centro de distribuição de chaves. Qual destas abordagens é mais adequada depende apenas da infraestrutura disponível no ambiente em que o protocolo será implementado; caso exista um ou mais servidores centrais, que compartilhem segredos com os demais usuários, a última abordagem pode ser mais indicada, enquanto que um ambiente sem servidores pode favorecer a geração de chaves pelas próprias entidades interessadas.

O Kerberos [27] é talvez um dos mais amplamente utilizados e estudados protocolos para estabelecimento de chaves de sessão, e conta com diversas versões que visam objetivos variados. A Figura 2.3 ilustra uma versão básica do protocolo, modificada através do acréscimo de uma última mensagem para garantir autenticação mútua entre entidades (i.e., tanto A quanto B são assegurados da identidade de seus pares).

[3]Protocolos de estabelecimento de chaves para múltiplas entidades utilizam variantes de abordagens para duas entidades [9].

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{K_{AB}, B, L, N_A\}_{K_{AS}}, \{K_{AB}, A, L\}_{K_{BS}}$
3. $A \rightarrow B : \{A, T_A\}_{K_{AB}}, \{K_{AB}, A, L\}_{K_{BS}}$
4. $B \rightarrow A : \{T_A\}_{K_{AB}}$

Figura 2.3: Protocolo Kerberos básico

Nesta versão, o protocolo consegue estabelecer uma chave de sessão entre as entidades, bem como a supracitada autenticação mútua, através de uma TTP S, com quem as entidades pré-compartilham chaves simétricas secretas. Primeiramente, o iniciador A contacta o centro gerador de chaves S para pedir uma chave de sessão boa para a comunicação com B. O *nonce* N_A funciona como desafio de A para S, e permite a A comprovar que a mensagem 2 foi gerada recentemente, após o recebimento da mensagem 1. Além disso, A sabe que S foi o gerador da chave K_{AB}, pois só A e S conhecem a chave K_{AS} utilizada no ciframento da mensagem 2.

Acordo de chaves

Diferentemente do transporte de chaves, em um acordo de chaves nenhuma entidade é responsável única pela geração da chave de sessão. Estes protocolos oferecem mecanismos de geração de chaves em múltiplos passos, que utilizam segredos gerados individualmente por cada entidade para a constituição de um segredo compartilhado apenas pelos colaboradores.

Existem diversos algoritmos que permitem o acordo de chaves sem servidores [14, 20]. A Figura 2.4 mostra o protocolo básico de Diffie-Hellman [14], baseado no problema do logaritmo discreto. A segurança deste criptossistema depende da dificuldade de obter-se os expoentes r_B e r_A, bem como o segredo $g^{r_A r_B}$, a partir de valores públicos g, g^{r_A} e g^{r_B}.

$$
\begin{array}{rcl}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
t_A = g^{r_A} & \xrightarrow{\quad t_A \quad} & r_B \in_R \mathbb{Z}_q \\
 & & t_B = g^{r_B} \\
Z_{AB} = t_B^{r_A} & \xleftarrow{\quad t_B \quad} & Z_{AB} = t_A^{r_B}
\end{array}
$$

Figura 2.4: O protocolo de acordo de chaves de Diffie-Hellman. O gerador g do grupo multiplicativo G é público.

Inicialmente, A deve escolher aleatoriamente um *nonce* secreto r_A, e a partir de um valor público g, calcular $t_A = g^{r_A}$. Como é computacionalmente difícil para qualquer adversário obter o valor de r_A a partir de t_A e g, A envia sua colaboração t_A para B em texto claro. Este, por sua vez, procede da mesma forma: Escolhe um r_B secreto, calcula $t_B = g^{r_B}$ e envia este último valor a A. De posse de seus próprios *nonces* secretos e dos valores originados por seus pares, cada entidade pode então calcular o segredo compartilhado Z_{AB}:

$$Z_{AB} = t_B^{r_A} = (g^{r_B})^{r_A} = (g^{r_A})^{r_B} = t_A^{r_B} = Z_{AB} \tag{2.1}$$

A Equação 2.1 mostra como as entidades A e B são capazes de derivar individualmente o segredo Z_{AB}, que pode então servir como chave de sessão. Embora este protocolo em si não ofereça garantias de autenticação – nenhuma das entidades pode ter certeza de que realmente estabeleceu uma chave de sessão com o interlocutor pretendido – existem inúmeras extensões que o tornam mais prático e seguro [9].

2.1.4 Taxonomias de ataques e falhas

Um **ataque** consiste em uma seqüência de passos realizados por um adversário, explorando uma ou mais **falhas de segurança**, com o objetivo de contornar um ou mais objetivos de segurança do protocolo. Embora ataques variem grandemente no que diz respeito à gravidade, metodologia e objetivos do adversário, existem estudos que pretendem categorizá-los com relação a características em comum. Nesta seção apresentaremos duas destas taxonomias [54, 23].

Em sua abordagem, Syverson [54] classifica uma ampla gama de ataques comuns, que inclui grande parte dos ataques incidentes sobre protocolos clássicos. Nesta classe de ataques, comumente denominados **ataques de *replay***, presume-se que o adversário tenha poderes de observação sobre os canais de comunicação, podendo capturar mensagens trocadas entre participantes honestos durante diversas execuções do protocolo atacado. A partir das mensagens observadas, ele constrói novas mensagens e as envia às demais entidades, a fim de passar-se por outro usuário (**ataque de *man in the middle*** [9]), forçar que um participante honesto forneça informação sensível (**ataque de oráculo** [9]) ou simplesmente inviabilizar o estabelecimento de uma chave de sessão para comunicação posterior entre dois participantes (**ataque de negação de serviço (DoS)** [9]).

A Figura 2.5 ilustra a taxonomia de Syverson, estabelecida sobre dois aspectos fundamentais: **origem de mensagens** (externa ou interna ao protocolo atacado), e **destino de mensagens** (que estabelece uma relação entre o participante que deveria

receber a informação interceptada pelo adversário, e o participante para o qual tal informação é retransmitida durante o ataque).

Em sua análise de origem das mensagens, Syverson conclui que as informações utilizadas pelo adversário podem provir de execuções externas (**ataques de execução externa**) ou de mensagens previamente enviadas durante a execução atual, sobre a qual o ataque está sendo realizado (**ataques de execução interna**). No primeiro caso, as execuções externas podem tanto estar ocorrendo paralelamente à execução atacada (**intercalações**), como podem ser execuções antigas (***replays* clássicos**) cujas mensagens foram observadas e armazenadas pelo adversário.

Quando o adversário intercepta mensagens do protocolo, sejam elas externas ou internas à execução atual, torna-se impossível prever se/quando tais mensagens chegarão ao seu destino original. Isso justifica o segundo aspecto analisado por Syverson em seu trabalho: o destino das mensagens. Existem três possibilidades, classificadas pelo autor em duas categorias: As informações podem sofrer **deflexões** – quando são desviadas de seu destino original – ou ***replays* diretos** – quando são transmitidas pelo adversário ao destinatário original, porém em algum ponto no futuro, após sofrerem atraso. No primeiro caso, as deflexões podem ser para o próprio remetente (**reflexões**) ou para participantes diferentes do remetente e destinatário, que incluem o próprio adversário (**deflexões para uma terceira parte**).

Outros autores apresentam abordagens diferentes da de Syverson para o estudo de ataques. Gritzalis et al. [23], por exemplo, categorizam falhas de segurança, ao invés dos ataques que as exploram. Dessa forma, os autores não se limitam apenas a ataques de replay, mas buscam analisar todo o processo de engenharia de protocolos de segurança – desde seu projeto até sua utilização pelo usuário final, passando pela implementação e escolha de criptossistemas. A Figura 2.6 ilustra a taxonomia de falhas proposta. Devemos notar que ela é fortemente baseada na de Syverson, especialmente o item 3 (falhas de mensagens).

2.2 Verificação de protocolos clássicos

Nesta seção apresentaremos algumas técnicas formais para a verificação de protocolos de autenticação e estabelecimento de chaves. O estudo das diferentes metodologias nos permitiu entrar em contato com o método de espaços de fitas [57], descrito na Seção 2.2.3, e identificá-lo como candidato a metodologia de verificação para protocolos de trocas justas.

Podemos classificar os métodos de verificação formal em três classes:

- **Verificação baseada em modelos**: Técnicas de verificação baseadas em modelos consideram um número grande, porém finito, de comportamentos possíveis dos

1. **Ataques de execução externa**: *replay* de mensagens externas à atual execução do protocolo
   (a) **Intercalações**: requerem execuções contemporâneas do protocolo
      i. **Deflexões**: mensagem é direcionada a um participante diferente do intencionado
         A. **Reflexões**: mensagem é enviada de volta ao rementente
         B. **Deflexões para uma terceira parte**: mensagem é enviada a uma entidade diferente do destinatário e do rementente
      ii. ***Replays* diretos**: participante intencionado recebe a mensagem, mas ela sofre atraso
   (b) ***Replays* clássicos**: execuções não precisam ser contemporâneas
      i. **Deflexões**: mensagem é direcionada a um participante diferente do intencionado
         A. **Reflexões**: mensagem é enviada de volta ao rementente
         B. **Deflexões para uma terceira parte**: mensagem é enviada a uma entidade diferente do destinatário e do rementente
      ii. ***Replays* diretos**: participante intencionado recebe a mensagem, mas ela sofre atraso
2. **Ataques de execução interna**: *replay* de mensagens internas à atual execução do protocolo
   (a) **Deflexões**: mensagem é direcionada a um participante diferente do intencionado
      i. **Reflexões**: mensagem é enviada de volta ao rementente
      ii. **Deflexões para uma terceira parte**: mensagem é enviada a uma entidade diferente do destinatário e do rementente
   (b) ***Replays* diretos**: participante intencionado recebe a mensagem, mas ela sofre atraso

Figura 2.5: Taxonomia de Syverson para ataques de *replay*

1. **Falhas elementares de protocolos**
2. **Falhas de adivinhação de senhas/chaves**
3. **Falhas de mensagens**
4. **Falhas de sessões paralelas**
5. **Falhas internas aos protocolos**
6. **Falhas de criptossistema**

Figura 2.6: Taxonomia de Gritzalis et al. para falhas de segurança

protocolos, e permitem verificar se tais comportamentos satisfazem determinadas condições de segurança. Estas técnicas são geralmente mais adequadas para a detecção de ataques em protocolos, do que para provar sua correção. Alguns exemplos de técnicas baseadas em modelos são Brutus [33], FDR [49] e Murϕ [15].

- **Verificação baseada em lógicas e provas de teoremas**: Estas técnicas permitem verificar um protocolo garantindo que um conjunto de condições de segurança, enunciadas na forma de teoremas, são satisfeitas pela especificação. Por isso, são geralmente mais adequadas para demonstrar a correção de protocolos, do que para identificar ataques. Alguns exemplos são a Lógica BAN [10] e a Lógica de Syverson (SVO) [55].

- **Híbridos**: São métodos que apresentam tanto características de modelagem quanto de lógicas. Tais técnicas normalmente traduzem a especificação do protocolo a ser analisado para um modelo bem definido, e então passam à prova de teoremas sobre este modelo. Alguns exemplos são o Método de Espaços de Fitas [57] e o analisador NRL [34].

Dedicaremos o restante da seção à descrição de três técnicas que, inicialmente, consideramos candidatas para a análise de protocolos de trocas justas. Tal discussão encontra-se disponível também em relatório técnico [46].

2.2.1 Lógica BAN

A Lógica de Michael Burrows, Martín Abadi e Roger Needham (BAN) [10] foi apresentada à comunidade em março de 1988, e utiliza um paradigma de análise baseada na crença de entidades; por exemplo, no caso de um protocolo destinado à autenticação mútua

entre dois participantes, é necessário que ao final da execução do protocolo cada um dos envolvidos **acredite** (isto é, tenha prova irrefutável de) que o outro participante também participou desta execução do protocolo e que ele interagiu, de maneira passiva ou ativa, nesta execução.

Sintaxe

Inspiradora de diversas lógicas subseqüentes – que não conseguiram manter sua intuitividade e simplicidade – a BAN conta com uma sintaxe própria para tornar o protocolo suscetível à análise. Algumas das notações mais utilizadas são:

- $P \models X$: P **acredita** em X, em particular, a entidade P acredita que X é verdadeiro.
- $P \triangleleft X$: P **vê** X. Alguém enviou a P uma mensagem contendo X e P consegue extrair X desta mensagem (possivelmente após um deciframento).
- $P \mid\sim X$: P **alguma vez disse** X. A entidade P enviou alguma mensagem contendo X, nesta ou em alguma execução passada do protocolo. Só o que se pode afirmar é que P acreditava em X quando enviou a mensagem que o continha.
- $P \Rightarrow X$: P tem **jurisdição** sobre X. A entidade P tem autoridade sobre X e é confiável para tal. Se todas as entidades acreditam que uma TTP tem jurisdição sobre a geração de chaves de sessão, por exemplo, então esta TTP deve gerar chaves boas, recentes e suficientemente aleatórias para a comunicação destas entidades.
- $\sharp(X)$: **frescor** de X. Significa que X é **recente**, isto é, não foi enviado em alguma mensagem de uma execução passada do protocolo. Essa construção é normalmente utilizada para *nonces* (números utilizados apenas uma única vez), *timestamps* e chaves sabidamente boas.
- $P \overset{K}{\longleftrightarrow} Q$: A chave K é **boa** (conhecida apenas pelas entidades honestas a que se destina) para a comunicação entre P e Q.
- $\{X\}_K$: X está **cifrado** com a chave K.

Postulados

Toda lógica é, em geral, composta por uma sintaxe própria (apresentada na subseção anterior) e por postulados que conduzem a análise em si. No restante desta seção, utilizaremos a seguinte notação para enunciar postulados e demonstrar propriedades:

$$\frac{exp_1, exp_2}{exp_3}$$

Esta notação significa que, se as expressões exp_1 e exp_2 valem, então exp_3 também vale. A capacidade de derivar novas expressões a partir de um conjunto de expressões conhecidas é a chave para as demonstrações em lógicas, já que as novas expressões podem ser incorporadas ao conjunto e, então, utilizadas na derivação de outras novas expressões, de forma iterativa.

A lógica BAN traz um número relativamente pequeno de postulados, o que contribui muito para sua natureza compacta. Os principais postulados são:

- **Significado de mensagem (*Message-meaning*)**: Este postulado diz respeito à interpretação de mensagens com campos cifrados. Ele deriva crença a partir de uma mensagem recebida.

$$\frac{P \models P \overset{K}{\leftrightarrow} Q, P \triangleleft \{X\}_K}{P \models Q \mid\sim X}$$

- **Verificação de *nonce* (*Nonce-verification*)**: Este postulado vincula o frescor de algo que foi dito à crença neste termo[4].

$$\frac{P \models \sharp(X), P \models Q \mid\sim X}{P \models Q \models X}$$

- **Jurisdição (*Jurisdiction*)**: Permite derivar crença a partir da jurisdição.

$$\frac{P \models Q \Rightarrow X, P \models Q \models X}{P \models X}$$

Alguns destes postulados apresentam variantes que os tornam aplicáveis a protocolos que utilizem chaves assimétricas. No entanto, tais formas não foram incluídas neste trabalho (os exemplos aqui tratados são de protocolos simétricos), mas podem ser estudadas na proposta original da lógica [10]. Além dos postulados apresentados, a lógica BAN conta ainda com uma série de propriedades mais simples que serão comentadas quando se fizerem necessárias.

[4]Este postulado é fundamental para a verificação de protocolos que têm como objetivo a autenticação de entidades. Em diversas demonstrações chegamos ao ponto em que uma expressão do tipo $P \models Q \models Nonce$ surge. Ao contrário da intuitividade característica da lógica BAN, o significado desta expressão não é nada evidente: o que pode significar o fato de P acreditar que Q crê em um *nonce*? Concluímos que esta expressão representa uma propriedade muito comum na autenticação de entidades: atividade (*liveness*). Se P acredita que Q acredita em algo comprovadamente recente, então é claro que P tem evidência de que Q está ativo. Esta equivalência foi também observada por Paul Syverson em sua discussão sobre a BAN no artigo sobre a Lógica SVO [55]. É interessante que este fato tenha recebido tão pouca atenção por parte dos autores da lógica.

Verificação do protocolo de Needham-Schroeder com chaves secretas (NSSK)

- **Descrição do protocolo**

  1. $A \rightarrow S : A, B, N_A$
  2. $S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
  3. $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
  4. $B \rightarrow A : \{N_B\}_{K_{AB}}$
  5. $A \rightarrow B : \{N_B - 1\}_{K_{AB}}$

- **Idealização do protocolo**

  1. $\langle \text{Vazio} \rangle$[5]
  2. $S \rightarrow A : \{N_A, A \overset{K_{AB}}{\longleftrightarrow} B, \sharp(A \overset{K_{AB}}{\longleftrightarrow} B), \{A \overset{K_{AB}}{\longleftrightarrow} B\}_{K_{BS}}\}_{K_{AS}}$
  3. $A \rightarrow B : \{A \overset{K_{AB}}{\longleftrightarrow} B\}_{K_{BS}}$
  4. $B \rightarrow A : \{N_B, A \overset{K_{AB}}{\longleftrightarrow} B\}_{K_{AB}}$
  5. $A \rightarrow B : \{N_B, A \overset{K_{AB}}{\longleftrightarrow} B\}_{K_{AB}}$

  Note que a mensagem 1 não é considerada na análise da lógica BAN. A BAN só considera na análise mensagens cifradas, descartando qualquer texto claro. Além disso, a mensagem 2 contém tanto $A \overset{K_{AB}}{\longleftrightarrow} B$ quanto $\sharp(A \overset{K_{AB}}{\longleftrightarrow} B)$. Se a segunda informação não fosse incluída na idealização da mensagem, não seria possível derivar que A confia que a chave K_{AB} é recente. Analisando o protocolo, sabemos que A pode ter esta confiança porque a chave está contida em uma resposta de S, que também contém o *nonce* N_A. Todavia, não conseguimos derivar esta crença (a de que a chave é recente) usando apenas os postulados da lógica, o que deveria ser natural. Acreditamos que uma possível solução para este impasse seria acrescentar o seguinte postulado à lógica:

$$\frac{P \models Q \mapsto Y, P \models Q \models (X, Y)}{P \models \sharp(Y)}$$

- **Premissas do protocolo**

[5]Por ser uma técnica que analisa propriedades de segurança a partir da crença de entidades, a Lógica BAN desconsidera mensagens iniciais que não contenham componentes cifrados, já que estas não contribuem para a derivação de crença alguma para o destinatário. Isso reflete a possibilidade de que componentes claros sejam forjados por algum adversário, com o intuito de personificar um participante honesto.

1. $A \models A \xleftrightarrow{K_{AS}} S$
2. $S \models A \xleftrightarrow{K_{AS}} S$
3. $B \models B \xleftrightarrow{K_{BS}} S$
4. $S \models B \xleftrightarrow{K_{BS}} S$
5. $S \models A \xleftrightarrow{K_{AB}} B$
6. $S \models \sharp(A \xleftrightarrow{K_{AB}} B)$
7. $A \models S \Rightarrow A \xleftrightarrow{K_{AB}} B$
8. $B \models S \Rightarrow A \xleftrightarrow{K_{AB}} B$
9. $A \models S \Rightarrow \sharp(A \xleftrightarrow{K_{AB}} B)$
10. $B \models S \Rightarrow \sharp(A \xleftrightarrow{K_{AB}} B)$
11. $A \models \sharp(N_A)$
12. $B \models \sharp(N_B)$
13. $B \models \sharp(A \xleftrightarrow{K_{AB}} B)$

A premissa 13 não é natural, mas sem ela não é possível garantir a B que a chave estabelecida é boa. Assim, para que o protocolo tenha sucesso no estabelecimento de chave de sessão entre A e B, é necessário supor que B utilizará a chave K_{AB} na encriptação da mensagem 4, mesmo sem a garantia de que ela foi recentemente gerada – uma afirmação um tanto duvidosa. A necessidade dessa premissa ficará mais clara no decorrer da análise do protocolo.

- **Análise do protocolo**

  Da mensagem 2 e da premissa 11 derivamos:

  – $A \models S \sim (N_A, A \xleftrightarrow{K_{AB}} B, \sharp(A \xleftrightarrow{K_{AB}} B), \{A \xleftrightarrow{K_{AB}} B\}_{K_{BS}})$

  – $$\frac{A \models \sharp(N_A)}{A \models \sharp(N_A, A \xleftrightarrow{K_{AB}} B, \sharp(A \xleftrightarrow{K_{AB}} B), \{A \xleftrightarrow{K_{AB}} B\}_{K_{BS}})}$$

  – $$\frac{A \models S \sim (N_A, A \xleftrightarrow{K_{AB}} B, \sharp(A \xleftrightarrow{K_{AB}} B), \{A \xleftrightarrow{K_{AB}} B\}_{K_{BS}}), A \models \sharp(N_A, A \xleftrightarrow{K_{AB}} B, \sharp(A \xleftrightarrow{K_{AB}} B), \{A \xleftrightarrow{K_{AB}} B\}_{K_{BS}})}{A \models S \models (N_A, A \xleftrightarrow{K_{AB}} B, \sharp(A \xleftrightarrow{K_{AB}} B), \{A \xleftrightarrow{K_{AB}} B\}_{K_{BS}})}$$

  – $$\frac{A \models S \models (N_A, A \xleftrightarrow{K_{AB}} B, \sharp(A \xleftrightarrow{K_{AB}} B), \{A \xleftrightarrow{K_{AB}} B\}_{K_{BS}})}{A \models S \models A \xleftrightarrow{K_{AB}} B}$$

  – $$\frac{A \models S \models A \xleftrightarrow{K_{AB}} B, A \models S \Rightarrow A \xleftrightarrow{K_{AB}} B}{A \models A \xleftrightarrow{K_{AB}} B} \qquad (2.1)$$

O resultado obtido (2.1) demonstra que o protocolo alcança o objetivo de garantir a A uma boa chave para comunicar-se com B. Só é possível estender esta conclusão para B se utililizarmos a premissa-problema 13:

$$\_\ \frac{B \models S \mid\sim A \overset{K_{AB}}{\longleftrightarrow} B, B \models \sharp(A \overset{K_{AB}}{\longleftrightarrow} B)}{B \models S \models A \overset{K_{AB}}{\longleftrightarrow} B}$$

Da premissa 8 derivamos:

$$\_\ \frac{B \models S \Rightarrow A \overset{K_{AB}}{\longleftrightarrow} B, B \models A \overset{K_{AB}}{\longleftrightarrow} B}{B \models A \overset{K_{AB}}{\longleftrightarrow} B} \qquad (2.2)$$

Concluímos de (2.1) e (2.2) que ambas as entidades acreditam na chave K_{AB}. Isso ainda não é suficiente para que a entidade A conclua que esta chave é boa – para tanto, resta derivarmos que tal chave é recente:

$$\_\ \frac{A \models S \models \{N_A, A \overset{K_{AB}}{\longleftrightarrow} B, \sharp(A \overset{K_{AB}}{\longleftrightarrow} B), \{A \overset{K_{AB}}{\longleftrightarrow} B\}_{K_{BS}}\}_{K_{AS}}}{A \models S \models \sharp(A \overset{K_{AB}}{\longleftrightarrow} B)}$$

$$\_\ \frac{A \models S \models \sharp(A \overset{K_{AB}}{\longleftrightarrow} B), A \models S \Rightarrow \sharp(A \overset{K_{AB}}{\longleftrightarrow} B)}{A \models \sharp(A \overset{K_{AB}}{\longleftrightarrow} B)} \qquad (2.3)$$

Observe que a expressão $\sharp(A \overset{K_{AB}}{\longleftrightarrow} B)$ da idealização da mensagem 2 exerce um papel fundamental na obtenção do resultado (2.3).

- **Conclusões**

  A análise mostrou que o NSSK consegue estabelecer uma chave de sessão boa para comunicação entre duas entidades A e B, e que ambas as entidades conseguem a crença de que esta chave de fato é boa.

Conclusão

A lógica BAN é a pioneira em um ramo da verificação formal que testa a correção de protocolos através de uma sintaxe lógica própria. Embora seus postulados sejam intuitivos e permitam análises diretas, este método ainda apresenta algumas fraquezas, como a falta de suporte a características especiais dos protocolos (*hashes*, MACs, funções não-inversíveis, algoritmos baseados em acordo de chaves, etc.), e mostra-se insensível à estrutura das mensagens, o que possibilita que ataques de tipagem passem despercebidos (tais ataques normalmente exploram a inversão na ordem de campos da mensagem ou violação do tipo esperado em algum campo das mesmas). No entanto, a BAN ainda

é uma ótima opção para a análise inicial de um novo protocolo, assim como para um primeiro contato com métodos formais de verificação.

2.2.2 Lógica de Syverson (SVO)

Proposta por Paul Syverson e Iliano Cervesato como uma extensão da lógica de Paul van Oorschot (nomeada VO), a SVO [55] é um exemplo evidente de lógica derivada da BAN. Destinada a responder algumas das perguntas deixadas em aberto pela BAN, a SVO fornece uma gama de postulados muito maior do que aquela fornecida por sua precursora.

Sintaxe

A sintaxe da SVO é basicamente a mesma utilizada pela BAN, com o acréscimo das seguintes notações:

- P *says* X: X é uma mensagem (ou parte de uma mensagem) que P **disse recentemente**. Esta construção engloba os operadores de **frescor** ($\sharp()$) e **alguma vez disse** ($\mid\sim$) da BAN.
- $\neg\phi$: **Negação** da fórmula ϕ.
- P *has* X: P **tem** a mensagem X. Esta mensagem pode ter sido recebida por P, gerada recentemente por P, ou simplesmente estar disponível a P inicialmente. X também pode ter sido construída por P a partir de mensagens que se enquadrem em quaisquer dos casos anteriores.
- $|X|_{*P}$: X, **segundo** P. Este operador confere a P a capacidade de reconhecer mensagens que não consegue decifrar (contendo portanto nossa noção de termos não-testáveis, apresentada no Capítulo 3), mas que tenham sido previamente recebidas por ele. Assim, embora P não saiba o significado de X, ele reconhecerá X como uma mensagem conhecida caso torne a recebê-la no futuro.
- $[X]_K$: X **assinado** com a chave K. A SVO diferencia ciframentos de assinaturas digitais.

Postulados

Assim como sua predecessora, a SVO também conduz a análise através de derivações a partir de postulados lógicos. No entanto, a SVO equipa o pesquisador com uma gama de postulados muito mais extensa:

- **Modus ponens**

  Sejam ϕ e ψ fórmulas quaisquer:

$$\frac{\phi, \phi \rightarrow \psi}{\psi}$$

- **Necessitação**

$$\frac{\phi \text{ é teorema}}{P \models \phi}$$

- **Axiomas de crença**

  1. $(P \models \phi \wedge P \models (\phi \rightarrow \psi)) \rightarrow P \models \psi$
  2. $P \models (\phi \rightarrow \psi)$
  3. $P \models \phi \rightarrow P \models (P \models \phi)$
  4. $\neg(P \models \phi) \rightarrow P \models \neg(P \models \phi)$

- **Axiomas de associação à origem**

  1. $(P \overset{K}{\leftrightarrow} Q \wedge R \triangleleft \{X \text{ from } Q\}_K) \rightarrow (Q \mid\sim X \wedge Q \text{ has } X)$

- **Axiomas de acordo de chaves**: Estes axiomas são interessantes para a análise de protocolos que se utilizem de acordo de chaves (Seção 2.1.3), especialmente aqueles que o fazem através do algoritmo de Diffie-Hellman [14]. Como o exemplo aqui tratado não utiliza tais axiomas, seguiremos a proposta inicial de não incluí-los em nossa discussão.

- **Axiomas de recebimento**

  1. $P \triangleleft (X_i, \ldots, X_n) \rightarrow P \triangleleft X_i$, para $i = 1, \ldots, n$;
  2. $(P \triangleleft \{X\}_K \wedge P \text{ has } K) \rightarrow P \triangleleft X$.

- **Axiomas de posse**

  1. $P \triangleleft X \rightarrow P \text{ has } X$;
  2. $P \text{ has } (X_i \ldots, X_n) \rightarrow P \text{ has } X_i$, para $i = 1, \ldots, n$;
  3. $(P \text{ has } X_1 \wedge \ldots \wedge P \text{ has } X_n) \rightarrow P \text{ has } F(X_1, \ldots, X_n)$, onde $F()$ é uma função computável por P.

- **Axiomas de discurso**

1. $P \mid\sim (X_1, \ldots, X_n) \rightarrow P \mid\sim X_i \wedge P \; has \; X_i$, para $i = 1, \ldots, n$;
2. $P \; says \; (X_1, \ldots, X_n) \rightarrow (P \mid\sim (X_i, \ldots, X_n) \wedge P \; says \; X_i)$, para $i = 1, \ldots, n$.

- **Axiomas de frescor (*freshness*)**

1. $\sharp(X_i) \rightarrow \sharp((X_i, \ldots, X_n))$, para $i = 1, \ldots, n$;
2. $\sharp((X_1, \ldots, X_n)) \rightarrow \sharp(F(X_i, \ldots, X_n))$.

- **Axiomas de jurisdição e verificação de *nonces***

1. $(P \mapsto \phi \wedge P \; says \; \phi) \rightarrow \phi$;
2. $(\sharp(X) \wedge P \mid\sim X) \rightarrow P \; says \; X$.

Como é possível observar, não só existem mais postulados na SVO do que na BAN, como tais postulados não são tão intuitivos. Isso é sem dúvida uma desvantagem da SVO em relação à BAN (no sentido de aprender e dominar a lógica), mas é o preço pago por uma lógica mais completa.

Verificação do protocolo de Needham-Schroeder com chaves secretas (NSSK)

- **Descrição do protocolo**

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
4. $B \rightarrow A : \{N_B\}_{K_{AB}}$
5. $A \rightarrow B : \{N_B - 1\}_{K_{AB}}$

- **Premissas do protocolo**

1. $A \mid\equiv A \overset{K_{AS}}{\longleftrightarrow} S$
2. $S \mid\equiv A \overset{K_{AS}}{\longleftrightarrow} S$
3. $B \mid\equiv B \overset{K_{BS}}{\longleftrightarrow} S$
4. $S \mid\equiv B \overset{K_{BS}}{\longleftrightarrow} S$
5. $S \mid\equiv A \overset{K_{AB}}{\longleftrightarrow} B$
6. $S \mid\equiv \sharp(A \overset{K_{AB}}{\longleftrightarrow} B)$
7. $A \mid\equiv S \mapsto A \overset{K_{AB}}{\longleftrightarrow} B$

8. $B \models S \mapsto A \overset{K_{AB}}{\longleftrightarrow} B$
9. $A \models S \mapsto \sharp(A \overset{K_{AB}}{\longleftrightarrow} B)$
10. $B \models S \mapsto \sharp(A \overset{K_{AB}}{\longleftrightarrow} B)$
11. $A \models \sharp(N_A)$
12. $B \models \sharp(N_B)$
13. $B \models \sharp(A \overset{K_{AB}}{\longleftrightarrow} B)$

Surge aqui a primeira "desilusão" com relação à SVO. A premissa 13, necessária para a conclusão da análise do NSSK com a BAN, é também necessária na verificação com a SVO. Seria de se esperar que em uma lógica melhor equipada, como a SVO, poderíamos conduzir a análise deste protocolo sem esta premissa artificial.

- **Anotação do protocolo**

  Este passo consiste unicamente em evidenciar quais mensagens foram recebidas por cada entidade durante a execução do protocolo:

  1. $S \triangleleft (A, B, N_A)$
  2. $A \triangleleft \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
  3. $B \triangleleft \{K_{AB}, A\}_{K_{BS}}$
  4. $A \triangleleft \{N_B\}_{K_{AB}}$
  5. $B \triangleleft \{N_B - 1\}_{K_{AB}}$

- **Compreensão**

  A partir das mensagens recebidas, cada entidade pode estabelecer algumas crenças. Tais crenças são explicitadas neste passo:

  1. $S \models S \triangleleft (A, B, |N_A|_{*S})$
  2. $A \models A \triangleleft \{N_A, B, |K_{AB}|_{*A}, |\{K_{AB}, A\}_{K_{BS}}|_{*A}\}_{K_{AS}}$
  3. $B \models B \triangleleft \{|K_{AB}|_{*B}, A\}_{K_{BS}}$
  4. $A \models A \triangleleft \{|N_B|_{*A}\}_{|K_{AB}|_{*A}}$
  5. $B \models B \triangleleft \{N_B - 1\}_{|K_{AB}|_{*B}}$

  É interessante observar como o *nonce* N_B é representado nas mensagens 4 e 5. Na mensagem 4, o *nonce* aparece com a notação de desconhecimento. Isto porque, embora A suponha que este valor seja um *nonce*, ele não tem significado algum

para esta entidade. Na mensagem 5, por outro lado, B reconhece o *nonce* como sendo o valor aleatoriamente gerado por ele.

O mesmo ocorre com a chave K_{AB}, recentemente gerada por S. Nem A nem B reconhecem este valor como algo previamente conhecido ou que apresente significado semântico.

- **Interpretação**

  Neste passo aplicamos os axiomas da linguagem às expressões derivadas nos três passos anteriores.

  1. $\langle \text{Vazio} \rangle$[6]
  2. $A \models A \triangleleft \{N_A, B, |K_{AB}|_{*A}, |\{K_{AB}, A\}_{K_{BS}}|_{*A}\}_{K_{AS}} \rightarrow$
     $A \models A \triangleleft \{N_A, B, A \xleftrightarrow{|K_{AB}|_{*A}} B, \sharp(|K_{AB}|_{*A}), |\{K_{AB}, A\}_{K_{BS}}|_{*A}\}_{K_{AS}}$
  3. $B \models B \triangleleft \{|K_{AB}|_{*B}, A\}_{K_{BS}} \rightarrow B \models B \triangleleft \{A \xleftrightarrow{|K_{AB}|_{*B}} B, \sharp(|K_{AB}|_{*B}), A\}_{K_{BS}}$
  4. $(A \models A \triangleleft \{|N_B|_{*A}\}_{|K_{AB}|_{*A}}) \wedge (A \models A \xleftrightarrow{|K_{AB}|_{*A}} B) \rightarrow$
     $A \models A \triangleleft \{|N_B|_{*A}, A \xleftrightarrow{|K_{AB}|_{*A}} B\}_{|K_{AB}|_{*A}}$
  5. $(B \models B \triangleleft \{N_B - 1\}_{|K_{AB}|_{*B}}) \wedge (B \models A \xleftrightarrow{|K_{AB}|_{*B}} B) \rightarrow$
     $B \models B \triangleleft \{N_B - 1, A \xleftrightarrow{|K_{AB}|_{*B}} B\}_{|K_{AB}|_{*B}}$

- **Derivação**

  Da Compreensão 2 e da Interpretação 2 derivamos, por *Modus Ponens*:

  $$A \models A \triangleleft \{N_A, B, A \xleftrightarrow{|K_{AB}|_{*A}} B, \sharp(|K_{AB}|_{*A}), |\{K_{AB}, A\}_{K_{BS}}|_{*A}\}_{K_{AS}} \quad (D1)$$

  De D1 e da Premissa 1, derivamos, por Associação à Origem:

  $$A \models S \mid\sim (N_A, B, A \xleftrightarrow{|K_{AB}|_{*A}} B, \sharp(|K_{AB}|_{*A}), |\{K_{AB}, A\}_{K_{BS}}|_{*A}) \quad (D2_a)$$
  $$A \models S \textit{ has } (N_A, B, A \xleftrightarrow{|K_{AB}|_{*A}} B, \sharp(|K_{AB}|_{*A}), |\{K_{AB}, A\}_{K_{BS}}|_{*A}) \quad (D2_b)$$

  A partir da Premissa 11, por Frescor:

  $$A \models \sharp(N_A) \rightarrow A \models \sharp(N_A, B, A \xleftrightarrow{|K_{AB}|_{*A}} B, \sharp(|K_{AB}|_{*A}), |\{K_{AB}, A\}_{K_{BS}}|_{*A}) \quad (D3)$$

  Pelas Derivações 2 e 3, por Verificação de *Nonces*, concluímos:

[6] Assim como a Lógica BAN, a SVO desconsidera mensagens iniciais em texto claro.

$$A \models S \text{ says } (N_A, B, A \overset{|K_{AB}|_{*A}}{\longleftrightarrow} B, \sharp(|K_{AB}|_{*A}), |\{K_{AB}, A\}_{K_{BS}}|_{*A}) \quad (D4)$$

Da Derivação 4, por Axioma de Discurso:

$$A \models S \text{ says } A \overset{|K_{AB}|_{*A}}{\longleftrightarrow} B \quad (D5_a)$$
$$A \models S \text{ says } \sharp(|K_{AB}|_{*A}) \quad (D5_b)$$

Partindo da Derivação 5 e das Premissas 7 e 9, por Jurisdição, obtemos:

$$A \models A \overset{|K_{AB}|_{*A}}{\longleftrightarrow} B \quad (D6_a)$$
$$A \models \sharp(|K_{AB}|_{*A}) \quad (D6_b)$$

Da Interpretação 4, por *Modus Ponens* da Derivação 6 e da Compreensão 4:

$$A \models A \triangleleft \{|N_B|_{*A}, A \overset{|K_{AB}|_{*A}}{\longleftrightarrow} B\}_{|K_{AB}|_{*A}} \quad (D7)$$

Das Derivações 7 e 6, por Associação à Origem:

$$A \models B \mid\sim (|N_B|_{*A}, A \overset{|K_{AB}|_{*A}}{\longleftrightarrow} B) \quad (D8_a)$$
$$A \models B \text{ has } (|N_B|_{*A}, A \overset{|K_{AB}|_{*A}}{\longleftrightarrow} B) \quad (D8_b)$$

Por Possessão:

$$A \models B \text{ has } (A \overset{|K_{AB}|_{*A}}{\longleftrightarrow} B) \quad (D8_c)$$

Da Derivação 6, por Recência, assumindo que $\sharp(|K_{AB}|_{*A}) \rightarrow \sharp(A \overset{|K_{AB}|_{*A}}{\longleftrightarrow} B)$:

$$A \models \sharp(|N_B|_{*A}, A \overset{|K_{AB}|_{*A}}{\longleftrightarrow} B) \quad (D9)$$

Finalmente, das Derivações 9 e 8, concluímos:

$$A \models B \text{ says } (|N_B|_{*A}, A \overset{|K_{AB}|_{*A}}{\longleftrightarrow} B) \quad (D10)$$

A Derivação 10 nos diz que A acredita que B disse que a chave K_{AB} é boa. Isso provê a A autenticação perante a entidade B. Vamos tentar obter conclusão semelhante para B:

Da Interpretação 3 e da Compreensão 3, por *Modus Ponens*:

$$B \models B \triangleleft \{A \xleftrightarrow{|K_{AB}|_{*B}} B, \sharp(|K_{AB}|_{*B}), A\}_{K_{BS}} \quad (D11)$$

A patir da Derivação 11 e da Premissa 3, por Associação à Origem:

$$B \models S \mid\sim (A \xleftrightarrow{|K_{AB}|_{*B}} B, \sharp(|K_{AB}|_{*B}), A) \quad (D12)$$

Da Derivação 12 e da Premissa 13, derivamos por Recência e Verificação de *Nonce*:

$$B \models S \textit{ says } (A \xleftrightarrow{|K_{AB}|_{*B}} B, \sharp(|K_{AB}|_{*B}), A) \quad (D13)$$

Note que esta derivação só foi possível graças à Premissa 13, a já discutida premissa-problema. Sem ela, a verificação pararia por aqui e a autenticação de A para B não seria obtida. Continuando:

Da Derivação 13 e das Premissas 8 e 10, derivamos por Jurisdição:

$$B \models (A \xleftrightarrow{|K_{AB}|_{*B}} B, \sharp(|K_{AB}|_{*B}), A) \quad (D14_a)$$
$$B \models (A \xleftrightarrow{|K_{AB}|_{*B}} B) \quad (D14_b)$$

A partir da Interpretação 5, por *Modus Ponens* da Derivação 14 e da Compreensão 5:

$$B \models B \triangleleft \{N_B - 1, A \xleftrightarrow{|K_{AB}|_{*B}} B\}_{|K_{AB}|_{*B}} \quad (D15)$$

Das Derivações 15 e 14, por Associação à Origem:

$$B \models A \mid\sim (N_B - 1, A \xleftrightarrow{|K_{AB}|_{*B}} B) \quad (D16_a)$$

Por Recência e Verificação de *Nonces*:

$$B \models A \textit{ says } (N_B - 1, A \xleftrightarrow{|K_{AB}|_{*B}} B) \quad (D16_b)$$

Conseguimos assim prover a B autenticação quanto à entidade A, através da Derivação $D16_b$.

- **Conclusão**

  A verificação do NSSK com a SVO, assim como com a BAN, só consegue alcançar garantias de segurança com a utilização da Premissa 13. As propriedades verificadas para o protocolo são exatamente as mesmas.

Conclusão

A lógica SVO indubitavelmente oferece inúmeros recursos inexistentes na BAN. O preço a ser pago por esta flexibilidade são demonstrações mais longas e complexas, o que a torna mais propensa a erros que a BAN, quando conduzida por analistas humanos. Particularmente, a SVO fornece sintaxe para representar termos desconhecidos, algo fundamental para o tratamento de propriedades como tempestividade (conforme discutido no Capítulo 3).

2.2.3 Método de espaços de fitas

O método de espaços de fitas (*Strand Spaces*) [58, 57] foi inicialmente proposto como uma técnica de verificação formal para protocolos de autenticação e estabelecimento de chaves. Embora fosse suficientemente robusto para detectar ataques baseados em multiplas execuções do mesmo protocolo (como ataques de *replay*, por exemplo), o método não permitia originalmente que instâncias de outros protocolos estivessem em execução no mesmo ambiente que o protocolo analisado – o que impedia a detecção de ataques de protocolo escolhido.

Uma verificação típica com o método de espaços de fitas é conduzida da seguinte forma: Inicialmente, a execução esperada do protocolo deve ser modelada em um subgrafo orientado acíclico (chamado *bundle* e exemplificado na Figura 2.7) do espaço de fitas, que preserva precedência causal entre as mensagens. O *bundle* é uma pequena porção do espaço de fitas total, e é utilizado para definirmos o comportamento esperado de cada entidade no protocolo. Cada vértice, ou **nó**, representa a transmissão ou o recebimento de uma mensagem, e pertence a apenas uma fita. Uma **fita** representa o comportamento de um participante no protocolo, e é definida como o conjunto de nós protagonizados por aquele participante. Existem dois tipos de arestas: arestas horizontais simples, que representam as mensagens do protocolo e conectam nós de fitas diferentes, sendo orientadas do remetente ao destinatário; e arestas verticais duplas, que representam precedência causal de eventos e conectam dois nós de uma mesma fita, orientadas do nó que representa o evento mais antigo ao nó que representa o evento mais recente. Após a modelagem do protocolo, as propriedades criptográficas desejadas devem ser enunciadas na forma de teoremas que explorem a estrutura daquele grafo em particular[7]. Por fim, os teoremas devem ser provados sobre o espaço de fitas recém-criado; o sucesso nesta tarefa significa que a propriedade representada é satisfeita pelo protocolo, enquanto a falha significa o oposto. A análise cuidadosa das razões que impediram o teorema de ser demonstrado revela informações cruciais sobre o que está errado na especificação, e como

[7]Tais teoremas variam, portanto, com o protocolo sendo analisado – protocolos diferentes podem apresentar teoremas diferentes para uma mesma propriedade.

corrigir a falha de segurança.

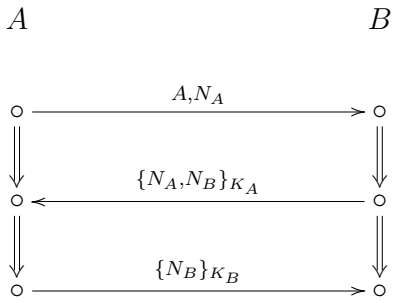


Figura 2.7: Exemplo de *bundle*: O protocolo clássico de Needham-Schroeder representado em espaços de fitas

No restante da seção, apresentaremos parte do formalismo por trás do método de espaços de fitas. Para maiores detalhes, o leitor deve referir à proposta original [57].

Definições

Um **espaço de fitas** Σ é definido como o conjunto das fitas que refletem o comportamento especificado dos participantes regulares, juntamente com qualquer combinação possível de fitas de intruso. A Figura 2.7 mostra as duas fitas regulares presentes no espaço de fitas do protocolo de Needham-Schroeder, uma para cada entidade do protocolo (A e B). As fitas ilustradas são ditas **regulares**, pois refletem o comportamento de entidades honestas. Uma **fita de intruso** reflete as capacidades do adversário, isto é, combinações das operações atômicas que ele é capaz de realizar e que compõem toda e qualquer tentativa de ataque ao sistema. Um ***bundle*** é um subespaço de algum espaço de fitas.

O conjunto de mensagens possíveis é definido como a álgebra de termos livres A, gerada pela operação de concatenação sobre um conjunto $\mathcal{A}$ de termos. A **relação de subtermo** sobre A, denotada por $t_0 \sqsubset t_1$, indica que t_0 é um subtermo de t_1. Um **termo sinalizado** identifica se um termo t está sendo enviado ($+t$) ou recebido ($-t$).

Um espaço de fitas sobre A é um conjunto Σ, junto a um mapeamento de rastros $tr : \Sigma \to (\pm\mathsf{A})^*$; o **rastro** de uma fita de Σ é a seqüência de termos sinalizados que compõem aquela fita. O **comprimento** de um rastro é o número de termos que compõem a seqüência, e é denotado por $length(\cdot)$. Um **nó** é determinado por um par $\langle s, i\rangle$, onde $s \in \Sigma$ é uma fita e i é um inteiro tal que $1 \leqslant i \leqslant length(tr(s))$. O conjunto de nós de Σ é dado por $\mathcal{N}$. Se $n = \langle s, i\rangle \in \mathcal{N}$ é um nó, então $index(n) = i$, $strand(n) = s$, $term(n) = (tr(s))_i$ e $uns\_term(n)$ é o **termo não-sinalizado** de $term(n)$.

Há em Σ uma aresta $n_1 \rightarrow n_2$ se e só se $term(n_1) = +a$ e $term(n_2) = -a$, o que indica que o termo a está sendo enviado de n_1 para n_2. Se, ao invés disso, $n_1 = \langle s, i\rangle$ e $n_2 = \langle s, i+1\rangle$, então existe uma aresta $n_1 \Rightarrow n_2$, significando que n_1 é um predecessor imediato de n_2 em s. Generalizando esta notação, dizemos que $n \Rightarrow^+ n'$ se n precede n' na mesma fita.

Seja I um conjunto de termos não-sinalizados. Então um nó $n \in \mathcal{N}$ é um **ponto de entrada** para I se e só se $term(n) = +t$ para algum $t \in I$, e para todo n' tal que $n' \Rightarrow^+ n$, $term(n') \notin I$. Um termo não sinalizado t é **originado** em $n \in \mathcal{N}$ se e só se n é um ponto de entrada para o conjunto $I = \{t' : t \sqsubset t'\}$. Um termo não-sinalizado t é **unicamente originado** se e só se t origina em um único $n \in \mathcal{N}$.

Sejam $\rightarrow_{\mathcal{C}} \subset \rightarrow, \Rightarrow_{\mathcal{C}} \subset \Rightarrow$ e $\mathcal{C} = \langle \mathcal{N}_{\mathcal{C}}, (\rightarrow_{\mathcal{C}} \cup \Rightarrow_{\mathcal{C}})\rangle$ um subgrafo de $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow)\rangle$. $\mathcal{C}$ é um *bundle* se:

1. $\mathcal{C}$ é finito;
2. se $n_2 \in \mathcal{N}_{\mathcal{C}}$ e $term(n_2)$ é negativo, então existe um n_1 único tal que $n_1 \rightarrow_{\mathcal{C}} n_2$;
3. se $n_2 \in \mathcal{N}_{\mathcal{C}}$ e $n_1 \Rightarrow n_2$, então $n_1 \Rightarrow_{\mathcal{C}} n_2$;
4. $\mathcal{C}$ é acíclico.

A **$\mathcal{C}$-altura** de uma fita s em um *bundle* $\mathcal{C}$ é o maior i tal que $\langle s, i\rangle \in \mathcal{C}$. Além disso, $\mathcal{C}\text{-}rastro(s) = \langle (tr(s))_1, \ldots, (tr(s))_m\rangle$, onde $m = \mathcal{C}$-altura(s).

Seja $\mathcal{S} \subseteq \rightarrow \cup \Rightarrow$ um conjunto de arestas. Então $\prec_{\mathcal{S}}$ é o fecho transitivo de $\mathcal{S}$, e $\preceq_{\mathcal{S}}$ é o fecho reflexivo e transitivo de $\mathcal{S}$. Tanto $\prec_{\mathcal{S}}$ quanto $\preceq_{\mathcal{S}}$ são subconjuntos de $\mathcal{N}_{\mathcal{S}} \times \mathcal{N}_{\mathcal{S}}$, onde $\mathcal{N}_{\mathcal{S}}$ é o conjunto de nós incidentes a alguma aresta em $\mathcal{S}$.

Apresentaremos a seguir alguns lemas úteis na verificação de protocolos clássicos. O Lema 2.2.1 declara que todo conjunto de nós em um *bundle* $\mathcal{C}$ contém um nó n $\preceq_{\mathcal{C}}$ -minimal. Já o Lema 2.2.2 declara que tal nó n é sempre positivo. Se o conjunto de nós é construído através da relação de subtermos, então, de acordo com o Lema 2.2.3, n é um nó originador.

Lema 2.2.1. *Seja $\mathcal{C}$ um bundle. Então $\preceq_{\mathcal{C}}$ é uma ordem parcial, i.e., uma relação reflexiva, anti-simétrica e transitiva. Todo subconjunto não-vazio de nós de $\mathcal{C}$ possui membros $\preceq_{\mathcal{C}}$ -minimais.*

Lema 2.2.2. *Seja $\mathcal{C}$ um bundle, e $\mathcal{S} \subseteq \mathcal{C}$ um conjunto de nós tal que*

$$\forall m, m'\ uns\_term(m) = uns\_term(m')\ \textit{implica em}\ (m \in \mathcal{S}\ \textit{iff}\ m' \in \mathcal{S}).$$

Se n é um membro $\preceq_{\mathcal{C}}$ -minimal de $\mathcal{S}$, então o sinal de n é positivo.

Demonstração. Suponha que $term(n)$ é negativo. Então, pela definição de *bundle*, $n' \rightarrow n$ para algum $n' \in \mathcal{C}$ e $uns\_term(n) = uns\_term(n')$. Portanto, $n' \in \mathcal{S}$, o que contradiz a minimalidade de n. □

Lema 2.2.3. *Seja $\mathcal{C}$ um bundle, $t \in \mathsf{A}$ e $n \in \mathcal{C}$ um elemento $\preceq_{\mathcal{C}}$ -minimal de $\{m \in \mathcal{C} : t \sqsubset term(m)\}$. Então, o nó n é uma ocorrência originadora de t.*

Demonstração. É evidente que $t \sqsubset term(n)$. Pelo Lema 2.2.2, o sinal de n é positivo. Se $n' \Rightarrow^+ n$, então $n' \in \mathcal{C}$. Por causa da minimalidade de n, $t \not\sqsubset term(n')$. Portanto, n é originador de t. □

Seja $\mathsf{T} \subseteq \mathsf{A}$ um conjunto de mensagens atômicas, e $\mathsf{K} \subseteq \mathsf{A}$ um conjunto de chaves criptográficas disjunto de T. Definimos um operador unário $inv : \mathsf{K} \rightarrow \mathsf{K}$ e dois operadores binários: $encr : \mathsf{K} \times \mathsf{A} \rightarrow \mathsf{A}$ e $join : \mathsf{A} \times \mathsf{A} \rightarrow \mathsf{A}$. Outra representação possível para estes operadores é K^{-1} para $inv(K)$, $\{m\}_K$ para $encr(K, M)$ e $a\ b$ para $join(a, b)$.

Os dois axiomas a seguir referem-se à livre geração da álgebra de termos A:

Axioma 2.2.4. *Para $m, m' \in \mathsf{A}$ e $K, K' \in \mathsf{K}$, $\{m\}_K = \{m'\}_{K'} \Rightarrow m = m' \wedge K = K'$.*

Axioma 2.2.5. *Para $m_0, m_0', m_1, m_1' \in \mathsf{A}$ e $K, K' \in \mathsf{K}$,*

1. $m_0 m_1 = m_0' m_1' \Rightarrow m_0 = m_0' \wedge m_1 = m_1'$
2. $m_0 m_1 \neq \{m_0'\}_{K'}$
3. $m_0 m_1 \notin \mathsf{K} \cup \mathsf{T}$
4. $\{m_0\}_K \notin \mathsf{K} \cup \mathsf{T}$.

A partir do Axioma 2.2.5, podemos definir a **largura** de um termo t (denotada por $width(t)$). Também definiremos mais precisamente a relação de subtermo.

Definição 2.2.6. *Se $m \in \mathsf{K} \cup \mathsf{T}$ ou se $m = \{m_0\}_K$, então $width(m) = 1$. Se $m = m_0 m_1$, então $width(m) = width(m_0) + width(m_1)$.*

Definição 2.2.7. *A relação de subtermo $\sqsubset$ é definida indutivamente como a menor relação que satisfaz, para um termo qualquer a:*

1. $a \sqsubset a$
2. $a \sqsubset \{g\}_K$ *se* $a \sqsubset g$
3. $a \sqsubset gh$ *se* $a \sqsubset g$ *ou* $a \sqsubset h$.

O modelo de intruso no método de espaços de fitas contém um conjunto de chaves conhecidas pelo adversário, $\mathsf{K_P}$, e um conjunto de **fitas de intruso** que representam todas as suas possíveis ações. Cada fita de intruso apresenta uma combinação dos seguintes **rastros de intruso**:

Mensagem de texto (M): $\langle +t \rangle$, onde $t \in \mathsf{T}$

Intervenção (F): $\langle -g \rangle$

Replicação (T): $\langle -g, +g, +g \rangle$

Concatenação (C): $\langle -g, -h, +gh \rangle$

Separação (S): $\langle -gh, +g, +h \rangle$

Chave (K): $\langle +K \rangle$, onde $K \in \mathsf{K_P}$

Ciframento (E): $\langle -K, -h, +\{h\}_K \rangle$

Deciframento (D): $\langle -K^{-1}, -\{h\}_K, +h \rangle$.

Definiremos agora um espaço de fitas infiltrado.

Definição 2.2.8. *Um **espaço de fitas infiltrado** é um par $(\Sigma, \mathcal{P})$ onde Σ é um espaço de fitas e $\mathcal{P} \subseteq \Sigma$, tal que $tr(p)$ é um rastro de intruso para toda fita $p \in \mathcal{P}$.*

*Uma fita $s \in \Sigma$ é uma **fita de intruso** se pertence a $\mathcal{P}$, e um nó é um **nó de intruso** se faz parte de alguma fita de intruso.*

*Um nó n é um nó **M, F,** etc. se n faz parte de alguma fita de intruso com rastro do tipo **M, F,** etc.*

A Figura 2.8 ilustra um *bundle* que contém um intruso, P, extraído de um espaço de fitas infiltrado. As entidades A e B são honestas de acordo com o protocolo clássico de Needham-Schroeder, cuja execução correta é ilustrada pela Figura 2.7. É possível perceber que o rastro da fita de P pode ser composto a partir de dois pares de nós dos tipos **D** e **E**, respectivamente. A partir das definições de intruso, fitas de intruso e rastros de intruso, é possível enunciar a seguinte proposição geral:

Proposição 2.2.9. *Seja $\mathcal{C}$ um bundle, e seja $K \in \mathsf{K} \setminus \mathsf{K_P}$.*

Se K nunca origina em um nó regular, então $K \not\sqsubset term(n)$ para todo $n \in \mathcal{C}$. Em particular, para todo $p \in \mathcal{C}, K \not\sqsubset term(p)$.

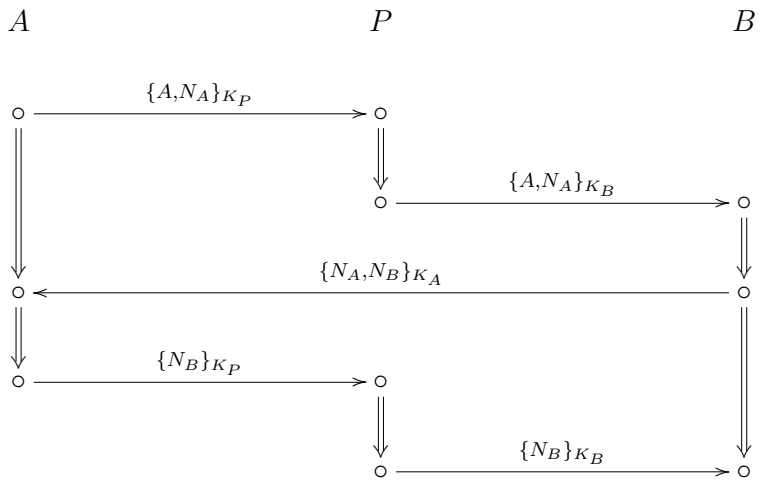


Figura 2.8: Exemplo de *bundle* infiltrado: O protocolo clássico de Needham-Schroeder representado por um espaços de fitas infiltrado

Se $\mathsf{k} \subseteq \mathsf{K}$, então um k-ideal de A é um subconjunto I de A tal que, para todo $h \in I, g \in \mathsf{A}$ e $K \in \mathsf{k}$, $hg, gh \in I$ e $\{h\}_K \in I$. O menor k-ideal contendo h é denotado $I_{\mathsf{k}}[h]$. A partir desta definição, declaramos as seguintes proposições, teoremas e corolários, considerando sempre um subconjunto S de termos possíveis da álgebra A. Suas demonstrações estão disponíveis na referência original [57].

Proposição 2.2.10. *Se $S \subseteq \mathsf{A}$, então $I_{\mathsf{k}}[S] = \bigcup_{x \in S} I_{\mathsf{k}}[x]$.*

Proposição 2.2.11. *Seja $K \in \mathsf{K}, S \subseteq \mathsf{A}$, e para todo $s \in S$, s é simples e não apresenta forma $\{g\}_K$. Se $\{h\}_K \in I_{\mathsf{k}}[S]$ para $K \in \mathsf{K}$, então $K \in \mathsf{k}$.*

Proposição 2.2.12. *Seja $S \subseteq \mathsf{A}$, tal que todo $s \in S$ é simples. Se $gh \in I_{\mathsf{k}}[S]$, então ou $g \in I_{\mathsf{k}}[S]$, ou $h \in I_{\mathsf{k}}[S]$.*

Teorema 2.2.13. *Seja $\mathcal{C}$ um bundle sobre $\mathsf{A}, S \subseteq \mathsf{T} \cup \mathsf{K}, \mathsf{k} \subseteq \mathsf{K}$, e $\mathsf{K} \subseteq S \cup \mathsf{k}^{-1}$. Então $I_{\mathsf{k}}[S]$ é **honesto**.*

Corolário 2.2.14. *Seja $\mathcal{C}$ um bundle, $\mathsf{K} = S \cup \mathsf{k}^{-1}$ e $S \cap \mathsf{K_P} = \emptyset$. Se existe um nó $m \in \mathcal{C}$ tal que $term(m) = I_{\mathsf{k}}[S]$, então existe um nó regular $n \in \mathcal{C}$ tal que n é ponto de entrada para $I_{\mathsf{k}}[S]$.*

Corolário 2.2.15. *Seja $\mathcal{C}$ um bundle, $\mathsf{K} = S \cup \mathsf{k}^{-1}; S \cap \mathsf{K_P} = \emptyset$, e nenhum nó regular em $\mathcal{C}$ é ponto de entrada para $I_{\mathsf{k}}[S]$. Então todo termo da forma $\{g\}_K$, para $K \in S$, não é originado em uma fita de intruso.*

2.2.4 Adaptações do método de espaços de fitas

Desde a proposta do método original, diversas extensões para a técnica foram sugeridas. A primeira alteração realmente impactante foi o suporte a execuções de múltiplos protocolos em um mesmo ambiente [56], o que viabilizou a detecção de ataques de protocolo escolhido (*chosen protocol attacks*) e a análise de protocolos compostos. A introdução dos testes de autenticação [25] tornou possível a determinação intuitiva de ambiguidades na origem de termos específicos, sem que fosse necessário utilizar o arcabouço algébrico complexo do método original; mais detalhes sobre os testes de autenticação serão fornecidos na Seção 2.2.5.

A primeira aplicação da técnica a protocolos de trocas justas [38] foi realizada com poucas alterações sobre o método original, mas não explorava as características especiais desta classe de protocolos; a primeira aplicação do método a levar em consideração a abordagem otimista e multi-protocolar dos protocolos de trocas justas foi sugerida em [48], e posteriormente explorada em [47]. Outras extensões são sugeridas em [26, 17, 31], assim como comparações a outras técnicas formais [11, 53, 13].

Como o processo de verificação em espaços de fitas é laborioso e susceptível a falha humana, alguns pesquisadores dedicaram-se à automatização do método – ou de parte dele, ao menos. Existem algumas ferramentas construídas a partir do método original, como a Athena [52], que não está disponível para estudo; tentativas de contato com o autor revelaram-se infrutíferas. Outra ferramenta inspirada pelo método é a ASPECT [36], classificada pelos autores como uma ferramenta de "*detecção de falhas para protocolos de segurança*" [36]. Isso significa que a atuação da ferramenta sobre a especificação do protocolo se limita a buscar padrões de ataque – do mesmo modo que as armadilhas propostas na Seção 3.2 – ao invés de realizar uma análise formal completa. Até o presente momento, todas as tentativas de automatização do método de espaço de fitas retêm a restrição original quanto à classe de protocolos passíveis de análise: apenas protocolos clássicos – de autenticação e estabelecimento de chaves – são suportados, o que exclui a possibilidade de verificação de protocolos diplomáticos.

2.2.5 Testes de autenticação

Na proposta original do método de espaços de fitas [57], os protocolos eram verificados através da determinação de valores para parâmetros em fitas regulares. Os Testes de Autenticação [25] visam estabelecer uma metodologia para tornar este processo mais intuitivo e menos propenso a erro humano. Para tanto, os autores definem **componentes de teste em** N, termos da forma $\{m\}_K$ que contêm um *nonce* N sabidamente recente como subtermo e cuja chave de ciframento K é boa para a comunicação com outras entidades regulares. Existem três tipos de testes de autenticação:

1. **Teste de autenticação de entrada**: Suponha que A seja uma entidade honesta que emite uma mensagem m, que contém um termo N sabidamente recente (N é um *nonce* recém-gerado por A, por exemplo, e $N \sqsubset m$). Futuramente, A recebe o componente de teste $\{h\}_K$ em N. Neste momento, A pode concluir que existe uma entidade regular (i.e., honesta) ativa, responsável pela geração do componente de teste.

2. **Teste de autenticação de saída**: Suponha que A seja uma entidade honesta que emite um componente de teste $\{h\}_K$ em N. Futuramente, A recebe a mensagem m, com $N \sqsubset m$. Neste momento, A pode concluir que existe uma entidade regular (i.e., honesta) ativa, que conhece K e decifrou o componente de teste, para então gerar m.

3. **Teste de autenticação não-solicitado**: Suponha que A receba um componente de teste $\{h\}_K$ em N. Então, a partir da garantia de que K é segura, A pode concluir que existe uma entidade regular ativa, responsável pela geração recente de $\{h\}_K$.

Na prática, os testes de autenticação permitem que uma entidade certifique-se de que seu interlocutor é uma entidade honesta, e não um adversário. Essa garantia provém do fato de que nenhuma combinação das ações unitárias permitidas ao adversário – introduzidas no método original de espaços de fitas [57] e descritas na Seção 2.2.3 – é capaz de alterar/forjar um componente de teste.

A análise de um protocolo ocorre através dos seguintes passos: Inicialmente, o analista deve traduzir a especificação do protocolo para a notação de espaços de fitas, da mesma forma que no método original. A seguir, deve definir os rastros regulares, que refletem o comportamento definido pelo protocolo para as entidades regulares; este passo determinará os parâmetros a serem derivados no restante da análise. Depois, o analista deve identificar os componentes de teste que estudará; a existência destes componentes, bem como o ponto do protocolo em que ocorrem, permite que o analista deduza logicamente informações importantes sobre segurança, como a existência de interlocutores válidos, o segredo de chaves trocadas e eventuais ataques. O leitor cuidadoso notará que a verificação do protocolo através de testes de autenticação foca-se no estudo dos componentes de teste, enquanto que o método original de espaços de fitas concentra-se na enunciação e prova de teoremas que representam as propriedades de segurança buscadas. Apesar desta aparente "simplificação" do processo de verificação, a técnica de testes de autenticação foi provada completa [16].

2.3 Protocolos de trocas justas

Nesta seção, apresentaremos o problema da troca justa, conforme definido por Asokan [4], bem como as discussões que o sucederam.

2.3.1 O problema da troca justa

Em um cenário clássico, as entidades honestas devem colaborar na obtenção de objetivos comuns; para que uma chave de sessão seja boa para A comunicar-se com B, por exemplo, é necessário que B também confie que sua geração ocorreu adequadamente. No entanto, em cenários diplomáticos, as entidades possuem interesses conflitantes, e preocupam-se em atingir seus próprios objetivos individuais. O problema da troca justa é justamente garantir que qualquer entidade só conseguirá atingir seus objetivos – i.e., obter a informação desejada no protocolo – caso os demais participantes também atinjam seus próprios objetivos no protocolo.

Embora Asokan [4] motive a discussão original sobre trocas justas no escopo do comércio eletrônico, existem diversas interações caracteristicamente diplomáticas. Algumas das mais usuais são:

1. **Assinatura de contratos:** As entidades concordam sobre os termos de um dado contrato eletrônico C, e devem assiná-lo digitalmente. No entanto, a assinatura assíncrona do contrato não é viável, já que nenhuma das entidades deseja ser a primeira a se comprometer com os termos da troca.

2. **Entrega certificada de mensagens:** Uma entidade deseja enviar uma mensagem a outra, mas deseja obter um recibo que comprove a entrega da mensagem (irretratabilidade de recebimento).

3. **Comércio eletrônico:** Um comprador deseja adquirir um item eletrônico junto a uma loja, mediante pagamento. Novamente, o comprador não deseja pagar antes de receber o produto, tampouco a loja deseja enviar o produto antes do pagamento ser realizado.

4. **Transações de cartão de débito:** O comprador deve fornecer seu cartão de débito físico e digitar sua senha no dispositivo, o que viabiliza o débito em conta. Em troca, deseja receber o código de aprovação da transação, bem como o recibo físico do pagamento.

Existem diversas abordagens que buscam estabelecer justiça em transações diplomáticas. A justiça nunca é, no entanto, o único objetivo a ser alcançado por

um protocolo, e vem freqüentemente associada a outras propriedades fundamentais. Na Seção 2.3.2 apresentaremos brevemente tais propriedades, conforme originalmente propostas por Asokan [4], bem como algumas discussões que o sucederam. Na Seção 2.3.3, apresentaremos diversas classes de protocolos de trocas justas.

2.3.2 Propriedades desejáveis de protocolos

Asokan [4] determinou originalmente cinco propriedades desejáveis para protocolos de trocas justas, que refletem o caráter diplomático das transações:

1. **Efetividade:** Remete ao caso base do protocolo, em que todas as entidades se comportam conforme previsto.
2. **Justiça:** Confere atomicidade à troca. Ou ambas as entidades obtêm os itens desejados, ou nenhuma o faz.
3. **Tempestividade:** Determina que o protocolo não pode permitir situações em que uma entidade honesta seja incapaz de terminar a troca de forma justa, ou precise esperar um tempo indeterminado para evitar perda de justiça.
4. **Irretratabilidade:** Vincula as entidades às suas ações no protocolo. As entidades não devem ser capazes de negar sua participação na troca.
5. **Verificabilidade de TTP:** Nos casos em que uma terceira parte confiável (TTP) participa da troca, é preciso garantir que seu eventual mau-comportamento possa ser provado a um árbitro externo.

Estas são as noções por trás de cada propriedade original de trocas justas. Na Seção 4.2 apresentaremos a declaração completa de cada propriedade, conforme definidas por Asokan [4].

As definições informais de Asokan permitiram a proliferação de diversas interpretações diferentes de algumas destas propriedades. Buscando um consenso sobre o significado formal de tais interpretações, alguns autores dedicaram-se a estudos exploratórios e taxonômicos [22, 44, 28].

Pagnia et al. [44] propõem uma hierarquia composta por seis graus de justiça, que variam sobre aspectos como necessidade de colaboração das entidades, qualidade da compensação oferecida em caso de falha na troca e época – durante ou após a transação – em que ocorrem eventuais disputas. A irretratabilidade é intensamente estudado por Kremer et al. [28], em que os autores não só categorizam diversas interpretações da propriedade, como também apresentam diversos exemplos de protocolos especializados em comprovar a participação de entidades em transações. Um estudo semelhante é apresentado no Capítulo 3, sobre tempestividade.

2.3.3 Tipos de protocolos

Existem diversas abordagens para o projeto de protocolos de trocas justas. Nesta seção apresentaremos as principais famílias de protocolos, com alguns exemplos. Nossa discussão será focada em protocolos para duas entidades; protocolos para mais de duas entidades normalmente são compostos de múltiplas interações dois-a-dois em topologias de anel (sendo, portanto, não mais que especializações dos primeiros).

Antes do problema da troca justa ser materializado como uma questão digna de estudo à parte, a solução padrão para a troca atômica de itens entre duas entidades baseava-se em uma TTP, que participava de cada troca. Estes protocolos, denominados **protocolos baseados em TTP *online*** ou **protocolos pessimistas**, se utilizam da TTP como um centro de retransmissão confiável de mensagens.

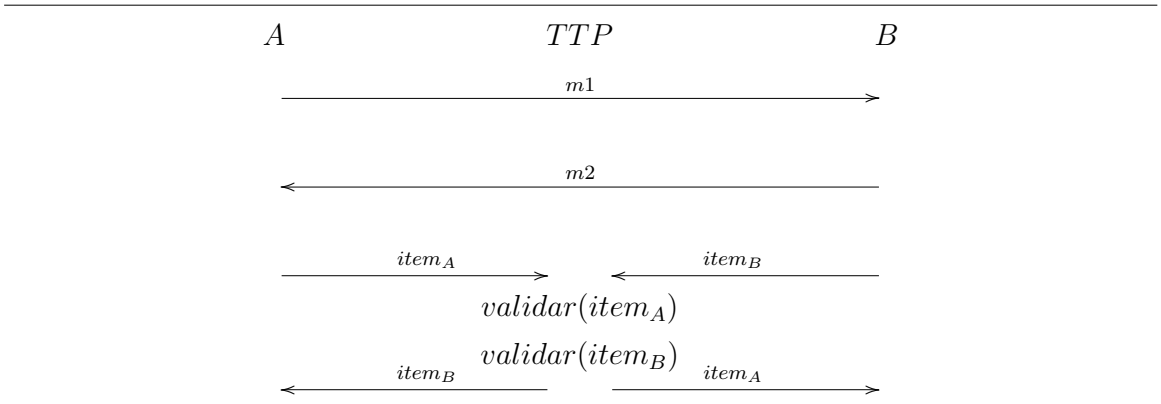


Figura 2.9: Um protocolo pessimista genérico de trocas justas. As descrições de $item_A$ e $item_B$ estão disponíveis para a TTP.

A Figura 2.9 ilustra o esquema geral dos protocolos baseados em TTP *online*. Inicialmente, as entidades trocam informações de natureza variada, possivelmente para autenticação ou acordo de parâmetros sobre a troca. A seguir, os itens são enviados para a TTP, bem como suas descrições (caso não sejam públicas). A TTP então realiza atomicamente o processo de validação sobre cada item, e os repassa às entidades destinatárias em caso de sucesso. Maiores detalhes sobre o processo de validação de itens serão fornecidos na Seção 2.3.4.

A maior desvantagem dos protocolos pessimistas é que, como a participação da TTP é exigida em todas as trocas, o sistema está sujeito a ataques de negação de serviço (DoS). Assim, embora tornem a troca completamente atômica sob o ponto de vista das entidades, há pouca aplicação prática para protocolos pessimistas – ainda que seja possível encontrar na literatura algumas propostas recentes [7].

Além de tornar o protocolo susceptível a ataques de DoS, o uso de TTPs encarece cada

transação individual (TTPs desejam ser pagas pelo serviço prestado). Assim, existem protocolos que procuram estabelecer justiça sem a necessidade de uma TTP de qualquer forma. O preço a ser pago por tal concessão é que os protocolos adquirem um caráter não-determinístico[8], conforme discutiremos nas duas classes a seguir.

Os **protocolos probabilísticos** [32, 28] pretendem garantir um elevado grau de justiça na maioria das trocas, sob a pena de admitir uma porcentagem de transações injustas.

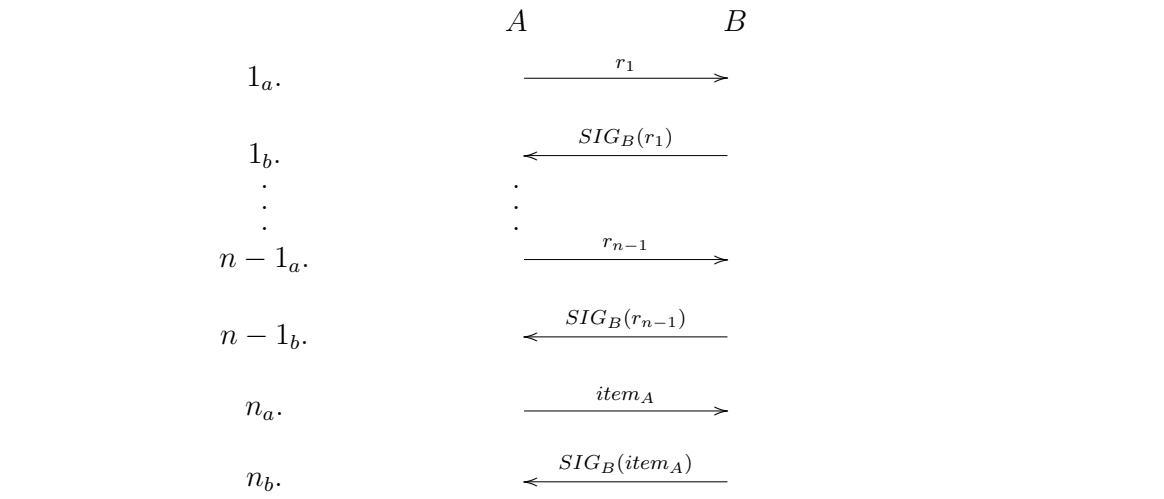


Figura 2.10: Um protocolo probabilístico genérico de trocas justas

A Figura 2.10 ilustra um exemplo de como os protocolos probabilísticos se comportam. Em tais protocolos, a troca é realizada em um certo número de trocas menores, denominadas turnos. Digamos que uma entidade A deseja, por exemplo, trocar um item $item_A$ por uma versão de $item_A$ digitalmente assinada por B (um protocolo de assinatura de contrato seria um bom exemplo). O número n de turnos é escolhido aleatoriamente pela entidade A no início do protocolo, e mantido em segredo com relação a B. A cada iteração (i_a, i_b), $0 < i < n$, A gera um novo item falso r_i indistingüível de $item_A$, e o envia a B, que o assina e retorna a A. No n-ésimo turno, A envia o verdadeiro item $item_A$ a B, que novamente assina o termo recebido e o retorna a A. O segredo de n garante que, se um interlocutor malicioso B tentar abandonar o protocolo de forma a obter vantagem sobre A, só poderá fazê-lo com probabilidade não maior que $\epsilon = 1/n$, já que B terá que advinhar o momento certo para parar a troca. Dizemos, portanto, que o protocolo é ϵ-justo.

[8]Protocolos baseados em TTP são projetados sob um ponto de vista determinístico – uma vez que se mostrem livre de ataques, o grau de justiça buscado pode ser obtido em todas as transações.

Outro exemplo de protocolos não-determinísticos são os **protocolos de trocas graduais**. Ao invés de procurarem garantir a atomicidade da troca, estes protocolos utilizam uma abordagem oposta – subdividindo cada item em partes menores, e trocando-os parte por parte. Dessa forma, se alguma entidade abandonar a troca antes do fim, conseguirá uma vantagem arbitrariamente pequena sobre a outra (terá conseguido no máximo uma parte a mais que seu interlocutor).

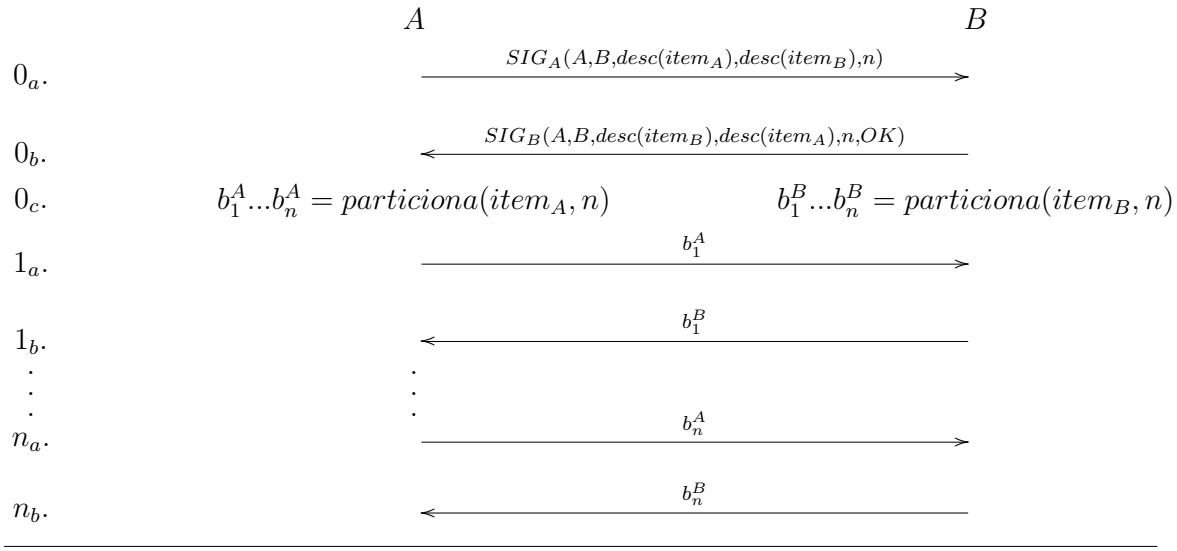


Figura 2.11: Um protocolo genérico de troca gradual

A Figura 2.11 ilustra o funcionamento básico de um protocolo gradual. Inicialmente, A e B combinam os parâmetros da transação, que incluem o número n de sub-trocas (partes) a serem realizadas. A seguir, as entidades particionam seus itens em n partes b_i, onde $1 \leq i \leq n$; estas partes podem ser tão pequenas quanto bits, dependendo do grau de justiça desejado. Por fim, A e B realizam n iterações, trocando b_i^A por b_i^B em cada iteração. Se alguma das entidades abandonar a troca antes de enviar a última parte de seu próprio item, terá ganho no máximo uma parte de vantagem sobre a outra (se as partes forem bits, a entidade prejudicada consegue deduzir o bit faltante com pouco poder computacional, já que ele será 0 ou 1). Assim, se ambas as entidades possuírem o mesmo poder computacional, pode-se garantir que a eventual desvantagem de uma entidade enganada será arbitrariamente pequena, variando sobre o tamanho das partes dos itens.

Tanto os protocolos probabilísticos quanto os graduais são, em geral, impraticáveis na maioria das aplicações reais. Embora seu projeto seja relativamente simples e não sejam necessárias TTPs de nenhuma forma, o grande número de mensagens trocadas nestes

protocolos representa um *overhead* de comunicação inaceitável.

Por um lado, temos protocolos pessimistas, que permitem elevados graus de justiça, mas são caros e frágeis devido à utilização intensa de uma TTP; por outro, temos as soluções não-determinísticas, que não dependem de TTPs mas não oferecem altos graus de justiça. A fim de proporcionar um compromisso aceitável entre estas duas abordagens, foram propostos os **protocolos otimistas** [4], que procuram oferecer as mesmas garantias que as alcançadas em protocolos pessimistas, mas sem depender tão cegamente de uma TTP quanto aqueles. Tais protocolos partem do princípio que, embora as entidades envolvidas em uma troca não sejam confiáveis, elas em geral se comportam bem. Em vista disso, os projetistas utilizam uma TTP *offline*; diferentemente da TTP *online*, esta terceira parte só se envolve na troca quando invocada por alguma das outras entidades, em caso de exceções. Para que isso seja possível, a especificação do protocolo deve fornecer, além do protocolo de troca em si, **subprotocolos auxiliares** para o tratamento de exceções.

Embora o número de subprotocolos auxiliares necessários dependa da especificação do protocolo principal, parece existir um consenso de que, em geral, dois subprotocolos são suficientes – um **subprotocolo de cancelamento**, em que a TTP interrompe a troca e impede que cada entidade obtenha informação adicional sobre o item da outra; e um **subprotocolo de conclusão**, em que a TTP tenta forçar o sucesso da troca, possivelmente enviando os itens desejados às respectivas entidades. O Capítulo 4 traz dois exemplos de protocolos otimistas, nominalmente o FPH [19], ilustrado pelas Figuras 4.1, 4.2 e 4.3, e o ZDB [62], ilustrado pelas Figuras 4.7, 4.8 e 4.9.

Nota sobre terceiras partes confiáveis

Embora existam protocolos capazes de estabelecer formas alternativas de justiça sem utilizar uma TTP, isso só é possível através do sacrifício de algumas das transações ocorridas no sistema. Para garantir que todas as transações sempre terminem de forma justa, é preciso disponibilizar às entidades meios para terminarem cada transação adequadamente – mesmo caso seu interlocutor torne-se indisponível por qualquer razão.

Pagnia et al. demonstram a impossibilidade de obter justiça forte na ausência de uma TTP [43], reduzindo o problema da troca justa para o problema de decisão de consenso. O argumento dos autores baseia-se na natureza assíncrona da troca, que cria para as entidades o dilema de decidir se alguma das demais foi desconectada da transação ou simplesmente é muito lenta. Tal argumento remete claramente à propriedade de tempestividade, tratada no Capítulo 3.

Os argumentos apresentados no decorrer da Seção 2.3.3 mostram como uma TTP pode tornar-se o elo mais fraco de um sistema, devido a ataques de DoS ou encarecimento dos custos de manutenção. Assim, é fundamental que o projetista considere as diversas formas

de TTP, levando em conta a aplicação para a qual o protocolo está sendo desenvolvido:

1. **TTP *inline*:** As primeiras soluções baseadas em TTPs as utilizavam como centro de retransmissão [12]. Além de participar de cada transação ocorrida no sistema, a TTP recebia e reenviava cada mensagem de cada transação, servindo como intermediária em cada mensagem. Além de desnecessariamente custoso, este processo está sujeito a todas as desvantagens das TTPs *online*, revisitadas a seguir.

2. **TTP *online*:** Uma TTP *online* participa ativamente de todas as transações ocorridas no sistema, como centro de validação ou centro de retransmissão de itens. Entretanto, diferentemente das TTPs *inline*, estas entidades intermediam apenas em mensagens críticas da transação, como o envio de um item, por exemplo; as demais mensagens são trocadas sem a intervenção da TTP. Embora tornem a transação completamente atômica sob o ponto de vista das entidades, TTPs *online* podem tornar-se gargalos caso o número de TTPs disponíveis num dado momento seja demasiadamente inferior ao número de transações ocorrendo simultaneamente. Além disso, o custo de cada transação com TTP *online* é consideravelmente mais alto que em outras abordagens, devido à inclusão dos honorários da TTP. TTPs *online* estão presentes em protocolos pessimistas [7], ilustrados pela Figura 2.9.

3. **TTP *offline*:** Atualmente, a abordagem mais eficiente a utilizar TTPs em trocas justas é a dos protocolos otimistas. TTPs otimistas devem ser invocadas por alguma entidade – caso contrário, sequer tomam conhecimento sobre a transação. Além de minimizar os danos causados por ataques de DoS, esta abordagem permite que o custo de cada transação seja minimamente afetado pelos honorários da TTP. Por outro lado, o projeto de protocolos otimistas exige a especificação de múltiplos subprotocolos auxiliares, o que aumenta consideravelmente a complexidade destes artefatos e sua suscetibilidade a erros de projeto.

4. **TTP transparente:** Em abordagens otimistas, a necessidade de intervenção de uma TTP na transação pode diminuir a confiança das entidades sobre o sistema, mesmo quando a justiça não é ameaçada. Assim, o uso de TTPs transparentes tem aumentado consideravelmente em aplicações voltadas a usuários leigos, como é o caso do comércio eletrônico. Em abordagens deste tipo, é impossível para uma entidade dizer, ao final da troca, se houve intervenção de uma TTP. Isso permite que falhas nos canais de comunicação sejam contornadas sem o conhecimento das entidades, o que pode evitar publicidade negativa sobre o serviço.

2.3.4 Problema da validação de itens

A validação de itens representa "*um problema importante, freqüentemente negligenciado, inerente a todos os protocolos de trocas justas*" [44]. Protocolos otimistas de trocas justas normalmente seguem uma seqüência comum de eventos: Suponhamos, por exemplo, que duas entidades P e Q desejam trocar dois itens indescritíveis i_P e i_Q. Uma premissa comum em aplicações do gênero é que P e Q conhecem a priori as descrições $desc(i_Q)$ e $desc(i_P)$, respectivamente, dos itens desejados. As entidades iniciam a troca coletando informações suficientes para comprovar sua participação na transação; este passo é crucial para viabilizar a resolução de disputas internas ou externas, em caso de exceções. A seguir, os participates passam à troca dos itens propriamente dita. Ao receber um item, uma entidade deve executar o procedimento de validação previsto pelo protocolo, que pode ser automático ou não-automático, conforme detalhado a seguir.

O sucesso da validação informará à entidade se o item recebido corresponde à descrição pré-conhecida, ou seja, se é de fato o item esperado. Em caso de falha, o protocolo deve ser suficientemente robusto para permitir que a entidade prejudicada utilize a informação reunida até o presente momento na recuperação do item correto – normalmente com o auxílio de uma TTP – ou impeça que seu interlocutor obtenha seu próprio item, dependendo do ponto em que o protocolo se encontrar.

É importante perceber que a disponibilidade de descrições unívocas dos itens trocados é fundamental para garantir justiça, neste tipo de protocolo. No entanto, existem itens – os denominados itens indescritíveis, descritos na Seção 2.3.5 – para os quais fornecer descrições satisfatórias pode ser impossível [7]. Qualquer descrição destes itens pode ser suficientemente vaga ao ponto de aplicar-se a dois ou mais itens similares, não importando quão detalhada seja.

No entanto, até mesmo para itens descritíveis o processo de validação pode ser consideravelmente complexo. Existem essencialmente três abordagens para a validação de itens, que apresentaremos no restante desta seção: validação automática baseada em TTP online, validação automática baseada em entidades e validação não-automática baseada em entidades. No restante desta seção, assumiremos que os itens i e i' a serem validados são descritíveis, e apresentam descrições públicas unívocas e exatas $desc(i)$ and $desc(i')$. Também suporemos que, para abordagens automáticas, o protocolo especifica uma função booleana $validar(i, j)$, que aceita como entradas um item i e uma descrição j. Esta função deve retornar $VERDADEIRO$, se j descreve corretamente o item i, ou $FALSO$, caso contrário.

1. **Validação automática baseada em TTP online**: A função $validar(i, j)$ deve estar disponível para a TTP online. Em abordagens deste tipo, as entidades enviam os itens i e i' à TTP, bem como as descrições $desc(i')$ e $desc(i)$. A TTP então

executa $validar(i, desc(i))$ e $validar(i', desc(i'))$, encaminhando os items aos seus destinatários em caso de sucesso mútuo. Esta abordagem só é possível em protocolos pessimistas de trocas justas, e portanto está sujeita às mesmas considerações que eles (Seção 2.3.3).

2. **Validação automática baseada em entidades**: A função $validar(i, j)$ deve ser pública, isto é, estar disponível para todas as entidades envolvidas na troca (inclusive a TTP, se necessário). Abordagens deste tipo têm a finalidade de minimizar a participação da TTP no protocolo, reduzindo portanto o impacto de ataques de DoS sobre o sistema. Assim, as próprias entidades são responsáveis pela validação dos itens, o que causa perda de atomicidade. Isso cria uma indesejável situação: Se a entidade tiver acesso completo ao item durante a validação, a justiça do sistema pode ser prejudicada. Assim, a validação deve em geral ser executada sobre uma versão alterada do item (uma versão cifrada, por exemplo), de forma a evitar que a entidade interessada obtenha o item antes de se comprometer a enviar o seu próprio. Um exemplo de validação automática baseada em entidades é o conceito de Verifiable Encryption [5, 41].

3. **Validação não-automática baseada em entidades**: Validação baseada em entidades elimina gargalos no sistema, e abordagens automáticas são perfeitamente compatíveis com itens descritíveis. No entanto, não é possível descrever univocamente itens indescritíveis, o que torna as técnicas de validação automática inadequadas. Em Bottoni et al. [7], os autores introduzem um método para a validação não-automática baseada em entidades, e o utilizam na proposta de um protocolo para a compra e venda justas de arquivos de imagem. Nesta abordagem, a entidade compradora recebe uma cópia reduzida, denominada *thumbnail*, do arquivo a ser trocado. Por simples inspeção visual da thumbnail, o comprador sabe se a foto em negociação é a esperada. Além de viabilizar a troca justa de imagens, que servem como exemplo de itens indescritíveis, este método é bastante interessante sob a perspectiva do usuário, já que permite que a validação seja executada de maneira simples, amigável e direta. Por outro lado, o protocolo proposto depende de uma TTP online, o que o torna impraticável para sistemas grandes reais.

2.3.5 Propriedades de items

Os itens a serem trocados em um protocolo de troca justa podem apresentar propriedades especiais, que devem ser exploradas pelo protocolo [44]. Nesta seção, apresentaremos três delas.

Descritividade

A descritividade de um item refere-se à possibilidade de se obter uma descrição unívoca dele. Como apresentado na Seção 2.3.4, tal descrição é essencial para a validação de itens, que por sua vez cumpre papel fundamental na troca justa de dois itens quaisquer. Embora seja prática comum assumir que tais descrições existem a priori, obtê-las constitui um problema digno de estudo. Arquivos de multimídia, tais como imagens, audio e vídeo, são exemplos de **itens indescritíveis** (i.e., é difícil obter uma descrição unívoca de tais itens). Como um exemplo, consideremos a situação em que uma entidade pretende comprar uma foto da modelo Lena Söderberg, consagrada como musa de trabalhos sobre processamento de imagens. Supondo que tal foto seja descritível, a entidade em questão poderia considerar satisfatória uma descrição que contivesse as expressões "Lena Söderberg", "musa de processamento de imagens", "modelo", "famosa", "chapéu" e "rosto". Ao final da compra, ela poderia esperar pela Figura 2.12(a), mas receber ao invés a imagem ilustrada pela Figura 2.12(b). A princípio, poderíamos tentar resolver este problema apenas acrescentando a expressão "colorida" à descrição do item, mas ainda teríamos a Figura 2.12(c) como candidata para recebimento.

(a) (b) (c)

Figura 2.12: Três possíveis fotos da modelo Lena Söderberg. As figuras (a) e (b) satisfazem igualmente descrições que não mencionem cores, e as Figuras (a) e (c) podem ser confundidas mesmo quando a cor é mencionada.

Não apenas imagens, mas também arquivos de audio e vídeo são indescritíveis. Uma edição mais curta para rádio (radio edit) ou versão ao vivo de determinada música poderiam ser erroneamente enviadas a um comprador que desejava a versão de álbum, ou uma versão remasterizada de um filme poderia ser recebida pelo comprador da versão original. De fato, não importa quão precisa a descrição de um item indescritível seja, sempre existirá ambigüidade suficiente para que uma família de objetos semelhantes satisfaçam aquela descrição [7]. Este fato, por si só, impede que validação baseada em TTP, descrita na Seção 2.3.4, seja utilizada para itens indescritíveis, e complica seriamente a resolução de eventuais disputas.

Gerabilidade

Um item é dito **gerável** se "*pode ser gerado pela TTP caso a entidade que o receberá possa provar que se comportou corretamente*" [44]. O grau de gerabilidade é definido de acordo com a possibilidade de sucesso: **gerabilidade forte** garante que a TTP *sempre* será capaz de gerar o item corretamente, enquanto que **gerabilidade fraca** permite falha caso alguma entidade tenha se comportado mal; neste caso, a TTP é capaz de detectar quem se comportou mal e fornecer evidência sobre sua identidade à entidade honesta, para que processos de disputa externos possam ser iniciados.

A maioria dos itens trocados em protocolos de trocas justas não apresentam gerabilidade inerente; se uma imagem ou arquivo de vídeo, por exemplo, não for corretamente entregue ao destinatário, a TTP não é capaz de recriar em tempo real uma cópia idêntica do arquivo. Em termos práticos, isso significa que se uma entidade é incapaz de obter o item desejado durante a execução do protocolo, a TTP pode não ser capaz de garantir justiça forte – a não ser que algum grau de gerabilidade seja conferido ao item. Embora existam técnicas disponíveis para atribuir gerabilidade a itens genéricos [59], elas em geral partilham de um mesmo problema: a necessidade de uma descrição unívoca do item em questão. Até onde pudemos verificar, não existem técnicas para conferir gerabilidade (forte ou fraca) a itens indescritíveis.

Revogabilidade

Um item é **revogável** se pode ser invalidado pela TTP, dadas determinadas circunstâncias. Da mesma forma que a gerabilidade, diferentes graus de revogabilidade podem ser obtidos. Uma TTP sempre terá sucesso em tornar um item **fortemente revogável** inútil para o receptor, mas pode falhar se o item for **fracamente revogável**; se este for o caso, a TTP sempre sabe se o receptor recebeu ou ainda pode receber o item.

Atribuir revogabilidade a um item genérico é uma tarefa ainda mais complexa do que atribuir gerabilidade. No entanto, itens revogáveis são relativamente comuns no contexto de comércio eletrônico e aplicações de pagamento digital. Na verdade, grande parte dos sistemas de pagamento eletrônico conferem algum grau de revogabilidade aos cheques emitidos. A combinação de gerabilidade e revogabilidade é comumente explorada por projetistas de protocolos de trocas justas; uma TTP pode, por exemplo, tentar gerar um item fracamente gerável para uma entidade prejudicada, e revogar o cheque emitido por ela como pagamento em caso de falha.

2.3.6 Verificação de protocolos de trocas justas

O estado-da-arte da verificação formal de protocolos de trocas justas é atualmente escasso. Até onde pudemos verificar, não existem técnicas desenvolvidas especificamente para protocolos diplomáticos; alguns autores têm sido bem sucedidos em utilizar técnicas baseadas em modelos com poucas ou nenhuma modificação no processo [50] [51], mas a adequação de tais técnicas para protocolos de trocas justas é questionável[9]. As lógicas, por outro lado, têm sido pouco utilizadas na verificação de protocolos de trocas justas [29, 63].

Em particular, os protocolos otimistas são em geral evitados pelos estudiosos de métodos formais, devido à sua estrutura multiprotocolar. Algumas exceções são a abordagem baseada em jogos de Kremer et al. [29], e o trabalho de Shmatikov et al. [51], que utiliza a modelagem Murϕ [15] para a detecção de alguns ataques em protocolos otimistas.

No Capítulo 4 descrevemos nossa adaptação do método de espaços de fitas, que nos permitiu aplicá-lo a protocolos otimistas de trocas justas de maneira semelhante aos testes de autenticação [25] descritos na Seção 2.2.5. Os ataques detectados por este processo também serão delineados naquele capítulo.

[9] Ainda que algumas destas técnicas sejam capazes de detectar ataques sobre os protocolos de trocas justas em questão, não podemos garantir que a ausência de falhas no processo de verificação representa que o protocolo é correto. Como discutido no decorrer desta seção, os objetivos e a estrutura dos protocolos de trocas justas é consideravelmente diferente dos protocolos clássicos, a que tais ferramentas se destinam.

Capítulo 3

Um estudo sobre tempestividade

Este capítulo apresenta um estudo qualitativo sobre uma das propriedades originais de trocas justas, a tempestividade.

3.1 Uma comparação entre interpretações de tempestividade

Diversas interpretações da propriedade tempestividade (também referida como *terminação*) surgiram desde a proposta de protocolos de trocas justas por Asokan [4]. Nesta seção, apresentaremos uma comparação destas definições informais.

De acordo com Asokan [4], um protocolo satisfaz tempestividade se e só se cada participante honesto tem a garantia de que atingirá, após uma quantidade finita de tempo, um ponto no protocolo em que o grau de justiça obtido até então é preservado (mesmo se o participante escolher abandonar honestamente a troca). Note que tal interpretação vê o problema da terminação como um objetivo de *segurança*, e não de *qualidade de serviço* – sem tempestividade, justiça não pode ser alcançada. Assim, para que um protocolo seja justo, é necessário que contenha em sua especificação instruções claras para que toda entidade honesta consiga terminar cada troca que inicia, independentemente das ações de seu interlocutor, ou mesmo de falhas de comunicação. Muitas vezes é possível permitir que uma entidade simplesmente interrompa o protocolo e abandone a troca, sem que haja danos a qualquer das duas partes. No entanto, existem situações em que a simples interrupção do protocolo permitiria que a outra entidade obtivesse vantagem sobre o participante que abandona a troca.

Como este é o significado originalmente pretendido para tempestividade no contexto de protocolos de trocas justas, nos referiremos a ele como a propriedade de *Tempestividade Original*.

Definição 3.1.1. ***Tempestividade Original:*** *"Uma entidade P pode estar certa de que o protocolo se completará eventualmente. A partir deste momento, o estado da troca, ou é final, ou é tal que qualquer alteração no estado não degrada o grau de justiça alcançado por P até aquele ponto." [4]*

No contexto de protocolos probabilísticos [32], descritos na Seção 2.3.3, é possível obter tempestividade com uma probabilidade ϵ após um número não-determinístico de rodadas, normalmente sem a necessidade de uma TTP; nos referiremos a esta propriedade como **Tempestividade Probabilística**, e dizemos que estes protocolos provêem ϵ-tempestividade.

Definição 3.1.2. ***Tempestividade Probabilística:*** *"Um protocolo provê ϵ-tempestividade a uma entidade P se é garantido a P que o protocolo tem probabilidade $1 - \epsilon$ de ser completado em um certo ponto no futuro, preservando justiça." [32]*

A mais interessante característica da tempestividade probabilística é que a probabilidade de uma entidade se encontrar em uma situação irremediavelemnte injusta pode ser arbitrariamente reduzida. No entanto, devido ao grande número de turnos normalmente exigidos, estes protocolos não são praticáveis para aplicações reais.

Algumas vezes, protocolos de troca requerem apenas que uma entidade seja capaz de eventualmente terminar o protocolo. Nos referimos a esta interpretação mais simples como **Tempestividade Fraca**.

Definição 3.1.3. ***Tempestividade Fraca:*** *"Nenhuma entidade deve ter que esperar indefinidamente para terminar o protocolo honestamente." [44]*

Nestes casos, a forma mais comum de garantir tempestividade é através de *timeouts* pré-definidos em cada passo do protocolo. No entanto, a tempestividade fraca não apresenta preocupações relativas à preservação de justiça – apenas declara a necessidade de pontos de terminação claros. Caso alguma forma de justiça deva ser preservada, como é sempre o caso em protocolos de trocas justas, deve-se buscar tempestividade original, probabilística, assíncrona ou forte.

Na prática, e particularmente no contexto de protocolos de trocas justas, buscamos tempestividade por sua íntima relação com a propriedade de justiça. Dessa forma, há uma tendência entre os projetistas de protocolos em se preocupar com a tempestividade apenas após o comprometimento de alguma entidade honesta com a troca; antes disso, como nenhum grau de justiça foi ainda obtido, a especificação do protocolo geralmente não explicita nenhuma forma para que as entidades possam interromper uma troca problemática – fato este que viola a idéia fundamental de que uma entidade não deveria ter que aguardar indefinidamente pela terminação do protocolo. Nestes casos, a justiça

é de fato preservada, mas o protocolo não apresenta um grau mínimo de qualidade de serviço.

Definição 3.1.4. ***Tempestividade Assíncrona:*** *"Qualquer entidade pode terminar o protocolo a qualquer momento sem que haja perda de justiça." [42]*

Garantindo que as entidades possam terminar o protocolo de forma justa, e a qualquer momento, nos aproximamos de uma definição mais usável de tempestividade. Todavia, devemos nos lembrar de que nem sempre as entidades envolvidas em protocolos ou situações de disputa são especialistas; para muitos usuários de comércio eletrônico, por exemplo, nem sempre é evidente o momento em que eles se tornam comprometidos à transação corrente [30, 2]. Em um contexto orientado à tempestividade assíncrona, uma entidade pode ainda encontrar-se em uma situação em que deve decidir se abandona a troca ou aguarda por uma possível resposta atrasada. Se esta entidade é um usuário comum, sem conhecimento criptográfico ou da especificação do protocolo, esta questão pode representar um dilema inconveniente.

Por isso, definiremos uma interpretação mais prática da tempestividade assíncrona, que incorpora tanto características de segurança quanto o aspecto de qualidade de serviço. Nos referiremos a ela por **Tempestividade Forte**.

Definição 3.1.5. ***Tempestividade Forte:*** *Em qualquer momento, durante uma execução do protocolo, uma entidade honesta P tem a garantia de que o protocolo será terminado automaticamente em algum momento no futuro. Se alguma ação de P é necessária para tal, ela deve ser claramente especificada pelo protocolo, bem como as circunstâncias em que deve ser tomada. Ao ser completada, o estado da troca ou é final ou qualquer alteração de estado não degradará o grau de justiça adquirido por P.*

Existe uma sutil, porém importante diferença entre as Definições 3.1.1 e 3.1.4, e a Definição 3.1.5. A tempestividade original não exige que os projetistas se preocupem com como e quando o protocolo será terminado até que alguma entidade tenha se comprometido com a troca, e a tempestividade assíncrona permite que tal decisão seja deixada a cargo das entidades. Imaginemos um protocolo para duas entidades em que a participante A é a primeira a se comprometer com o protocolo, através do envio de seu item, digamos, na terceira mensagem. Se o protocolo busca tempestividade original ou assíncrona, então não são necessários mecanismos especiais para terminação após a primeira e segunda mensagens, já que nenhum comprometimento foi ainda realizado. Assim, se algum problema ocorrer antes do envio da terceira mensagem, alguma entidade honesta pode se encontrar em uma situação em que deve decidir se voluntariamente interrompe a troca, fechando o canal de comunicação, ou se aguarda um pouco mais por uma resposta. Se o protocolo busca tempestividade forte, por outro lado, a especificação

deve explicitar mecanismos de terminação também após a primeira e a segunda mensagens. Como nestes pontos nenhuma entidade ainda se comprometeu com o protocolo, tais mecanismos seriam provavelmente diferentes daqueles responsáveis pela terminação após a terceira mensagem. Uma boa estratégia geral seria utilizar *timeouts* pré-definidos na terminação de trocas em que nenhuma entidade estivesse ainda comprometida, e a intervenção de uma TTP somente após o primeiro comprometimento. Em termos práticos, a tempestividade forte garante que o usuário sempre receberá feedback sobre o estado final do protocolo, e jamais deveria ter que decidir por si só se aguarda um período indefinido por algum evento, ou se abandona a troca. O fato de que o protocolo deve especificar quando e como as entidades devem terminar o protocolo é o que torna a Definição 3.1.5 mais forte que a Definição 3.1.4.

O leitor cuidadoso notará que a Definição 3.1.2 diferencia-se das demais pela completa ausência de uma TTP. No entanto, a justiça forte (assim como a tempestividade forte) só pode ser obtida para ambas as entidades se uma TTP de alguma forma (online, inline ou offline) estiver disponível. Caso contrário, a entidade que primeiro abrir mão de seu item no protocolo sempre se encontrará em desvantagem, já que sua interlocutora poderia simplesmente abandonar a troca tão logo receba tal item. Este fato está diretamente relacionado à necessidade de uma TTP para estabelecer justiça determinística [43], conforme discutido na Seção 2.3.3.

3.2 Armadilhas comuns em tempestividade

No Capítulo 4, apresentamos uma forma de como modelar protocolos otimistas de trocas justas na forma de rastros comportamentais de entidades, para então analisá-los com o método de espaços de fitas. Este procedimento revelou ataques de naturezas diversas, e em especial um ataque de tempestividade que pudemos posteriormente detectar em uma série de outros protocolos otimistas. A Seção 3.2.3 apresenta tal ataque, bem como uma lista dos protocolos acometidos por ele. Nesta seção apresentaremos, na forma de armadilhas, duas falhas de projeto recorrentes.

3.2.1 Armadilha 1: Especificação do processamento de mensagens

A primeira armadilha se refere à especificação dos testes que devem ser realizados pelas entidades ao receber cada mensagem. Sabe-se que o processamento incorreto de mensagens pode resultar em falhas de segurança, o que pode ser particularmente

problemático no que diz respeito à tempestividade. Mais especificamente, deve estar claro que ações a TTP realiza ao receber um pedido de intervenção, no caso de alguma exceção. Um *pedido de intervenção* (a primeira mensagem de um subprotocolo auxiliar para tratamento de exceções) é geralmente mais extenso e complexo do que as demais mensagens do protocolo; isso ocorre devido à grande quantidade de informação necessária para que a TTP offline avalie a validade da transação (como o termo *offline* sugere, a TTP nem sequer tinha conhecimento da troca em questão até o momento da invocação). Para verificar se uma transação é válida, a TTP deve muitas vezes recalcular hashes, decifrar com sua chave privada textos cifrados por outras entidades, comparar valores de campos diferentes das mensagens, checar *timeouts*, e assim por diante. Cada um destes testes e das ações associadas a eles devem ser tão cuidadosamente especificadas quanto qualquer outro aspecto do protocolo; caso contrário, a implementação pode não refletir com fidelidade os objetivos originais do protocolo, o que pode resultar em ataques triviais [47].

3.2.2 Armadilha 2: não-testabilidade de termos

A segunda armadilha se refere à não-testabilidade de termos que compõem os pedidos de intervenção. Quando uma entidade se encontra em uma situação possivelmente injusta, ela invoca a TTP através da execução de um subprotocolo auxiliar. Ao receber o pedido de intervenção, a TTP realizará uma série de testes para decidir se a troca em questão é de fato válida. Se algum desses testes falhar, a TTP considera a transação como sendo inválida, e não intervirá nela. Por isso, é fundamental que os testes realizados pela TTP sobre o pedido de intervenção estejam também disponíveis ao solicitante.

Considere o seguinte exemplo: Uma entidade A tenta invocar uma TTP, através de um pedido de intervenção $M = M_1|M_2|M_3|M_4$ (M_1, M_2, M_3, M_4 são os termos que, concatenados, formam o pedido). Suponha que, para determinar se uma transação é válida, a TTP deve testar se $M_1 = M_2$ e se o deciframento de M_3 sob uma certa chave K_1 é igual ao deciframento de M_4 sob uma certa chave K_2, ambas conhecidas pela TTP. Se A é honesta e gerou todos os quatro termos, então A sabe que tais termos são o que de fato deveriam ser, e que a TTP terá sucesso em todos os testes sobre o pedido M. Se por outro lado alguns dos termos, digamos M_1 and M_3, forem gerados pelo interlocutor possivelmente malicioso de A, então o seguinte problema pode ocorrer: Ao receber M_1 ou gerar M_2 (o que ocorrer por último), A pode comparar M_1 com M_2 e parar imediatamente caso não sejam idênticos. O teste envolvendo M_3 e M_4, no entanto, é mais problemático para A; embora A seja capaz de gerar M_4, ela não pode gerar M_3 – e também não pode decifrar M_3, já que não conhece K_1. O segundo teste não é, portanto, verificável por A, pois M_3 é não-testável para ela. Assim, o interlocutor de A poderia desonestamente

enviar a ela um termo inválido no lugar de M_3, possivelmente lixo, e A não seria capaz de perceber a ação. Mais tarde, quando A tentasse invocar a TTP, o segundo teste falharia e a TTP pararia, deixando a entidade honesta A em um estado de injustiça.

Este é, na realidade, um problema recorrente em protocolos otimistas. Podemos encontrar na literatura diversos exemplos de protocolos susceptíveis a tal ataque [35, 62, 64, 45], bem como discussões sobre o assunto [48, 47, 6].

3.2.3 Casos de estudo e o ataque do termo não-testável

Um estudo sobre os protocolos ZDB [62], BWZZ [6], BWZZ melhorado [45], Zuo-Li [64], CEM1 [35], CEM2 [35], CEM tempestivo [42], e FPH [19], revelou que a maioria deles apresenta termos não-testáveis. Nesta seção, apresentaremos quais destes termos são não-testáveis e como estas falhas podem ser exploradas de forma a prejudicar três formas de tempestividade (Definições 3.1.1, 3.1.4 e 3.1.5 da Seção 3.1). As descrições detalhadas destes protocolos encontram-se no Apêndice A, à exceção do FPH [19] e do ZDB [62], que se encontram descritos nas Seções 4.3 e 4.4, respectivamente.

1. **ZDB [62]**: Termos L, $PU_T(K)$ e sub_K são não-testáveis por B após a segunda mensagem do protocolo.

2. **BWZZ [6]** e **BWZZ melhorado [45]**: Termo $PU_T(A, B, Hash(C), M)$ é não-testável por B após a segunda mensagem do protocolo.

3. **Componente de pagamento do Zuo-Li [64]**: Termo $PU_T(A, B, K)$ é não-testável por B após a segunda mensagem do protocolo.

4. **CEM1 [35]**: Termo $PU_T(A, B, M, R)$ é não-testável por B após a segunda mensagem do protocolo.

5. **CEM2 [35]** e **CEM tempestivo [35]**: Termo $PU_T(A, B, PU_B(M))$ é não-testável por B após a segunda mensagem do protocolo.

6. **FPH [19]**: Termo $PU_T(K)$ é não-testável por B após a terceira mensagem do protocolo.

Todos estes protocolos apresentam termos não-testáveis – interessantemente, até mesmo uma versão modificada do CEM, alterada para garantir tempestividade assíncrona (Definição 3.1.4). Isso significa que todos são suceptíveis a um ataque simples de tempestividade, ilustrado pela Figura 3.1. Além disso, dependendo do grau de comprometimento da entidade honesta B com o protocolo no momento da invocação

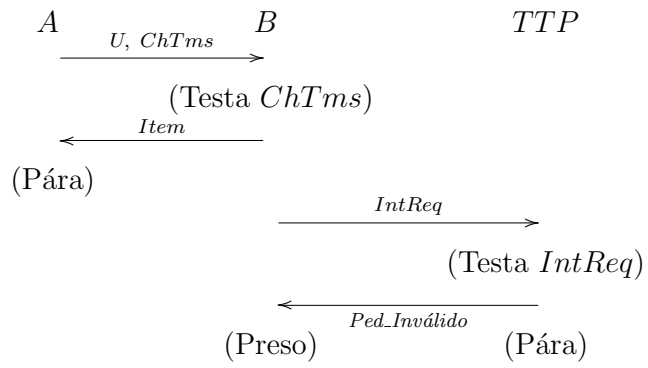


Figura 3.1: Forma geral do ataque do termo não-testável, U sendo o termo não-testável em cada protocolo.

da TTP, o grau obtido de justiça também pode ser afetado. Para um exemplo detalhado, refira à Seção 4.4.

Na Figura 3.1, U representa os termos não testáveis, e $ChTms$ são os termos testáveis que podem ser verificados por B no momento do recebimento. A primeira mensagem de qualquer subprotocolo auxiliar é denotada por $IntReq$, um pedido de intervenção que, no caso, contém U.

O ataque procede da seguinte forma: Uma entidade honesta B é contactada por uma participante mal-intencionada A para uma troca justa. A envia alguma informação inicial a B, mas substitui U por lixo. Quando B recebe esta informação inicial, verifica a validade de $ChTms$, já que U é não-testável. Embora o termo recebido no lugar de U não seja válido, B não é capaz de percebê-lo, e prossegue com a troca – possivelmente enviando seu próprio $Item$ na segunda mensagem (o que costuma ser uma abordagem comum em protocolos de três mensagens). Neste ponto, A maliciosamente abandona a troca, ao passo que B já se encontra comprometido com o protocolo. Eventualmente, B decide contactar a TTP e pedir auxílio para terminar a troca – simplesmente fechar o canal de comunicação violaria a especificação do protocolo e deixaria B em uma posição possivelmente injusta. Ele então compõe $IntReq$, utilizando o falso termo U, e envia o pedido à TTP. Esta, então, verifica cada informação contida em $IntReq$, inclusive U – que ela reconhece como inválido. Esta falha é suficiente para que a TTP declare como inválido o pedido de intervenção de B, e interrompa a execução do subprotocolo auxiliar. B, por sua vez, se encontra preso na execução do protocolo, incapaz de abandonar a troca de maneira justa.

3.3 Diretrizes orientadas à tempestividade para projeto de protocolos

Diretrizes informais para projeto de protocolos de segurança são amplamente propostas [30, 3, 1, 61] e representam princípios de boa prática, que podem auxiliar projetistas a evitarem armadilhas e outros erros comuns no processo. Nesta seção apresentaremos diretrizes específicas para a obtenção de tempestividade em protocolos otimistas para duas entidades, com base nos argumentos apresentados até agora neste capítulo.

3.3.1 Uso de *timeouts* pré-definidos

Protocolos de segurança (e particularmente protocolos de trocas justas) constantemente fazem uso de mecanismos de timeout como solução para tempestividade. No entanto, é preciso escolher cuidadosamente quando e como utilizar tais mecanismos, de forma que a justiça seja preservada. Digamos que B tenha acabado de enviar seu item a A, e está esperando que A lhe envie seu próprio item. Garantir que o protocolo será abortado após uma quantidade de tempo finita, caso B não receba a mensagem esperada de A, garante apenas justiça fraca (já que B ainda seria deixado em uma situação potencialmente injusta).

Sugerimos que *timeouts* pré-definidos sejam utilizados para abortar protocolos apenas nos estágios iniciais da troca, quando nenhum item foi ainda enviado (ao menos não em alguma forma recuperável sem informação adicional). Após o envio do primeiro item, o destinatário pode ter obtido alguma vantagem sobre o remetente; *timeouts* tornam-se portanto inúteis como formas diretas de terminação (embora sejam ainda úteis como forma de indicar às entidades quando iniciar a execução de subprotocolos auxiliares).

3.3.2 Mensagens de terminação

Protocolos de trocas justas são em geral implementados como parte de aplicações (um navegador de internet, ou mesmo um software disponibilizado por alguma loja online). Embora isso seja interessante sob o ponto de vista da transparência, projetistas tanto de protocolos como de aplicações devem ter em mente que as verdadeiras entidades de uma troca podem ser usuários comuns, como é o caso dos clientes em uma transação de comércio eletrônico, ao invés de sistemas automáticos executando as implementações do protocolo. Isso torna necessário que o protocolo informe o usuário caso qualquer ponto de terminação seja atingido – seja devido a algum timeout, seja por razão de falha. Suponhamos que o protocolo em questão tenha por objetivo a compra de conteúdo digital. É provável que alguma das entidades seja um usuário humano com pouca ou nenhuma

experiência técnica, ou mesmo conhecimento da especificação do protocolo. Suponhamos que ocorra o vencimento de algum timeout logo após o usuário ter entrado seu número de cartão de crédito na aplicação, mas antes deste ser enviado para a loja através da rede. O protocolo poderia simplesmente abortar, sem exibir ao usuário mensagem alguma, e ainda garantiria um estado final justo (afinal, nenhum item chegou a ser enviado). O usuário, por sua vez, não saberia se seu item – o número de cartão de crédito – teria sido comprometido pela aplicação, e portanto teria que escolher entre aguardar pela continuação da transação, ou correr o risco de abandonar a troca por si só.

A maneira mais simples de evitar tais dilemas é garantir que implementações do protocolo sempre informem quando o protocolo termina, seja pela efetivação da troca, seja devido a alguma falha prevista na especificação. Note que, em vista disso, é desaconselhado que a TTP – ou qualquer entidade honesta – simplesmente pare em caso de erro; uma mensagem deveria sempre ser enviada ao interlocutor para informar que o protocolo está sendo abortado.

3.3.3 Especificação clara de todos os testes, até os mais óbvios

Ao receber uma mensagem, uma entidade geralmente realiza uma série de testes para decidir se continua ou abandona a troca (quando a mensagem recebida contém uma assinatura digital, por exemplo). Uma enumeração detalhada de tais testes é fundamental em cada passo do protocolo, e pode permitir que termos não-testáveis sejam percebidos mais facilmente. Além disso, esta prática permite que a implementação final do protocolo seja a mais próxima possível da especificação original.

3.3.4 Evitar termos não-testáveis

Em protocolos otimistas de trocas justas, a TTP pode ser invocada através de subprotocolos auxiliares, que devem ser fornecidos pela especificação do protocolo. No entanto, é necessário garantir que as entidades serão capazes de iniciar uma execução auxiliar sempre que necessário. Falha em executar um subprotocolo representaria para a entidade em questão a possibilidade de ficar presa no protocolo, conforme discutido na Seção 3.2.2.

3.4 Um protocolo tempestivo genérico

Nesta seção apresentaremos um protocolo otimista para a troca justa de dois itens eletrônicos genéricos. Mais especificamente, nosso protocolo é projetado para prover tempestividade forte (Definição 3.1.5), seguindo as diretrizes discutidas neste capítulo.

3.4.1 Descrição de contexto

Em adição às funções $PU_O(), PR_O()$ de ciframento/deciframento assimétrico, precisaremos de uma função $f()$ que seja: (i) **Não-inversível:** Dado $f(M)$, é computacionalmente infactível deduzir M; (ii) **Comutativa com ciframento:** $f(PU_O(M)) = PU_O(f(M))$; e (iii) **Resistente a colisões:** É difícil encontrar $N \neq M$ tal que $f(M) = f(N)$.

Tal função $f()$ permitirá à entidade verificar informação secreta sem de fato conhecê-la. Exemplos de esquemas que apresentam estas propriedades são o Custódia Verificável (VE) [5] e o Ciframento Verificável e Recuperável (VRE) [41, 39, 40]; este último utiliza exponenciações baseadas em RSA tanto como $PU_O()$, quanto $f()$.

Assumiremos que o item de troca X seja descrito por $desc(X)$, e que o certificado $[desc(X), f(X), f()]$ esteja disponível em algum diretório público; o certificado deve estar assinado por uma autoridade certificadora. Quanto à natureza de X e Y, eles podem tanto ser os próprios itens como – mais provavelmente – chaves para o deciframento de textos cifrados previamente obtidos, que resultam nos reais itens quando decifrados.

Como exemplo, suponhamos que X seja um arquivo de música digital a ser trocado por um cheque assinado Y de um comprador. Então, $desc(X)$ incluiria o título da música, o artista intérprete e o nome do álbum que a contém, e X seria o arquivo da música em si ou uma chave simétrica para deciframento posterior de conteúdo cifrado pré-adquirido (possivelmente através de *download* realizado na própria página da loja). Assumindo que uma implementação do RSA seja usada para ciframento/deciframento simétrico, $f(M)$ poderia ser definida como $f(M) = M^e mod(n)$ para algum par (e, n) escolhido de maneira compatível ao RSA[1].

Neste exemplo, a companhia detentora dos direitos de distribuição sobre a música seria a autoridade a certificar a tupla $[desc(X), X^e mod(n), (e, n)]$, assinando-a. Este procedimento aconteceria apenas uma vez para cada música individual, durante o processo de contratação da loja A como vendedora daquela música. Neste processo, a companhia detentora seria a responsável por gerar o par (e, n), bem como os dois outros termos da tupla, antes de assiná-la.

3.4.2 Descrição do protocolo

A Figura 3.2 mostra o protocolo principal de troca da nossa proposta. Aqui, A e B representam as aplicações que implementam o protocolo em cada extremo da troca (iniciador e respondente, respectivamente) e SIG() é uma função de assinatura

[1]No VRE, este par é escolhido dentro dos limites estabelecidos pela Teoria da Validação Cruzada. O leitor interessado deve referir-se a [41, 39, 40] para maiores informações sobre o assunto.

digital. Também supomos que todas as mensagens enviadas através do canal alcançarão eventualmente seu destino, possivelmente fora de ordem.

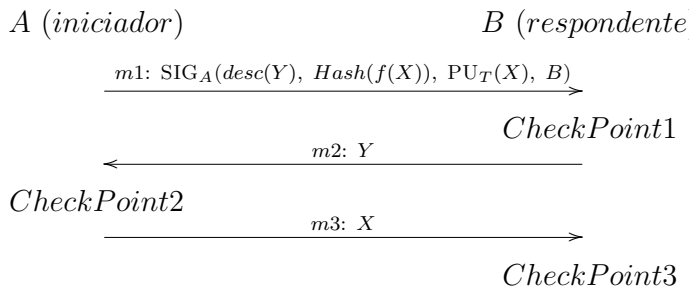


Figura 3.2: Protocolo principal

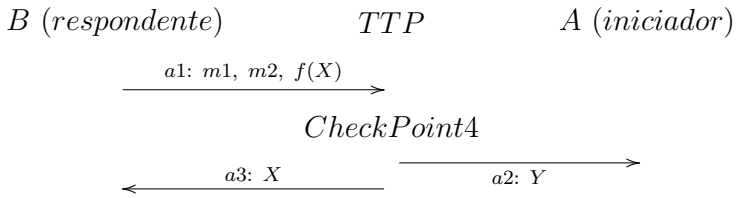


Figura 3.3: Subprotocolo auxiliar

Após cada mensagem, há um *CheckPoint*, ponto no qual a entidade que recebe a mensagem deve realizar uma série de testes pré-definidos, para então decidir se a mensagem é válida e se a entidade deve ou não prosseguir. Os *CheckPoints* serão definidos a seguir, com $f(X)$ e $f()$ estando publicamente disponíveis no certificado de X, $desc(Y)$ e $desc(X)$ sendo respectivamente as descrições previamente conhecidas de Y e X, e $PU_T(X)$ o ciframento de X com a chave pública da TTP:

- *CheckPoint1*: Se algum dos seguintes testes falhar, B abandona a troca.
  1. Verifica a validade da assinatura de A, $\mathrm{SIG}_A(desc(Y),\ Hash(f(X)),\ \mathrm{PU}_T(X),\ B)$, e se o último termo assinado é a identidade de B;
  2. Computa o hash de $f(X)$ e o compara com $Hash(f(X))$;
  3. Verifica se $f(\mathrm{PU}_T(X)) = \mathrm{PU}_T(f(X))$;
  4. Verifica se $desc(Y)$ descreve seu próprio item, Y;

5. Testa um *timeout* pré-definido.

- *CheckPoint2*: Se algum dos seguintes testes falhar, A abandona a troca.
  1. Testa Y contra $desc(Y)$;
  2. Testa um *timeout* pré-definido.
- *CheckPoint3*: Se algum dos seguintes testes falhar, B executa o subprotocolo auxiliar.
  1. Testa X contra $desc(X)$;
  2. Testa um *timeout* pré-definido.
- *CheckPoint4*: Se algum dos seguintes testes falhar, a TTP nega o pedido de intervenção de B e lhe envia uma mensagem de erro.
  1. Verifica a validade da assinatura de A, $\mathrm{SIG}_A(desc(Y),\ Hash(f(X)),\ \mathrm{PU}_T(X),\ B)$, e se o último termo assinado é a identidade de B;
  2. Testa Y em $m2$ contra $desc(Y)$ em $m1$;
  3. Testa se $f(\mathrm{PU}_T(X)) = \mathrm{PU}_T(f(X))$;
  4. Decifra $\mathrm{PU}_T(X)$ com sua própria chave privada.

3.4.3 Análise do protocolo

Os itens a serem trocados são X, de A, e Y, de B. Se B é honesto, uma entidade A mal-intencionada poderia tentar obter o item de B sem revelar X. Isso poderia ser tentado de algumas maneiras: (a) usando um item falso $X' \neq X$ na mensagem $m1$; (b) usando uma função$f()$ falsa na mensagem $m1$; (c) usando um X falso na mensagem $m3$; (d) se passando por B e enviando a mensagem $a1$ à TTP antes de contactar B para uma troca. Todos estes casos resultam em falha, pois: nos casos (a) e (b), os testes 2 e 3 do $CheckPoint1$ falhariam, portanto B teria abandonado a troca antes de enviar $m2$; no caso (c), o teste 1 do $CheckPoint3$ falharia, portanto B invocaria a TTP através do envio da mensagem $a1$; como a mensagem $m1$ estaria correta (caso contrário, B teria abandonado a troca antes de enviar $m2$), e assumindo que B se comporta honestamente, a TTP seria capaz de recuperar X a partir de $\mathrm{PU}_T(X)$; (d) isso só seria possível se A conhecesse Y de antemão, o que viola a especificação do protocolo.

Se A é honesto, por outro lado, um atacante B poderia tentar obter o item de A de apenas uma forma: enviando a mensagem $a1$ à TTP sem enviar a correta $m2$ a A. Neste caso, o teste 3 do $CheckPoint4$ falharia.

Assim, concluímos que nosso protocolo provê justiça forte, com garantias de tempestividade forte, conforme definido na Seção 3.1.

3.5 Conclusão

Neste capítulo apresentamos uma discussão sobre tempestividade, uma das três propriedades de segurança requeridas pelos protocolos de trocas justas. Reunimos inicialmente uma série de interpretações informais do problema de tempestividade, e discutimos como as diferenças entre tais interpretações afetam os objetivos do protocolo. A seguir, apontamos algumas falhas recorrentes no projeto de protocolos, que podem viabilizar ataques de termo não-testável. Finalmente, sugerimos algumas diretrizes para projeto que auxiliam a evitar tais armadilhas, e propusemos um protocolo genérico que acreditamos satisfazer tanto tempestividade, como justiça.

Embora não seja estritamente formal, acreditamos que este estudo contribui para a formalização da tempestividade no contexto de trocas justas. Outras abordagens informais [28, 30, 22] foram fundamentais no entendimento de problemas similares, e serviram como base para um processo de formalização adequado [21, 44]. Quanto às diretrizes propostas, esperamos que auxiliem até mesmo projetistas experientes a evitar falhas sutis, da mesma forma que o trabalho de outros [28, 30, 1, 61].

O protocolo proposto na Seção 3.4 respeita as diretrizes propostas, de forma a evitar as armadilhas discutidas na Seção 3.2. Existem alternativas projetadas com o objetivo de prover tempestividade e justiça, tal como o protocolo ASW [5]. Este protocolo foi inicialmente proposto para a troca justa de assinaturas digitais, ao passo que o nosso permite a troca de itens genéricos. Além disso, nosso protocolo utiliza apenas três mensagens, ao invés de quatro, como ocorre no ASW. Nosso subprotocolo auxiliar é suficiente para a invocação da TTP sempre que necessário; no ASW, três subprotocolos auxiliares são necessários para garantir justiça. Em compensação, o ASW possui uma análise formal de segurança, que nos falta no presente momento.

Capítulo 4

Espaços de fitas e trocas justas: modelagem e ataques

Este capítulo descreve como o método de espaços de fitas pode ser utilizado na verificação de protocolos de trocas justas otimistas entre duas entidades.

4.1 Papéis em trocas justas com parâmetros gerais

Da mesma forma que as propriedades clássicas de autenticação, é possível representar no modelo de espaços de fitas as propriedades de trocas justas [48], utilizando uma abordagem semelhante à dos testes de autenticação descritos na Seção 2.2.5. Em espaços de fitas, as demonstrações baseiam-se em provas de teoremas através da determinação de parâmetros de rastros, que descrevem as ações de cada entidade no protocolo. Propriedades de trocas justas podem ser verificadas da mesma forma.

Protocolos otimistas entre duas entidades normalmente envolvem três papéis: um **iniciador**, que é a entidade que dá início à troca; um **respondente**, que é a entidade contactada pelo iniciador; e uma **terceira parte confiável (TTP)**, responsável por resolver eventuais conflitos mediante invocação por alguma das outras partes.

Embora os parâmetros que formam uma fita regular possam variar consideravelmente de um protocolo para outro, há um certo padrão quando tratamos protocolos otimistas para duas entidades. Os seguintes parâmetros estarão presentes na maioria dos rastros:

$$[X, Y, T, o, o', d, d', C, F],$$

onde X, Y e T são as identidades do iniciador, do respondente e da TTP, respectivamente; o é o item do iniciador, que deve ser entregue ao respondente; o' é o item do respondente, que deve ser entregue ao iniciador; d e d' (equivalentes a $desc(o)$ e

$desc(o')$) são as descrições públicas de o e o', respectivamente, previamente conhecidas (são as informações necessárias para a validação dos itens recebidos, conforme argumentado na Seção 2.3.4); C é o token de cancelamento, fornecido pela TTP à entidade invocadora do subprotocolo de cancelamento; e F é o token de encerramento, fornecido pela TTP à entidade invocadora do subprotocolo de encerramento.

Estes são, em geral, os parâmetros necessários durante a verificação de um protocolo otimista de trocas justas. É importante notar que alguns deles podem não importar em determinados momentos da análise (C e F são ignorados durante uma verificação de efetividade, por exemplo), mas é prática comum representá-los por "*", ao invés de simplesmente ignorá-los.

4.2 Representação de propriedades de trocas justas no método de espaços de fitas

Nesta seção apresentaremos cada uma das propriedades de trocas justas propostas por Asokan [4], bem como suas representações na forma de teoremas gerais sobre espaços de fitas. Para que o protocolo satisfaça quaisquer destas propriedades, é necessário que as satisfaça para ambas as entidades envolvidas na troca. Aqui, consideraremos cada teorema sob o ponto de vista do iniciador. Em uma verificação completa do protocolo, os mesmos teoremas devem ser enunciados para o respondente.

4.2.1 Efetividade

Sejam A e B duas entidades envolvidas em uma troca, onde A é honesta. Se a entidade B também for honesta, e tanto A quanto B não desejarem abandonar a troca, então A terá obtido o' tal que $d' = desc(o')$, quando o protocolo estiver completo.

Teorema 4.2.1. *Seja A uma entidade associada a uma fita $s_A \in Init[A, *, T, o, *, d, d', *, *]$ e B uma entidade associada a uma fita $s_B \in Resp[*, B, T, *, o', d, d', *, *]$. Então, considerando-se apenas o protocolo principal de troca, $s_A \in Init[A, *, T, o, o', d, d', *, *]$ e $s_B \in Resp[*, B, T, o, o', d, d', *, *]$, onde d e d' são as descrições dos itens inicialmente possuídos por A e B, respectivamente, e $validar(o, d)$ e $validar(o', d')$ são verdadeiras.*

4.2.2 Justiça forte

Seja A uma entidade honesta. Então, quando o protocolo se completar, ou A tem o' que satisfaça $validar(o', d')$, ou B não obteve informação adicional relevante sobre o.

Teorema 4.2.2. *Seja A uma entidade associada a uma fita $s_A \in Init[A, B, T, o, *, d, d', *, *]$. Então ou $s_A \in Init[A, B, T, o, o', d, d', *, *]$ com $validar(o', d') = TRUE$, ou $s_B \notin (Resp[*, B, *, o, o', d, d', *, *] \cup Init[B, *, *, *, o, *, d, *, *])$, onde $validar(o, d) = TRUE$.*

4.2.3 Justiça fraca

Seja A uma entidade honesta. Então, quando o protocolo se completar, ou A tem o' que satisfaça $validar(o', d')$, ou B não obteve informação adicional relevante sobre o, ou A consegue provar a um juiz que B recebeu (ou ainda pode receber) o que satisfaça $validar(o, d)$, sem qualquer intervenção futura de A.

Teorema 4.2.3. *Seja A uma entidade associada a uma fita $s_A \in Init[A, B, T, o, *, d, d', *, *]$. Então ou vale o Teorema 4.2.2, ou $\exists\ s_B \in (Resp[*, B, *, o, o', d, d', *, *] \cup Init[B, *, *, *, o, *, d, *, *])$, onde $validar(o, d) = TRUE$.*

4.2.4 Irretratabilidade

Seja A uma entidade honesta. Então, após uma troca efetiva (i.e., A recebeu o' que satisfaz $validar(o', d')$ ao final da troca), A pode provar que

- **Irretratabilidade de origem**: o' foi enviado por B, e
- **Irretratabilidade de recebimento**: B recebeu o que satisfaz $validar(o, d)$.

Teorema 4.2.4. *Seja A uma entidade associada a uma fita $s_A \in Init[A, B, T, o, o', d, d', *, *]$ com $validar(o', d') = TRUE$ e $validar(o, d) = TRUE$. Então:*

- ***Irretratabilidade de origem:*** *$\exists\ s_B$ tal que $s_B \in (Resp[*, B, *, *, o', *, d', *, *] \cup Init[B, *, *, o', *, *, d, *, *])$, com $validar(o', d') = TRUE$.*
- ***Irretratabilidade de recebimento:*** *$\exists s_B$ tal que $s_B \in Resp[*, B, *, o, *, d, *, *, *] \cup Init[B, *, *, *, o, *, d, *, *])$, com $validar(o, d) = TRUE$.*

4.2.5 Verificabilidade de TTP

Assumindo-se que a terceira parte T possa ser forçada a eventualmente enviar uma resposta válida[1] a toda requisição, esta propriedade exige que caso T se comporte mal,

[1]Como o conceito de **resposta válida** varia de protocolo para protocolo, não é possível enunciar um teorema geral que represente esta propriedade. Veja a Seção 4.3 para um exemplo.

resultando na perda de justiça para A, então A pode provar tal mau comportamento a um juiz em uma disputa externa. Em outras palavras, cada uma das outras entidades tem garantia de justiça fraca mesmo no evento de uma TTP mal comportada.

4.2.6 Tempestividade

Seja A uma entidade honesta. Então A pode ter a certeza de que o protocolo será completado em algum momento. Quando isso ocorrer, o estado da troca é final, ou qualquer alteração neste estado não reduz o grau de justiça adquirido por A.

A propriedade de tempestividade apresenta sutilezas que impedem, no momento, uma interpretação formal satisfatória. Apresentaremos uma discussão detalhada sobre esta propriedade no Capítulo 3.

4.3 O protocolo FPH

O FPH [19] foi proposto como um protocolo otimista para duas entidades, visando a entrega certificada de mensagens. Os itens a serem trocados são uma mensagem M (originada pelo iniciador) e seu recibo $|h|_B$ (pelo respondente). Ambos os tokens C e F fornecidos pela TTP são morfologicamente idênticos, o que resulta nos ataques descritos na Seção 4.3.3. Nesta seção, utilizaremos a seguinte notação específica, além daquela apresentada na Seção 2.1.1:

i. $|h|_A = |H(H(\{M\}_K), PU_T(K))|_A$: primeira parte da evidência de irretratabilidade de origem da mensagem M, obtida por B;

ii. $|h|_B = |H(H(\{M\}_K), PU_T(K))|_B$: parte da evidência de irretratabilidade de recebimento da mensagem M, obtida por A;

iii. $|\text{chave} = K|_A$: segunda parte da evidência de irretratabilidade de origem de M, obtida por B;

iv. $|\text{chave} = K|_T$: segunda parte alternativa da evidência de irretratabilidade de origem de M, obtida por B;

v. $|H(H(\{M\}_K), PU_T(K), |h|_A)|_A$: evidência de que A solicitou a intervenção da TTP;

vi. $|H(H(\{M\}_K), PU_T(K), |h|_A, |h|_B)|_B$: evidência de que B solicitou a intervenção da TTP;

vii. $|H(|h|_B)|_T$: assinatura da TTP sobre $|h|_B$, que prova sua intervenção.

4.3.1 Descrição do protocolo

Nesta seção descreveremos os três componentes do FPH.

1. $A \rightarrow B : \{M\}_K, PU_T(K), |h|_A$ (em caso de exceção, B pára)
2. $B \rightarrow A : |h|_B$ (em caso de exceção, A executa o protocolo de cancelamento)
3. $A \rightarrow B : |\text{chave} = K|_A$ (em caso de exceção, B executa o protocolo de conclusão)

Figura 4.1: FPH: protocolo principal de troca

1. $B \rightarrow T : H(\{M\}_K), PU_T(K), |h|_A, |h|_B, |H(H(\{M\}_K), PU_T(K), |h|_A, |h|_B)|_B$
2. $T \rightarrow B : \dfrac{|H(\textit{cancelled}, |h|_B)|_T \text{ (se } \textit{cancelled} = \text{TRUE)}}{|\text{chave} = K|_T \text{ (se } \textit{cancelled} = \text{FALSE)} \quad \textit{finished} := TRUE}$

Figura 4.2: FPH: protocolo de conclusão

1. $A \rightarrow T : H(\{M\}_K), PU_T(K), |h|_A, |H(H(\{M\}_K), PU_T(K), |h|_A)|_A$
2. $T \rightarrow A : \dfrac{|h|_B, |H(|h|_B)|_T \text{ (se } \textit{finished} = \text{TRUE)}}{|H(\textit{cancelled}, |h|_A)|_T \text{ (se } \textit{finished} = \text{FALSE)} \quad \textit{cancelled} := TRUE}$

Figura 4.3: FPH: protocolo de cancelamento

4.3.2 Representação em espaços de fitas e definição de fitas regulares

Nesta seção definiremos os rastros das fitas regulares para o protocolo principal e seus subprotocolos (conclusão e cancelamento). O espaço de fita Σ pode ser formado por qualquer combinação de tais fitas.

Protocolo principal

A Figura 4.4 ilustra o protocolo principal do FPH.

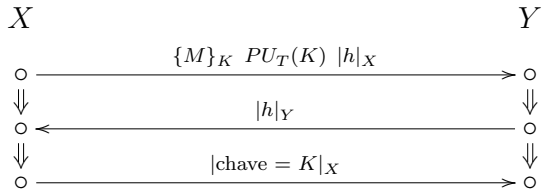


Figura 4.4: Protocolo principal

1. Uma fita $s \in Init[X, Y, *, M, |h|_Y, \{M\}_K, h, *, *, K]$ se e só se s tem rastro da forma

$$\langle +\{M\}_K\ PU_T(K)\ |h|_X,\ \ -|h|_Y,\ \ +|\text{chave} = K|_X\ \rangle$$

onde $X, Y \in T_{name}$ com $X \neq Y$, $T \in T_{ttp}$ com T_{name} e T_{ttp} disjuntos, $h = H(\{M\}_K, PU_T(K))$ e $H()$ é uma função de hash não-inversível e resistente a colisões, M é o item inicial de X, com descrição $\{M\}_K$, $|h|_Y$ é o item inicial de Y, com descrição h e $K \in \mathcal{K}$. A entidade associada à fita $s \in Init$ é A.

2. Uma fita $s \in Resp[X, Y, *, M, |h|_Y, G, h, *, *, G', K]$ se e só se s apresenta rastro da forma

$$\langle -G\ G'\ |h|_X,\ \ +|h|_Y,\ \ -|\text{chave} = K|_X\ \rangle$$

A entidade associada a uma fita $s \in Resp$ é B.

Protocolo de conclusão

A Figura 4.5 ilustra o protocolo de conclusão do FPH.

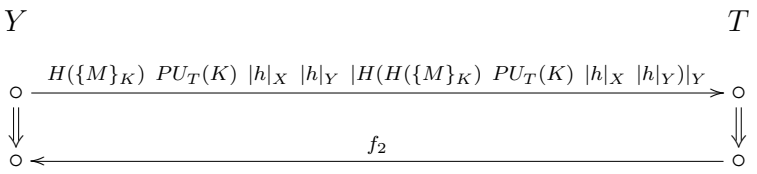


Figura 4.5: Protocolo de conclusão

1. Uma fita $s \in Resp[X, Y, T, M, |h|_Y, G, h, C_R, F_R, G', K]$ se e só se s apresenta rastro da forma

$$\langle +H(G)\ G'\ |h|_X\ |h|_Y\ |H(H(\{M\}_K)\ PU_T(K)\ |h|_X\ |h|_Y)|_Y,\ \ -f_2\rangle$$

onde $f_2 = C_R = |(H(\text{cancelled},\ |h|_Y))|_T$ é um token de cancelamento fornecido pela TTP ao respondente, caso o protocolo tenha sido cancelado, e $f_2 = |\text{chave} = K|_T$ caso contrário. A entidade associada a uma fita $s \in Resp$ é B.

2. Uma fita $s \in Serv[X, Y, T, M, |h|_Y, \{M\}_K, h, C_R, F_R, K]$ se e só se s apresenta rastro da forma

$$\langle -H(\{M\}_K)\ PU_T(K)\ |h|_X\ |h|_Y\ |H(H(\{M\}_K)\ PU_T(K)\ |h|_X\ |h|_Y)|_Y,\ \ +f_2\rangle$$

A entidade associada a uma fita $s \in Serv$ é T.

Protocolo de cancelamento

A Figura 4.6 ilustra o protocolo de cancelamento do FPH.

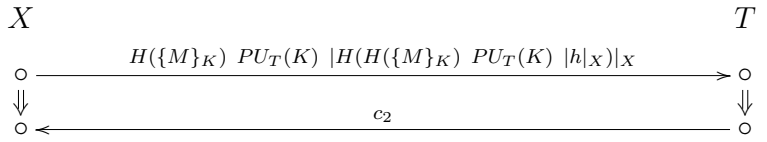


Figura 4.6: Protocolo de cancelamento

1. Uma fita $s \in Init[X, Y, T, M, G'', \{M\}_K, h, C_I, F_I, K]$ se e só se s apresenta rastro da forma

$$\langle +H(\{M\}_K)\ PU_T(K)\ |H(H(\{M\}_K)\ PU_T(K)\ |h|_X)|_X,\ \ -c_2\rangle$$

onde $c_2 = C_I = |(H(\text{cancelled},\ |h|_X))|_T$ é um token de cancelamento fornecido pela TTP ao iniciador, caso o protocolo não tenha sido ainda concluído (através da execução do protocolo de conclusão) e $c_2 = |h|_Y\ |H(|h|_Y)|_T$ caso contrário. A entidade associada a uma fita $s \in Init$ é A.

2. Uma fita $s \in Serv[X, Y, T, M, G'', \{M\}_K, h, C_I, F_I, K]$ se e só se s rastro da forma

$$\langle -H(\{M\}_K)\ PU_T(K)\ |H(H(\{M\}_K)\ PU_T(K)\ |h|_X)|_X,\ \ +c_2\rangle$$

A entidade associada a uma fita $s \in Serv$ é T.

Os conjuntos *Serv, Resp* e *Init* são dois a dois disjuntos.

4.3.3 Verificação do protocolo FPH

Nesta seção provaremos que o FPH falha em prover verificabilidade de TTP, mas consegue garantir efetividade.

Efetividade

Para verificar a efetividade do protocolo, demonstraremos que se tanto A quanto B completarem a troca sem tentarem abandonar o protocolo, então é possível obter $s_A \in Init[A, *, T, M, |h|_B, \{M\}_K, h, *, *, K]$ e $s_B \in Resp[*, B, T, M, |h|_B, G, h, *, *, G', K]$, com $G = \{M\}_K$ e $G' = PU_T(K)$.

Teorema 4.3.1. *Seja A uma entidade associada a uma fita $s_A \in Init[A, *, T, M, *, \{M\}_K, h, *, *, K]$, M, K originados unicamente em Σ e K uma chave boa. Suponha que s_A apresenta $\mathcal{C}$-altura ≥ 2 e a chave privada de B é segura*[2]*. Então $s_A \in Init[A, *, T, M, |h|_B, \{M\}_K, h, *, *, K]$.*

Demonstração. A primeira mensagem recebida por A (nó $\langle s_A, 2\rangle$) contém o termo $|h|_B$, que pode ser interpretado como um teste de autenticação não-solicitado [25] (h é recentemente gerado por A a partir dos valores originados unicamente $\{M\}_K$ e $PU_T(K)$). Isso significa que existe um participante honesto B (tal que $B \neq A$), representado por uma fita $s_B \in \Sigma$ em que $|h|_B$ é originado. Devemos considerar dois casos para s_B (não consideraremos o caso em que B pode ser a TTP, pois consideramos os conjuntos T_{name} e T_{ttp} como sendo disjuntos):

1. $s_B \in Init[B, *, T, M, *, \{M\}_K, h, *, *, K]$ e $|h|_B \in term(\langle s_B, 1\rangle)$. Pela origem única de M, K concluímos que $A = B$, o que é uma contradição.

2. $s_B \in Resp[*, B, T, *, |h|_B, G, h, *, *, G', *]$ e $|h|_B = term(\langle s_B, 2\rangle)$. Note que G e G' são não-testáveis por B. Isso significa que B só pode recuperar M e testar a igualdade $G = \{M\}_K$ ao receber K, ao passo que a igualdade $G' = PU_T(K)$ jamais poderá ser verificada por ele. Por outro lado, a igualdade $h = H(G, G')$ é testável. Portanto, sob o ponto de vista de A, o recebimento de $|h|_B$ significa que B verificou que $h = H(G, G')$. Como A conhece os valores de G e G', pode concluir que $s_B \in Resp[*, B, T, *, |h|_B, \{M\}_K, h, *, *, PU_T(K), *]$. Pela origem única de M e pelo fato de K ser boa, A sabe de $\{M\}_K$ que B está envolvido na mesma execução que ela, e portanto $s_A \in Init[A, B, T, M, |h|_B, \{M\}_K, h, *, *, K]$. ■

[2] *Segura, neste contexto, significa conhecida apenas por B.*

Teorema 4.3.2. *Seja B uma entidade associada a uma fita $s_B \in Resp[*, B, T, X, |h|_B, G, h, *, *, G', *]$. Suponha que s_B apresenta $\mathcal{C}$-altura $= 3$ e que a chave privada de A é segura. Então $s_B \in Resp[*, B, T, M, |h|_B, G, h, *, *, G', K]$, com $G = \{M\}_K$.*

Demonstração. A última mensagem recebida por B (nó $\langle s_B, 3\rangle$) pode ser interpretada como um teste de autenticação não-solicitado [25]. Isso significa que existe um participante honesto A (com $B \neq A$), representado por uma fita $s_A \in \Sigma$ em que $|chave = K|_A$ é originado. Apenas fitas de iniciador possuem um termo com este formato, portanto $s_A \in Init[A, *, *, *, *, *, *, *, *, *, K]$ (note que B deve confiar em A quanto à geração de uma chave K boa). Agora, B pode verificar se os demais valores estão corretos. Ele verifica que $G = \{M\}_K$ através do deciframento de G com K. Pelo fato de K ser boa, B conclui que $G = \{M\}_K$ foi originado por A, bem como h. Portanto, $s_A \in Init[A, *, T, M, *, \{M\}_K, h, *, *, K]$, e $s_B \in Resp[A, B, T, M, |h|_B, G, h, *, *, G', K]$, with $G = \{M\}_K$. ■

Verificabilidade de TTP

Nesta seção demonstraremos que o protocolo FPH não provê verificabilidade de TTP.

Teorema 4.3.3. *O FPH provê verificabilidade de TTP nas seguintes circunstâncias:*

- ***quanto à irretratabilidade de origem:*** *$\exists s_B$ tal que $s_B \in Resp[A, B, T, M, *, G, h, *, *, G', K]$, com $G = \{M\}_K, G' = PU_T(K)$ e K boa. Segue que $\nexists\ s_A \in Init[A, *, T, M, *, \{M\}_K, h, C, *, K]$ com um token de cancelamento C válido (um token de cancelamento C é considerado válido se $C = |(H(cancelled, |h|_A))|_T$).*
- ***quanto à irretratabilidade de recebimento:*** *$\exists s_A$ tal que $s_A \in Init[A, B, T, M, |h|_B, \{M\}_K, h, *, *, K]$, com $PR_B()$ segura. Segue que $\nexists s_B \in Resp[*, B, T, M, *, G, h, C, *, G', K]$ com $G = \{M\}_K, G' = PU_T(K)$ e um token de cancelamento C válido (um token de cancelamento C é considerado válido se $C = |(H(cancelled,\ |h|_B))|_T$).*

Demonstração. Suponha que o protocolo provê verificabilidade de TTP quanto à irretratabilidade de recebimento. Então, $\exists s_A \in Init[A, B, T, M, |h|_B, \{M\}_K, h, *, *, K]$, e como $PR_B()$ é segura, $|h|_B$ deve ter sido originada no segundo nó de uma fita de respondente, em uma execução do protocolo principal, ou no primeiro nó de uma fita de respondente, em uma execução do protocolo de conclusão (neste caso, A estaria participando como TTP, o que é impossível devido ao fato de T_{name} e T_{ttp} serem disjuntos).

Portanto, concluímos que s_A apresenta $\mathcal{C}$-altura ≥ 2 em uma execução do protocolo principal.

Como supomos que o protocolo provê verificabilidade de TTP, não pode existir uma entidade B que conhece um C válido para a execução em que A está envolvido (ou seja, $C = |(H(\text{cancelled}, |h'|_B))|_{T'}$ com $T' = T$, $h' = h = H(\{M\}_K, PU_T(K)))$. Se tal entidade existisse, então C teria sido originado em um dos seguintes casos:

1. protocolo de conclusão: C foi produzido por T em uma instância do protocolo de conclusão, após a execução de uma instância do protocolo de cancelamento por algum participante Z. Portanto, $s_B \in Resp[*, B, T, *, |h|_B, G, h, C, *, G', *]$ e $\exists s_Z \in Init[Z, *, T, *, *, \{M\}_K, h, C', *, K]$, onde $C' = |(H(\text{cancelled},|h|_Z))|_T$. Note que os parâmetros $\{M\}_K$ e K derivam do acordo entre as entidades B e Z, associada a s_Z, sobre os valores $H(\{M\}_K)$ e $PU_T(K)$, enviados na primeira mensagem da execução do protocolo de cancelamento. Neste caso, pela origem única de M, ou $Z = A$, ou $Z = B$.

2. protocolo de cancelamento: C foi produzido por T em uma instância do protocolo de cancelamento, executada antes de alguma instância do protocolo de conclusão para os mesmos valores de T and h. Portanto, $s_B \in Init[B, *, T, *, *, \{M\}_K, h, C, *, K]$. ■

O protocolo FPH permite ambos os cenários, o que representa falha em garantir verificabilidade de TTP. No primeiro caso, seja $Z = B$. A entidade A inicia uma troca com o participante B, desonesto. B então finge ser o iniciador de uma troca falsa, e utiliza os valores recebidos de A na troca real para requerer o cancelamento da troca falsa junto à TTP. A TTP fornece o token de cancelamento $C' = |(H(\text{cancelled},|h|_B))|_T = C$ a B e marca a troca falsa como cancelada. Agora, B invoca a TTP para requerer a conclusão da troca real. A TTP identifica A como sendo o iniciador e verifica que esta troca não foi cancelada, fornecendo K a B, e marca a troca real como concluída. Agora B pode prosseguir com o restante do protocolo principal, enviando $|h|_B$ a A, ou pode simplesmente não enviar mais nada a A, o que faria com que A invocasse a TTP para cancelamento. Como a execução já foi concluída por B, A obtém $|h|_B$ da TTP. Ao final da troca, $s_A \in Init[A, B, T, M, |h|_B, \{M\}_K, h, *, *, K]$ e $s_B \in Resp[*, B, T, M, |h|_B, G, h, C, *, G', K]$, com $G = \{M\}_K, G' = PU_T(K)$ e um token de cancelamento $C = |(H(\text{cancelled}, |h|_B))|_T$ válido. Isso viola o Teorema 4.3.3 e reflete um ataque previamente conhecido [37] ao protocolo FPH.

Se considerarmos o último cenário, um ataque [48] semelhante torna-se possível. Se B continuasse a execução do protocolo principal logo após o recebimento do token de cancelamento C da TTP, ao invés de invocar o protocolo de conclusão, a entidade A também obteria h_B e B também conseguiria C.

O que torna ambos os ataques possíveis é a similaridade dos tokens de cancelamento fornecidos pela TTP em ambos os subprotocolos. Se a identidade do iniciador fosse inserida nos tokens, nenhum dos ataques detectados seriam possíveis (C' e C seriam diferentes).

4.4 O protocolo ZDB

O ZDB [62] foi proposto como um protocolo otimista para duas entidades, visando irretratabilidade. Existem alguns ataques publicados ao ZDB, bem como versões corrigidas [8, 24]. A troca ocorre em dois passos: Primeiro, o iniciador troca um termo C (que é uma mensagem M cifrada sob uma chave K, inicialmente desconhecida pelo respondente) por um recibo EOR_C. A seguir, o iniciador troca a chave K por outro recibo, EOR_K. Estes dois recibos formam o item o' do respondente. Nesta seção, utilizaremos a seguinte notação específica, além daquela apresentada na Seção 2.1.1:

i. $e_K(M)$ e $d_K(M)$: ciframento e deciframento da mensagem M com a chave K;

ii. $C = e_K(M)$: texto cifrado contendo a mensagem M;

iii. $L = H(M, K)$: identificador que liga C a K ($H()$ é uma função de *hash* criptográfico);

iv. $f_1, f_2, \ldots, f_8$: marcadores de mensagens, utilizados para indicar o propósito das mensagens que os contêm;

v. $PU_T(K)$ ciframento da chave K sob a chave pública da TTP;

vi. $sig_A(W)$: assinatura digital da entidade A sobre a mensagem W, utilizando a chave privada de A. O esquema utilizado não deve permitir que W seja recuperável a partir da assinatura;

vii. $EOO_C = sig_A(f_1, B, L, C)$: evidência da origem de C;

viii. $EOR_C = sig_B(f_2, A, L, EOO_C)$: evidência do recebimento de C;

ix. $EOO_K = sig_A(f_3, B, L, K)$: evidência da origem de K;

x. $EOR_K = sig_B(f_4, A, L, EOO_K)$: evidência do recebimento de K;

xi. $sub_K = sig_A(f_5, B, L, K, TTP, EOO_C)$: evidência da submissão de K à TTP;

xii. $con_K = sig_{TTP}(f_6, A, B, L, K)$: evidência da confirmação de K pela TTP;

xiii. $abort = sig_{TTP}(f_8, A, B, L)$: evidência de cancelamento;

4.4.1 Descrição do protocolo

Nesta seção descreveremos os três componentes do ZDB.

1. $A \rightarrow B : f_1, f_5, B, L, C, T, PU_T(K), EOO_C, sub_K$ (em caso de exceção, B pára)
2. $B \rightarrow A : f_2, A, L, EOR_C$ (em caso de exceção, A executa o protocolo de cancelamento)
3. $A \rightarrow B : f_3, B, L, K, EOO_K$ (em caso de exceção, B executa o protocolo de conclusão)
4. $B \rightarrow A : f_4, A, L, EOR_K$ (em caso de exceção, A executa o protocolo de conclusão)

Figura 4.7: ZDB: protocolo principal

1. $U \rightarrow T : f_1, f_2, f_5, A, B, L, T, PU_T(K), sub_K, EOO_C, EOR_C$
2. $T \rightarrow U : \dfrac{f_8, A, B, L, abort \text{ (se } aborted = \text{TRUE)}}{f_2, f_6, A, B, L, K, con_K, EOR_C \text{ (se } aborted = \text{FALSE)} \;\; finished := TRUE}$

Figura 4.8: ZDB: protocolo de conclusão

1. $A \rightarrow T : f_7, B, L, sig_A(f_7, B, L)$
2. $T \rightarrow A : \dfrac{f_2, f_6, A, B, L, K, con_K, EOR_C \text{ (se } finished = \text{TRUE)}}{f_8, A, B, L, abort \text{ (se } finished = \text{FALSE)} \;\; aborted := TRUE}$

Figura 4.9: ZDB: protocolo de cancelamento

4.4.2 Representação em espaços de fitas e definição de fitas regulares

Nesta seção definiremos os rastros das fitas regulares para o protocolo principal e seus subprotocolos (conclusão e cancelamento). O espaço de fita Σ pode ser formado por qualquer combinação de tais fitas.

Protocolo principal

A Figura 4.10 ilustra o protocolo principal do ZDB.

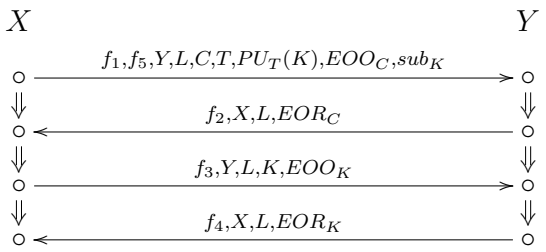


Figura 4.10: Protocolo principal

1. Uma fita $s \in Init[X, Y, T, M, K, EOR_C, EOR_K, C, PU_T(K), EOO_C, EOO_K, *, *, L]$ se e só se s apresenta rastro da forma

$$\begin{aligned}&\langle +\{f_1\ f_5\ Y\ L\ C\ T\ PU_T(K)\ EOO_C\ sub_K,\\ &-f_2\ X\ L\ EOR_C,\\ &+f_3\ Y\ L\ K\ EOO_K,\\ &-f_4\ X\ L\ EOR_C\rangle\end{aligned}$$

   onde $X, Y \in T_{name}$ com $X \neq Y$, $T \in T_{ttp}$ com T_{name} e T_{ttp} disjuntos, $K \in \mathcal{K}$. A entidade associada a uma fita $s \in Init[X, Y, T, M, K, EOR_C, EOR_K, C, PU_T(K), EOO_C, EOO_K, *, *, L]$ é A.

2. Uma fita $s \in Resp[X, Y, T, M, K, EOR_C, EOR_K, G_2, G_3, EOO_C, EOO_K, *, *, G_1, G_4]$ se e só se s apresenta rastro da forma

$$\begin{aligned}&\langle -\{f_1\ f_5\ Y\ G_1\ G_2\ T\ G_3\ EOO_C\ G_4,\\ &+f_2\ X\ G_1\ EOR_C,\\ &-f_3\ Y\ G_1\ K\ G_5,\\ &+f_4\ X\ G_1\ EOR_C\rangle\end{aligned}$$

   onde $G_1, G_2, G_3, G_4, G_5 \in \mathcal{A}$ (que é o conjunto de todos os termos possíveis em Σ [57]) não são testáveis por Y. Embora alguns destes termos possam tornar-se testáveis por B em algum momento (como $G_5 = EOO_K$, que torna-se testável tão logo B recebe K), eles não são testáveis no momento em que B os recebe, e portanto são representados desta forma. A entidade associada à fita $s \in Resp[X, Y, T, M, K, EOR_C, EOR_K, G_2, G_3, EOO_C, EOO_K, *, *, G_1, G_4]$ é B.

Protocolo de conclusão

A Figura 4.11 ilustra o protocolo de conclusão do ZDB.

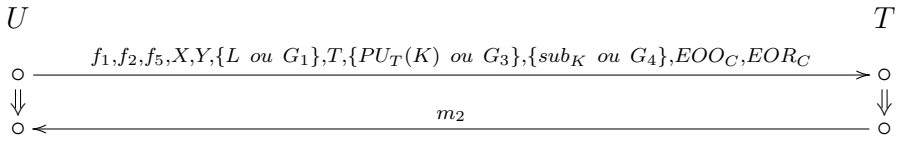


Figura 4.11: Protocolo de conclusão

1. Uma fita $s \in Init[X, Y, T, M, K, EOR_C, EOR_K, C, PU_T(K), EOO_C, EOO_K, abort, con_K, L]$ se e só se s apresenta rastro da forma

$$\langle +f_1\ f_2\ f_5\ X\ Y\ L\ T\ PU_T(K)\ sub_K\ EOO_C\ EOR_C,\ \ -f_2\rangle$$

   onde $m_2{=}f_8, X, Y, L, abort$ se o protocolo foi cancelado, e $m_2 = f_2, f_6, X, Y, L, K, con_K, EOR_C$ caso contrário. A entidade associada a uma fita $s \in Init[X, Y, T, M, K, EOR_C, EOR_K, C, PU_T(K), EOO_C, EOO_K, abort, con_K, L]$ é A.

2. Uma fita $s \in Resp[X, Y, T, M, K, EOR_C, EOR_K, G_2, G_3, EOO_C, EOO_K, abort, con_K, G_1, G_4]$ se e só se s apresenta rastro da forma

$$\langle +f_1\ f_2\ f_5\ X\ Y\ G_1\ T\ G_3\ G_4\ EOO_C\ EOR_C,\ \ -m_2\rangle$$

   A entidade associada a uma fita $s \in Resp[X, Y, T, M, K, EOR_C, EOR_K, G_2, G_3, EOO_C, EOO_K, abort, con_K, G_1, G_4]$ é B.

3. Uma fita $s \in Serv[X, Y, T, M, K, EOR_C, C, PU_T(K), EOO_C, abort, con_K, L, sub_K]$ se e só se s apresenta rastro da forma

$$\langle -f_1\ f_2\ f_5\ X\ Y\ L\ T\ PU_T(K)\ sub_K\ EOO_C\ EOR_C,\ \ +m_2\rangle$$

   A fita asociada a uma fita $s \in Serv[X, Y, T, M, K, EOR_C, C, PU_T(K), EOO_C, abort, con_K, L, sub_K]$ é T.

Protocolo de cancelamento

A Figura 4.12 ilustra o protocolo de cancelamento do ZDB.

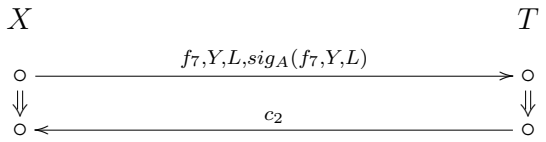


Figura 4.12: Protocolo de cancelamento

1. Uma fita $s \in Init[X, Y, T, M, K, EOR_C, EOR_K, C, PU_T(K), EOO_C, EOO_K, abort, con_K, L]$ se e só se s apresenta rastro da forma

$$\langle +f_7\ Y\ L\ sig_A(f_7\ Y\ L),\ \ -c_2 \rangle$$

onde $c_2 = f_2, f_6, A, B, L, K, con_K, EOR_C$ se o protocolo foi concluído, e $c_2 = f_8, A, B, L, abort$ caso contrário. A entidade associada a uma fita $s \in Init[X, Y, T, M, K, EOR_C, EOR_K, C, PU_T(K), EOO_C, EOO_K, abort, con_K, L]$ é A.

2. Uma fita $s \in Serv[X, Y, T, M, K, EOR_C, C, PU_T(K), EOO_C, abort, con_K, G_1, sub_K]$ se e só se s apresenta rastro da forma

$$\langle -f_7\ Y\ G_1\ sig_A(f_7\ Y\ G_1),\ \ +c_2 \rangle$$

A entidade associada a uma fita $s \in Serv[X, Y, T, M, K, EOR_C, C, PU_T(K), EOO_C, abort, con_K, G_1, sub_K]$ é T.

Os conjuntos *Serv, Resp* e *Init* são dois a dois disjuntos.

4.4.3 Verificação do protocolo ZDB

Na Seção 3.2.3, apresentamos a estrutura geral do nosso ataque do termo não-testável, que explora uma falha de segurança característica de diversos protocolos otimistas de trocas justas. Nesta seção, exemplificaremos tal ataque, detalhando como o ZDB falha em garantir tempestividade [24].

Tempestividade

A definição dos rastros de respondente do ZDB evidencia três termos não-testáveis a B – G_1, G_3 e G_4 – presentes no pedido de intervenção do protocolo de conclusão. Ao ser invocada, a TTP T realizaria, entre outros, os seguintes procedimentos:

1. Decifrar G_3 e obter K'; se A é honesto, $K' = K$, caso contrário, podemos ter $K' \neq K$;
2. $G_1 = L = H(M, K')$;
3. $G_4 = sub_K = sig_A(f_5, B, L, K, T, EOO_C)$.

Se algum dos testes acima falhar, T declara que o pedido de intervenção é inválido e não se involve na troca. Assim, um iniciador mal-intencionado poderia forjar qualquer destes três termos, e B só perceberia tarde demais. A Figura 4.13 ilustra o ataque.

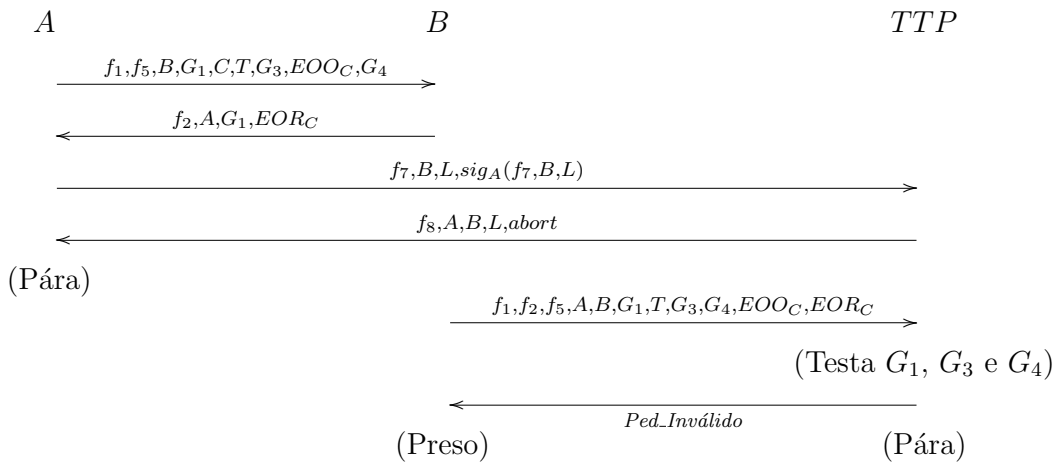


Figura 4.13: Ataque do termo não-testável sobre o ZDB, com G_1, G_3 e G_4 não-testáveis.

Como podemos perceber, um iniciador A malicioso poderia enviar um L, sub_K ou $PU_T(K)$ ilegítimos a B, na primeira mensagem do protocolo. Se os demais termos fossem válidos, B prosseguiria para a segunda mensagem do protocolo, revelando seu item parcial EOR_C. Se A abandonasse a execução do protocolo neste momento, simplesmente fechando o canal de comunicação, B estaria preso na execução, incapaz de construir um pedido de intervenção com a informação reunida. Note que A poderia ainda executar o protocolo de cancelamento junto à TTP, obtendo um o *token* de cancelamento *abort*. Este token constitui evidência de cancelamento para A, que pode então comprovar seu

"bom-comportamento" a um juiz externo, enquanto B – a entidade de fato honesta – não obtém nenhuma evidência de que participou da troca.

Embora esta falha não implique em perda de justiça, já que B nunca revela a outra metade EOR_K de seu item, ela pode resultar em uma entidade honesta ser julgada inadequadamente, além de evitar que ela possa terminar o protocolo de maneira adequada.

Capítulo 5

Conclusão e trabalhos futuros

Embora de aparência enganosamente simples, os protocolos criptográficos são artefatos difíceis de se projetar corretamente. Por esta razão, técnicas de verificação formal fazem-se necessárias não só para auxiliar na detecção e correção de falhas de segurança, mas também como argumento para correção de novos protocolos. Existem registros de sucesso na aplicação de algumas técnicas a protocolos de trocas justas – ainda que tais técnicas tenham sido desenvolvidas para protocolos clássicos – conforme discutido na Seção 2.3.6. A adaptação proposta para o método de espaços de fitas, apresentada no Capítulo 4, foi capaz de detectar falhas de segurança desconhecidas até então. No entanto, a completude de tal adaptação sob o ponto de vista formal é ainda um problema em aberto, que só pode ser devidamente estudado sob a luz de propriedades de trocas justas mais bem-definidas que aquelas atualmente disponíveis.

A ausência de consenso na interpretação das propriedades de trocas justas não prejudica apenas a aplicação de técnicas de verificação formal sobre estes protocolos. Seu projeto também se torna mais sucetível a falhas, gerando especificações ambíguas e incompletas, conforme discutido no Capítulo 3 sobre tempestividade. Buscando resolver este impasse, diversos autores têm buscado formalizar ao menos parcialmente tais propriedades [22] [21] [28].

Além de técnicas formais, outras metodologias podem ser utilizadas para auxiliar o de projeto de protocolos livres de falhas. Ao estudar protocolos previamente propostos e ataques sobre eles, alguns autores têm contribuído com diretrizes para projeto [30] [1]. Embora representem conceitos a princípio informais, diretrizes de projeto condensam lições aprendidas por pesquisadores e podem evitar que projetistas incidam nos mesmos erros que seus antecessores. Na Seção 3.3, apresentamos nossa própria contribuição na forma de diretrizes para projeto de protocolos tempestivos de trocas justas.

Na Seção 2.3.4 apresentamos o problema da validação de itens, procedimento fundamental para o estabelecimento de justiça em protocolos de trocas justas. Embora

existam diversas abordagens para a validação, todas dependem da disponibilidade de uma descrição unívoca do item a ser validado; de que tal descrição constitui é assunto raramente tratado.

Assim, a troca justa de itens indescritíveis é ainda um problema em aberto. Itens de multimídia, como arquivos de música, vídeo e imagens, se enquadram nesta classificação, e são atualmente o carro-chefe do comércio eletrônico. Assim, o estudo de soluções justas para a troca e compra/venda destes itens se faz necessário. Tentativas recentes aplicaram protocolos pessimistas para este fim [7], ainda que tal solução não seja eficiente, tampouco adequada sob o ponto de vista do usuário. Idealmente, os trabalhos futuros nesta área de pesquisa deveriam focar em métodos de validação para itens indescritíveis, orientados ao projeto de protocolos otimistas de trocas justas.

Referências Bibliográficas

[1] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[2] Ross J. Anderson. Liability and computer security: Nine principles. In *Computer Security — ESORICS 94, Springer LNCS*, volume 875, pages 231–245. Springer-Verlag, 1994.

[3] Ross J. Anderson and Roger M. Needham. Robustness principles for public key protocols. In *CRYPTO '95: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, pages 236–247, London, UK, 1995. Springer-Verlag.

[4] A. Asokan. *Fairness in Electronic Commerce*. PhD thesis, University of Waterloo, 1998.

[5] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 1999.

[6] Feng Bao, Guilin Wang, Jianying Zhou, and Huafei Zhu. Analysis and improvement of Micali's fair contract signing protocol. In *In: Information Security and Privacy (ACISP'04), LNCS 3108*, pages 176–187. Springer-Verlag, 2004.

[7] Andrea Bottoni, Gianluca Dini, and Tage Stabell-Kulø. A methodology for verification of digital items in fair exchange protocols with active trustee. *Electronic Commerce Research*, 7(2):143–164, 2007.

[8] Colin Boyd and Peter Kearney. Exploring fair exchange protocols using specification animation. In *ISW '00: Proceedings of the Third International Workshop on Information Security*, pages 209–223, London, UK, 2000. Springer-Verlag.

[9] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer-Verlag, 2003.

[10] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[11] Iliano Cervesato, Nancy Durgin, Max I. Kanovich, and Andre Scedrov. Interpreting Strands in Linear Logic. In H. Veith, N. Heintze, and E. Clark, editors, *2000 Workshop on Formal Methods and Computer Security — FMCS'00*, Chicago, IL, 20 July 2000.

[12] Tom Coffey and Puneet Saidha. Non-repudiation with mandatory proof of receipt. *SIGCOMM Comput. Commun. Rev.*, 26(1):6–17, 1996.

[13] Federico Crazzolara and Glynn Winskel. Petri nets in cryptographic protocols. In *IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 149, Washington, DC, USA, 2001. IEEE Computer Society.

[14] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[15] David L. Dill. The murphi verification system. In *In Computer Aided Verification. 8th International Conference*, pages 390–393. Springer-Verlag, 1996.

[16] Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Completeness of the authentication tests. In Joachim Biskup and Javier Lopez, editors, *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 106–121. Springer, 2007.

[17] Nancy Durgin, John Mitchell, and Dusko Pavlovic. A compositional logic for proving security properties of protocols. *J. Comput. Secur.*, 11(4):677–721, 2003.

[18] F. Javier Thayer Fabrega, Jonathan Herzog, and Joshua D. Guttman. Strand space pictures. In *Proceedings, Workship on Formal Methods and Security Protocols*, June 1998. Co-located with LICS'98.

[19] J. L. Ferrer-Gomila, M. Payeras-Capellà, and L.H. Rotger. An efficient protocol for certified electronic mail. In *Third International Workshop - ISW 2000*, volume 1975 of *Lecture Notes in Computer Science*, pages 237–248, Berlin, 2000. Springer-Verlag.

[20] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[21] Felix Gärtner. The problem of fair exchange, its formalization, and its relation to other problems in distributed computing, 2002.

[22] Felix C. Gartner, Henning Pagnia, and Holger Vogt. Approaching a formal definition of fairness in electronic commerce. In *SRDS '99: Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, page 354, Washington, DC, USA, 1999. IEEE Computer Society.

[23] Stefanos Gritzalis and Diomidis Spinellis. Cryptographic protocols over open distributed systems: A taxonomy of flaws and related protocol analysis tools. In *16th International Conference on Computer Safety, Reliability and Security: SAFECOMP '97*, pages 123–137, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1997. Springer Verlag.

[24] Sigrid Gürgens, Carsten Rudolph, and Holger Vogt. On the security of fair non-repudiation protocols. *Int. J. Inf. Secur.*, 4(4):253–262, 2005.

[25] Joshua D. Guttman and F. Javier Thayer. Authentication tests and the structure of bundles. *Theor. Comput. Sci.*, 283(2):333–380, 2002.

[26] Joshua D. Guttman, F. Javier Thayer, Jay A. Carlson, Jonathan C. Herzog, John D. Ramsdell, and Brian T. Sniffen. Trust management in strand spaces: A rely-guarantee method. In David A. Schmidt, editor, *ESOP*, volume 2986 of *Lecture Notes in Computer Science*, pages 325–339. Springer, 2004.

[27] J. Kohl and C. Neuman. The kerberos network authentication service (v5), 1993.

[28] Steve Kremer, Olivier Markowitch, and Jianying Zhou. An intensive survey of fair non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, 2002.

[29] Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. *J. Comput. Secur.*, 11(3):399–429, 2003.

[30] Panagiotis Louridas. Some guidelines for non-repudiation protocols. *SIGCOMM Comput. Commun. Rev.*, 30(5):29–38, 2000.

[31] Alfred P. Maneki. Honest functions and their application to the analysis of cryptographic protocols. In *CSFW '99: Proceedings of the 12th IEEE workshop on Computer Security Foundations*, page 83, Washington, DC, USA, 1999. IEEE Computer Society.

[32] Olivier Markowitch and Yves Roggeman. Probabilistic non-repudiation without trusted third party. In *Second Workshop on Security in Communication Network*, September 1999.

[33] Wilfredo Marrero. Brutus: A model checker for security protocols. Technical report, Process Capability, Release 3.0, Bell Canada Acquisitions, 2001.

[34] Catherine A. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26:113–131, 1996.

[35] Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 12–19, New York, NY, USA, 2003. ACM Press.

[36] Brian Monahan. Introducing ASPECT - a tool for checking protocol security. Technical Report HPL-2002-246, HP Technical Report, 2002.

[37] José R.M. Monteiro and Ricardo Dahab. An attack on a protocol for certified delivery. *Information Security - 5th International Conference, ISC 2002, Proceedings, LNCS 2433*, 2002.

[38] Aybek Mukhamedov, Steve Kremer, and Eike Ritter. Analysis of a multi-party fair exchange protocol and formal proof of correctness in the strand space model. In Andrew S. Patrick and Moti Yung, editors, *Revised Papers from the 9th International Conference on Financial Cryptography and Data Security (FC'05)*, volume 3570 of *Lecture Notes in Computer Science*, pages 255–269, Roseau, The Commonwealth Of Dominica, August 2005. Springer.

[39] A. Nenadic, N. Zhang, and S. K. Barton. FIDES: A middleware e-commerce security solution. In *The 3rd European Conference on Information Warfare and Security*, pages 295–304, June 2004.

[40] A. Nenadic, N. Zhang, and S. K. Barton. A security protocol for certified e-goods delivery. In *Information Assurance and Security*, pages 22–28. IEEE Computer Society, April 2004.

[41] A. Nenadic, N. Zhang, Q. Shi, and C. Goble. DSA-based verifiable and recoverable encryption of signatures and its application in certified e-goods delivery. In *EEE '05: Proceedings of IEEE Conference on e-Technology, e-Commerce and e-Service*, pages 94–99. IEEE Computer Society, 2005.

[42] Jose Onieva, Jianying Zhou, and Javier Lopez. Enhancing certified email service for timeliness and multicasting. In *Steven M. Furnell and Paul S. Dowland (Eds.). Proceedings of the Fourth International Network Conference, INC'04, 6-9 July, Plymouth, UK*, pages 327–335, October 2004.

[43] Henning Pagnia and Felix C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Darmstadt, Germany, 1999.

[44] Henning Pagnia, Holger Vogt, and Felix C. Gaertner. Fair Exchange. *The Computer Journal*, 46(1):55, 2003.

[45] Magdalena Payeras-Capell, Josep Llus Ferrer-Gomila, and Llorenc Huguet-Rotger. Achieving fairness and timeliness in a previous electronic contract signing protocol. In *ARES*, pages 717–722. IEEE Computer Society, 2006.

[46] Fabio R. Piva, Augusto J. Devegili, and Ricardo Dahab. Verification of three authentication protocols usin BAN, SVO and Strand Spaces. Technical Report IC-06-017, Institute of Computing, University of Campinas, 2006.

[47] Fabio R. Piva, José R. M. Monteiro, and Ricardo Dahab. Strand spaces and fair exchange: More on how to trace attacks and security problems. In *Anais do VII SBSeg, Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 80–93. UFRJ/NCE, September 2007.

[48] Fabio R. Piva, José R. M. Monteiro, Augusto J. Devegili, and Ricardo Dahab. Applying strand spaces to certified delivery proofs. In *Anais do VI SBSeg, Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. UFRJ/NCE, September 2006.

[49] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *CSFW: Proceedings of the IEEE workshop on Computer Security Foundations*, pages 98–107. IEEE Computer Society, 1995.

[50] Steve Schneider. Formal analysis of a non-repudiation protocol. In *CSFW: Proceedings of the IEEE workshop on Computer Security Foundations*, pages 54–65. IEEE Computer Society, 1998.

[51] Vitaly Shmatikov and John C. Mitchell. Analysis of a fair exchange protocol. In *NDSS*. The Internet Society, 2000.

[52] Dawn Xiaodong Song. Athena: a new efficient automatic checker for security protocol analysis. In *CSFW '99: Proceedings of the 12th IEEE workshop on Computer Security Foundations*, page 192, Washington, DC, USA, 1999. IEEE Computer Society.

[53] P. Syverson. Towards a strand semantics for authentication logics, 1999.

[54] Paul F. Syverson. A taxonomy of replay attacks. In *CSFW: Proceedings of the IEEE workshop on Computer Security Foundations*, pages 187–191, 1994.

[55] Paul F. Syverson and Iliano Cervesato. The logic of authentication protocols. In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design (FOSAD 2000), Tutorial Lectures*, volume LNCS 2171 of *Lecture Notes in Computer Science*, pages 63–136. Springer-Verlag, September 2000.

[56] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. In *CSFW: Proceedings of the IEEE workshop on Computer Security Foundations*, pages 72–82, 1999.

[57] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2–3):191–230, 1999.

[58] J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 1999.

[59] Holger Vogt. Asynchronous optimistic fair exchange based on revocable items. In *Financial Cryptography*, pages 208–222, 2003.

[60] Thomas Y. C. Woo and Simon S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, 1992.

[61] Thomas Y. C. Woo and Simon S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 28(3):24–37, 1994.

[62] Jianying Zhou, Robert H. Deng, and Feng Bao. Evolution of fair non-repudiation with ttp. In *ACISP '99: Proceedings of the 4th Australasian Conference on Information Security and Privacy*, pages 258–269, London, UK, 1999. Springer-Verlag.

[63] Jianying Zhou and Dieter Gollmann. Towards verification of non-repudiation protocols. In *In Proceedings of 1998 International Refinement Workshop and Formal Methods Pacific*, pages 370–380. Springer-Verlag, 1998.

[64] Min Zuo and Jianhua Li. Constructing fair-exchange P2P file market. In Hai Zhuge and Geoffrey Fox, editors, *GCC*, volume 3795 of *Lecture Notes in Computer Science*, pages 941–946. Springer, 2005.

Apêndice A

Descrição de protocolos susceptíveis ao ataque do termo não-testável

Neste apêndice estão disponíveis as descrições dos protocolos que serviram como casos de estudo para nossa discussão sobre o ataque do termo não-testável, apresentada no Capítulo 3. As únicas exceções são o FPH [19] e o ZDB [62], que se encontram descritos nas Seções 4.3 e 4.4, respectivamente.

A.1 BWZZ [6]

A.1.1 Notação

i. C: contrato a ser assinado por A e B;

ii. (M, R): par de *nonces* escolhido aleatoriamente por A;

iii. $(SIG_A(A, B, T, C, Z), M, R)$: assinatura de A sobre o contrato; é o item a ser trocado por $SIG_B(A, B, T, C, Z)$;

iv. $SIG_B(A, B, T, C, Z)$: assinatura de B sobre o contrato; é o item a ser trocado por $(SIG_A(A, B, T, C, Z), M, R)$;

v. $Z = PU_T(A, B, H(C), M, R)$;

A.1.2 Descrição

1. $A \rightarrow B : SIG_A(A, B, T, C, Z)$ (B checa a assinatura; pára em caso de exceção)
2. $B \rightarrow A : SIG_B(A, B, T, C, Z)$ (A checa a assinatura; pára em caso de exceção)
3. $A \rightarrow B : M, R$ (B computa Z, compara o resultado com o Z recebido previamente; executa o protocolo de conclusão em caso de exceção)

Figura A.1: BWZZ: protocolo principal de troca

1. $B \rightarrow T : SIG_A(A, B, T, C, Z), SIG_B(A, B, T, C, Z)$ (T checa ambas as assinaturas, decifra Z, checa o resultado; pára em caso de exceção)
2. $T \rightarrow B : SIG_A(A, B, T, C, Z), M, R$
3. $T \rightarrow A : SIG_B(A, B, T, C, Z)$

Figura A.2: BWZZ: protocolo de conclusão

A.2 BWZZ melhorado [45]

A.2.1 Notação

i. C: contrato a ser assinado por A e B;

ii. (M, R): par de *nonces* escolhido aleatoriamente por A;

iii. (sig_A, M, R): assinatura de A sobre o contrato; é o item a ser trocado por $SIG_B(A, B, T, C, Z)$;

iv. $sig_B = SIG_B(A, B, T, C, Z)$: assinatura de B sobre o contrato; é o item a ser trocado por $(SIG_A(A, B, T, C, Z), M, R)$;

v. $sig_A = SIG_A(A, B, T, C, Z)$

vi. $Z = PU_T(A, B, H(C), M, R)$;

vii. $reqproof_X$: evidência de que X está de fato recorrendo à TTP por auxílio.

A.2.2 Descrição

1. $A \rightarrow B : sig_A$ (B checa a assinatura; pára em caso de exceção)
2. $B \rightarrow A : sig_B$ (A checa a assinatura; executa o protocolo de cancelamento em caso de exceção)
3. $A \rightarrow B : M, R$ (B computa Z, compara o resultado com o Z recebido previamente; executa o protocolo de conclusão em caso de exceção)

Figura A.3: BWZZ melhorado: protocolo principal de troca

1. $B \rightarrow T : sig_A, sig_B, reqproof_B$
2. $T \rightarrow B : \dfrac{SIG_T(\text{“cancelled”}, sig_B) \text{ (se } cancelled = \text{TRUE)}}{M, R \text{ (se } cancelled = \text{FALSE)} \quad finished := TRUE}$

Figura A.4: BWZZ melhorado: protocolo de conclusão

1. $A \rightarrow T : sig_A, reqproof_A$
2. $T \rightarrow B : \dfrac{sig_B \text{ (se } finished = \text{TRUE)}}{SIG_T(\text{“cancelled”}, sig_A) \text{ (se } finished = \text{FALSE)} \quad cancelled := TRUE}$

Figura A.5: BWZZ melhorado: protocolo de cancelamento

A.3 Componente de pagamento do Zuo-Li [64]

A.3.1 Notação

i. $C = (A, B, TTP, ChequeSN, Price, TS, H(KP), H(\{KP\}_K))$;

ii. $SIG_B(C, Z)$: cheque digital de B; é o item a ser trocado por KP;

iii. KP: arquivo de A; é o item a ser trocado por $SIG_B(C, Z)$;

iv. $\{KP\}_K$: KP, cifrado com uma chave simétrica K; B obteve este termo previamente à execução do protocolo de pagamento;

v. $ChequeSN$: número de série do cheque digital de B;

vi. $Price$: preço de KP, acordado previamente por B e A;

vii. TS: *timestamp*;

viii. $Z = PU_T(A, B, K)$;

A.3.2 Descrição

1. $A \rightarrow B : C, Z, SIG_A(C, Z)$ (B checa cada campo de C e a assinatura; pára em caso de exceção)
2. $B \rightarrow A : SIG_B(C, Z)$ (A checa a assinatura; pára em caso de exceção)
3. $A \rightarrow B : K$ (B decifra $\{KP\}_K$, aplica o *hash* sobre o resultado e compara com $H(KP)$; executa o protocolo de conclusão em caso de exceção)

Figura A.6: Componente de pagamento do Zuo-Li: protocolo principal de troca

1. $B \rightarrow T : C, Z, SIG_A(C, Z), SIG_B(C, Z)$ (T checa cada campo de C, checa ambas as assinaturas, decifra Z; pára em caso de exceção)
2. $T \rightarrow B : K$
3. $T \rightarrow A : SIG_B(C, Z)$

Figura A.7: Componente de pagamento do Zuo-Li: protocolo de conclusão

A.4 CEM1 [35]

A.4.1 Notação

i. M: mensagem de A; é o item a ser trocado por $SIG_B(Z)$;

ii. $SIG_B(Z)$: confirmação de que B recebeu a mensagem M; é o item a ser trocado por M;

iii. R: *nonce* gerado por A;

iv. $Z = PU_T(A, B, M, R)$;

A.4.2 Descrição

1. $A \rightarrow B : Z$ (B pára em caso de exceção)
2. $B \rightarrow A : SIG_B(Z)$ (A checa a assinatura; pára em caso de exceção)
3. $A \rightarrow B : M, R$ (B computa Z, compara o resultado com o Z recebido previamente; executa o protocolo de conclusão em caso de exceção)

Figura A.8: CEM1: protocolo principal de troca

1. $B \rightarrow T : Z, SIG_B(Z)$ (T checa a assinatura, decifra Z; pára em caso de exceção)
2. $T \rightarrow B : M, R$
3. $T \rightarrow A : SIG_B(Z)$

Figura A.9: CEM1: protocolo de conclusão

A.5 CEM2 [35]

A.5.1 Notação

i. M: mensagem de A; é o item a ser trocado por $SIG_B(Z)$;

ii. $SIG_B(Z)$: confirmação de que B recebeu a mensagem M; é o item a ser trocado por M;

iii. $Z = PU_T(A, B, PU_B(M))$;

A.5.2 Descrição

1. $A \rightarrow B : Z$ (B pára em caso de exceção)
2. $B \rightarrow A : SIG_B(Z)$ (A checa a assinatura; pára em caso de exceção)
3. $A \rightarrow B : PU_B(M)$ (B computa Z, compara o resultado com o Z recebido previamente,decifra $PU_B(M)$; executa o protocolo de conclusão em caso de exceção)

Figura A.10: CEM2: protocolo principal de troca

1. $B \rightarrow T : Z, SIG_B(Z)$ (T checa a assinatura, decifra Z; pára em caso de exceção)
2. $T \rightarrow B : PU_B(M)$
3. $T \rightarrow A : SIG_B(Z)$

Figura A.11: CEM2: protocolo de conclusão

A.6 CEM tempestivo [35]

A.6.1 Notação

i. M: mensagem de A; é o item a ser trocado por $SIG_B(Z)$;

ii. $SIG_B(Z)$: confirmação de que B recebeu a mensagem M; é o item a ser trocado por M;

iii. t_A: *timeout* de A; representa o tempo máximo que A está disposta a aguardar pela conclusão da troca;

iv. $Z = PU_T(A, B, PU_B(M))$;

A.6.2 Descrição

1. $A \rightarrow B : SIG_A(t_A, Z)$ (B checa a assinatura, checa se ainda tem tempo hábil para concluir a troca; pára em caso de exceção)
2. $B \rightarrow A : SIG_B(Z)$ (A checa a assinatura; pára em caso de exceção)
3. $A \rightarrow B : PU_B(M)$ (B computa Z, compara o resultado com o Z recebido previamente, decifra $PU_B(M)$; executa o protocolo de conclusão em caso de exceção)

Figura A.12: CEM tempestivo: protocolo principal de troca

1. $B \rightarrow T : SIG_A(t_A, Z), SIG_B(Z)$ (T checa ambas as assinaturas, checa se t_A ainda não ocorreu, decifra Z; pára em caso de exceção)
2. $T \rightarrow B : PU_B(M)$
3. $T \rightarrow A : SIG_B(Z)$

Figura A.13: CEM tempestivo: protocolo de conclusão