Uma Abordagem à Comunicação Segura em Aplicações Distribuídas

José Carlos Rufino Amaro

Mestrado em Informática

Área de Especialização em Sistemas Distribuídos, Comunicações por Computador e Arquitecturas de Computadores

Universidade do Minho

Setembro 1997

Uma Abordagem à Comunicação Segura em Aplicações Distribuídas

José Carlos Rufino Amaro*

Mestrado em Informática

Área de Especialização em Sistemas Distribuídos, Comunicações por Computador e Arquitecturas de Computadores

Tese submetida à Universidade do Minho para obtenção do grau de Mestre em Informática, elaborada sob a orientação do Prof. Dr. Francisco Soares Moura.

Setembro 1997

(versão revista de Junho de 1998)

*Financiado pelo PRAXIS XXI, bolsa BM/2273/94

Abstract

This work presents S3L, a typical application–layer API providing security services (Privacy, Authentication, Integrity and Sequencing — against replay attacks —) immediately above the Berkeley sockets programming interface with which shares great resemblance, both in syntatic and in semantic ways. S3L allows migration, with a minimal effort, for non–secure Berkeley sockets based applications to a more secure operation mode where the exchange of security items (*e.g.*, keys, algorithms, sequencing data, authentication codes, etc.) happens in an almost completely stateless way. As long as an initial run of the proposed Skeyx protocol takes place (allowing for the secure exchange of Diffie–Hellman non–certified public keys) then, any future contacts between two communicating entities can take place, both over connectionless (IP/UDP) — including multicast mode — and connection–oriented (TCP) protocols, without a prior secure session establishment phase. Security items specific for each packet travel side by side with user data and the unique (but mandatory) run of the Skeyx protocol takes place transparently, on–the–fly. Benchmarking proves that, although relying on software based encryption, the benefits derived from the extra qualities of service available upon the use of the S3L API surpass the inevitable overhead introduced.

Resumo

Nesta dissertação apresenta–se o S3L, fundamentalmente uma API que fornece serviços de segurança (Privacidade, Autenticação, Integridade e Sequenciação — contra *ataques–de–repetição* —) imediatamente acima da interface de programação dos *sockets* de Berkeley, preservando, tanto quanto possível, grandes semelhanças sintácticas e semânticas com o dito modelo de programação. O S3L permite a migração, com um mínimo de esforço, de aplicações não seguras baseadas em *sockets* de Berkeley, para um modo de operação mais seguro, no qual a troca de informação de segurança (*e.g.*, chaves, algoritmos, números de sequência, códigos de autenticação, etc.) ocorre de uma forma quase *stateless*. Desde que tenha lugar uma execução inicial do protocolo proposto Skeyx (o qual permite a troca segura de chaves públicas de Diffie–Hellman não certificadas) então, quaisquer contactos futuros entre duas entidades comunicantes poderão ocorrer, quer sobre protocolos *não–orientados–à–conexão* (IP/UDP) — incluindo a variante multiponto —, quer sobre protocolos *orientados–à–conexão* (TCP), sem que para tal seja necessário o estabelecimento prévio de uma sessão segura. A informação de segurança específica para cada pacote viaja lado a lado com os dados do utilizador e a única (mas obrigatória) execução do protocolo Skeyx tem lugar de uma forma transparente e automática. A análise de desempenho ao S3L prova que, apesar da sua dependência de criptografia por *software*, os benefícios obtidos com as qualidades de serviço extra ultrapassam a inevitável sobrecarga introduzida.

Para os meus pais, pelo apoio incondicional e pela paciência demonstrada durante a gestação desta tese.

Agradecimentos

Ao meu orientador, Prof. Dr. Francisco Soares Moura, pelo tema de investigação e pelo interesse permanente na melhoria da qualidade desta dissertação.

A todos os elementos do Grupo de Sistemas Distribuídos, pela sua disponibilidade constante em fornecer ajuda.

Aos colegas de mestrado, especialmente à Maria João, ao Exposto e ao Albano, agora colegas de profissão, pelo bom ambiente de trabalho proporcionado.

A Andrew Romanenko, pelo auxílio prestado no desenvolvimento do código *IP Multicast*.

Ao ilustre deputado Dr. José Magalhães, por ter respondido com solicitude às minhas dúvidas sobre a legalidade de um trabalho numa área potencialmente controversa como é a Criptografia.

À Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança, pelas facilidades concedidas durante o período de elaboração desta tese.

À Xana, pela revisão do documento.

Conteúdo

Capítulo 1

Introdução

Em meados dos anos setenta, a entidade governamental norte–americana *Defense Advanced Research Projects Agency* (DARPA) iniciou um projecto do qual resultaria a definição de um novo e revolucionário conjunto de protocolos na área das Comunicações por Computador, hoje em dia conhecido por TCP/IP [Pos80c, Pos80b]. Esta pilha de protocolos transformou–se actualmente no *standard de facto* que permite a interligação, à escala global, de redes de computadores baseadas nas mais diversas tecnologias, servindo organizações e indivíduos de interesses muitos diversificados. Essa imensa rede é universalmente conhecida por *Internet*.

Apesar da influência que os meios militares tiveram no desenho original dos protocolos (e da própria infra–estrutura da *Internet*), tal não impediu que posteriormente se identificassem alguns problemas de segurança na pilha TCP/IP [Bel89]. Contudo, foi só na década presente que a preocupação com a segurança das comunicações ganhou o protagonismo de que hoje goza. Certamente, tal facto é indissociável do crescimento exponencial da Internet[1], o qual reforça ainda mais a necessidade de as organizações e os indivíduos garantirem a segurança dos seus dados, face a uma exposição cada vez maior a ataques.

A importância da Criptografia no contexto actual das Comunicações por Computador é, portanto, inquestionável, suscitando um interesse crescente fora dos meios que tradicionalmente se lhe associam: Defesa, Espionagem e Diplomacia. Pode–se inclusivamente afirmar que a Criptografia e, em geral, as questões associadas à Segurança no domínio da Informática estão "na moda". Por exemplo, a definição de um mecanismo de transacções seguras universal (que parece bem encaminhado através do protocolo *Secure Electronic Transactions* (SET))[2] tem sido objecto de intensa actividade, sendo esse mecanismo considerado decisivo para a consolidação da exploração comercial da Internet, permitindo a migração definitiva da comunidade financeira para a "rede das redes". Intimamente associada a esta iniciativa, surge, inevitavelmente, a omnipresente questão da Certificação que, todavia, não se limita ao domínio financeiro. A operação e a relação entre Autoridades de Certificação a vários níveis (organizacionais, nacionais ou mesmo supra–nacionais, de âmbito público ou privado — no contexto de uma Intranet, por exemplo —) é também um problema em aberto, sendo ainda em número reduzido (e de reponsabilidade/credibilidade limitada) as Autoridades de Certificação em actividade. Este cenário torna–se ainda mais

[1] Hoje reconhecidamente atribuído ao advento da *World Wide Web* (WWW).

[2] `http://www.visa.com/cgi-bin/vee/nt/ecomm/set/intro.html?2+0` disponibiliza a sua especificação mais recente.

complexo porquanto a legislação que regulamenta a produção, exportação e utilização de ferramentas criptográficas está longe de ser uniforme[3] (sendo, por vezes, inexistente ou desconhecida, como é o caso de Portugal). Apresentando como justificação as dificuldades acrescidas que a liberalização completa neste campo introduziria no combate ao crime, certos governos têm insistido em controlar a utilização/exportação de tecnologia do foro criptográfico[4]. Ao nível empresarial, mecanismos como as *Redes Privadas Virtuais* (VPNs[5]) e as *firewalls* são de uso cada vez mais vulgarizado, como forma de manter a privacidade das comunicações entre e dentro das organizações. No domínio pessoal, é de assinalar a utilização crescente de ferramentas de encriptação de ficheiros e de correio electrónico (como por exemplo o popular PGP), a preocupação em assegurar a confidencialidade de sessões remotas (materializada em aplicações como o *secure shell* (ssh)) e a previsível vulgarização dos *smartcards*. Em termos mais mediáticos, são frequentes as notícias de "buracos de segurança" em ferramentas de uso diário (como os *browsers* da Microsoft e da Netscape) e em sistemas operativos, bem como de ataques bem sucedidos a servidores organizacionais e fornecedores de acesso à Internet.

A área em que se insere este trabalho, a da comunicação segura em aplicações distribuídas, é assim de uma actualidade indiscutível e de uma importância cada vez maior, à medida que a operação em rede se consolida como modelo dominante em praticamente todas as áreas onde a Informática se "infiltra", fazendo corresponder aos seus enormes benefícios outra tanta dose de preocupação pelas dificuldades em assegurar a Privacidade da informação, alguma da qual governa as nossas vidas.

1.1 Motivação

A motivação original para a realização deste trabalho partiu da constatação da possibilidade de desenvolver um protocolo de gestão de réplicas de objectos num ambiente nómada[6], tomando como ponto de partida a comunicação multiponto[7] oferecida de base pelo *IP Multicast* [Dee89].

A utilização do *IP Multicast* colocava alguns desafios interessantes, nomeadamente a sua natureza *não–orientada–à–conexão*[8] e o facto de a entrega de pacotes se basear numa política de *melhor esforço*[9].

Contudo, havia ainda um importante obstáculo a vencer: dada a natureza aberta dos grupos *IP Multicast*, segundo a qual não há restrições à geração e auscultação do tráfego associado a qualquer grupo, tornou–se evidente a necessidade de providenciar mecanismos de protecção desse tráfego.

Nesta linha, foi levado a cabo um estudo inicial [Ama96], a fim de averiguar a viabilidade da aplicação de esquemas de segurança (Privacidade, Autenticação, etc.) ao *IP*

[3] A este propósito, `http://cwis.kub.nl/%7Efrw/people/koops/lawsurvy.html` oferece um estudo bastante completo que cobre a legislação existente num conjunto diversificado de países.

[4] Os Estados Unidos constituem um exemplo paradigmático deste tipo de atitude. Todavia, recentemente, o debate sobre a questão foi relançado, perspectivando–se evoluções a curto prazo no sentido de uma maior liberalização neste sector.

[5] Do inglês *Virtual Private Networks*.

[6] Do inglês *mobile*.

[7] Do inglês *multicast*.

[8] Do inglês *connectionless*.

[9] Do inglês *best effort*.

Multicast, do qual resultou uma primeira versão de um protocolo de comunicação segura em grupo, embora com bastantes limitações. Nesse trabalho, concluiu-se da necessidade de desenvolver um protocolo mais genérico, que contemplasse, de base, segurança sobre a tradicional comunicação ponto–a–ponto[10] e que fornecesse os alicerces para as extensões multiponto.

O desenho e a implementação desse protocolo, que se convencionou designar de *Stateless Secure Sockets Layer* (S3L), suscitaram questões em torno das quais acabou por se centrar a presente tese e que justificaram o abandono do objectivo inicial relativo à gestão de réplicas.

Assim sendo, esta dissertação deve ser encarada como uma descrição do processo de investigação, concepção, implementação e análise de desempenho do S3L.

1.2 Apresentação

Basicamente, o S3L é uma Interface de Programação de Aplicações (API[11]) que, em conjunto com alguns demónios e utilitários de apoio, fornece serviços de segurança imediatamente acima da interface de programação dos *sockets* de Berkeley [LMKQ89, LFJL86], mantendo com estes grandes semelhanças sintácticas e semânticas nas operações de escrita e leitura. Tais semelhanças contribuem decisivamente para a facilidade com que se faz a migração de aplicações do "modelo de Berkeley" para o "modelo S3L".

As qualidades de serviço oferecidas pelo S3L compreendem a Privacidade, a Autenticação (incluindo a Não–Repudiação), a Integridade, a Sequenciação (contra *ataques–de–repetição*) e a Compressão dos dados trocados, oferecendo–se ainda, na variante multiponto, Controle de Acesso na fase de junção a um grupo *IP Multicast* seguro.

O S3L distingue–se de outras abordagens afins[12] em dois aspectos fundamentais:

- pelo seu carácter *stateless*;
- pelo seu suporte explícito à comunicaçao segura em grupo sobre *IP Multicast*.

O carácter *stateless* atribuído ao S3L resulta do seu protocolo auxiliar *Secure Key Exchange* (Skeyx), responsável por definir um estado mínimo comum entre as partes intervenientes numa comunicação S3L. Esta aparente contradição explica–se facilmente: toda a segurança do S3L assenta, em última instância, no acordo de Diffie–Hellman; logo, a fim de evitar que a transmissão de toda e qualquer mensagem S3L seja precedida da troca de chaves públicas de Diffie–Hellman, executa–se o protocolo Skeyx uma única vez, entre as duas entidades comunicantes. O Skeyx é executado transparentemente, *on–the–fly*, durante a primeira comunicação S3L entre duas entidades. A chave acordada é preservada, por cada entidade, numa *cache* própria, a fim de ser eventualmente reutilizada, numa comunicação S3L futura, sem necessidade de repetir o processo de acordo. Nessa *cache*, são também preservados outros elementos resultantes desta negociação prévia, como por exemplo o tipo de algoritmos criptográficos suportados pela entidade remota.

As chaves públicas de Diffie–Hellman trocadas pelo Skeyx não necessitam de ser certificadas pelo que o Skeyx consiste, basicamente, num protocolo do tipo *desafio–resposta*,

[10] Do inglês *unicast*.
[11] Do inglês *Application Programming Interface*.
[12] Analisadas no capítulo 3.

onde cada interveniente tem que provar que possui a chave privada correspondente à chave pública que envia ao outro. Ataques do tipo *homem–no–meio* durante este processo são evitados através da utilização de assinaturas digitais baseadas na aplicação de chaves públicas RSA certificadas. Todavia, o Skeyx foi concebido de forma a que uma entidade dispense o conhecimento prévio da chave pública RSA da outra, sendo a única restrição a de que a cadeia de certificação de ambas tenha pelo menos um ponto comum.

A vantagem imediata de um acordo de Diffie–Hellman entre duas entidades comunicantes é a obtenção de uma chave secreta comum (a chave acordada) que, doravante, poderá ser usada na protecção das mensagens trocadas entre ambas. As mensagens S3L podem assim circular de uma forma expedita, sem necessidade de estabelecimento prévio de sessões seguras dedicadas, sendo possível a escolha unilateral, por parte de cada entidade, de items de segurança específicos para cada mensagem (como, por exemplo, chaves, algoritmos, números de sequência, códigos de autenticação, etc.) que viajam lado a lado com os dados do utilizador.

O S3L oferece funcionalidades ponto–a–ponto e multiponto. Basicamente, estas funcionalidades resumem–se ao encapsulamento dos dados do utilizador num formato próprio e à posterior invocação de primitivas de leitura e escrita sobre os *sockets* de Berkeley tradicionais. É ainda possível a assemblagem de mensagens S3L sem entrega directa à rede, constituindo–se dessa forma em contentores seguros de informação, destinados a armazenamento ou a formas alternativas de transmissão.

Assente em serviços criptográficos fundamentais fornecidos pelo *Secure Development Environment* (SecuDE) [Sch95a], o S3L é conceptualmente baseado no (mas não é compatível nem interoperável com o) *Simple Key Management for Internet Protocols* (SKIP) [AP95, AMP95c] da Sun, que fornece segurança ao nível da Camada de Rede. Todavia, o S3L transporta (e estende) os conceitos do SKIP para a Camada de Aplicação, na vizinhança da interface de programação com *sockets*, representando uma forma mais flexível de implementar segurança. Este ganho em flexibilidade deriva da não imposição de um modelo global (embora reconhecidamente mais transparente) de segurança, como acontece com o SKIP.

As diferenças de posicionamento em termos arquitecturais, entre o SKIP e o S3L, conduzem, inevitavelmente, a diferenças de fundo entre ambos. Basta dizer que, no SKIP a segurança é estabelecida em função do sistema (ou melhor, do seu endereço IP) na sua globalidade. As listas de controle de acesso do SKIP definem regras aplicáveis a todo o tráfego de e para uma máquina. No S3L, o sujeito da segurança é a entidade[13], não havendo limite teórico ao número de entidades por máquina. Tal permite a gestão individualizada de qualidades de serviço como a Privacidade, Autenticação, Integridade, Compressão, etc.

Outros aspectos, como por exemplo a Sequenciação de mensagens, corroboram esta distinção que se pretende vincar entre o SKIP e o S3L. Concretizando, o SKIP é bastante tolerante, lidando com granularidades demasiado altas para serem efectivas no combate a *ataques–de–repetição*. O S3L, por seu turno, enfrenta a velha questão da sincronização de relógios, redefinindo e parametrizando o mecanismo do SKIP. Dispensando a sincronização de relógios, as soluções preconizadas pelo S3L acabam por abrir janelas de ataque mas constituem, como se poderá verificar, um risco bem calculado.

Na variante multiponto as diferenças não se limitam ao mecanismo de Sequenciação.

[13] Ver definição na secção 4.6.

As listas de controle de acesso lidam, como seria de esperar, com entidades (e não com endereços IP). O tempo de vida de um grupo é definido de forma independente do tempo de vida da *chave–do–grupo*, sendo o cálculo desses tempos de vida independente da sincronização de relógios em todos os elementos do grupo. A (re)junção ao grupo constitui sempre um processo automático, realizado *por necessidade*, *on–the–fly*, e quando ocorre em face da obsolescência da *chave–do–grupo* prevê–se a possibilidade de usar, temporariamente, essa chave já obsoleta, enquanto a rejunção não se concretiza. O processo de criação de um novo grupo procura evitar a sobreposição de dois grupos diferentes no mesmo endereço *IP Multicast*.

Acrescente–se que o S3L não faz assunções relativamente à segurança dos protocolos de base em que assenta (pilha IP/TCP) uma vez que toda a informação que lhe é relevante (e que basicamente partilha, com os dados do utilizador, o campo de dados dos pacotes desses protocolos) é sujeita a mecanismos próprios de Autenticação.

1.3 Estrutura da Dissertação

O resto deste documento divide–se, informalmente, em três partes distintas. A primeira contempla o capítulo 2 e o capítulo 3 nos quais se faz, genericamente, o enquadramento da dissertação na sua área de investigação específica. A segunda parte compreende os capítulos 4, 5, 6, 7 e 8 ao longo dos quais se processa a concepção, o desenvolvimento e a análise dos protocolos criados no âmbito do S3L. A terceira parte consiste nos apêndices A e B, que contêm indicações essenciais à instalação e utilização do S3L. Concretamente:

- no capítulo 2 sintetizam–se as noções fundamentais do ramo do conhecimento (Criptografia) em que se insere o presente trabalho. O capítulo começa por definir algumas qualidades de serviço básicas (Confidencialidade, Autenticação, etc.) e respectivos ataques, no contexto da Segurança em Sistemas Distribuídos. De seguida descrevem–se os blocos construtores de qualquer sistema criptográfico moderno: sistemas simétricos, assimétricos e híbridos, métodos de Autenticação, de Sequenciação e de gestão de chaves;

- o capítulo 3 apresenta as perspectivas mais correntes do posicionamento das funções de encriptação na arquitectura do Modelo de Comunicações e do Sistema Operativo. Nomeadamente, são discutidas as variantes Ligação–a–Ligação e Fim–a–Fim. A atenção acaba por se concentrar em algumas das propostas Fim–a–Fim mais representativas, sendo efectuada uma análise comparativa resumida das abordagens analisadas. O capítulo termina procurando justificar a oportunidade do S3L bem como a escolha do SKIP e do SecuDE como pontos de partida teórico e prático, respectivamente, para o seu desenvolvimento;

- a descrição do protocolo auxiliar Skeyx constitui o foco do capítulo 4. Nele se justifica a necessidade deste protocolo para em seguida se apresentar, com detalhe, o seu modelo de operação. O serviço de reencaminhamento de pedidos Skeyx bem como a infra–estrutura de gestão de chaves em *cache* são também objecto de atenção. O capítulo prossegue com uma discussão sobre o conceito de entidade no contexto do S3L e suas consequências na eventualidade da sua replicação. Finalmente, o

algoritmo de actualização de *caches* é posto à prova numa situação específica com elevado grau de concorrência e imprevisibilidade;

- o capítulo 5 introduz a variante ponto–a–ponto do S3L. A sua relação com o SKIP é clarificada através do confronto entre a estrutura das mensagens (e seus mecanismos de sequenciação) de ambos. De seguida introduz–se a API do S3L ponto–a–ponto, sugere–se uma metodologia de programação e estabelecem–se relações entre as "primitivas S3L" e as primitivas de Berkeley. O capítulo termina com a descrição do mecanismo de execuções implícitas do protocolo Skeyx, desencadeadas *on–the–fly* pela execução das "primitivas S3L";

- o capítulo 6 encerra a apresentação do protocolo S3L com a discussão das suas extensões multiponto que lhe permitem operar sobre o *IP Multicast*. As extensões multiponto do SKIP são em primeiro lugar objecto de análise para de seguida se introduzir a proposta específica do S3L. O processo de junção a um grupo S3L é então dissecado: define–se o papel do "dono do grupo", estende–se a função do serviço de reencaminhamento introduzido no capítulo 4, apresenta–se a estrutura dos pedidos (e respostas) de junção e descreve–se o mecanismo de controle de acesso ao grupo. De seguida apresenta–se a estrutura das mensagens multiponto geradas pelos membros e justifica–se a abordagem seguida na gestão de chaves de grupo e na sequenciação de mensagens multiponto. O capítulo prossegue apresentando as extensões multiponto à API do S3L e definindo um modelo de programação com essas extensões. Por fim, a viabilidade de detecção de grupos pré–activos num determinado endereço *IP Multicast* é discutida;

- a análise de desempenho do S3L é efectuada no capítulo 7 onde se descreve a metodologia seguida e os resultados obtidos nos vários testes realizados sobre ambas as modalidades ponto–a–ponto e multiponto do S3L;

- no capítulo 8 apresentam–se as conclusões do trabalho e apontam–se linhas de desenvolvimento futuro;

- os apêndices A e B são de leitura obrigatória para os utilizadores do S3L. No apêndice A descrevem–se, com pormenor, todos os procedimentos necessários à instalação e configuração do S3L. No apêndice B documentam–se todos os utilitários e todas as funções da API do S3L sendo fornecidos exemplos de código comentados.

1.4 Convenções

Esta dissertação assume determinadas convenções relativas à tradução de termos ou expressões para a língua portuguesa, tipos de letra usados e abreviaturas.

É feito um esforço no sentido de traduzir a maioria dos termos e designações técnicas. A qualidade dessa tradução reflecte necessariamente as limitações do autor como tradutor. Assim, na ausência de uma tradução satisfatória, será mantido o termo ou expressão na sua língua de origem. A primeira ocorrência de uma tradução será sempre acompanhada de uma nota de rodapé assinalando o termo ou expressão original.

Quanto aos tipos de letra usados, o seu contexto de aplicação é o seguinte:

- *itálico*: para termos na sua língua original ou para conferir ênfase a determinados termos ou frases de maior relevância;
- `letra de máquina de espaçamento fixo`: aplicada à representação de código, nomes e conteúdo de ficheiros, comandos e outros "objectos" de natureza informática.

A introdução de abreviaturas é feita pela apresentação das palavras que estão na sua origem, por extenso, seguidas da abreviatura, entre parênteses. Por vezes, opta–se pela utilização de abreviaturas correspondentes a termos na língua original, ainda que se tenham apresentado as suas traduções para a língua portuguesa, com o fim de tornar a notação mais clara e compatível com as fontes seguidas.

Capítulo 2

Elementos de Segurança

São dois os pilares teóricos em que assenta a presente dissertação: Segurança e *IP Multicast*. Neste capítulo pretende–se apenas condensar elementos teóricos fundamentais de Segurança em Sistemas Distribuídos. Dada a vastidão actual desta área, apenas serão cobertos os tópicos directamente relacionados com o trabalho. Relativamente ao *IP Multicast*, este será objecto de atenção posteriormente (capítulo 6), durante a descrição das extensões ao S3L que suportam grupos *IP Multicast* seguros.

2.1 Segurança em Sistemas Distribuídos

No contexto dos Sistemas Distribuídos modernos, o conceito de Segurança compreende algumas qualidades de serviço[1] básicas, entre as quais [Sta95]:

- *Confidencialidade/Privacidade*: assegura o acesso (de leitura) a determinado recurso ou mensagem apenas por entidades devidamente autorizadas;
- *Autenticação*: implica a identificação correcta do originador (*Autenticação da Origem*) e/ou do destinatário (*Autenticação do Destino*) de uma mensagem;
- *Integridade*: assegura que qualquer modificação (*via* acesso de escrita) no estado de um recurso ou na estrutura de uma mensagem é feita apenas por entidades com autorização adequada;
- *Não–Repudiação*: corresponde à capacidade de provar que um determinado emissor e/ou um determinado receptor de uma mensagem estão efectivamente comprometidos na sua troca, ainda que neguem tal facto;
- *Controle de Acesso*: implica a possibilidade de colocar barreiras selectivas no acesso (incluindo na própria natureza desse acesso) a determinado recurso (serviço ou dados);
- *Disponibilidade*: assegura a manutenção do acesso a um determinado recurso e a sua operação contínua, inclusivamente quando sujeito a ataques que visem a sua inoperacionalidade parcial ou total.

[1] Do inglês *Qualities–of–Service* (QoS).

É precisamente sobre estas qualidades de serviço que incide a maioria dos ataques feitos à Segurança de um Sistema Distribuído: *Intercepção* (ataque à *Confidencialidade*), *Interrupção* (ataque à *Disponibilidade*), *Modificação* (ataque à *Integridade*) e *Fabricação/Falsificação*[2] (ataque à *Autenticidade*).

A *Intercepção* enquadra–se numa categoria mais vasta de ataques, designados por *ataques passivos*, que visam fundamentalmente a observação da informação (por exemplo, durante o acto da sua transmissão) e sua possível divulgação. Uma vez que não modifica a informação, este género de ataques é difícil de detectar, colocando-se particular ênfase na sua prevenção.

Por oposição, os *ataques activos* compreendem a *Interrupção*, a *Modificação* e a *Fabricação* e apoiam-se em técnicas como:

- *Personificação/Simulação*[3]: em que uma entidade consegue criar a ilusão de que é outra entidade;
- *Repetição*: em que ocorre a retransmissão de uma mensagem (capturada através de um ataque passivo), como forma de, por exemplo, viabilizar um ataque de *Personificação*;
- *Adulteração*: em que se modifica o corpo de uma mensagem ou se introduz um atraso deliberado na sua propagação ou, inclusivamente, se reordenam mensagens a fim de obter efeitos que não os originalmente pretendidos na transmissão da mensagem original;
- *Negação do Serviço*[4]: em que se procura evitar que os potenciais clientes de um determinado serviço a ele tenham acesso, quer gerando um grande número de falsos pedidos, quer desviando os pedidos originais, quer destruindo a própria infra–estrutura fornecedora do serviço.

Os *ataques activos* são de difícil prevenção, o que leva a que as contra–medidas se baseiem, maioritariamente, em procedimentos de recuperação, uma vez detectado o ataque. Adicionalmente, os ataques ocorrem frequentemente em forma combinada, a fim de se revelarem mais eficazes, o que dificulta ainda mais o seu combate.

Obviamente, estas classificações (quer das qualidades de serviço, quer dos ataques) relativas à Segurança de um Sistema Distribuído, estão longe de ser exaustivas, já que não colocámos nos objectivos da presente tese a elaboração de classificações desse tipo. Fundamentalmente, elas servem para sensibilizar o leitor para a necessidade de se encontrarem mecanismos apropriados à implementação de uma política de Segurança capaz de imunizar uma comunidade, (seja ela de utilizadores, máquinas, aplicações, objectos ou outras entidades), relativamente a interferências indesejadas de agentes internos ou externos.

É precisamente neste contexto que se justifica um breve resumo de algumas das técnicas e conceitos em que assenta a Criptografia moderna. A apresentação que se segue reflecte, necessariamente, alguma da investigação na área feita pelo autor, uma vez que a maioria dos tópicos abordados encontram concretização na parte prática da presente dissertação, como oportunamente se verificará.

[2] Do inglês *Fabrication*.
[3] Do inglês *Impersonation/Masquerade*.
[4] Do inglês *Denial of Service*.

2.2 Criptografia Simétrica

A essência da criptografia simétrica (também designada por "convencional" ou "de chave secreta") reside no uso de uma chave comum ao processo de encriptação e desencriptação[5]. Enquanto essa chave não for comprometida, existirá um canal seguro entre os produtores e os consumidores de informação que partilham a chave. O conhecimento do algoritmo de encriptação/desencriptação e a posse da mensagem cifrada nunca deverão ser suficientes para determinar a mensagem original.

A comunicação entre duas entidades — $\mathcal{A}$ e $\mathcal{B}$ — que protegem as mensagens trocadas usando criptografia simétrica segue, em geral, um protocolo que, na direcção $\mathcal{A} \rightarrow \mathcal{B}$, poderá enunciar–se da seguinte forma:

1. $\mathcal{A}$ e $\mathcal{B}$ concordam previamente num algoritmo simétrico e numa chave K comuns;
2. $\mathcal{A}$ efectua $C = E_K(P)$ e envia C a $\mathcal{B}$;
3. $\mathcal{B}$ efectua $P = D_K(C)$, recuperando P;

em que $E \equiv$ encriptação, $D \equiv$ desencriptação, $K \equiv$ chave secreta, $P \equiv$ mensagem original (ou *texto plano*[6]) e $C \equiv$ mensagem cifrada (ou *texto cifrado*[7]).

Os métodos criptográficos de chave secreta sofrem, porém, de alguns problemas fundamentais [Sch96]:

- a chave secreta tem de ser distribuída antecipadamente a todos os potenciais intervenientes numa comunicação segura. Essa distribuição está, naturalmente, sujeita a ataques que, se bem sucedidos, comprometerão irremediavelmente qualquer comunicação futura;
- como toda a segurança do processo reside na ocultação de uma única chave, se essa chave é comprometida, todo o processo é comprometido;
- mesmo que se opte pelo uso de chaves separadas para qualquer par de entidades comunicantes, tal pode não ser viável pelo elevado número de chaves possivelmente necessárias: $n(n-1)/2$ chaves para n entidades.

2.2.1 Data Encryption Standard

O *Data Encryption Standard* (DES) [oS77] é o algoritmo de criptografia simétrica de uso mais vulgarizado. Datando já de 1977 a sua adopção pela organização norte-americana *National Bureau of Standards*[8], o DES baseia–se numa chave de 56 bits e num algoritmo que opera com blocos de 64 bits[9].

O DES goza de vários modos de operação (definidos originalmente em [oS80]), consoante o tipo de aplicação alvo, que lhe permitem agir sobre mensagens de dimensões

[5] O caso mais genérico é aquele em que, apesar de as chaves não serem iguais, qualquer uma delas pode obter–se facilmente com base no conhecimento da outra.

[6] Do inglês *plaintext*.

[7] Do inglês *ciphertext*.

[8] Actualmente *National Institute of Standards and Technology* (NIST).

[9] Diz-se, então, que o DES é uma *cifra de bloco* (do inglês *block cipher*).

arbitrárias, ao mesmo tempo que lhe conferem resistência a alguns tipos de ataques. No contexto do presente trabalho, apenas os seguintes modos de operação do DES são relevantes[10]:

- *Electronic Codebook* (ECB): cada bloco de 64 bits é cifrado individualmente e de uma forma independente dos outros blocos (embora com a mesma chave), produzindo–se outro bloco de 64 bits. O mesmo bloco de entrada origina sempre o mesmo bloco à saída, para uma dada chave, provindo daí a designação *Electronic Codebook*, uma vez que, teoricamente, seria possível manter uma tabela que fizesse a correspondência entre um bloco não cifrado e o respectivo bloco cifrado. O modo ECB deve ser usado preferencialmente na encriptação de pequenas mensagens, onde a probabilidade de repetição de blocos é menor. Assim, se por acaso se descobrir a correspondência entre um determinado bloco não cifrado e o respectivo bloco cifrado, então a análise de uma mensagem cifrada, onde esse bloco ocorra frequentemente — ou mesmo sistematicamente (no cabeçalho, por exemplo) —, poderá ficar substancialmente facilitada [Sch96];

- *Cipher Block Chaining* (CBC): a chave mantém-se ao longo de todo o processo. Porém, a encriptação de um determinado bloco depende do resultado da encriptação do bloco anterior. Assim, para cada bloco é calculado o *ou lógico exclusivo* (*XOR*) com o bloco anterior cifrado, constituindo o bloco resultante uma nova entrada do algoritmo. Para o primeiro bloco, o bloco anterior deverá consistir num bloco gerado aleatoriamente (o Vector de Inicialização (IV)[11], que deverá ser conhecido por ambas as partes), a fim de forçar sempre a produção de resultados diferentes para os mesmos blocos de entrada (ou então a construção de uma tabela semelhante à referida no modo ECB poderia tornar–se viável).

Apesar de o DES ser bastante popular, a sua aceitação nunca foi completamente pacífica, devido, por um lado, ao receio de que a dimensão (actualmente considerada reduzida) das chaves se traduza numa vulnerabilidade a ataques do tipo *força–bruta*[12] (cuja viabilidade foi demonstrada recentemente em [Wie93] e posteriormente confirmada[13] no âmbito de uma série de desafios lançados pela firma americana RSA[14]), por outro lado, ao facto de certos detalhes da estrutura interna do DES terem vindo a ser ocultados, o que levantou suspeições relativamente à segurança efectiva do algoritmo.

3DES

Uma alternativa interessante à aplicação simples do DES (e que no fundo constitui uma forma de prolongar a vida útil do algoritmo) baseia–se num método de *encriptação múltipla*, segundo o qual se aplica o mesmo algoritmo um determinado número de vezes, fazendo variar a chave em cada aplicação. O 3DES (ou *triplo DES*) é uma variante do DES que usa duas (ou mesmo três) chaves diferentes em modo *Encrypt–Decrypt–Encrypt* (EDE).

[10] Porque são precisamente os modos oferecidos pelo pacote criptográfico usado (ver secção 3.3.6).

[11] Do inglês *Initialization Vector*.

[12] Do inglês *brute–force–atacks*. Neste tipo de ataques, é feita uma procura exaustiva da chave, o que só é viável se o espaço das chaves for de dimensões "reduzidas".

[13] `http://www.frii.com/~rcv/deschall.htm`

[14] `http://www.rsa.com/rsalabs/97challenge/`

O modo EDE (originalmente sugerido por [Tuc79]) consiste em cifrar a mensagem com a primeira chave, decifrar com uma segunda e voltar a cifrar com a primeira (ou com uma terceira[15]). Uma discussão mais completa das técnicas de *encriptação múltipla* (incluindo possíveis ataques) pode ser encontrada em [Sch96].

2.2.2 International Data Encryption Algorithm

O *International Data Encryption Algorithm* (IDEA) é um algoritmo simétrico recente [LM90], afigurando–se como um sério candidato à disputa da hegemonia do DES.

À semelhança do DES, o IDEA é também um algoritmo que consome e produz blocos de 64 bits, usando, porém, uma chave de 128 bits, o que, conjuntamente com a própria estrutura interna do algoritmo, contribui para a boa imagem (em termos de Segurança) de que o IDEA goza actualmente. O IDEA também pode basear–se nos mesmos modos de operação (ECB, CBC, CFB e OFB) definidos para o DES.

As implementações em *software* do IDEA chegam a ser duas vezes mais rápidas que as implementações do DES [Sch96]. No entanto, provavelmente, o principal factor de sucesso do IDEA reside na sua utilização pela aplicação criptográfica de uso mais vulgarizado: o *Pretty Good Privacy* (PGP) [Zim95].

2.3 Criptografia Assimétrica

A criptografia assimétrica (ou "de chave pública") é conhecida desde 1976[16] [DH76], constituíndo um dos marcos mais importantes na evolução da ciência da Criptografia.

Este paradigma baseia-se no uso de um par de chaves: uma de encriptação e outra de desencriptação[17]. Uma delas é tornada pública e a outra deverá permanecer privada. Acresce que deverá ser computacionalmente impraticável obter a chave privada, conhecida a respectiva chave pública.

A comunicação entre duas entidades $\mathcal{A}$ e $\mathcal{B}$ que protegem as mensagens trocadas usando criptografia assimétrica poderá basear–se no seguinte protocolo, na direcção $\mathcal{A} \rightarrow \mathcal{B}$:

1. $\mathcal{A}$ e $\mathcal{B}$ concordam previamente num algoritmo de chave pública;
2. $\mathcal{A}$ toma conhecimento de K_{pub_B} e assegura–se de que essa chave é de $\mathcal{B}$;
3. $\mathcal{A}$ efectua $C = E_{K_{pub_B}}(P)$ e envia C a $\mathcal{B}$;
4. $\mathcal{B}$ efectua $P = D_{K_{prv_B}}(C)$, recuperando P;

em que $K_{pub_B} \equiv$ chave pública de $\mathcal{B}$ e $K_{prv_B} \equiv$ chave privada de $\mathcal{B}$.

[15] Note–se que se todas as chaves forem iguais, o 3DES em modo EDE equivale a uma aplicação simples do DES, o que encontra justificação na necessidade de assegurar retro–compatibilidade com implementações antigas.

[16] A organização governamental norte–americana *National Security Agency* (NSA) reclama esse conhecimento desde 1966, sem que, contudo, tivesse fornecido provas públicas desse facto.

[17] Alguns esquemas (como por exemplo o RSA) permitem que uma chave possa ser usada em ambos os processos: *o que uma chave cifra, a outra chave decifra.* Na secção 2.5 demonstra-se como esta propriedade torna possível uma verificação expedita da Autenticidade das mensagens trocadas.

Desta forma, garante-se um canal seguro sem ser necessário estabelecê-lo à partida, ou seja, resolve–se um problema fundamental dos sistemas simétricos: a distribuição de chaves.

São pelo menos três as aplicações possíveis dos algoritmos de chave pública: Privacidade, Autenticação e Troca de Chaves. Contudo, nem todos os algoritmos de chave pública são capazes de suportar em simultâneo todas estas aplicações[18]. Adicionalmente, os esquemas de criptografia assimétrica actuais traduzem–se, geralmente, num grande esforço computacional, não sendo por isso frequente a sua aplicação como forma de assegurar Privacidade.

Do conjunto de algoritmos de chave pública actualmente disponíveis, a concretização prática da presente dissertação faz uso apenas do *Rivest–Shamir–Adleman* (RSA) e do *Diffie–Hellman* (DH).

2.3.1 Rivest–Shamir–Adleman

O *Rivest–Shamir–Adleman* (RSA) [RSA78] está para a criptografia assimétrica assim como o DES está para a criptografia simétrica: ambos constituem os algoritmos mais usados e estudados nas suas respectivas áreas, sendo considerados seguros até à data. À semelhança do IDEA, a aceitação e a disseminação do RSA foi também favorecida pela sua inclusão no PGP.

A segurança do RSA reside na dificuldade em factorizar números grandes: a ideia base é a de que é "fácil" encontrar dois números primos grandes, mas é impraticável (em termos computacionais) factorizar o seu produto.

Para além da abordagem da factorização, é possível conceber outras categorias de ataques ao RSA, por exemplo dirigidos à sua concretização e não ao algoritmo em si. [Sch96] descreve uma série de ataques desse género. A título meramente ilustrativo, refira–se o ataque clássico do tipo *texto–plano–escolhido*[19], cuja aplicação, aliás, é extensível a outros algoritmos para além do RSA: seja $C = E_{K_{pub}}(P)$ (em que $C \equiv$ mensagem cifrada, $P \equiv$ mensagem plana, $E \equiv$ encriptação e $K_{pub} \equiv$ uma chave pública). Deste modo, se há no máximo n Ps diferentes e conhecidos, bastará cifrar todos os Ps possíveis com K_{pub} e comparar o resultado com C, para se determinar, assim, qual o P que corresponde a C. A viabilidade deste ataque depende, obviamente, de uma dimensão favorável n do espaço das mensagens planas P. Se n for pequeno (*i.e.*, se n for compatível com os meios de cálculo disponíveis), então torna–se evidente que nem sempre é necessário recorrer à força bruta ou à factorização para derrubar esquemas de segurança baseados em algoritmos aparentemente tão robustos como o RSA.

O RSA é um algoritmo através do qual é possível assegurar Privacidade e realizar Autenticação e Troca de Chaves. Contudo, o RSA é particularmente vocacionado para estas duas últimas vertentes, uma vez que, inclusivamente em *hardware*, o RSA é aproximadamente 1000 vezes mais lento que o DES [Sch96].

[18] Para uma descrição praticamente exaustiva dos algoritmos de chave pública existentes, recomenda–se a consulta de [Sch96].

[19] Do inglês *chosen–plaintext*.

2.3.2 Diffie–Hellman

O algoritmo de *Diffie–Hellman*[20] (DH) foi o primeiro algoritmo de chave–pública a ser publicado [DH76].

O seu propósito é apenas o de permitir a troca de uma chave entre duas entidades[21], para o que é suficiente que as chaves pública e privada das entidades envolvidas sejam geradas com base em certos parâmetros especiais comuns (conhecidos antecipadamente por ambas as entidades) e que uma entidade faça chegar à outra, de uma forma fiável, a sua chave pública.

A chave trocada[22] (ou uma outra dela derivada) poderá então ser empregue por um algoritmo de encriptação (em geral simétrico) do tráfego entre as duas entidades.

A segurança do algoritmo DH deriva da dificuldade em calcular logaritmos discretos num campo finito[23]. Relativamente à sua concretização, um ataque do tipo *homem–no–meio*[24] é evidentemente viável se as mensagens envolvidas na troca das chaves não forem assinadas pelos seus originadores. Num ataque deste género, supõe-se que exista uma terceira entidade que tem a capacidade de interceptar uma mensagem e modificá–la sem que o destinatário seja capaz de detectar esse facto. Um dos objectivos pode ser a Personificação, levando a outra entidade a interactuar com um impostor.

2.4 Sistemas Híbridos

Tradicionalmente, os sistemas de criptografia pública são usados para trocar chaves simétricas de sessão e não tanto para garantir a Privacidade das mensagens. Nesta perspectiva, os sistemas assimétricos são vistos como ferramentas que tornam mais seguras as trocas de chaves simétricas. Fundamentalmente, a razão para tal procedimento assenta no grande esforço computacional em que se traduzem os algoritmos assimétricos. Adicionalmente, já vimos que nem todos os algoritmos assimétricos suportam operações de encriptação.

Num sistema híbrido em que a entidade $\mathcal{A}$ usa criptografia assimétrica para enviar à entidade $\mathcal{B}$ uma chave de sessão simétrica, um protocolo básico na direcção $\mathcal{A} \rightarrow \mathcal{B}$ poderá ser:

1. $\mathcal{A}$ e $\mathcal{B}$ concordam previamente num algoritmo de chave pública e noutro de chave secreta;
2. $\mathcal{A}$ toma conhecimento de K_{pub_B} (e, idealmente, certifica–se de que essa chave é realmente de $\mathcal{B}$); decide ainda que a chave secreta é K;
3. $\mathcal{A}$ efectua $C = E_{K_{pub_B}}(K)$ e envia C a $\mathcal{B}$;
4. $\mathcal{B}$ efectua $K = D_{K_{prv_B}}(C)$, tomando assim conhecimento de K;
5. doravante, $\mathcal{A}$ e $\mathcal{B}$ cifram com K o tráfego que trocarem entre si.

[20] Que rigorosamente deve ser designado por "Troca de Chaves de *Diffie–Hellman*".

[21] Não suportando directamente operações de encriptação, como acontece, por exemplo, com o RSA.

[22] Neste contexto o termo *acordada* talvez fosse mais adequado, visto que, na realidade, a chave em si não é trocada, mas sim gerada por ambas as entidades com base em informação de domínio público e também em informação privada.

[23] Assunto abordado por qualquer bom livro de Teoria de Números, como por exemplo [Lev90].

[24] Do inglês *man–in–the–middle*.

Porém, este protocolo está longe de ser satisfatório[25], já que, apesar de fornecer Confidencialidade, carece de um mecanismo de Autenticação, tópico a abordar de seguida.

2.5 Autenticação

No contexto de uma troca segura de mensagens, Autenticação refere–se ao processo através do qual o receptor pode concluir com segurança que o emissor é quem afirma ser e que a mensagem recebida não foi adulterada em trânsito. Adicionalmente, poderá ser necessário assegurar que o emissor não possa negar a transmissão de uma mensagem e/ou o receptor não possa negar a sua recepção. Resumindo, a Autenticação pretende combater pelo menos ataques de Personificação, Modificação (podendo estes últimos incidir sobre o conteúdo, sequência ou temporização das mensagens) e Repudiação.

Uma primeira abordagem (embora ingénua, como se constatará de seguida) à Autenticação consiste em conceber a mensagem cifrada como o próprio elemento autenticador. Desta forma, se o destinatário consegue decifrar uma mensagem recebida, acredita que essa mensagem é proveniente de um originador legítimo (*i.e.*, que possui a chave correcta). A fraqueza deste esquema é evidente: se uma dada mensagem puder ser uma sequência arbitrária de bits, então é impossível distinguir uma mensagem original de uma mensagem forjada ou adulterada em trânsito.

A solução deste problema passa pela adição de um bloco de controle à mensagem, cuja verificação, no destinatário, confirmará a pretensa origem. Actualmente, esses blocos de controle são produzidos maioritariamente pela aplicação de Funções de Hashing Unidireccionais[26], eventualmente combinadas com encriptação (ver secção 2.5.2). Relativamente ao problema da Repudiação, este pode ser resolvido com o auxílio de Assinaturas Digitais.

2.5.1 Funções de Hashing Unidireccionais

Uma Função de Hashing Unidireccional[27] H goza das seguintes propriedades básicas [Sch96]:

- dada uma entrada M, de dimensão arbitrária, a saída $h = H(M)$ tem uma dimensão fixa;
- a determinação de h com base em M é expedita;
- é difícil determinar M com base em h; ou seja, na prática, H não é reversível (propriedade da Unidireccionalidade);
- para uma determinada mensagem M, é difícil encontrar outra mensagem M' tal que $H(M) = H(M')$;
- é difícil encontrar duas mensagens aleatórias M e M' tal que $H(M) = H(M')$; *i.e.*, H é resistente a colisões no espaço das imagens.

[25] É evidente que é vulnerável a um ataque do tipo *homem–no–meio*: a mensagem C pode ser interceptada e substituída sem que o destinatário $\mathcal{B}$ consiga detectar esse facto.

[26] Do inglês *One–Way Hash Functions*. Dada a dificuldade em traduzir o termo *Hash*, optou–se pela manutenção dos termos dessa família na sua língua original.

[27] [Pre93] constitui uma referência bastante completa e actual do tema.

O cumprimento deficiente do requisito da resistência à colisão de imagens constitui uma porta aberta a uma categoria particular de ataques denominada *ataques–de–aniversário*[28]. Um ataque deste género, aplicado a uma função H que produz um *hash* de m bits, resultaria na determinação de duas mensagens aleatórias, M e M', tal que $H(M) = H(M')$, sendo de 50% a probabilidade de sucesso do ataque ao fim de $2^{m/2}$ tentativas. Uma forma de minimizar o perigo de colisão consiste, por exemplo, no incremento do número de bits do *hash* produzido pela função.

São várias as aplicações das Funções de Hashing Unidireccionais à Autenticação de mensagens (a qual ocorre frequentemente combinada com mecanismos que asseguram a Privacidade das mesmas[29]). Basicamente, todas elas envolvem o recálculo do *hash* da mensagem pelo destinatário e sua comparação com o *hash* que acompanha a mensagem. Se coincidirem, o destinatário conclui que não só a mensagem original foi preservada *en route* (teste à Integridade), como também é proveniente de um determinado originador conhecido (Autenticação).

MD5

O *Message Digest #5* (MD5) [Riv92c] é um exemplo concreto de um algoritmo baseado numa Função de Hashing Unidireccional, sendo, provavelmente, o mais conhecido e usado dos algoritmos desta categoria. Trata–se de uma evolução do MD4 [Riv92b], este mais rápido mas menos seguro que o MD5.

O MD5 gera uma síntese[30] (uma espécie de "impressão digital") de 128 bits de uma mensagem de dimensão arbitrária, tendo sido projectado para oferecer uma grande resistência a colisões. Todavia, tal não impediu a detecção recente de vulnerabilidades a esse nível [Rob94, Dob96], sendo já sugerida a adopção de outros algoritmos em futuras aplicações[31].

2.5.2 Códigos de Autenticação de Mensagens

Um Código de Autenticação de uma Mensagem (MAC[32]) resulta da aplicação de uma Função de Hashing Unidireccional combinada com uma operação de encriptação.

A forma mais expedita de produzir um MAC consiste em cifrar com uma chave simétrica a síntese da mensagem. O destinatário, que deverá também possuir essa chave, recalcula a síntese, cifra–a e compara–a com o MAC recebido. Se os MACs coincidirem, tal constitui prova da Integridade e Autenticidade da mensagem[33].

À semelhança das Funções de Hashing Unidireccionais, um esquema de produção de MACs deve também respeitar certas propriedades, sob pena de ser vulnerável a determinados ataques. Assim, segundo [Sta95]:

[28] Do inglês *birthday–attacks*. [Sta95] descreve com algum pormenor os fundamentos matemáticos e probabilísticos deste ataque.

[29] [Sta95] lista algumas dessas possibilidades.

[30] Do inglês *digest*, correspondente ao *hash* de que temos vindo a falar.

[31] A concretização prática da presente tese faz ainda uso do MD5 por não ter aparecido, até à data, uma alternativa suficientemente consensual, embora seja crescente o protagonismo de algoritmos como o *Secure Hash Algorithm* (SHA) [oST95] ou o RIPEMD160 [DBP96].

[32] Do inglês *Message Authentication Code*.

[33] [Sch96] refere ainda outras possibilidades para o cálculo de MACs, assinalando também alguns ataques possíveis.

- dados M (mensagem) e $MAC_K(M)$ (MAC de M baseado na aplicação da chave secreta K), então o esforço computacional envolvido na determinação de M', tal que $MAC_K(M') = MAC_K(M)$, deverá ser incomportável;
- $MAC_K(M)$ deve ser uniformemente distribuído no sentido de que, dadas duas mensagens aleatórias M e M', a probabilidade de que $MAC_K(M') = MAC_K(M)$ é de 2^{-n} (com n = número de bits da síntese);
- se $M' = f(M)$, em que f corresponde a determinada transformação sobre M[34], então a probabilidade de $MAC_K(M') = MAC_K(M)$ deverá ser 2^{-n} (ou seja, o algoritmo do MAC não deverá ser mais vulnerável em certas partes da mensagem do que noutras).

2.5.3 Assinaturas Digitais

Uma Assinatura Digital é uma técnica de Autenticação que pretende combater a Repudiação, ou seja, a negação da transmissão de uma mensagem pelo emissor ou da sua recepção pelo destinatário.

[Sta95] enumera algumas propriedades que se desejam presentes num esquema de Assinaturas Digitais:

- a assinatura deve ser uma sequência de bits que depende da mensagem a assinar e de algo único relativamente ao originador (a fim de prevenir Falsificação e Repudiação);
- a produção, reconhecimento e verificação de assinaturas deverá ser fácil;
- deverá ser computacionalmente impraticável forjar uma assinatura, quer procurando uma mensagem para uma assinatura preexistente, quer gerando uma assinatura para uma mensagem preexistente;
- deverá ser possível armazenar uma cópia da assinatura para futura reutilização.

A título exemplificativo, apresenta–se, de seguida, um protocolo simples em que a Assinatura Digital resulta da aplicação da chave privada do originador sobre uma síntese da mensagem. Assim, se $\mathcal{A}$ deseja enviar a $\mathcal{B}$ uma mensagem M aberta (*i.e.*, não cifrada) mas assinada, um protocolo na direcção $\mathcal{A} \rightarrow \mathcal{B}$ poderá ser:

1. $\mathcal{A}$ e $\mathcal{B}$ concordam previamente num sistema assimétrico e numa Função de Hashing Unidireccional, H, comuns;
2. $\mathcal{A}$ gera a síntese $h = H(M)$;
3. $\mathcal{A}$ efectua $ds = E_{K_{prv_A}}(h)$, gerando a assinatura digital ds;
4. $\mathcal{A}$ envia $M|ds$[35] (a mensagem e a sua assinatura digital) a $\mathcal{B}$;
5. $\mathcal{B}$ efectua $h' = H(M)$, gerando uma síntese local, h';

[34] Por exemplo, inverter um ou mais bits específicos.
[35] Em que o símbolo "|" significa concatenação.

6. $\mathcal{B}$ (que deve conhecer antecipadamente K_{pub_A}) efectua $h = D_{K_{pub_A}}(ds)$, recuperando a síntese original, h;

7. $\mathcal{B}$ compara h' com h; se forem iguais, então a Autenticidade de M está comprovada. $\mathcal{A}$ não pode negar a origem da mensagem uma vez que só $\mathcal{A}$ (que é a única entidade que possui K_{prv_A}) poderia ter cifrado ds de forma a que $h' = h$.

O protocolo anterior fornece Autenticação mas não Privacidade, a qual pode ser conseguida cifrando a mensagem M com uma chave simétrica comum ou com a chave pública do destinatário. É prudente que este procedimento seja extensível à própria assinatura digital ds, ou seja, deve–se cifrar o conjunto $M|ds$, em vez de deixar a assinatura a descoberto. Tal dificulta a remoção ou substituição da assinatura por parte de um adversário. Existem ainda outras razões que justificam esta prática, entre as quais:

- se a mensagem a assinar não é visível ao assinante, então a assinatura pode ter pouco valor legal [Rih94];

- alguns algoritmos de chave pública, como por exemplo o RSA, são vulneráveis a certos ataques quando se opta por cifrar antes de assinar [AN95].

Assim, se $\mathcal{A}$ deseja enviar a $\mathcal{B}$ uma mensagem M assinada e cifrada, um protocolo na direcção $\mathcal{A} \rightarrow \mathcal{B}$ poderá ser:

1. $\mathcal{A}$ e $\mathcal{B}$ concordam previamente num sistema assimétrico e numa Função de Hashing Unidireccional, H, comuns. Deverão ainda conhecer as chaves públicas um do outro;

2. $\mathcal{A}$ efectua $h = H(M)$, gerando a síntese h;

3. $\mathcal{A}$ efectua $ds = E_{K_{prv_A}}(h)$, gerando a assinatura digital ds;

4. $\mathcal{A}$ efectua $C = E_{K_{pub_B}}(M|ds)$ e envia $C \equiv ||M|ds||$[36] a $\mathcal{B}$;

5. $\mathcal{B}$ efectua $D_{K_{prv_B}}(C)$ e obtém $M|ds$;

6. $\mathcal{B}$ efectua $h' = H(M)$, gerando uma síntese local, h';

7. $\mathcal{B}$ efectua $h = D_{K_{pub_A}}(ds)$, recuperando a síntese original, h;

8. $\mathcal{B}$ compara h' com h; se forem iguais, então a Autenticidade de M está comprovada, tendo sido garantida Privacidade na sua transmissão.

Estes protocolos caem na categoria dos Protocolos de Assinatura Digital Directa, assim designados porque o originador e o receptor da mensagem são as únicas entidades participantes no protocolo. A segurança da chave privada de cada entidade constitui o ponto crítico destes protocolos: uma entidade pode argumentar que a sua chave privada foi comprometida, repudiando assim a autoria de todas as mensagens posteriores a esse evento.

Alternativamente, um Protocolo de Assinatura Digital Arbitrada pode impedir a Repudiação. Genericamente, sempre que o originador $\mathcal{A}$ pretender enviar uma mensagem

[36] Em que "||" delimita elementos cifrados de uma mensagem.

assinada ao destinatário $\mathcal{B}$, remete–a para um árbitro $\mathcal{C}$, o qual deverá submeter a mensagem a uma série de verificações que confirmem o seu conteúdo e a sua origem como genuínos, após o que lhe aplica uma espécie de certificado de autenticidade e a reencaminha para $\mathcal{B}$. Desta forma, nenhum originador poderá negar a autoria de uma mensagem. Porém, a presença obrigatória de um intermediário em qualquer troca de mensagens introduz novos problemas:

- é difícil manter uma confiança total e permanente na entidade árbitro. Todo o esquema ruirá à mínima suspeita de que o ponto intermédio foi comprometido;
- uma vez que todas as mensagens são serializadas através de um árbitro, este constitui um ponto de estrangulamento natural do sistema, não só em termos de desempenho[37] como de disponibilidade[38].

Existem políticas de gestão de chaves que procuram, de alguma forma, dar uma resposta satisfatória a esta classe de problemas. Retomaremos este assunto na secção 2.7.

2.6 Sequenciação

Por vezes, não basta ter a certeza de que uma mensagem recebida permaneceu secreta e íntegra e que o originador é quem afirma ser (e que não o pode negar). Pode ser necessário assegurar que a troca de mensagens segue uma determinada sequência ou ainda que não há réplicas em circulação. Neste contexto, uma mensagem autêntica deverá ser rejeitada caso viole a sequência ou constitua uma cópia de uma mensagem recebida anteriormente[39].

Existem ataques, compreensivelmente denominados por *ataques–de–repetição*[40], que pretendem explorar as fraquezas de um protocolo ou algoritmo precisamente neste domínio. Basicamente, um *ataque–de–repetição* assenta na retransmissão de uma mensagem por parte de uma entidade que, de alguma forma, a interceptou. [Gon93] fornece alguns exemplos de *ataques–de–repetição* possíveis:

- *repetição simples:* uma entidade captura uma cópia de uma mensagem e retransmite-a posteriormente, em qualquer instante;
- *repetição limitada:* uma marca temporal[41] mantém–se válida nos limites de determinada janela de tempo, o que permite a retransmissão de mensagens com essa marca, ao longo desse intervalo de tempo;
- *repetição indetectável:* uma mensagem que foi interceptada e impedida de atingir o seu destino é posteriormente retransmitida; sob o ponto de vista do destinatário, esta mensagem corresponde à mensagem original;
- *repetição dirigida à origem:* uma mensagem é retransmitida em direcção à sua origem, na tentativa de a fazer passar por mensagem válida ou de provocar uma situação não prevista no protocolo.

[37] *Do inglês performance.*
[38] *Do inglês availability.*
[39] Cópia exacta, entenda–se. Se houver necessidade de retransmitir uma mensagem, ela deverá ser individualizada de alguma forma, a fim de se distinguir da cópia anteriormente emitida.
[40] Do inglês *replay–attacks*.
[41] Do inglês *timestamp*.

Tradicionalmente, os mecanismos de combate a esta classe de ataques assentam na utilização de:

- *números de sequência:* uma mensagem é aceite se o seu número de sequência corresponder ao esperado, o que normalmente exige que ambas as partes mantenham o registo da sequência de números correcta, em ambos os sentidos;

- *marcas temporais:* uma mensagem é aceite se possuir uma marca temporal (colocada pelo originador) suficientemente próxima do tempo actual do receptor. A sincronização dos relógios de ambas as partes é, assim, um requisito fundamental à aplicação desta técnica, sendo necessários protocolos adicionais de acesso a servidores de tempo. Esses protocolos já não podem depender de relógios sincronizados; devem ser seguros, tolerantes a faltas e lidar com atrasos variáveis e imprevisíveis na propagação das mensagens;

- *desafios/respostas*[42]*:* uma mensagem é aceite se incluir um determinado valor que o receptor fez chegar previamente ao emissor; esse valor é gerado, no receptor, especificamente para cada troca[43]. Note–se que não é essencial que esse valor seja imprevisível (gerado aleatoriamente, por exemplo): um simples contador pode servir. Contudo, valores previsíveis devem ser usados com prudência, uma vez que podem constituir oportunidade para ataques [AN94].

2.7 Gestão de Chaves

A gestão de chaves diz respeito aos seus processos de geração, transmissão e armazenamento e é, reconhecidamente, uma das tarefas mais árduas da Criptografia. Com efeito, de nada valem protocolos sofisticados e algoritmos seguros se as respectivas chaves forem comprometidas. Dado que é precisamente na gestão de chaves que muitos ataques se concentram, ela deve ser feita com um grau de segurança não inferior àquele que se exige para os dados que essas chaves protegem.

2.7.1 Geração

Um bom algoritmo de geração de chaves não deve ser permeável à análise criptográfica, ou seja, não deverá ser possível a determinação de chaves com base na análise algorítmica do seu processo de geração. Obviamente, a concretização do algoritmo também deverá estar à altura das qualidades que governaram a sua concepção. Factores como a dimensão do espaço das chaves (que determina a viabilidade de um ataque exaustivo, pela *força bruta*), a vulnerabilidade a *ataques–de–dicionário*[44] e o grau de aleatoriedade (quando necessária) das chaves determinam a eficácia de um algoritmo de geração de chaves. Acresce ainda que há algoritmos para os quais certas chaves são consideradas "fracas" (como é o caso de DES [MS87]), no sentido de que a sua utilização pode ser detectada, sendo por isso

[42] Do inglês *challenges/responses*.

[43] Diz–se que é um *valor de ocasião* (do inglês *nonce*).

[44] Do inglês *dictionary–attacks*. São ataques baseados numa selecção de chaves comummente usadas e podem ser bastante eficazes: [Kle90] descreve uma aplicação deste método sobre *passwords* com uma taxa de sucesso de 40%.

fortemente desaconselhada. Finalmente, a própria geração das chaves pode ser difícil se lhes forem exigidas determinadas propriedades matemáticas (recordem–se, por exemplo, os requisitos do RSA em termos de números primos grandes, cujo produto deverá ser difícil de factorizar).

2.7.2 Transmissão

A troca de chaves entre duas entidades a fim de estabelecerem um canal seguro e a submissão de uma chave a uma entidade que proceda à sua certificação[45] (e eventualmente armazenamento) são, entre outras, circunstâncias em que se dá lugar ao trânsito de uma chave, provavelmente através de um meio hostil. Consequentemente, torna-se necessário providenciar mecanismos de transmissão segura das chaves. Esses mecanismos variam conforme o tipo de chaves em questão.

Troca de Chaves Simétricas

Para chaves simétricas, [Sta95] assinala as seguintes alternativas básicas:

1. a entidade $\mathcal{A}$ selecciona uma chave e entrega–a à entidade $\mathcal{B}$ através de um encontro face–a–face ou de outra técnica de distribuição manual;

2. uma terceira entidade $\mathcal{C}$ gera a chave e entrega–a a $\mathcal{A}$ e a $\mathcal{B}$ também segundo uma técnica de distribuição manual;

3. se $\mathcal{A}$ e $\mathcal{B}$ já partilham uma chave, esta pode ser usada para proteger a troca de uma nova chave. Nesta perspectiva, as chaves também podem classificar–se em *chaves--de–encriptação–de–chaves*[46] (usadas para proteger a distribuição de outras chaves, sendo distribuídas manual e pouco frequentemente) e *chaves–de–dados*[47] (usadas para proteger o tráfego dito "normal");

4. se $\mathcal{A}$ e $\mathcal{B}$ já detêm um canal seguro com $\mathcal{C}$, então podem trocar–se chaves através desse canal. A entidade $\mathcal{C}$ é vulgarmente designada por *Centro de Distribuição de Chaves* (KDC)[48] e partilha com cada entidade uma *chave mestra*[49]. Genericamente, quando uma entidade pretende estabelecer com outra(s) um canal seguro, solicita ao KDC uma *chave–de–sessão*[50] que deverá ser comunicada (pelo KDC ou pela entidade que tomou a iniciativa) a todas as partes interessadas. Todas as mensagens trocadas entre uma entidade e o KDC são cifradas com base na *chave mestra* que partilham. Obviamente, este esquema pressupõe uma distribuição inicial segura (possivelmente manual) das *chaves mestras*. É ainda possível (e até desejável, não só em termos de distribuição de carga, como também em termos de segurança[51]) uma gestão distribuída das chaves, baseada numa hierarquia de KDCs, cada qual responsabilizando–se

[45] O conceito de certificação será esclarecido ainda nesta secção.
[46] Do inglês *key–encryption keys*.
[47] Do inglês *data keys*.
[48] Do inglês *Key Distribution Center*.
[49] Do inglês *master key*.
[50] Do inglês *session key*.
[51] Um ataque bem sucedido a um KDC que concentre todas as chaves é potencialmente mais grave que um ataque a um outro que detém apenas um conjunto parcial dessas chaves.

por determinado domínio: duas entidades em domínios diferentes poderiam ainda estabelecer um canal seguro se os KDCs respectivos cooperarem nesse sentido.

Troca de Chaves Assimétricas

Ao permitirem a divulgação pública de uma das chaves, os sistemas assimétricos resolvem o principal "quebra–cabeças" dos sistemas simétricos: como criar um canal seguro sem depender de uma troca segura de chaves[52]? No contexto dos sistemas assimétricos, a distribuição de chaves envolve não só as chaves públicas, mas também possíveis aplicações à distribuição de chaves simétricas. Em relação à distribuição de chaves públicas, [Sta95] resume as técnicas conhecidas nas seguintes categorias:

- *anúncio público*: uma entidade anuncia publicamente[53] a sua chave pública. Contudo, esse anúncio pode ser forjado: a entidade $\mathcal{A}$ pode fazer–se passar pela entidade $\mathcal{B}$ e anunciar uma chave pública que, na realidade, não é de $\mathcal{B}$. Nitidamente, esta técnica carece de alguma forma de Autenticação;

- *directoria de chaves públicas*: uma determinada entidade, universalmente confiável, é responsável pela manutenção de uma base de dados/directoria (possivelmente distribuída) que armazena as chaves públicas de todas as entidades que se registaram (supostamente de forma segura e autenticada[54]). Sempre que uma entidade necessita de uma certa chave pública, pode solicitá–la à directoria (através de um protocolo seguro, necessariamente) ou consultar uma cópia da directoria se esta for periodicamente publicada. Em qualquer altura, uma entidade pode requisitar, na directoria, a substituição da sua chave pública por outra. Os ataques mais óbvios a este esquema concentram–se na determinação da chave privada da autoridade que mantém a directoria ou na substituição de entradas da base de dados[55]. É possível considerar ainda uma variante mais robusta da *directoria de chaves públicas*: a *autoridade para as chaves públicas*, em que se assegura que a chave pública da autoridade que mantém a directoria chega ao conhecimento de cada entidade através de um procedimento seguro e autenticado;

- *certificados*: nos esquemas centralizados descritos anteriormente, a requisição da chave pública de uma entidade junto de uma autoridade central antecede, pelo menos, o primeiro contacto com essa entidade. A autoridade central pode, por conseguinte, ser bastante sobrecarregada. Uma alternativa que minimiza a intervenção de terceiros consiste na certificação das chaves públicas; ou seja, cada entidade submete (através de um protocolo seguro e autenticado) a sua chave pública a uma *autoridade de certificação*[56], a qual comprova a origem da chave e lhe acrescenta determinada informação de controle (como por exemplo uma assinatura baseada na sua chave privada e ainda um período de validade). Este certificado é devolvido à entidade

[52] Dito de outra forma: "como criar um canal seguro sem ter, à partida, um canal seguro"?

[53] Através da sua página WWW ou de um grupo de *news*, por exemplo.

[54] Através de um encontro face–a–face, por exemplo.

[55] Em ambos os casos seria então possível disseminar falsas chaves públicas e consequentemente ter acesso às mensagens cifradas com base nessas chaves.

[56] Que não é necessariamente única. Em geral, esta autoridade fará parte de uma hierarquia de certificação que, normalmente, culmina numa autoridade de confiança máxima.

que o requisitou, a qual poderá exibi–lo junto de outra entidade a fim de comprovar a associação do seu nome à chave pública em questão[57]. Essa comprovação serve–se da chave pública da autoridade de certificação, supostamente conhecida por todas as entidades (e, obviamente, certificada). Resumidamente, os requisitos mínimos de um esquema de certificação são:

- um certificado deverá poder ser lido por qualquer entidade que pretenda saber o nome e a chave pública que ele encerra;
- deverá ser possível verificar se o certificado é genuíno e não uma falsificação;
- só uma autoridade de certificação poderá criar certificados;
- qualquer entidade deverá poder verificar se um determinado certificado está dentro do prazo de validade.

Troca de Chaves Simétricas com base em Chaves Assimétricas

Com base numa infra–estrutura pré–montada de chaves assimétricas, é possível levar a cabo a distribuição de chaves simétricas de uma forma segura e autenticada.

O protocolo que a seguir se apresenta é uma evolução do protocolo apresentado na secção 2.4 e demonstra a aplicação deste conceito. Assim, se $\mathcal{A}$ deseja enviar a $\mathcal{B}$ uma chave de sessão simétrica K, combinando Confidencialidade, Autenticação (incluindo Não-Repudiação), Integridade e Sequenciação, então um protocolo na direcção $\mathcal{A} \rightarrow \mathcal{B}$ poderá ser:

1. $\mathcal{A}$ e $\mathcal{B}$ concordam previamente num sistema assimétrico e numa Função de Hashing Unidireccional, H, comuns. Deverão ainda conhecer as chaves públicas, um do outro, devidamente certificadas;
2. $\mathcal{A}$ constrói a mensagem $M = id_A|K_{AB}|t_A$, em que id_A corresponde ao identificador da entidade $\mathcal{A}$ presente no certificado da sua chave pública, K_{AB} é a chave de sessão e t_A é uma marca temporal (ou contador, caso os relógios de $\mathcal{A}$ e $\mathcal{B}$ não se suponham sincronizados) que identifica esta transacção;
3. $\mathcal{A}$ gera a síntese $h = H(M)$ e a assinatura digital $ds = E_{K_{prv_A}}(h)$;
4. $\mathcal{A}$ efectua $C = E_{K_{pub_B}}(M|ds)$ e envia $C \equiv ||M|ds||$ a $\mathcal{B}$;
5. $\mathcal{B}$ efectua $D_{K_{prv_B}}(C)$ e obtém $M|ds \equiv id_A|K_{AB}|t_A|ds$;
6. $\mathcal{B}$ constata, pela leitura de id_A, que, aparentemente, a mensagem vem de $\mathcal{A}$[58] e verifica se possui uma cópia do certificado da chave pública de $\mathcal{A}$. Em caso negativo,

[57] Note–se que a posse de um certificado não implica a posse da chave privada correspondente à chave pública certificada. Basta recordar que os certificados podem circular livremente, uma vez que a informação que encerram é pública. Um certificado estabelece apenas uma associação entre um identificador — da entidade que solicitou a certificação — e uma chave pública, associação essa que deverá ter sido verificada pela autoridade de certificação.

[58] Facto do qual ainda não pode ter a certeza, já que id_A pode ter sido forjado ou adulterado em trânsito. Contudo, se isso tiver acontecido, a verificação da assinatura digital ds irá falhar, detectando–se uma eventual fraude.

pode solicitá–la a uma autoridade de certificação que mantenha uma cópia dos certificados[59] ou mesmo executar um outro protocolo com $\mathcal{A}$ destinado especificamente a esse efeito;

7. $\mathcal{B}$ efectua $h' = H(M)$, gerando uma síntese local, h', e efectua $h = D_{K_{pub_A}}(ds)$, recuperando a síntese original, h;

8. $\mathcal{B}$ compara h' com h; se forem iguais, então a Autenticidade (incluindo a Não–Repudiação) e a Integridade da mensagem estão garantidas;

9. $\mathcal{B}$ verifica ainda se a mensagem não é repetida através da análise da marca t_A (se for uma marca temporal verifica se faz parte de uma janela de tempo válida e se for um número de sequência assegura–se de que não faz parte de uma gama ultrapassada);

10. doravante, $\mathcal{B}$ pode usar a chave simétrica K_{AB} para cifrar o tráfego com $\mathcal{A}$.

2.7.3 Armazenamento

O armazenamento deve garantir a Privacidade, a Integridade e a Autenticidade das chaves. Existe uma grande variedade de formas de armazenamento: cartões magnéticos/*smart-cards*, circuitos integrados, partição da chave e armazenamento separado das partes[60], ficheiros em disco (ou outro suporte) cifrados com chaves mais simples de recordar, bases de dados (centralizadas ou distribuídas, individuais ou públicas), a própria memória humana, etc.

Resumidamente, todos os cuidados devem ser postos no sentido de evitar que as chaves sejam comprometidas através de ataques aos locais e aos algoritmos empregues no seu armazenamento.

2.7.4 Autenticação

A transmissão de uma chave deve ser complementada por um processo que permita ao destinatário verificar se a chave pertence realmente ao originador e não a um impostor. Nesta secção são referidos alguns exemplos concretos de protocolos em que a troca de chaves é feita de uma forma que conjuga a Confidencialidade com a Autenticação[61]. Dos exemplos a seguir apresentados, o Kerberos é o único que não tem uma relação directa com a concretização prática da presente dissertação. Contudo, impõe–se–lhe uma referência, ainda que breve, dada a importância de que continua a gozar nesta área e, de certa forma, devido ao seu papel precurssor na mesma.

[59] Alternativamente, em vez de id_A, $\mathcal{A}$ poderia ter enviado a $\mathcal{B}$ o próprio certificado em questão.

[60] Um exemplo paradigmático é fornecido pelo *Escrowed Encryption Standard* (EES) [oST94b], recentemente proposto pelo governo norte–americano com o objectivo de assegurar a Privacidade das comunicações telefónicas do comum dos cidadãos, ao mesmo tempo que permite a sua desencriptação por parte de agências de combate ao crime, quando devidamente autorizadas para o efeito. Neste esquema, uma das chaves secretas envolvidas é dividida em duas partes, cada qual entregue a uma agência governamental "de confiança", durante a manufactura dos dispositivos de encriptação. A junção das duas metades da chave é imprescindível para decifrar o tráfego do cidadão ou entidade em questão, e teria que ser "devidamente" autorizada e justificada. [LKB+94, Sch96] apresentam discussões interessantes de algumas das inúmeras questões que a implementação de um esquema (no mínimo polémico) deste género suscita.

[61] Para uma informação mais exaustiva recomenda–se a consulta de [Sch96].

Kerberos

O Kerberos [MNSS88] é um serviço de Autenticação centralizado, nitidamente na linha dos KDCs, actuando como um intermediário ou árbitro entre os clientes que pretendem aceder a determinado serviço e os servidores que providenciam esse serviço. Baseado num modelo de autenticação originalmente apresentado em [NS78], o Kerberos foi desenvolvido no seio do projecto Athena, do MIT, tendo sido introduzida recentemente a sua quinta versão [KN93], após se terem detectado alguns problemas de segurança nas versões anteriores [BM91].

A motivação inicial para o desenvolvimento do Kerberos foi a tentativa de resolver o problema do acesso autenticado a serviços espalhados pela rede, a partir de estações de trabalho remotas. Num ambiente deste género, um servidor não pode confiar na identificação de um cliente apresentada pela sua estação de trabalho, uma vez que:

- um utilizador pode obter acesso à conta de outrem na tentativa de se fazer passar por aquele junto do servidor;
- é possível alterar o endereço de rede de uma estação de trabalho, de forma a enganar o servidor quanto à origem dos pedidos;
- um utilizador pode interceptar pedidos de acesso válidos e posteriormente tentar reutilizá–los (*ataque–de–repetição*).

O Kerberos é baseado em criptografia simétrica (concretamente no DES) e, para além de um serviço de autenticação, providencia também *chaves–de–sessão*. Cada entidade (cliente ou servidor) partilha uma chave secreta (do tipo *chave mestra*) com um servidor especial, que guarda essas chaves numa base de dados. Esse servidor designa–se, precisamente, por Kerberos. Quando um cliente pretende usufruir de um determinado serviço, requisita ao Kerberos um *ticket*[62] para aceder ao *Ticket Granting Service* (TGS). Esse *ticket* é devolvido ao cliente cifrado com a *chave mestra* que partilha com o Kerberos. O cliente usa então esse *ticket* para solicitar ao TGS um outro *ticket* para aceder ao serviço pretendido. Se tudo correr bem, o TGS devolve esse novo *ticket*, que o cliente apresentará, conjuntamente com um *autenticador*[63], ao servidor pretendido[64].

Troca de Chaves Diffie–Hellman

Tal como já foi referido na secção 2.3.2, o algoritmo de chave pública de Diffie–Hellman (DH) é especificamente vocacionado para a troca de chaves.

Existem algumas abordagens possíveis à utilização do DH[65]. Uma delas baseia–se na disponibilização de uma base de dados (supostamente inviolável) com todas as chaves públicas DH de uma comunidade, à guarda de uma entidade na qual deverão confiar todos os elementos dessa comunidade. Nitidamente, é uma abordagem centralizada, que

[62] Que é uma espécie de credencial, de validade limitada, que prova a identidade do cliente junto de um servidor, sendo assim específica para um par (cliente, servidor).

[63] Outra credencial, gerada cada vez que um cliente pretende aceder a um serviço. Ao contrário de um *ticket*, um autenticador só pode ser usado uma vez.

[64] Esta descrição é, obviamente, superficial, recomendando–se, por exemplo, a consulta de [NT94] para uma apresentação actual e completa do Kerberos.

[65] [Sch96] refere um bom número delas.

se enquadra na categoria dos KDCs. Sempre que uma entidade desejar estabelecer um canal seguro com outra, solicita ao KDC a chave pública DH do seu par, determinando–se então a chave secreta DH partilhada, sem que para isso tenha havido a necessidade de estabelecer previamente um contacto com a outra entidade. Esta entidade–destino necessita apenas de saber a identificação da entidade–origem das mensagens que recebe cifradas, a fim de, por sua vez, gerar, localmente, a chave secreta comum. Este esquema (que é apresentado na literatura [Sch96] sob a designação de "Troca de Chaves sem Trocar Chaves"[66]) pressupõe, naturalmente, a existência de um canal seguro entre as entidades e o KDC. Alternativamente, poderia aplicar–se um esquema de certificação às chaves públicas DH, eliminando assim, na maioria dos casos, a necessidade de contactar uma autoridade central.

A troca de chaves Diffie-Hellman assume um papel de especial protagonismo nos protocolos desenvolvidos no âmbito desta dissertação, que serão introduzidos oportunamente. Porém, refira–se desde já que a abordagem seguida não depende nem da existência de um KDC, nem de certificação de chaves públicas DH (pouco usual). Resumidamente, assume-se a existência de uma infra–estrutura de certificação X.509 (ver secção a seguir) de chaves públicas RSA, sendo com base nestas chaves que opera um protocolo de troca de chaves públicas DH. Este protocolo segue um esquema de *desafio–resposta* (que, em certa medida, recorda protocolos da classe *station–to–station* [DOW92]) e assenta na premissa de que pode confiar–se na associação de uma entidade a uma chave pública DH se essa entidade provar que possui a chave privada correspondente.

X.509

A recomendação X.509 [CCI89] do CCITT[67] define uma infra–estrutura de certificação de chaves públicas assente no Serviço de Directoria X.500, ou seja, a Directoria passa a ser também um repositório de certificados de chaves públicas. Adicionalmente, o X.509 prevê três protocolos de autenticação (*One–Way Authentication, Two–Way Authentication* e *Three–Way Authentication*), todos baseados na utilização de assinaturas geradas por chaves públicas devidamente certificadas[68].

Os certificados X.509 são emitidos por uma Autoridade de Certificação (CA[69]) e colocados na Directoria por essa autoridade ou pela própria entidade que solicitou o certificado. Naturalmente, uma CA deverá ser confiável por toda a comunidade de entidades que serve e as chaves públicas a certificar deverão ser transmitidas à CA de uma forma segura.

O formato geral de um certificado X.509 compreende os seguintes campos:

- *versão*: distingue sucessivas versões do formato;
- *número de série*: valor inteiro, único no conjunto dos certificados emitidos pela CA;
- *identificador do algoritmo*: identifica o algoritmo (e respectivos parâmetros) usado para assinar o certificado;

[66] Do inglês *Key Exchange Without Exchanging Keys.*
[67] *Consultation Committee, International Telephone and Telegraph.*
[68] Esses protocolos diferem nas qualidades de serviço prestadas, em relação às quais não entraremos em detalhes.
[69] Do inglês *Certification Authority.*

- *emissor*[70]: identifica a CA que assinou o certificado;
- *período de validade*: consiste em duas datas que definem o tempo de vida útil do certificado;
- *sujeito*: identifica a entidade que solicitou o certificado, ou seja, a detentora da chave pública em questão;
- *informação da chave pública*: consiste na chave pública (e respectivos parâmetros) e num identificador do algoritmo de chave pública;
- *assinatura*: abarca os outros campos e resulta da encriptação do seu *hash* com a chave privada da CA.

A verificação da Autenticidade de um certificado só é possível na posse da chave pública da CA que emitiu esse certificado. Consequentemente, essa chave deverá ser comunicada de forma segura às entidades interessadas. Uma vez que a detecção de certificados falsificados/adulterados é imediata (pela não verificação da assinatura respectiva), torna–se possível não só mantê–los na Directoria como transmiti–los a quem os solicita sem preocupações especiais de segurança. Assim, quando uma entidade necessita da chave pública de outra, pode pedir o seu certificado à Directoria ou recebê–lo directamente dessa entidade. Uma vez na posse desse certificado, pode mantê–lo numa *cache* local e reutilizá–lo posteriormente, mas sempre com a preocupação de verificar a sua validade. Para que um certificado se mantenha válido não basta que o seu tempo de vida não tenha expirado. Um certificado pode ser revogado, porque, entre outras razões, a chave privada correspondente à chave pública do certificado foi comprometida, a CA deixou de certificar o sujeito do certificado ou a chave privada da CA foi comprometida. Periodicamente, uma CA publica uma *Lista de Certificados Revogados* (CRL[71]) (mas não–expirados), os quais são removidos da Directoria[72]. Essa lista deverá ser acessível a partir da Directoria (e poderá ser também mantida numa *cache* local), de modo a permitir que uma entidade confirme a validade de um certificado antes de o utilizar.

O X.509 prevê a existência de várias CAs, organizadas numa hierarquia em árvore. Cada CA certifica e controla uma comunidade restrita de entidades que conhecem a sua chave pública. Por seu turno, cada CA certifica também as chaves públicas das CAs imediatamente acima na hierarquia[73] e possui também certificados da sua chave pública gerados por essas CAs[74]. Uma cadeia de certificação ocorre sempre que uma entidade necessita de verificar o certificado de outra entidade que pertence a uma comunidade distinta. Por exemplo, na notação X.509, em que $y << x >>$ corresponde ao certificado de x emitido pela CA y, $X_1 << X_2 >> X_2 << B >>$ exprime uma cadeia de certificação pela qual uma entidade A, certificada pela CA X_1, consegue verificar o certificado de B emitido pela CA X_2. Para isso, A necessitou de obter um certificado da chave pública de X_2 emitido por X_1 (*reverse certificate*), após o que pode verificar qualquer certificado emitido por X_2. Acresce ainda que estas cadeias de certificação são arbitrariamente longas.

[70] Do inglês *issuer*.
[71] Do inglês *Certificate Revocation List*.
[72] Contudo, a CA deve manter uma cópia desses certificados, dado que podem ser necessários para resolver disputas futuras.
[73] Resultando desse processo os chamados *reverse certificates*.
[74] Designando–se esses certificados por *forward certificates*.

O X.509 tem sido objecto de sucessivas revisões[75], desde que [IM90] detectou problemas de segurança na recomendação inicial. Entre outras novidades, as extensões introduzidas pelo X.509v2 traduzem–se em novos campos nas CRLs enquanto que as novas extensões que caracterizam o X.509v3 afectam sobretudo a estrutura dos certificados.

[75] Sendo [IT97] a mais recente.

Capítulo 3

Abordagens à Comunicação Segura em Sistemas Distribuídos

O desenvolvimento de um novo protocolo de comunicação segura, no contexto dos Sistemas Distribuídos, encerra decisões de vária ordem. No início desse processo confrontamo–nos, inevitavelmente, com algumas questões de fundo, entre as quais:

- Valerá a pena desenvolver (mais) um novo protocolo? Qual será a sua mais valia relativamente aos que já existem?
- Será necessário escrever, de raiz, o código respeitante aos algoritmos criptográficos escolhidos, ou poderemos confiar na reutilização de bibliotecas, eventualmente de domínio público, com essas funcionalidades?

Justificar o esforço de desenvolvimento de um novo protocolo não é trivial. No âmbito da presente dissertação, essa tarefa afigura–se, aparentemente, mais complicada, uma vez que o protocolo proposto (o S3L) tem raízes num outro preexistente (o *Simple Key Management for Internet Protocols* (SKIP) [AP95, AMP95c]).

Relativamente à possível reutilização de código, tal decisão só pode ser tomada mediante a análise de um conjunto suficientemente representativo de *kits* de programação criptográfica. Para já, e antecipando a decisão tomada a este respeito, diremos que a opção caiu sobre o *Secure Development Environment* (SecuDE) [Sch95a].

A escolha do SKIP e do SecuDE como pontos de partida para o desenvolvimento teórico e prático, respectivamente, do S3L resultou, então, da comparação de diversas propostas que actualmente se encontram disponíveis no domínio da comunicação segura em Sistemas Distribuídos. Essas propostas correspondem, tipicamente, a visões diferentes do posicionamento das funções de Segurança na arquitectura do modelo de Comunicações e, concomitantemente, do próprio Sistema Operativo. Neste capítulo, começaremos por identificar alguns desses pontos de vista, para depois apresentarmos algumas abordagens que lhes dão corpo. Essa apresentação serve, fundamentalmente, dois objectivos:

1. proporcionar uma visão global (embora necessariamente superficial) do estado da arte neste domínio;
2. possibilitar a compreensão das opções que determinaram a eleição do SKIP e do SecuDE, como pontos de partida para o desenvolvimento do S3L.

3.1 Encriptação Ligação–a–Ligação

A aplicação de encriptação na Camada Física de uma pilha de protocolos de comunicação[1] garante a Confidencialidade de todo o tráfego trocado entre dois equipamentos que partilham uma ligação física directa. Esta forma de encriptação, designada por "Encriptação Ligação–a–Ligação[2]", pode ser conseguida de uma forma expedita, pela colocação de equipamento(s) de encriptação ao nível da interface com o meio de transmissão. Naturalmente, os dois extremos de uma ligação deverão partilhar uma chave, cuja aquisição se supõe ter sido feita de uma forma segura. Alternativamente, a Encriptação Ligação–a–Ligação também poderá ser levada a cabo ao nível da Camada de Ligação de Dados[3] e, como veremos na próxima secção (3.2), é possível argumentar que ela também é viável na Camada de Rede. Sob o ponto de vista das camadas superiores, a encriptação realizada numa camada inferior processa–se de uma forma transparente[4].

A Encriptação Ligação–a–Ligação não evita que os dados fiquem temporariamente expostos nos nós intermédios entre o originador primário e o destino final, já que num nó intermédio é necessário decifrar a mensagem que se recebeu para voltar a cifrá–la antes de enviá-la ao próximo nó[5]. Além disso, todas as ligações devem estar igualmente protegidas[6], o que, para redes de grandes dimensões, pode ter custos económicos demasiado elevados em termos do número de dispositivos de encriptação necessários.

Contudo, ao nível da ligação, esta técnica de encriptação é bastante efectiva, já que nega, a quem eventualmente observe o tráfego de mensagens, o conhecimento não só do conteúdo das mesmas, como também da sua estrutura (originador primitivo, destino final, dimensão, frequência com que a troca de mensagens se processa entre duas entidades Fim-a–Fim, etc.). [Sch96] designa este nível de segurança por "segurança do fluxo de tráfego". Na eventualidade de se quererem prevenir ataques baseados na análise da quantidade de tráfego entre dois nós directamente ligados, é ainda possível conceber esquemas que geram tráfego aleatório a fim de manter a ligação permanentemente activa [Sta95].

3.2 Encriptação Fim–a–Fim

A Encriptação Fim–a–Fim[7] estabelece um canal seguro entre dois sistemas finais[8]. Uma vez que a Camada de Rede chama a si a responsabilidade de fazer chegar as mensagens de Transporte (ou Unidades de Dados do Protocolo de Transporte (TPDUs)[9], na terminologia OSI), de um sistema final (originador) a outro (destino), efectuando, se necessário, o seu

[1] Ao longo deste capítulo, e salvo indicação em contrário, adoptaremos a divisão sugerida pelo Modelo de Referência OSI [CCI84].

[2] Do inglês *Link–to–Link*.

[3] Do inglês *Data Link Layer*.

[4] Tradicionalmente, no contexto do Modelo de Referência OSI, a encriptação é enquadrada na Camada de Apresentação. Entretanto, esse posicionamento rígido tem sido questionado, havendo quem advogue a sua extensão às outras camadas [Bra87].

[5] Para além de as chaves serem, em geral, diferentes para ligações físicas diferentes, pode ser necessário abrir as mensagens para efectuar, por exemplo, detecção e/ou correcção de erros.

[6] Sob pena de haver caminhos com troços inseguros, o que pode acabar por colocar em causa a segurança de toda a rede.

[7] Do inglês *Ent–to–End*.

[8] Do inglês *end systems*.

[9] Do inglês *Transport Protocol Data Units*.

encaminhamento através dos sistemas intermediários que unem redes diferentes, então só a partir da Camada de Transporte (inclusive, e em sentido ascendente) é que se consideram disponíveis conexões Fim–a–Fim[10].

Assim, o primeiro ponto onde se afigura possível a aplicação da Encriptação Fim–a–Fim situa–se "algures" entre a Camada de Transporte e a Camada de Rede: as TPDUs poderão ser cifradas antes de serem integradas nas Unidades de Dados do Protocolo de Rede (NPDUs)[11]. Se a encriptação ocorrer efectivamente na Camada de Rede[12], então, desde que a chave usada seja apenas partilhada com a Camada de Rede do sistema de destino, é assegurada a Confidencialidade dos TPDUs perante as Camadas de Rede dos sistemas intermediários. Neste contexto, a Encriptação Fim–a–Fim ocorreu, efectivamente, na Camada de Rede[13]. Por outro lado, ao nível da Camada de Rede é obviamente possível Encriptação Ligação–a–Ligação, considerando chaves partilhadas entre sistemas intermediários, usadas para cifrar as NPDUs antes de serem entregues à Camada de Ligação de Dados[14].

Focando agora a nossa atenção sobre a Encriptação Fim–a–Fim, verificamos que não são necessários equipamentos de encriptação dedicados para cada ligação física. Contudo, toda a informação específica da Camada de Rede (e inferiores) fica vulnerável a ataques, uma vez que deixa de haver "segurança do fluxo de tráfego": a análise do padrão do tráfego de mensagens bem como de alguma da sua estrutura é agora possível já que a encriptação não afecta a informação específica das camadas abaixo da Camada de Transporte. Adicionalmente, a gestão de chaves torna–se mais complexa. Agora, em cada sistema final não basta uma chave para cada ligação física que esse sistema estabelece. Poderá haver múltiplas conexões de Transporte, Sessão, Apresentação ou Aplicação, cada qual necessitando, eventualmente, de uma chave própria, que deverá ser acordada com a entidade correspondente no outro extremo da conexão Fim–a–Fim.

A Encriptação Fim–a–Fim não se limita, portanto, à protecção de TPDUs. Ela poderá ocorrer sobre unidades de dados de protocolos de mais alto nível. Tal é desejável se se pretender uma maior independência em relação aos pormenores tecnológicos e de implementação das camadas inferiores. Por outro lado, a Encriptação Fim–a–Fim ao mais baixo nível possível proporciona segurança de uma forma mais transparente às aplicações, minimizando (ou mesmo evitando) a sua reprogramação a fim de se adaptarem a um determinado modelo de segurança para o qual não foram originalmente concebidas.

A tabela 3.1, extraída de [Pfl89], compara diversos aspectos da Encriptação Ligação–a–Ligação e Fim–a–Fim. Idealmente, seria desejável a combinação das duas políticas de encriptação a fim de auferir das vantagens de ambas. Porém, uma opção desse género nem sempre é viável. [Sch96] salienta, por exemplo, os custos económicos adicionais que uma

[10] Ou seja, as entidades de Transporte, Sessão, Apresentação e Aplicação, que constituem o extremo de uma conexão do nível correspondente, estão localizadas nos sistemas finais.

[11] Do inglês *Network Protocol Data Units*.

[12] O termo "algures" terá que ser necessariamente concretizado e, sob o Modelo de Referência OSI, essa concretização corresponde à implementação do serviço de encriptação das TPDUs na Camada de Transporte ou na Camada de Rede, não havendo meio termo.

[13] As abordagens swIPe (ver secção 3.4.1) e SKIP (ver secção 3.4.2) constituem exemplos concretos desta opção.

[14] Obviamente, a localização exacta das funções de encriptação/desencriptação teria que ser novamente decidida entre a Camada de Rede ou a Camada de Ligação de Dados. Independentemente da escolha, teríamos sempre Encriptação Ligação–a–Ligação.

Ligação–a–Ligação	Fim–a–Fim
Segurança em Sistemas Finais (SFs) e Intermediários (SIs)	
Mensagem exposta no SF originador.	Mensagem cifrada no SF originador.
Mensagem exposta num SI.	Mensagem cifrada num SI.
Papel do Utilizador	
Assumido pelo SF originador.	Assumido pelo *processo* originador.
Transparente ao utilizador.	Encriptação aplicada pelo utilizador.
Cada nó é responsável pela manutenção da sua infra–estrutura de encriptação.	O utilizador determina o algoritmo de encriptação a usar.
Uma infra–estrutura de encriptação serve vários utilizadores.	O utilizador selecciona os esquemas que lhe convêm.
Realizável em *hardware*.	Preferentemente em *software*.
Todas ou nenhumas mensagens cifradas.	Encriptação selectiva.
Detalhes de Implementação	
Necessita de uma chave por cada par de sistemas.	Necessita de uma chave por cada par de utilizadores.
Fornece autenticação dos sistemas.	Fornece autenticação dos utilizadores.

Tabela 3.1: Comparação da Encriptação Ligação–a–Ligação e Fim–a–Fim.

abordagem combinada acarreta.

3.3 Abordagens Fim–a–Fim

As abordagens Fim–a–Fim que iremos analisar de seguida constituem apenas uma amostra[15], julgada representativa, de um universo cada vez maior e pretendem ilustrar a diversidade de perspectivas com que é encarada a comunicação segura, acima da Camada de Rede.

3.3.1 Generic Security Service API

A *Generic Security Service Application Program Interface* (GSS–API) [Lin93b] especifica uma interface de acesso a uma série de serviços de segurança que as aplicações podem invocar, de uma forma genérica, independente das linguagens de programação[16] e dos mecanismos de segurança subjacentes efectivamente usados (*e.g.*, algoritmos criptográficos, políticas de gestão de chaves, etc.). Consequentemente, são assegurados bons níveis de portabilidade das aplicações entre ambientes que suportam tecnologias de segurança diferentes.

A concepção da GSS–API perseguiu alguns objectivos fundamentais [Lin90]:

- *independência dos mecanismos criptográficos*: a GSS–API define uma interface genérica de acesso a funcionalidades criptográficas independente das opções concretas

[15] Uma vez que não é intenção da presente dissertação a elaboração de classificações exaustivas a este respeito.

[16] Embora se tenham definido associações da especificação da GSS–API a linguagens de programação concretas, como por exemplo o C [Wra91, Wra93].

tomadas a este nível[17];

- *independência dos protocolos*: a GSS–API é independente dos protocolos particulares em que possa ser usada. Uma pequena camada interposta entre as aplicações e o protocolo de comunicação usado (*e.g.*, RPC) poderá providenciar as necessárias chamadas à GSS–API;

- *independência de protocolos*: o contexto seguro fornecido pela GSS–API poderá ser usado pelas aplicações em circunstâncias em que não há obrigatoriamente lugar à operação de um protocolo de comunicação[18];

- *adaptabilidade a uma gama diversa de localizações*: os clientes da GSS–API não estão necessariamente confinados a um perímetro de computação seguro[19].

A GSS–API permite a uma aplicação autenticar–se perante outra, delegar–lhe direitos e assegurar Confidencialidade, Autenticação e Integridade para cada mensagem trocada. A utilização da GSS–API prevê, assim, quatro etapas [Wra91]:

1. cada aplicação adquire um conjunto de credenciais, com as quais pode provar a sua identidade;

2. um par de aplicações estabelecem um contexto seguro comum, usando as respectivas credenciais. Durante esse processo, a aplicação iniciadora apresenta as suas credenciais à outra aplicação, a qual, por sua vez, pode ser solicitada a efectuar também a sua Autenticação. A aplicação iniciadora pode ainda delegar à outra o direito de, futuramente, tomar a iniciativa no estabelecimento de contextos seguros comuns;

3. invocação de serviços de Integridade, Autenticação e Confidencialidade sobre cada mensagem trocada. A GSS–API prevê duas combinações básicas destas qualidades de serviço. Ambas fornecem Integridade e Autenticação da origem, mas só uma delas assegura Confidencialidade;

4. durante a terminação de uma sessão (que pode incluir várias conexões), as aplicações eliminam o(s) contexto(s) entretanto criado(s).

A tabela 3.2, extraída de [Lin90], sumariza as principais funções[20] disponibilizadas pela versão 1 da GSS–API. Recentemente, foi apresentada uma proposta para a versão 2 [Lin96] que permanece ainda em discussão.

[17] Por exemplo, [Lin93c] refere a possibilidade de acomodar a GSS–API sobre o Kerberos ou sobre o *Distributed Authentication Security Service* (DASS) [Kau93] (este último baseado no X.509) e em [Lin93b] essas possibilidades são discutidas mais pormenorizadamente.

[18] Todavia, em [Lin93b] reconhece–se que a utilização típica da GSS–API será feita por aplicações que implementam determinados protocolos (*e.g.*, *telnet*, *ftp*) onde se deseja a integração de Autenticação, Integridade e/ou Confidencialidade.

[19] Do inglês *trusted computing base*.

[20] Não foram consideradas funções auxiliares.

Gestão de credenciais	
GSS_Acquire_cred	adquirir credencial;
GSS_Release_cred	eliminar credencial;
Gestão de contextos	
GSS_Init_sec_context	iniciar contexto no originador;
GSS_Accept_sec_context	aceitar contexto no respondedor;
GSS_Delete_sec_context	eliminar contexto;
GSS_Process_context_token	processar *token* de controlo recebido num contexto;
GSS_Context_time	indicar tempo de vida de um contexto;
Funções sobre mensagens	
GSS_Sign	assinar uma mensagem;
GSS_Verify	verificar a assinatura de uma mensagem;
GSS_Seal	assinar e cifrar uma mensagem;
GSS_Unseal	decifrar uma mensagem e verificar a sua assinatura.

Tabela 3.2: Funções principais da interface GSS–API.

3.3.2 Secure Network Programming

O *Secure Network Programming* (SNP) [WBSL94] fornece uma abstracção de alto nível, sob a forma de uma biblioteca de funções, sobre a qual é possível conseguir comunicação segura Fim–a–Fim. O SNP foi concebido para apresentar uma interface semelhante à fornecida pelo mecanismo de *sockets* de Berkeley, facilitando assim a sua integração em aplicações originalmente baseadas naquele modelo. Contudo, o SNP e o mecanismo de *sockets* diferem em dois pontos cruciais:

- o SNP fornece serviços adicionais de segurança (nomeadamente, Autenticação, Integridade e Confidencialidade das mensagens) sobre os serviços usuais de *streams* e datagramas proporcionados pelos *sockets*;
- o SNP oferece comunicação segura Fim–a–Fim ao nível da Camada de Aplicação, enquanto os *sockets* são considerados abstracções de acesso à Camada de Transporte[21].

Com o SNP, são possíveis diversas combinações das seguintes qualidades de serviço [WBSL94]:

- *entrega persistente*: o emissor efectuará a retransmissão da mensagem enquanto não obtiver do destinatário a confirmação da sua entrega;
- *entrega do tipo melhor–esforço*: não se garante a entrega da mensagem ao seu destinatário. Cada um dos nós intermédios pode reencaminhar ou desprezar a mensagem;
- *entrega sequenciada*: uma mensagem deverá ser entregue ao seu destino final segundo a ordem pela qual foi enviada, ou seja, não se admite reordenação e duplicação de mensagens;

[21] Isto não é inteiramente correcto, uma vez que os *sockets* também podem fornecer acesso a camadas inferiores (e.g., *raw sockets*). Todavia, os *sockets* são considerados, tipicamente, interfaces de acesso à Camada de Transporte.

- *confidencialidade dos dados*: o conteúdo de uma mensagem só deverá ser acessível ao verdadeiro destinatário;
- *integridade dos dados*: as mensagens não deverão ser adulteradas em trânsito de forma a que tal passe despercebido ao receptor;
- *autenticação da origem dos dados*: o receptor de uma mensagem só deverá aceitá–la se a origem se confirmar como válida;
- *autenticação do destino dos dados*: o receptor de uma mensagem deverá confirmar, sem ambiguidades, que é o verdadeiro destinatário dessa mensagem;
- *autenticação da conexão*: uma conexão só deverá ser realizada entre as partes legitimamente interessadas.

O SNP assenta sobre a GSS–API[22] o que, à partida, lhe confere uma portabilidade razoável. Contudo, a transparência do SNP não é completa[23]: o SNP assume um modelo de comunicação explícito, *i.e.*, os utilizadores das funções SNP são directamente responsáveis pela iniciação de conexões, troca de mensagens e terminação das conexões. Este facto, aliado à semelhança sintáctica e semântica entre as funções SNP e as suas homónimas do mecanismo de *sockets*, sugere ao programador que está a lidar com "*sockets* seguros", embora, na realidade, tudo se passe na Camada de Aplicação.

A tabela 3.3 sumariza a especificação da interface fornecida pelo SNP[24].

3.3.3 Secure Sockets Layer

O *Secure Sockets Layer* (SSL) [FKK96] é um protocolo, da iniciativa da Netscape Communications, sobre o qual é possível construir aplicações, segundo o modelo Cliente–Servidor, que comunicam de forma segura. O SSL é pois, nitidamente, uma abordagem Fim–a–Fim, integrada na Camada de Aplicação.

Os objectivos primários do SSL são, segundo [FKK96]:

- *segurança criptográfica*: estabelecimento de uma conexão segura entre duas entidades (cliente e servidor);
- *interoperabilidade*: compatibilidade entre implementações diversas da mesma especificação do SSL;
- *extensibilidade*[25]: permitir a introdução de novos algoritmos criptográficos com o menor esforço e perturbação possíveis (evitando assim, provavelmente, a introdução de novas fraquezas);

[22] Isolando, porém, o programador de aplicações de alguns detalhes da GSS–API (*e.g.*, credenciais e gestão de contextos).

[23] Embora se reconheça que não era esse um dos objectivos dos seus autores.

[24] À semelhança da tabela apresentada para a GSS–API, não são apresentados os parâmetros específicos de cada função. Acresce ainda que as funções apresentadas como semelhantes às suas homónimas do mecanismo de *sockets* diferem destas em determinados aspectos semânticos que aqui não se julgou oportuno inventariar, recomendando–se a consulta a [WBSL94] para um esclarecimento cabal desta questão.

[25] Do inglês *extensibility*.

Inicialização	
`snp()`	semelhante a `socket()` mas devolve uma referência[a] de Aplicação;
`snp_bind()`	semelhante a `bind()`;
`snp_listen()`	semelhante a `listen()`;
`snp_attach()`	especifica a identidade com que se deseja ser autenticado bem como a identidade do outro extremo;
Estabelecimento de conexão	
`snp_connect()`	semelhante a `connect()`;
`snp_accept()`	semelhante a `accept()`;
Transferência de dados	
`snp_write()`	semelhante a `write()`;
`snp_read()`	semelhante a `read()`;
`snp_send()`	semelhante a `send()`;
`snp_recv()`	semelhante a `recv()`;
`snp_sendto()`	semelhante a `sendto()`;
`snp_recvfrom()`	semelhante a `recvfrom()`;
Terminação da conexão	
`snp_close()`	semelhante a `close()`;
`snp_shutdown()`	semelhante a `shutdown()`.

[a]Do inglês *handle*. Essa referência corresponde a um dos extremos da comunicação.

Tabela 3.3: Algumas funções da interface do SNP.

- *eficiência*: minimizar as operações de chave pública, à custa de mecanismos de *caching* que evitem a repetição desnecessária de uma negociação[26] completa de qualidades de serviço.

Basicamente, o SSL compreende duas camadas: ao nível mais baixo, assente sobre um protocolo de transporte fiável (*e.g.*, o TCP), encontra–se o *SSL Record Protocol* (SSL–RP). O SSL–RP é usado para encapsular todas as mensagens oriundas da camada superior, entre as quais se encontram as do *SSL Handshake Protocol* (SSL–HP). O SSL–HP permite a um cliente e a um servidor autenticarem–se mutuamente[27] e negociar um conjunto de diversas qualidades de serviço (algoritmos e respectivas chaves, entre outros parâmetros de segurança) necessárias para que as aplicações assentes no SSL troquem mensagens de uma forma segura.

Da negociação levada a cabo pelo SSL–HP resulta o estabelecimento de uma sessão[28] entre o cliente e o servidor, eventualmente com múltiplas conexões. O cliente e o servidor podem agora trocar mensagens de uma forma segura, contando com os préstimos do SSL--RP para encapsular essas mensagens.

Basicamente, a segurança de uma conexão SSL assenta nas seguintes funcionalidades [FKK96]:

[26]Do inglês *handshake*.

[27]A autenticação de um servidor é obrigatória, mas a autenticação de um cliente é opcional, ocorrendo apenas por solicitação explícita do servidor.

[28]São ainda possíveis várias sessões em simultâneo, entre o mesmo par de entidades.

- encriptação simétrica do canal após uma negociação prévia que inclui a definição de uma chave secreta de sessão;
- autenticação da identidade de cada parte com base em criptografia assimétrica e certificados;
- fiabilidade[29] da conexão com base na utilização de MACs.

Sessões e conexões definem estados, sendo que uma conexão define um estado mais efémero que uma sessão, já que uma sessão poderá ser suspendida e retomada posteriormente. Neste contexto, é lícito afirmar que o SSL é *statefull*[30].

A Netscape Communications disponibiliza uma implementação própria do SSL, designada por SSLRef[31]. Apesar de ser gratuita a sua utilização para aplicações não comerciais, porventura o maior *handicap* da SSLRef são as restrições à sua exportação para fora dos Estados Unidos e que se traduzem na menor segurança dos algoritmos da versão exportável[32].

Em resposta às restrições de exportação da SSLRef, surgiu o SSLeay[33]. O SSLeay já suporta a versão 3 do SSL, é de domínio público, grátis (inclusivamente para aplicações comerciais) e a sua adopção é generalizada, fora dos Estados Unidos, na programação de aplicações que necessitam do protocolo SSL.

No contexto do SSL, o estabelecimento de uma sessão segura entre um cliente e um servidor é algo complexa. Por outro lado, uma vez estabelecida a sessão, a troca de mensagens é relativamente expedita. A título meramente ilustrativo, apresentam–se de seguida os passos fundamentais, em termos das funções invocadas, tomando como referência a implementação do SSLeay[34]:

1. `SSL_CTX_new()`: criar uma estrutura `SSL_CTX` onde será guardada toda a informação da sessão, eventualmente a partilhar por várias conexões;
2. `SSL_load_verify_location()`, `SSL_set_default_verify_paths()`: especificar a localização da informação necessária à verificação dos certificados eventualmente apresentados pela outra parte;
3. `socket()`, `connect()`, `bind()`, `listen()`, etc.: cliente e servidor estabelecem uma conexão TCP com base em chamadas às primitivas de *sockets*;
4. `SSL_new()`: criar uma estrutura `SSL` e inicializá–la com os parâmetros da conexão estabelecida;
5. `SSL_set_fd()`: associar a conexão SSL à referência[35] do *socket* TCP conectado;

[29] Do inglês *reliability*.

[30] E é também num contexto semelhante que o S3L será classificado como *stateless* (ver secção 4.1).

[31] `http://home.netscape.com/newsref/std/sslref.html`

[32] Ver, por exemplo, o desafio `http://hwww.portal.com/~hfinney/sslchal.html` e a respectiva solução em `http://pauillac.inria.fr/~doligez/ssl/announce.txt` através da qual se demonstra a vulnerabilidade de uma transacção comercial assente no SSL com algoritmos licenciados para exportação.

[33] `http://psych.psy.uq.oz.au/~ftp/Crypto/`

[34] Não se apresentam os parâmetros das funções. Note–se ainda que a descrição que se segue não foi possível de efectuar para a SSLRef, dada a sua indisponibilidade fora dos Estados Unidos.

[35] Do inglês *handle*.

6. `SSL_set_verify()`: estabelecer o modo de verificação dos certificados;
7. `SSL_set_pref_cipher()`: estabelecer as preferências em termos de algoritmos criptográficos a usar;
8. `SSL_use_certificate_file()`, `SSL_use_Private_key_file()`: especificar a localização do nosso certificado (e da chave privada correspondente à chave pública do certificado), caso seja necessário fornecê–lo à outra parte;
9. `SSL_connect()`, `SSL_accept()`: efectivar a conexão SSL; corresponde à realização do protocolo SSL-HS;
10. `SSL_write()`, `SSL_read()`: trocar mensagens através da conexão SSL estabelecida;
11. `shutdown(SSL_get_fd())`, `SSL_free()`, `SSL_CTX_free()`: terminar a conexão e libertar as suas estruturas de controle.

O SSL tem sido usado não só na programação de novas aplicações, como também se tem assistido ao aparecimento de versões de alguns dos protocolos/aplicações mais populares da Internet (*e.g.*, HTTP, FTP e Telnet) especialmente adaptadas para operarem sobre o SSL.

3.3.4 Segurança em Invocações Remotas de Procedimentos

Originalmente apresentado por [BN84], o modelo de Invocação Remota de Procedimentos (RPC)[36] permite a um programa que executa num determinado sistema invocar, de forma relativamente transparente, uma subrotina, cuja execução poderá ocorrer noutro sistema. Ou seja, o mecanismo de RPCs permite a distribuição da execução de um programa. Adicionalmente, facilita a programação de aplicações segundo o paradigma Cliente–Servidor.

Actualmente, existem diversas implementações do modelo RPC. A mais divulgada é, provavelmente, a da Sun [Sun88], dado que alguns dos seus serviços de rede mais populares (como é o caso do *Network File System* (NFS) e do *Network Information System* (NIS)[37]), assentam na utilização do Sun RPC como mecanismo de comunicação. Mais recentemente, a Sun introduziu o *Transport Independent RPC* (TI–RPC) [Sun91], que, como o nome indica, pretende ser mais versátil em relação à gama de protocolos de Transporte que o suporta[38]. Na prática, o TI–RPC encontra–se ainda confinado ao sistema operativo Solaris, razão pela qual não será objecto de análise nesta secção. A *Open Software Foundation* (OSF) avançou também com uma proposta de um mecanismo de RPC no âmbito do *Distributed Computing Environment* (DCE) e que passaremos a designar por DCE RPC. Existe ainda uma especificação da ISO [ISO91] para um mecanismo de RPCs, não sendo, porém, segundo [Bar93], conhecidas implementações.

Nesta secção será feita uma breve análise à segurança do mecanismo de RPCs original da Sun[39] e do DCE RPC.

[36] Do inglês *Remote Procedure Call*.
[37] Previamente conhecido por *Yellow Pages* (YP).
[38] Recorde–se que o Sun RPC original assenta no TCP [Pos80c] ou no UDP [Pos80a].
[39] Em concreto, será analisada uma variante conhecida por "Secure RPC".

Secure RPC

A introdução de segurança no mecanismo original de RPCs da Sun[40] foi feita, fundamentalmente, ao nível da Autenticação entre clientes e servidores.

Tipo	Técnica de Autenticação	Comentários
`AUTH_NONE`	Nenhuma.	Acesso anónimo.
`AUTH_UNIX` /`AUTH_SYS`	Cliente apresenta o UID e GIDs de tipo UNIX ao servidor.	Inseguro. Servidor confia de forma implícita na identidade do cliente.
`AUTH_DES`	Baseada em criptografia pública e de chave secreta.	Segurança razoável. Propriedade da SUN.
`AUTH_KERB`	Baseada no Kerberos.	Bastante segura. Necessita de um serviço Kerberos. Pouco divulgada. Recente.

Tabela 3.4: Tipos de autenticação no mecanismo de RPCs.

As modalidades de Autenticação actualmente previstas pelo mecanismo de RPCs são apresentadas na tabela 3.4, extraída de [GS96]. Dessas modalidades, a `AUTH_DES` é também conhecida por Secure RPC. A adopção do Secure RPC, contudo, não tem conhecido o sucesso que se verificou para a especificação RPC inicial, encontrando–se praticamente restrito às plataformas SunOS e Solaris. Segundo [GS96], as razões para tal assentam na necessidade de licenciar o produto directamente com a Sun (uma vez que não foram produzidas versões de domínio público[41]) e de utilizar o NIS ou o NIS+ [Ram94] em conjunto com o Secure RPC.

No contexto do Secure RPC, a Autenticação assenta, em primeira instância, numa Troca de Chaves de Diffie–Hellman. Cada entidade — seja ela um utilizador ou uma máquina[42] — tem associado um par de chaves — pública e privada — de Diffie–Hellman. A chave privada encontra–se cifrada, *via* DES, com a *password* do utilizador. O triplo <*nome canónico*[43]*, chave pública, chave privada cifrada*> resume a informação relevante, no contexto do Secure RPC, para cada utilizador e consta de uma base de dados, distribuída através do NIS ou NIS+[44].

Resumidamente, o processo de Autenticação contempla as seguintes fases [BCK+94]:

1. *login*:

   i. o utilizador fornece um *username* e uma *password* a fim de aceder a uma máquina onde o serviço Secure RPC é disponibilizado;

[40] Doravante, nesta secção, o termo RPC será assumido como uma referência ao mecanismo original de RPCs da Sun e não ao TI–RPC.

[41] Em parte devido às restrições de exportação de determinados algoritmos criptográficos empregues no Secure RPC.

[42] Neste caso, o super–utilizador e a máquina confundem–se.

[43] Por exemplo, `fred.sun.com`, ou seja, o utilizador `fred` no domínio `sun.com`.

[44] No que se segue, o termo NIS é usado indistintamente para NIS ou NIS+.

ii. a *password* fornecida é usada para decifrar, *via* DES, a chave privada do triplo <*nome canónico, chave pública, chave privada cifrada*>, o qual é obtido *via* NIS com base no nome canónico;

iii. a chave privada é guardada, em memória, pelo processo *keyserv*;

2. negociação[45] entre o cliente c e o servidor s:

   i. com base no acordo de Diffie–Hellman, o cliente combina a sua chave privada com a chave pública do servidor (obtida *via* NIS), gerando uma chave de sessão comum, SK;

   ii. o cliente gera uma chave de conversação CK, aleatória, e envia–a, cifrada com a chave de sessão, ao servidor, numa mensagem de formato $\{c, \{CK\}_{SK}, \{window\}_{CK}, \{t_1, window + 1\}_{CK}\}$, em que $window$ especifica o tempo de vida de CK e t_1 é uma marca temporal;

   iii. o servidor obtém a chave pública do cliente (via NIS) e gera a chave de sessão comum, SK;

   iv. o servidor decifra a chave de conversação CK e com ela decifra o resto da mensagem, obtendo $window$, t_1 e $window + 1$;

   v. se o valor $(window + 1)$ fornecido for efectivamente igual à soma de $window$ com 1, então a mensagem só poderá ter vindo de c;

   vi. o servidor guarda $< c, CK, window, t_1 >$ na entrada ID de uma tabela de credenciais;

   vii. o servidor responde ao cliente com $\{\{t_1 - 1\}_{CK}, ID\}$;

   viii. se o cliente confirmar que o valor $(t_1 - 1)$ fornecido é efectivamente igual à diferença de t_1 com 1, então conclui que ambas as partes estão autenticadas, entregando ID e CK ao processo $keyserv$, para armazenamento;

3. tráfego "normal" entre o cliente e o servidor:

   i. na direcção cliente $\rightarrow$ servidor: conjuntamente com a mensagem, o cliente envia $(ID, \{t_n\}_{CK})$. O servidor usa ID para aceder à tabela de credenciais e verificar se $t_n < t_1 + window$, em cujo caso a mensagem é aceite;

   ii. na direcção servidor $\rightarrow$ cliente: conjuntamente com a mensagem, o servidor envia $(\{t_n - 1\}_{CK}, ID)$. A comparação do valor $(t_n - 1)$ recebido com o t_n local permitirá autenticar a origem como sendo o servidor;

4. *logout*: a informação do cliente no *keyserv* (triplo $< \ldots >$ e pares (ID, CK)) é eliminada.

Pelo exposto, é evidente que o Secure RPC não fornece serviços de Confidencialidade nem de Integridade sobre as mensagens trocadas. Essa é, porventura, uma das suas maiores fraquezas. [GS96, BCK+94] chamam ainda a atenção para outras limitações, entre as quais:

[45] Ocorre na altura do primeiro contacto ou sempre que a chave de sessão expira.

- as aplicações programadas segundo o modelo RPC necessitam de ser individualmente modificadas a fim de indicarem, explicitamente, o tipo de autenticação (ver tabela 3.4) que pretendem;
- ao depositar no DES uma parte significativa da sua segurança, o Secure RPC tem um tempo de vida provavelmente mais limitado que o previsto inicialmente, uma vez que é previsível, a médio prazo, a viabilidade de *ataques–de–força–bruta* ao DES [Wie93];
- um *ataque–de–dicionário* sobre a *password* de um utilizador pode ser suficiente para recuperar a sua chave privada;
- se o acesso à zona de memória do processo *keyserv* não for devidamente protegido, as chaves privadas aí mantidas ficam expostas.

A Sun encontra–se, actualmente, a desenvolver o GSS RPC, o qual representa a integração no mecanismo de RPCs da generalidade de mecanismos de Autenticação possíveis de aceder através da GSS–API. O GSS RPC é, por assim dizer, o próximo passo na evolução do Sun RPC, após a introdução do TI–RPC.

DCE RPC

O *Distributed Computing Environment* (DCE), da *Open Software Foundation* (OSF), define uma arquitectura de serviços que suportam o desenvolvimento, utilização e manutenção de aplicações distribuídas. A arquitectura do DCE é independente do sistema operativo e da rede, colocando assim particular ênfase na interoperabilidade entre tecnologias diferentes. A disponibilização de um conjunto uniforme de serviços, em qualquer parte da rede, deverá abilitar as aplicações a usar melhor os recursos da rede.

Os serviços oferecidos pelo DCE dividem–se, fundamentalmente, em Serviços Distribuídos Fundamentais e Serviços de Partilha de Dados [Fou92a]. Dos primeiros, constam serviços como a Invocação Remota de Procedimentos, Serviço de Directoria, Serviço de Tempo, Serviço de Segurança e Serviço de Fios de Execução[46]. Na segunda categoria enquadram–se o Sistema de Ficheiros Distribuído e o Suporte na Ausência de Disco[47].

O Serviço de Segurança do DCE assenta, essencialmente, em três subserviços: Autenticação, Autorização e Controle de Acesso [Loc94].

A Autenticação baseia–se no Kerberos (versão 5) mas existem diferenças determinadas por requisitos próprios do serviço de Autorização. Assim, o *Ticket Granting Ticket* (TGT) que um cliente recebe do Kerberos[48] durante o processo de *login* não é usado directamente para contactar o *Ticket Granting Service* (TGS) a fim de receber dele um *ticket* de acesso ao servidor pretendido. Em vez disso, o TGT é usado para aceder novamente ao Kerberos, requisitando–lhe um *Privilege Service Ticket* (PST) de acesso ao *Privilege Service*[49]. O *Privilege Service* devolve, então, um *Privilege–Ticket–Granting Ticket* (PTGT). Um PTGT contém um *Privilege Attribute Certificate* (PAC) que define a classe do cliente em

[46] Do inglês *Threads*.
[47] Do inglês *Diskless Support*.
[48] Neste contexto, Kerberos refere–se ao servidor que gera TGTs e não ao conjunto dos serviços de Autenticação designado, genericamente, de Kerberos.
[49] Em si, um subserviço do Kerberos.

termos de privilégios/autorizações genéricos[50]. Finalmente, o cliente apresenta o PTGT ao TGS, pedindo–lhe um *ticket* de acesso ao servidor pretendido.

O Controle de Acesso baseia–se na utilização de Listas de Controlo de Acesso (ACLs)[51]. As ACLs associam recursos a uma entidade ou grupo de entidades, definindo o tipo de operações que essas entidades estão autorizadas a realizar sobre esses recursos. O Controle de Acesso é levado a cabo nos servidores pela confrontação dos privilégios contidos nos PACs e das acções pretendidas pelos clientes, com as restrições impostas pelas ACLs no acesso aos recursos em causa.

O DCE RPC foi concebido, à partida, para usufruir de um bom nível de segurança, fazendo–o de uma forma plenamente integrada com o Serviço de Segurança do DCE [Fou92b]. Ao contrário do Secure RPC da Sun, que contempla apenas a Autenticação mútua entre clientes e servidores, o DCE RPC pode assegurar, em simultâneo, Autenticação, Integridade e Confidencialidade. O serviço de Autenticação do DCE baseia–se, como seria de esperar, no Kerberos. As *chaves–de–sessão* DES estabelecidas durante o processo de Autenticação podem, se assim se entender, assegurar Confidencialidade[52]. A verificação da Integridade das mensagens baseia–se na utilização do algoritmo MD5 (rever secção 2.5.1).

A partir da versão 1.1, o DCE forneceu uma implementação da GSS–API. Tal veio não só permitir o acesso aos Serviços de Segurança do DCE por parte de aplicações que não utilizam o DCE RPC como mecanismo de comunicação nativo, como também possibilitou a utilização directa, pelos programadores, de funcionalidades que antes se limitavam a poder ser invocadas, directamente, pelos DCE RPCs.

Actualmente (versão 1.2) a utilização do DES para assegurar Confidencialidade é a única opção de criptografia simétrica fornecida de origem com o DCE[53]. Adicionalmente, a utilização do DES no DCE está condicionada ao território dos Estados Unidos e do Canadá.

3.3.5 Kerberos

O Kerberos já foi apresentado, embora superficialmente, na secção 2.7.4. Nesta secção será analisado do ponto de vista da programação de aplicações que necessitam de segurança na comunicação.

O Kerberos, recorde–se, é um serviço de Autenticação que, adicionalmente, permite o estabelecimento de um canal seguro, entre um cliente e um servidor, com base em *chaves--de–sessão* simétricas, tipicamente DES.

Na prática, o Kerberos é geralmente usado por aplicações especialmente modificadas (*e.g.*, *telnet*, *rlogin*, *rsh*, *ftp*, servidores de *mail*[54], NFS[55], etc.) para tirarem proveito da

[50] O que geralmente corresponde à integração do cliente num ou mais grupos de entidades que partilham esse privilégios.

[51] Do inglês *Access Control Lists*.

[52] Opcionalmente, pode optar–se apenas pela Autenticação e Integridade, dispensando Confidencialidade, a fim de acelerar a troca de mensagens, desde que os riscos resultantes da não encriptação se considerem reduzidos, o que se pode assumir, por exemplo, para redes relativamente isoladas.

[53] O que não impede determinados fabricantes de acrescentarem outros algoritmos, em versões "proprietárias" do ambiente de desenvolvimento DCE.

[54] Por exemplo, servidores do tipo *Post–Office–Protocol* (POP).

[55] A este propósito, [SNS88] apresenta uma breve discussão sobre as modificações efectuadas ao NFS da

Autenticação que o Kerberos oferece. O Kerberos, portanto, é predominantemente usado por entidades/protocolos da Camada de Aplicação, o que não impede que desempenhe o seu papel também a níveis mais baixos (*e.g.*, DCE RPC e variante `AUTH_KERB` do Sun RPC[56]).

O processo de reconversão de uma aplicação que lhe permitirá beneficiar da Autenticação do Kerberos designa–se, na gíria própria do Kerberos, de "kerberização" [SNS88]. A "kerberização" de uma aplicação envolve chamadas às funções disponibilizadas pelas bibliotecas que fazem parte do ambiente de desenvolvimento do Kerberos. Essas funções[57] desempenham, entre outras tarefas, a Autenticação entre as partes, a Encriptação (usualmente DES) do tráfego subsequente com a chave de sessão estabelecida durante a Autenticação e também a geração de MACs como forma de assegurar Integridade das mensagens trocadas entre as partes autenticadas.

Actualmente, o acesso directo às APIs do Kerberos deixou de ser necessário[58] dado que, conjuntamente com a versão 5 do Kerberos, foi disponibilizada uma implementação da GSS–API. A "kerberização" de aplicações torna–se assim automática desde que essa aplicações tenham sido programadas com base em invocações à GSS–API.

Apesar da sua crescente popularidade, o Kerberos não é imune a algumas críticas, entre as quais [GS96]:

- cada serviço/aplicação deve ser individualmente modificado para ser usado com o Kerberos. A "kerberização" implica, obviamente, a disponibilidade do código fonte o que nem sempre é viável;

- o Kerberos não foi concebido para ambientes multi–utilizador. O ambiente nativo do Kerberos pressupõe estações de trabalho dedicadas, num dado instante, a um só utilizador, porque a coexistência de vários utilizadores pode constituir uma oportunidade para que um deles, caso a segurança da estação de trabalho tenha falhas[59], se aproprie dos *tickets* de outro, os quais eram, nas primeiras versões do Kerberos, mantidos em *cache* na directoria `/tmp`[60];

- o Kerberos necessita de um servidor central — o Kerberos, usualmente acompanhado do TGS —, localizado numa área física segura (*i.e.*, de acesso físico restrito) e exclusivamente dedicado às tarefas de autenticação e gestão da base de dados com as chaves secretas de todos os clientes. A disponibilidade de instalações com níveis de segurança tão exigentes, a fim de albergar a(s) máquina(s) com o Kerberos, nem sempre é viável para as organizações, quer sob o ponto de vista económico, quer sob o ponto de vista administrativo;

Sun a fim de integrá–lo com o Kerberos.

[56] O mecanismo de RPCs situa–se algures, acima da Camada de Transporte e abaixo da Camada de Aplicação, sendo comummente classificado como pertencente à Camada de Apresentação [Ste90] dado que faz uso de uma representação (*e.g.*, XDR [Sun87], no caso do Sun RPC) dos argumentos e resultados dos procedimentos, independente de questões de ordenação dos bits (*big endian versus little endian*) e outros pormenores de baixo nível.

[57] Cuja descrição completa, para a versão 5 do Kerberos, pode ser observada em [MIT91].

[58] E até aconselhado, a não ser por questões de desempenho.

[59] Basta que a *password* do super–utilizador seja comprometida.

[60] A este propósito, [BM91] salienta que mesmo usando memória partilhada, os riscos subsistem dada a possibilidade de paginação sobre o disco. Se as estações de trabalho não tiverem disco, então os riscos agravam–se porque se gera tráfego não protegido, em direcção ao servidor de ficheiros.

- o servidor Kerberos necessita de estar continuamente disponível. Caso contrário, os serviços que dele dependem ficam inacessíveis;
- no servidor Kerberos, as chaves dos clientes são todas cifradas com a *chave mestra* do servidor, a qual se encontra guardada na mesma máquina onde reside a base de dados das chaves dos clientes. Se essa *chave mestra* for comprometida, as chaves de todos os clientes ficarão também comprometidas;
- o Kerberos não oferece protecção contra modificações no *software* do sistema operativo. Sem autenticação da estação de trabalho perante o utilizador, mecanismos tão simples como máscaras de *login* são suficientes para comprometer a segurança do Kerberos[61];
- o Kerberos não é tão transparente como seria desejável. O tempo de vida de um *ticket* está limitado a oito horas, ao fim das quais o serviço correspondente deixa de estar disponível, sendo necessário repetir o procedimento de *login* ou então invocar uma série de comandos que permitem, manualmente, requisitar novos *tickets* de acesso aos serviços pretendidos.

Adicionalmente, em [BM91] faz–se uma análise detalhada a determinadas insuficiências detectadas nos protocolos de base do Kerberos (algumas das quais foram corrigidas na versão mais recente do Kerberos [KN93]) e que não serão aqui discutidas, pelo elevado nível de detalhe que envolvem em torno da especificação interna do Kerberos.

3.3.6 Secure Development Environment

O *Secure Development Environment* (SecuDE) [Sch95a] é um *kit* de segurança que incorpora uma colecção de utilitários, APIs e bibliotecas de suporte à programação de aplicações em relação às quais se pretendem serviços acrescentados de Autenticação, Integridade e Confidencialidade.

As funcionalidades oferecidas pelo SecuDE compreendem, basicamente:

- funções de criptografia básica simétrica (DES, 3DES, IDEA, RC2, RC4), assimétrica (RSA, DSA, DSS, Diffie–Hellman) e funções de *hashing* unidireccionais (MD2, MD4, MD5, SHA, RIPEMD160);
- funções capazes de assegurar Autenticação (incluindo Não–Repudiação) da origem, verificação da Integridade e Confidencialidade, com base em assinaturas digitais e criptografia simétrica e assimétrica;
- funções de manipulação de certificados X.509, contemplando caminhos de certificação, certificação cruzada e revogação de certificados;
- utilitários e funções que permitem operar as autoridades de certificação e fazer a sua interacção com os utilizadores;
- utilitários de assinatura, verificação, encriptação, desencriptação e de aplicação de funções de *hashing* unidireccionais, sobre ficheiros;

[61] [GS96] salienta que tal é possível porque muitas das estações de trabalho mantêm cópias locais de parte ou da totalidade do sistema operativo.

- processamento de mensagens de acordo com o standard *Privacy–Enhanced Mail* (PEM) [Lin93a, Ken93, Bal93, Kal93];
- acesso seguro à Directoria X.500 [CCI88] a fim de armazenar ou recuperar certificados e informação associada (*e.g.*, listas de revogação);
- codificação/descodificação segundo as *Basic Encoding Rules* (BER) [ISO87b] da *Abstract Sintaxe Notation One* (ASN.1) [ISO87a];
- protecção de toda a informação de segurança relevante de um utilizador num Ambiente Pessoal de Segurança (PSE)[62], implementado em *hardware*, através de um *smartcard* (SC–PSE), ou em *software*, através de um conjunto de ficheiros localizados numa directoria específica (SW–PSE). O acesso ao PSE é feito de uma forma transparente à sua implementação (*hardware/software*) e está condicionado à apresentação de um Número de Identificação Pessoal (PIN)[63] do qual se deriva uma chave DES que protege a informação preservada no PSE. Essa informação inclui, tipicamente, a(s) chave(s) pública(s) (devidamente certificada(s)) e privada(s) do utilizador, a chave pública da autoridade máxima de certificação[64], o nome distinto X.500 do utilizador, o nome de *login*, um caminho de certificação directa[65] entre a CA máxima e a CA que certificou a(s) chave(s) públicas do utilizador, listas de revogação de certificados e chaves públicas certificadas de outros utilizadores.
- interface de suporte à especificação GSS–API.

Na secção 3.6 voltaremos a falar do SecuDE. Nomeadamente, apresentaremos as razões que ditaram a sua escolha como *kit* criptográfico de suporte à implementação do S3L.

3.4 Abordagens na Camada de Rede

O fornecimento de segurança ao nível da Camada de Rede tem sido, ultimamente, objecto de intensa actividade de investigação e desenvolvimento. Nesse sentido, têm sido sugeridas extensões ao IP actual (*e.g.*, [Atk95a, Atk95b, Atk95c]), enquanto a migração para a versão 6 [Dee95] (que contempla de base opções de segurança[66]) não se processa.

As abordagens que iremos apresentar de seguida constituem exemplos nessa linha recorrendo a mecanismos de encapsulamento capazes de oferecer segurança e transparência em simultâneo. Em particular, o SKIP será alvo de uma análise mais detalhada, uma vez que serviu de ponto de partida à concepção do S3L.

3.4.1 swIPe

O swIPe [IB93] pretende ser uma resposta às dificuldades sentidas pela aplicações, num Sistema Distribuído, relativamente ao estabelecimento de comunicações seguras com base

[62] Do inglês *Personal Security Environment*.
[63] Do inglês *Personal Identification Number*.
[64] Chave essa conhecida por *Public Root Key* e que é a única chave pública de uma autoridade de certificação (CA) que não foi certificada, uma vez que diz respeito à CA que encabeça a hierarquia de certificação.
[65] Do inglês *Forward Certification Path*.
[66] As extensões referidas para o IPv4 coincidem com as opções de base fornecidas pelo IPv6.

numa infra–estrutura TCP/IP, e opta pelo fornecimento de segurança a um nível suficientemente baixo para que esse processo seja, na prática, transparente às aplicações.

Concretamente, o swIPe é um protocolo de encapsulamento de datagramas IP, capaz de assegurar Autenticação, Integridade e Confidencialidade ao nível da Camada de Rede[67]. O encapsulamento dos datagramas IP (possivelmente cifrados) é feito no interior de outros datagramas IP[68] de um novo tipo (`IPPROTO_SWIPE`), conjuntamente com a possível adição de informação de controlo de Autenticação e Integridade. Esta abordagem tem, segundo [IB93], algumas vantagens evidentes:

- uma vez que os datagramas IP são encapsulados dentro de outros datagramas IP, eles podem transitar por troços da rede que desconhecem o swIPe;
- dado que o encapsulamento e desencapsulamento pode ocorrer em qualquer ponto onde se processem datagramas IP[69] (*i.e.*, nos *routers* ou nos sistemas finais) então, com o swIPe, é possível segurança Fim–a–Fim (encapsulamento no sistema de origem e desencapsulamento no sistema de destino) ou Ligação–a–Ligação (encapsulamento e desencapsulamento também nos nós intermédios);
- é possível implementar uma variedade razoável de políticas de segurança pela decisão de encapsular (ou não) datagramas ou de aceitar (ou não) datagramas (que podem estar ou não encapsulados), com base no seu endereço de destino e origem, respectivamente.

Sob Unix, o swIPe é implementado com base numa interface de rede virtual[70], `sw`n (com n igual a 0, 1, etc.), e em direcção à qual são estabelecidas rotas por omissão. O processamento de pacotes IP é feito no núcleo[71] do sistema operativo, devidamente modificado para o efeito, enquanto o tratamento de excepções, a gestão de chaves e a política de segurança individual com cada máquina ficam a cargo de demónios do nível utilizador.

Apesar dos grandes benefícios que a sua abordagem transparente pressupõe, não podemos deixar de referir alguns aspectos menos positivos do swIPe:

- a gestão de chaves implica, frequentemente, a comunicação entre demónios em máquinas diferentes. Essa comunicação não usa (declaradamente) as qualidades de serviço oferecidas pelo swIPe. Logo, caso não se aplique um outro mecanismo capaz de assegurar Autenticação, Integridade e Confidencialidade do tráfego das mensagens de gestão de chaves, toda a estrutura de segurança do swIPe poderá ficar em risco;
- no swIPe, as políticas de segurança são estabelecidas tendo como sujeito a máquina/sistema, ou seja, todo o tráfego de e para esse sistema é afectado pelas exigências da política individual adoptada em relação a esse sistema. Logo, para aplicações que

[67] Ou seja, o swIPe não contempla outra protecção que não a de datagramas IP. Ficheiros e outros objectos de um Sistema Distribuído têm quer ser protegidos com recurso a outros mecanismos complementares.

[68] De uma forma análoga ao mecanismo posteriormente "normalizado" pelo protocolo *IP–inside–IP* (IPIP) [Sim95, Per95].

[69] Desde que haja suporte instalado para o swIPe, obviamente.

[70] Do inglês *virtual network interface*, em termos conceptuais semelhante ao mecanismo sugerido por [Bel90].

[71] Do inglês *kernel*.

não necessitem de segurança (ou necessitam apenas de um subconjunto das qualidades de serviço automaticamente asseguradas), existirá sempre uma penalização indesejada.

3.4.2 Simple Key Management for Internet Protocols

O *Simple Key Management for Internet Protocols* (SKIP) [AP95, AMP95c] é, primariamente, um esquema de distribuição de chaves, com base nas quais se torna possível assegurar Autenticação, Integridade e Confidencialidade na troca de pacotes IP. Particularmente bem adaptado a protocolos *não–orientados–à–conexão* (como o IP), em ambas as suas modalidades de comunicação ponto–a–ponto e multiponto, o SKIP fornece, neste último caso, uma boa escalabilidade na distribuição de chaves.

Ao contrário de outras abordagens à segurança na Camada de Rede (*e.g.*, swIPe), o SKIP não necessita de uma comunicação prévia entre as partes interessadas, a fim de estabelecer e actualizar chaves de sessão. O SKIP usa chaves individuais para cada pacote, que viajam integradas no próprio pacote.

Este esquema de operação é possível porque o SKIP recorre à Troca de Chaves de Diffie-Hellman, a qual, recorde–se (ver secção 2.7.4.), permite o estabelecimento de uma chave secreta, entre duas partes, desde que cada uma delas conheça a chave pública de Diffie-Hellman da outra[72]. Adicionalmente, o SKIP pressupõe a disponibilidade de certificados X.509 dessas chaves públicas. Uma vez estabelecida a chave secreta de longa duração (chave essa que permanece válida enquanto forem válidas as chaves pública e privada de Diffie-Hellman de cada uma das partes), dela faz–se derivar uma *chave mestra* K_{ij}, a qual será usada como *chave–de–encriptação–de–outras–chaves*.

Cada pacote IP é protegido com uma chave K_p, gerada aleatoriamente e por sua vez cifrada com K_{ij}. Uma vez que as chaves K_{ij} podem ser mantidas em *cache*, então a protecção individual de cada pacote torna–se viável, dado que não é necessário efectuar operações de chave pública (lentas, por natureza) a fim de gerar K_{ij}, cada vez que K_{ij} é necessária para cifrar K_p.

Tudo o que o receptor do pacote tem a fazer é: identificar o emissor[73], adquirir a sua chave pública de Diffie–Hellman devidamente certificada[74], combiná–la com a chave privada de Diffie–Hellman própria, gerar a chave secreta comum, dela derivar K_{ij}, usar K_{ij} para decifrar K_p (que viaja no pacote) e finalmente usar K_p para decifrar o resto do pacote e comprovar a sua Autenticidade e Integridade.

Note–se que, apesar de K_p poder ser gerada individual e aleatoriamente para cada pacote, tal não é estritamente necessário. Pode optar–se por uma política que conceda uma determinada vida útil (em termos de tempo ou de quantidade de tráfego protegido) a K_p, só ao fim da qual será gerada uma nova chave K_p.

Por sua vez, é também conveniente mudar a própria *chave mestra* K_{ij}. Tal não só dificulta a análise criptográfica dos pacotes como também previne a reutilização de chaves

[72] Note–se que o SKIP não depende exclusivamente, em termos conceptuais, da Troca de Chaves de Diffie-Hellman. Qualquer algoritmo de chaves públicas que permita a geração de uma chave secreta comum pela combinação da chave privada de uma parte com a chave pública de outra, é igualmente candidato a ser usado numa implementação do SKIP.

[73] O que, nas implementações actuais do SKIP, corresponde a identificar o endereço IP de origem.

[74] Junto de uma autoridade de certificação ou através da execução de um protocolo específico com o detentor do certificado (que é o originador do pacote).

K_p eventualmente comprometidas[75]. A mudança da chave K_{ij} pode ser induzida pela actualização das chaves pública e privada de Diffie–Hellman de qualquer uma das partes. Em relação à chave pública isso significa também a emissão de um novo certificado e a revogação do anterior. Se se desejar uma actualização rápida de K_{ij}, este processo pode introduzir demoras inaceitáveis, pelo que o SKIP opta por associar um contador n a K_{ij} e que serve, fundamentalmente, para definir a sua versão, quando gerada por um determinado processo que deverá ser comum a ambas as partes. Se o n viajar em cada pacote, então o destinatário fica a saber qual a versão de K_{ij} usada e gera, localmente, K_{ijn}. Note–se que, para além de evitar a reutilização de chaves K_{ijn} (caso tenham sido comprometidas), o incremento de n fornece também uma protecção contra *ataques–de–repetição*.

O SKIP apresenta também uma solução própria ([AMP95e]) para a distribuição de chaves num grupo onde a comunicação se faz com a variante multiponto do IP (*IP Multicast*). Tradicionalmente, supõe–se a existência de um *Centro de Distribuição de Chaves* (KDC)[76] que fornece a chave de sessão aos elementos do grupo. O tráfego multiponto é então cifrado com essa chave. Segundo [AP95], esta abordagem tem pelo menos dois inconvenientes:

1. a (re)distribuição de chaves pode consumir demasiado tempo em grupos de dimensões consideráveis. Mesmo em grupos pequenos, se a política de gestão de chaves ditar actualizações frequentes por parte do KDC, então a actividade do grupo pode ser penalizada, uma vez que a troca de mensagens poderá ficar temporariamente suspensa até que a nova chave seja disseminada por todo o grupo (a não ser que se opte pela utilização da chave antiga enquanto não se receber a nova);

2. a utilização da mesma chave por todos os elementos do grupo pode impedir a adopção de determinadas cifras de fluxo[77], conhecidas que são as vulnerabilidades desta classe de cifras no que diz respeito à análise criptográfica do seu produto cifrado, quando se efectua a reutilização de chaves[78].

O SKIP resolve estes problemas através de mecanismos semelhantes aos usados na comunicação ponto–a–ponto. Uma entidade designada por "dono do grupo" (no fundo, um KDC adaptado à gestão das chaves que circulam num grupo) detém uma Chave de Intercâmbio do Grupo (GIK)[79], que mais não é que uma *chave–de–encriptação–de–chaves*, semelhante à chave K_{ij}. Por forma a enviar mensagens cifradas para o grupo, um membro deve solicitar ao dono do grupo (cuja identidade deverá ser, antecipadamente, bem conhecida) a chave GIK, o que deverá ser feito recorrendo a comunicação ponto–a–ponto protegida pelo próprio SKIP. Com base na consulta a uma Lista de Controle de Acesso (ACL), o dono do grupo decide (ou não) fornecer a GIK. Uma vez na posse da GIK, um membro pode gerar aleatoriamente chaves K_p, específicas para cada pacote e que o acompanham, devidamente cifradas com a GIK (a qual, recorde–se, é compartilhada por todos os elementos do grupo). Assim sendo, cada membro usa uma chave K_p diferente,

[75] Uma vez que estas, recorde–se, são cifradas com K_{ij}.
[76] Do inglês *Key Distribution Center*.
[77] Do inglês *stream ciphers*.
[78] Ver [Sch96], págs. 197–199.
[79] Do inglês *Group Interchange Key*.

cujo tempo de vida fica ao seu critério. Quanto à GIK, ela poderá também ter um tempo de vida limitado, ao fim do qual todos os membros devem solicitar ao dono do grupo uma nova GIK. Naturalmente, espera–se que esse tempo de vida seja suficientemente lato para que o processo de requisição de uma nova GIK não ocorra tão frequentemente de modo a diluir as vantagens que a sua utilização pressupõe.

Sob UNIX[80], a implementação do SKIP comporta três módulos [AP95]:

1. demónio de gestão de chaves (`skip_keymgrd`), no nível do utilizador. Mantém uma base de dados com os certificados das chaves públicas de Diffie–Hellman, gera as chaves K_{ij} (e conserva–as em *cache*) e também as chaves K_p;

2. módulo genérico de encriptação/desencriptação, no núcleo do sistema operativo, que opera com base nas chaves fornecidas pelo `skip_keymgrd` ou pelo módulo de *streams*[81] (ver a seguir);

3. módulo de *streams*, também no núcleo do sistema operativo, inserido entre o IP e a interface de rede. Com base no endereço IP de destino (durante a emissão) ou de origem (durante a recepção) dos pacotes, e na política ditada pelas ACLs, este módulo solicita a encriptação ou desencriptação, respectivamente, dos pacotes IP.

Esta abordagem, claramente integrada na Camada de Rede, é completamente transparente às aplicações, que beneficiam assim de segurança nas suas comunicações sem necessidade de serem modificadas. À semelhança do swIPe, o SKIP também é uma abordagem que pode fornecer não só segurança Ligação–a–Ligação (bastando que se configurem todos os nós intermédios a fim de executarem o SKIP) como também Fim–a–Fim (em que o SKIP actua apenas na origem primária e no destino final dos pacotes), dado que, fundamentalmente, ambas as abordagens optam por mecanismos de intercepção do tráfego IP conceptualmente semelhantes. Essa semelhança estende–se também à utilização do endereço IP[82] como determinante da (des)encriptação (ou não) dos pacotes, o que pode ser indesejável para certas aplicações que não querem tirar partido da encriptação automática.

Porém, o SKIP vai mais longe ao sugerir a extensão do seu mecanismo de segurança de base à comunicação multiponto o que, em conjunção com a política de gestão de chaves, perfeitamente adaptada a aplicações *não–orientadas–à–conexão*[83], influenciou decisivamente a concepção e o desenvolvimento do S3L[84]. Recorde–se que um dos objectivos perseguidos pela presente dissertação é precisamente a segurança da comunicação em grupo, tomando o *IP Multicast* — que é *não–orientado–à–conexão* — como protocolo base de comunicação.

[80] Concretamente, em Solaris.

[81] A tradução "módulo de fluxo" perderia significado, pelo que se optou por conservar o termo *streams*.

[82] De origem ou de destino, conforme as circunstâncias.

[83] A este propósito, recorde–se o exemplo contrário fornecido pelo SSL, que necessita de trocar chaves secretas e outros parâmetros criptográficos (ou seja, necessita de criar estado) sempre que pretende estabelecer uma sessão — embora se reconheça que todas as conexões dentro dessa sessão irão beneficiar dessa troca prévia.

[84] A relação entre o SKIP e o S3L será esclarecida progressivamente ao longo da dissertação.

3.5 Análise Comparativa

As abordagens apresentadas constituem uma pequena amostra das propostas actualmente disponíveis em termos de Segurança em Sistemas Distribuídos. Um facto é indiscutível: não existe uma solução ou modelo ideal para o problema da Segurança em Sistemas Distribuídos. Abordagens de alto e de baixo nível têm os seus méritos e fraquezas próprias. Resumindo [CLA+96]:

- vantagens da segurança integrada nas aplicações:
  - as aplicações podem decidir quais os dados que deverão estar sujeitos a medidas especiais de segurança;
  - assumindo que as aplicações se encarregam de todas as operações de segurança, então não há necessidade de depositar confiança em entidades de mais baixo nível;
  - não é necessário transferir chaves para camadas mais baixas e, portanto, a visibilidade desta informação (crítica) fica limitada;
- desvantagens da segurança integrada nas aplicações:
  - ao ligar intimamente a infra–estrutura de segurança com as aplicações (na própria aplicação ou numa biblioteca), o comportamento dessa infra–estrutura é susceptível de ser influenciado por outra aplicação;
  - num contexto em que diferentes arquitecturas e algoritmos de segurança emergem rapidamente, a interoperabilidade entre aplicações é não só um requisito cada vez mais solicitado como também exige um esforço crescente por parte dos programadores de aplicações;
  - a escalabilidade das aplicações pode ficar comprometida se tomarem a seu cargo tarefas como a gestão de chaves (geração, armazenamento, troca, etc.);
- vantagens da segurança ligada à Camada de Rede:
  - mecanismos e interfaces são iguais para todas as aplicações que residem na mesma máquina. Uma vez provada a correcção e segurança desses mecanismos, todas as aplicações podem tirar deles proveito de igual forma;
  - o acesso a bases de dados (possivelmente distribuídas) de chaves é transparente, sob o ponto de vista das aplicações. Tal facilita a gestão de chaves, a qual pode ser feita com base em mecanismos mais uniformes;
  - sistemas intermediários (*e.g.*, *routers*) podem aplicar operações de segurança nos pacotes, facilitando a implementação de redes privadas virtuais e *firewalls*;
- desvantagens da segurança ligada à Camada de Rede:
  - as aplicações têm que confiar incondicionalmente na Segurança supostamente fornecida pelos níveis hierárquicos mais baixos;

- a Segurança é levada a cabo sempre da "mesma" forma, sem levar em conta o significado (que só as aplicações conhecem) dos dados, o que representa uma perda de flexibilidade em relação à aplicação de Segurança directamente pelas aplicações;
- a utilização de chaves específicas para cada aplicação/utilizador é mais difícil de concretizar. Na realidade, isso representaria uma violação da divisão que os modelos de comunicação por camadas estabelecem, uma vez que a Camada de Rede teria que levar em conta necessidades específicas de entidades da Camada de Aplicação.

Adicionalmente, é previsível que a clivagem entre abordagens de alto nível (maioritariamente ao nível da Camada de Aplicação) e baixo nível (tipicamente ao nível da Camada de Rede, em contacto íntimo com o núcleo dos sistemas operativos) se vá progressivamente esbatendo. A tendência será, portanto, para modelos que integrem os diversos aspectos da Segurança de um Sistema Distribuído nas camadas apropriadas, mas de uma forma complementar. Assim, por exemplo [CLA+96], as operações de encriptação e desencriptação seriam realizadas ao nível da Camada de Rede, usualmente integrada no núcleo do sistema operativo, o que lhes permitiria atingir níveis de desempenho adequados ao tratamento de grandes volumes de dados. Já o controlo de dispositivos de *hardware*, eventualmente dedicados a essas tarefas[85], seria feito, apropriadamente, pelo sistema operativo. Políticas globais de Segurança (tipo de algoritmos, chaves, combinações diversas de Autenticação, Integridade e Confidencialidade) podem também ser impostas a um nível suficientemente baixo de forma a que as aplicações/utilizadores não as possam contornar, se assim se desejar. Todavia, ao nível das aplicações, deverá ser feita a disponibilização de APIs, bibliotecas e utilitários que permitam um nível satisfatório de programação/configuração das necessidades específicas de Segurança de cada aplicação.

Por enquanto, e na ausência de uma abordagem que proporcione este nível de integração, há que optar. Nesse sentido, as tabelas 3.5 e 3.6 poderão ser úteis, na medida em que resumem algumas das características das abordagens tradicionais apresentadas neste capítulo.

Em relação à tabela 3.5, o significado dos termos "implícito" e "explícito" com que se classifica o Modelo de Comunicação de cada abordagem é o seguinte [WBSL94]:

- *implícito*: diz–se do modelo de comunicação para o qual as aplicações/utilizadores não necessitam de gerir os pormenores da troca de mensagens, lidando com abstracções de alto nível. A programação segundo o modelo de RPCs é um exemplo típico de uma abordagem implícita à comunicação;

- *explícito*: diz–se do modelo de comunicação para o qual as aplicações/utilizadores são responsáveis pela invocação dos serviços de transferência de dados, incluindo a possível iniciação e terminação de conexões. A programação com *sockets* constitui um exemplo paradigmático deste modelo.

[85] E a outras, como por exemplo a geração de números aleatórios.

Abordagem	Nível OSI	Modelo de Comunicação	Níveis de Segurança	Esforço de Migração
GSS–API	7 (*e.g.*, sobre RPC, Kerberos)	Independente do modelo de comunicação.	Autenticação, Integridade. e Confidencialidade.	(Re)Programação com chamadas à GSS–API.
SNP	7 (sobre GSS–API)	Explícito. (interface tipo *sockets*)	Autenticação, Integ. e Conf. (derivadas da GSS–API)	(Re)Programação com chamadas à API do SNP. KDC necessário.
SSL	>4 (7)	Explícito. (sobre *sockets* TCP)	Autenticação, Integridade e Confidencialidade.	(Re)Programação com chamadas à API do SSL.
Secure RPC	≥6	Implícito.	Autenticação. Nova variante `AUTH_KERB` fornece Integridade e Confidencialidade.	(Re)Programação segundo o modelo RPC. Adição explícita de opções de Autenticação.
DCE RPC	≥6 (sobre Kerberos)	Implícito.	Autent., Integ., Autorização, e Controle de Acesso. Confidencialidade opcional.	(Re)Programação segundo o modelo DCE RPC. KDC (Kerberos) necessário.
Kerberos	>4 (7)	Explícito (sobre *sockets*)	Autenticação, Integridade e Confidencialidade.	(Re)Programação com chamadas à API do Kerberos (kerberização). KDC (Kerberos) necessário.
swIPe	3	Implícito.	Autenticação, Integridade e Confidencialidade.	Modificação do *kernel*. Necessita de alguns demónios.
SKIP	3	Implícito.	Autenticação, Integridade e Confidencialidade.	Modificação do *kernel*. Necessita de alguns demónios.
SecuDE	>4 (7)	Explícito.	Autenticação, Integridade e Confidencialidade.	(Re)Programação com chamadas às APIs do SecuDE.

Tabela 3.5: Comparação de algumas abordagens à Segurança em Sistemas Distribuídos.

Abordagem	Algoritmos de Suporte	Disponibilidade/Exportabilidade	Portabilidade/Plataformas
GSS–API	Independente.	Especificação pública. Exportabilidade de algumas implementações condicionada.	Potencialmente boa.
SNP	"Independente" *via* GSS–API (actualmente DES, RSA e MD5)	Exportabilidade condicionada pela implementação da GSS–API sobre RSAREF.	Potencialmente boa. (via GSS–API)
SSL	DES, 3DES, RC2, RC4, IDEA, RSA, Diffie–Hellman, Fortezza, MD5, SHA.	Especificação pública. SSLRef público mas não exportável. SSLeay público e exportável (origem australiana).	SunOS, Solaris, HP–UX, IRIX DGUX, OSF, AIX, Win16/32.
Secure RPC	DES e Diffie–Hellman.	Propriedade da Sun. Licenciamento necessário.	Solaris e SunOS. NIS/NIS+ necessário.
DCE RPC	"Independente" (DES e MD5 nas implementações actuais)	Versão de domínio público sem Segurança. Versões comerciais sujeitas às restrições de exportação do DES.	Maioria das variantes UNIX. OpenVMS, Win95, WinNT, MacOS, OS/2, AS/400, etc.
Kerberos	DES. Uso de algoritmos assimétricos (*e.g.*, RSA) na fase de autenticação previsto em próximas versões.	Versão pública do MIT. Várias versões comerciais. Todas sujeitas às restrições de exportação do DES, excepto a "versão internacional" eBones (Bones+DES).	Boa (*e.g.*, versão do OSF DCE; portabilidade da versão do MIT comparável à da OSF DCE).
swIPe	Arquitectura "independente". (DES, RSA e Diffie–Hellman na implementação actual)	Especificação e implementação públicas. Restrições de exportação em função dos algoritmos criptográficos empregues.	SunOS, Mach, BSDI.
SKIP	DES, 3DES, SAFER 128SK, RC2, RC4, Diffie–Hellman, MD2, MD4, SHA, DSA.	Especificação e implementação públicas. Restrições de exportação em função dos algoritmos criptográficos empregues.	SunOS, Solaris, FreeBSD, IRIX, Windows 3.11/95/NT, JavaOS.
SecuDE	DES, 3DES, IDEA, RC2/4, RSA, DSA, DSS, Diffie–Hellman, MD2/4/5, SHA, RIPEMD160.	Código de domínio público e de utilização gratuita para fins não comerciais. Exportável (origem alemã).	SunOS, Solaris, HP–UX, AIX, IRIX, OSF1, Linux, AS/400, OS/2, Win95/NT, MSDOS 6.

Tabela 3.6: Comparação de algumas abordagens à Segurança em Sistemas Distribuídos (continuação).

3.5.1 Outras Abordagens

Existem muitas outras abordagens à Segurança em Sistemas Distribuídos (a maior parte das quais acabam por se enquadrar nas mesmas categorias das que analisámos), mas que, por limitações de espaço e de tempo, não foi possível analisar em pormenor. A título meramente ilustrativo, ficam aqui referências a algumas delas:

- *autenticação e distribuição de chaves*: SPX [TA91], KryptoKnight [MTHZ92], autenticação no sistema operativo Taos [WABL93], SESAME [KPP94];
- *segurança na Camada de Rede*: SKEME [Kra95], Photuris[KS], IPSP [Atk95a, Atk95b, Atk95c], NLSP [ISO92b];
- *segurança na Camada de Transporte*: TLSP [ISO92a], TLS [DA97];
- *bibliotecas de funções/toolkits*: Cryptolib [LMS93], Crypto++[86], Python[87], RSAEURO[88], RSAREF[89], CryptoAPI[90], BSAFE[91];
- *aplicações*: secure shell (ssh)[92], Pretty Good Privacy (PGP) [Zim95] [93], Privacy Enhanced Mail (PEM) [Lin93a, Ken93, Bal93, Kal93].

3.6 Oportunidade do S3L

Recorde–se que era objectivo inicial desta dissertação fornecer uma contribuição para a resolução do problema da comunicação segura num grupo *IP Multicast*. Das abordagens apresentadas neste capítulo, só o SKIP propõe soluções concretas a esse nível.

O que poderá então justificar a elaboração de um novo protocolo (o S3L) a fim de resolver o mesmo problema, com a "agravante" de que esse novo protocolo assume[94] a adopção de muitos dos mecanismos de segurança sugeridos pelo SKIP?

Fundamentalmente:

- o SKIP é um protocolo de baixo nível que, devido à abordagem transparente que assume, leva a cabo uma política de "tudo ou nada" em relação à Segurança. Como já constatámos, no SKIP, as ACLs baseiam–se apenas no endereço IP. Assim, a possibilidade de que as aplicações/utilizadores possam escolher, à medida das suas necessidades específicas, o nível de Segurança que desejam (Autenticação, Integridade e Confidencialidade, combinadas de várias formas) constitui um aspecto em que o SKIP é passível de aperfeiçoamento;

[86] `http://www.eskimo.com/~weidai/cryptlib.html`
[87] `http://www.python.org/~pct/`
[88] `http://www.sourcery.demon.co.uk/index.html`
[89] `ftp://ftp.rsa.com/rsaref`
[90] `http://www.microsoft.com/security/tech/misf6.htm`
[91] `http://www.rsa.com/rsa/prodspec/bsafe/bsafe_3_0_f.htm`
[92] `http://www.cs.hut.fi/ssh`
[93] Ver também `http://www.ifi.uio.no/pgp`
[94] Como se poderá constatar ao longo dos próximos capítulos.

- uma vez que o SKIP actua apenas ao nível dos pacotes IP, não foi concebido para uma utilização mais genérica, que contemple outros objectos de um Sistema Distribuído. Assim, a extensão dos mecanismos de Segurança do SKIP a outro tipo de aplicações poderá ser bem–vinda. No fundo, trata–se de aplicar o encapsulamento que o SKIP efectua sobre um pacote IP (e que lhe garante Autenticação, Integridade e Privacidade) a objectos de outras categorias;

- o SKIP necessita de chaves públicas Diffie–Hellman devidamente certificadas. Ora, por um lado, a certificação é um processo que ainda carece de estruturas nacionais (ou mesmo internacionais) devidamente implementadas e suficientemente divulgadas. Por outro, a haver certificação, ela efectua–se predominantemente com base em assinaturas RSA e, usualmente, sobre chaves também RSA[95]. O SKIP usa certificados X.509 com assinaturas feitas segundo o algoritmo *Digital Signature Standard* (DSA) [oST94a] sobre chaves públicas de Diffie–Hellman, o que contraria de alguma forma a prática de certificação corrente[96]. Aliás, a utilização do DSA em vez do RSA não é isenta de polémicas[97]. Por estes motivos, a concepção de mecanismos alternativos, no contexto do SKIP, capazes de ultrapassar estas limitações afigura–se desejável. Como veremos, o S3L "resolve o problema" dispensando a certificação de chaves públicas de Diffie-
  -Hellman, embora dependendo da certificação de chaves públicas RSA;

- o SKIP tem, como seria de esperar, restrições de exportação. A utilização do SKIP tal como originalmente implementado pela Sun[98] é assim impraticável fora dos Estados Unidos e do Canadá[99]. Resta então recodificá–lo, com base na sua especificação[100] e na utilização de versões de domínio público (e disponíveis na Europa) dos algoritmos que usa. Contudo, e atendendo aos motivos apresentados anteriormente, não foi essa a decisão do autor.

Existem ainda outras insuficiências do SKIP, estas ligadas ao estado actual das suas implementações. Assim, e em relação à implementação da Sun para a versão 1.0[101], temos:

- a chave privada de Diffie–Hellman é guardada em disco não cifrada;

- o protocolo que permite a um sistema descobrir os algoritmos criptográficos que outro suporta ainda não está implementado. Essa negociação tem que ocorrer *fora–de-*

[95] Embora se reconheça que, tecnicamente, nada impede a submissão a uma autoridade de certificação de um protótipo de certificado contendo uma chave pública Diffie–Hellman.

[96] O SKIP reconhece a presença de um certificado com uma assinatura RSA mas não efectua a sua verificação.

[97] Ver [Sch96], págs. 484–485.

[98] Ver `http://skip.incog.com`.

[99] Muito recentemente, o SKIP passou a ter versões exportáveis que, todavia, utilizam algoritmos de menor qualidade (RC2 e RC4 de 40 bits).

[100] O que aliás acabou por ser feito na Suiça (`http://www.tik.ee.ethz.ch/%7Eskip/`) e na Rússia (`http://www.elvis.ru`). Neste último caso, a implementação da firma Elvis+ — sem quaisquer limitações em termos algorítmicos — consta, desde Maio de 1997, do "catálogo" de produtos da própria Sun, artifício que lhe permitiu tornear as ditas restrições.

[101] Versão em vigor à data do desenvolvimento desta tese. Entretanto, segundo `http://www.sun.com/sunworldonline/swol-06-1997/swol-06-skip.html`, a versão mais recente (1.1) não trouxe alterações significativas.

-banda[102];

- o protocolo de descoberta de certificados[103] [AMP95a] baseia–se em mensagens não cifradas.

O S3L, que *grosso modo* é uma transposição de muitas das funcionalidades do SKIP para a Camada de Aplicação, não padece destas insuficiências.

Uma vez definido o SKIP como ponto de partida teórico para a concepção do S3L, apontam–se de seguida as razões que ditaram a escolha do SecuDE como *kit* criptográfico de apoio à sua codificação:

- código de domínio público e utilização gratuita para fins não comerciais;
- isento de problemas de patentes e restrições à exportação (origem alemã);
- ambiente rico em termos da variedade de algoritmos suportados e de APIs fornecidas;
- boa portabilidade, cobrindo uma grande variedade de ambientes Unix, Wintel, OS/2, etc.;
- documentação bastante completa e de boa qualidade [Sch95a, Sch95b, Sch95d, Sch95c], o que não é muito frequente em pacotes de domínio público;
- bom nível de protecção da informação criptográfica associada a um utilizador, através do PSE[104];
- boa gestão da certificação X.509, com integração (opcional) com o Serviço de Directoria X.500;
- boas perspectivas de que a sua solução de certificação seja adoptada no nosso país, nomeadamente ao nível da FCCN (provisoriamente o topo da hierarquia de certificação nacional), no âmbito do projecto europeu de certificação ICE–TEL[105].

Saliente–se que a escolha do SecuDE só foi feita após a análise de outros *kits* similares, alguns dos quais são referenciados na secção 3.5.1 e cuja descrição, tal como foi ali referido, não foi viável enquadrar neste capítulo.

Os próximos capítulos dedicam–se, exclusivamente, à descrição do processo de concepção e implementação do S3L.

[102] Do inglês *out–of–band*.

[103] Basicamente, este protocolo permite que um sistema solicite a outro o certificado da sua chave pública de Diffie–Hellman. No próximo capítulo, este protocolo será confrontado com a solução própria que o S3L propõe.

[104] Muitos dos *kits* de domínio público concentram–se quase exclusivamente na disponibilização dos algoritmos criptogáficos, descurando ou mesmo ignorando a segurança no armazenamento de chaves e outra informação crítica.

[105] `http://www.fccn.pt/ice--tel`.

Capítulo 4

Skeyx: Gestão de Chaves no S3L

A génese do S3L envolveu, essencialmente, a concepção e o desenvolvimento de três protocolos:

1. o *Secure Key Exchange* (Skeyx);
2. a versão ponto–a–ponto do S3L;
3. as extensões multiponto do S3L.

Em termos arquitecturais, a relação existente entre os componentes fundamentais do S3L pode ser observada na representação de alto nível fornecida pela figura 4.1.

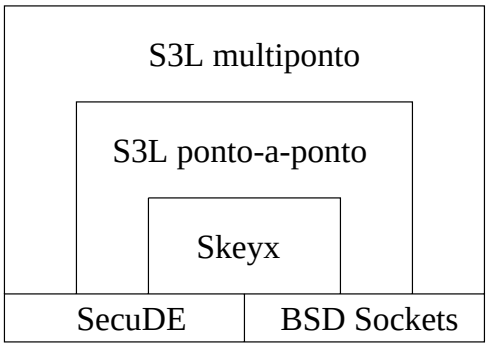

Figura 4.1: Arquitectura do S3L.

Assim, relativamente às comunicações, a base é fornecida pelos *sockets* de Berkeley. Sob o ponto de vista criptográfico, o SecuDE proporciona as funcionalidades básicas necessárias, não só no que diz respeito a APIs, como também em termos de um ambiente de operação seguro. Segue–se o Skeyx, protocolo que foi desenvolvido para permitir a troca segura de chaves públicas de Diffie–Hellman (não certificadas), sem as quais não seria possível ao S3L (em ambas as versões ponto–a–ponto e multiponto) operar. Note–se ainda que a versão multiponto do S3L pressupõe a disponibilidade da versão ponto–a–ponto[1]. As versões ponto–a–ponto e multiponto do S3L serão discutidas com mais pormenor em capítulos subsequentes. Este capítulo foca apenas os pormenores de concepção e desenvolvimento relativos ao Skeyx.

[1]Algumas tarefas de gestão de um grupo assentam em comunicação segura ponto–a–ponto.

4.1 Necessidade do Skeyx

À semelhança do SKIP, o S3L depende do acordo de Diffie-Hellman como ponto de partida para a geração de chaves específicas para cada mensagem individual de uma comunicação ponto–a–ponto. Adicionalmente, e uma vez que as extensões multiponto do S3L dependem, em parte, das funcionalidades ponto–a–ponto, torna–se visível a importância que aquele acordo assume no contexto do S3L.

Para que um acordo de Diffie–Hellman seja levado a cabo com segurança, é necessário que:

1. os parâmetros públicos de Diffie–Hellman[2], com base nos quais as chaves são geradas, sejam publicamente conhecidos e distribuídos de forma segura (*e.g.*, distribuição manual ou *via* PEM) às partes interessadas;

2. uma vez geradas as chaves pública e privada de Diffie–Hellman, cada parte envolvida na obtenção de um acordo esteja segura de que a chave pública de Diffie–Hellman da outra é autêntica e não um logro. Ou seja as chaves públicas de Diffie–Hellman devem ser certificadas.

Um bom mecanismo candidato ao cumprimento destes dois requisitos é o Serviço de Directoria X.500 [CCI88]. Os parâmetros de Diffie–Hellman seriam obtidos através duma consulta à Directoria, o mesmo ocorrendo com as chaves públicas de Diffie–Hellman, devidamente certificadas, segundo a norma X.509, por exemplo. Obviamente, as consultas à Directoria deveriam ser protegidas de forma a evitar ataques do tipo *homem–no–meio*. Um entrave a uma solução deste género reside na pouca divulgação de que o Serviço de Directoria (ainda) dispõe[3], com a agravante de que eventuais versões seguras desse serviço serão, inicialmente, de uso ainda mais restrito.

Outras hipóteses para a certificação das chaves públicas passariam, por exemplo, pelos "certificados PGP" ou por *resource records* do Secure DNS [EK95].

A certificação e a distribução de chaves públicas constitui assim uma questão para a qual não existe (e nem se vislumbra a breve trecho) uniformidade em termos de tratamento. A ausência de uma estrutura de certificação universal traduz–se, naturalmente, em problemas de interoperabilidade e de confiança. Uma entidade poderá argumentar que não reconhece como válidos certificados de formato diferente do seu ou provenientes de uma infra–estrutura de certificação à qual não pertence. Naturalmente, poderá ser possível a concepção de um esquema de certificação cruzada entre as duas infra–estruturas, mas, quando as CAs[4] são de natureza diferente, mesmo que os problemas de confiança se resolvam, colocam–se inevitavelmente problemas de interoperabilidade entre formatos e domínios de nomeação diferentes.

No contexto do SKIP foi proposto o *Certificate Discovery Protocol* (CDP) [AMP95a] a fim de lidar com esta heterogeneidade. O CDP foi concebido de forma a permitir a pesquisa de certificados junto de fontes tão diversas como o Serviço de Directoria, o Serviço de Nomes (DNS) ou até mesmo junto do próprio sujeito de um certificado. Adicionalmente,

[2] Um número primo grande, p, e um número gerador pequeno, g, que é uma *raiz primitiva* de p ([Sta95] oferece um breve resumo das fundações matemáticas do acordo de Diffie–Hellman).

[3] Embora, em princípio, qualquer outra base de dados distribuída, desde que segura, seja ainda um bom candidato.

[4] Do inglês *Certificate Authorities*.

o CDP é capaz de lidar com uma boa variedade de formatos de certificados e domínios de identificação.

Apesar da generalidade com que lida com o problema da pesquisa de certificados, a especificação do CDP é imprecisa em determinados aspectos, nomeadamente no que diz respeito à obrigatoriedade da assinatura digital das mensagens CDP como forma de assegurar a sua Autenticidade e Integridade[5]. Aparentemente, a assinatura não é obrigatória, sendo possivelmente considerada um pormenor de implementação.

Certamente que é discutível a eficácia de um ataque do tipo *homem–no–meio* sobre certificados em relação aos quais a chave pública da respectiva autoridade de certificação se supõe bem conhecida e não comprometida. Em princípio, tentativas de manipulação dos certificados por uma entidade intermédia serão prontamente detectadas. Saliente-se porém que, a ser bem sucedido, um ataque aos certificados durante uma troca CDP coloca em risco toda a segurança do SKIP, uma vez que é com base nas chaves públicas de Diffie–Hellman transportadas nesses certificados que se geram as chaves acordadas, as quais servem de base à segurança das mensagens SKIP.

Além disso, a identificação dos participantes (iniciador e respondedor) numa troca CDP resume–se ao seu endereço IP e a um de dois portos UDP bem conhecidos (conforme se trate da emissão ou recepção). Consequentemente, o mecanismo de *cookies* previsto pelo CDP[6] como forma de combate a *ataques–de–repetição* e *de–saturação*, bem como para a identificação unívoca das trocas[7], suscita alguma dúvidas quanto à sua eficácia, dado que não é possível identificar com precisão a entidade originadora, cuja utilização do porto UDP de emissão é efémera[8]. Assim, por um lado, a rejeição de mensagens CDP originadas num determinado par (endereço IP, porto UDP) pode ser injusta, porque entretanto a entidade que fazia uso daquele porto pode ter mudado. Por outro lado, uma entidade à escuta no mesmo par (endereço IP, porto UDP) que outra pode apoderar–se do *cookie* que lhe é destinado e, antecipando–se à entidade legítima, usar esse *cookie* numa resposta que o destinatário julgará proveniente da entidade legítima (a quem o *cookie* era destinado originalmente). Mesmo que essa entidade legítima ainda envie a sua resposta, ela será ignorada porque o *cookie* será visto como uma repetição do enviado pelo impostor.

Obviamente, a utilização do UDP, o qual não oferece garantias de entrega, constitui uma preocupação adicional, em termos de implementação do CDP, uma vez que é necessário o tratamento explícito de situações de retransmissão[9].

O CDP contempla também a troca de chaves públicas de Diffie–Hellman, não certificadas pelos métodos "tradicionais" (*e.g.*, X.509), desde que codificadas segundo a especificação prevista em [AMP95b]. Nesta especificação, a associação entre uma entidade e uma chave pública de Diffie–Hellman baseia–se na utilização de um *hash*[10] da chave[11]

[5] A Encriptação não é necessária uma vez que toda a informação trocada é de domínio público.

[6] No contexto do qual um *cookie* consiste num *hash* MD5 dos endereços IP de origem e destino, portos UDP de origem e destino e um valor aleatório gerado localmente.

[7] Sendo neste aspecto que o conceito de *cookie* do CDP mais se aproxima do originalmente introduzido pela Netscape (`http://www.netscape.com/newsref/std/cookie_spec.html`) e segundo o qual os *cookies* complementam o carácter *stateless* do *Hypertext Transfer Protocol* (HTTP) [FIG+97] ao permitir a salvaguarda de informação de estado, quer no cliente, quer no servidor.

[8] Como veremos, no Skeyx este problema não se coloca porque a identificação das entidades participantes se baseia no seu nome distinto.

[9] A este propósito, refira–se desde já que o Skeyx usa o TCP.

[10] Gerado *via* MD5, por exemplo.

[11] E de informação relevante associada, como os parâmetros públicos de Diffie–Hellman e os prazos de

como identificador dessa entidade. Todavia, não só esses identificadores são, na maioria dos casos, pouco amigáveis, como a sua utilização pressupõe a sua distribuição segura, *fora–de–banda*. Outra desvantagem evidente da sua utilização é a de que a mudança de qualquer elemento em que se baseia o *hash* implica o seu recálculo e logo a mudança da identidade de uma entidade. Consequentemente, seria necessário repropagar a nova identidade e a chave pública da entidade.

As restrições apontadas ao CDP podem assim considerar–se algo limitativas no que concerne à divulgação e utilização de chaves públicas de Diffie–Hellman não certificadas[12], e justificam, por si só, a busca de uma solução alternativa, a qual, na presente dissertação conduziu à concepção e desenvolvimento do Skeyx.

O Skeyx "perde" em relação ao CDP em termos de generalidade, para além de estar intimamente ligado a um modelo de segurança imposto pelo SecuDE. Porém, ao recorrer à utilização de assinaturas digitais minimiza alguns dos receios suscitados em relação ao CDP. Acresce ainda que o Skeyx é independente de qualquer serviço de bases de dados distribuídas (como o Serviço de Directoria), colocando em cada entidade a responsabilidade de manter em execução um servidor individual de chaves públicas, bem como de gerir a sua *cache*.

O Skeyx combina ainda outra funcionalidade em paralelo com a troca de chaves não certificadas de Diffie–Hellman, concretamente, a negociação dos algoritmos simétricos suportados por ambas as partes[13]. Pelo contrário, no SKIP encontramos estas duas actividades realizadas por protocolos individualizados e que ocorrem em circunstâncias diferentes. Assim, se por um lado a aquisição[14] do certificado da chave pública de Diffie–Hellman da entidade remota deve preceder, obrigatoriamente, o envio da primeira mensagem SKIP para essa entidade, já a negociação de algoritmos, através de outro protocolo específico (o *Algorithm Discovery Protocol* (ADP) [AMP95d]), tem lugar *à posteriori*, *por necessidade*, quando o receptor da mensagem constata que não suporta um ou mais dos algoritmos que presidiram à assemblagem da mensagem SKIP que recebeu. O Skeyx tanto pode ocorrer *por antecipação*, executado de uma forma explícita, antes de qualquer troca S3L, como *por necessidade*, executado de uma forma implícita e automática[15]. Em ambos os casos, a totalidade do estado (chaves e algoritmos) necessário à operação subsequente do S3L entre as entidades em questão fica definido (salvaguardado em *cache*), dispensando–se qualquer negociação posterior entre essas entidades. É por esta razão que a designação *stateless* se aplica ao S3L.

Antes de prosseguir com a descrição do Skeyx, convém referir que a execução deste protocolo pressupõe a realização prévia da configuração descrita no apêndice A, até ao passo A.2.2, inclusive, o que é doravante assumido.

---

validade.

[12] Situação mais provável que a sua certificação.

[13] A negociação de outras qualidades de serviço (como algoritmos de MAC — recordar a secção 2.5.2 — e de compressão) seria também viável, mas não foi concretizada na versão actual do Skeyx. Por simplicidade assumem-se algumas dessas qualidades por omissão.

[14] Neste caso realizada através do CDP.

[15] Imediatamente antes do envio ou após a recepção de uma mensagem S3L. Este modo de operação será apresentado na secção 5.4.

4.2 Modelo de Operação do Skeyx

O Skeyx é um protocolo que, com base na certificação das suas chaves públicas RSA, permite a duas entidades efectuarem a troca das respectivas chaves públicas de Diffie–Hellman, não certificadas, ao mesmo tempo que se informam mutuamente sobre os algoritmos simétricos suportados.

A figura 4.2 constitui uma representação da troca de mensagens entre duas entidades, efectuada no contexto do Skeyx e que, basicamente, segue o modelo Cliente–Servidor.

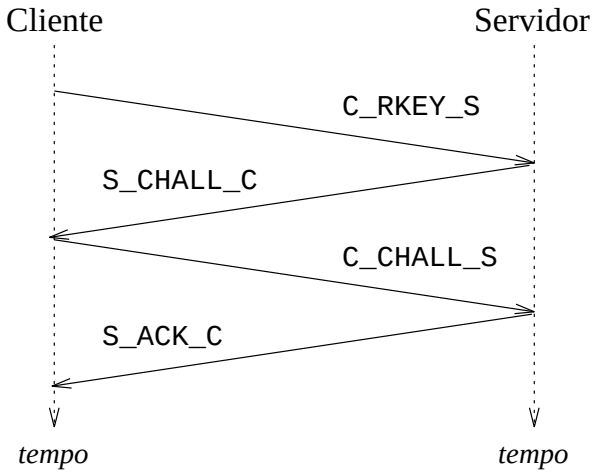

Figura 4.2: Modelo de operação do Skeyx.

Resumidamente, um Cliente inicia o protocolo Skeyx pelo envio de uma mensagem do tipo *Client_RequestKey_to_Server* (`C_RKEY_S`) a um determinado Servidor. Através dessa mensagem, o Cliente fornece ao Servidor a sua chave pública de Diffie–Hellman e a lista dos algoritmos simétricos suportados. Implicitamente, o Cliente solicita ao Servidor o mesmo tipo de informação, que lhe é devolvida numa mensagem do tipo *Server_Challenge_Client* (`S_CHALL_C`), conjuntamente com um desafio (um número aleatório cifrado com uma chave simétrica derivada da chave provisoriamente[16] acordada de Diffie–Hellman) lançado ao Cliente para que este prove a posse da chave privada de Diffie–Hellman correspondente à chave pública enviada ao Servidor. O Cliente deriva também uma chave simétrica a partir da chave acordada de Diffie–Hellman provisória e de seguida decifra com ela o desafio e devolve–o ao Servidor numa mensagem do tipo *Client_Challenge_Server* (`C_CHALL_S`). Através dela responde ao desafio do Servidor ao mesmo tempo que lhe lança também um desafio em moldes semelhantes (embora baseado noutro número aleatório). Comprovada a correcção da resposta ao desafio lançado ao Cliente, o Servidor responde ao desafio daquele com uma mensagem do tipo *Server_Acknowledge_Client* (`S_ACK_S`).

De seguida, analisa–se a estrutura de cada uma das mensagens do Skeyx, em paralelo com uma descrição mais detalhada da operação desse protocolo.

[16] A chave acordada é, sob o ponto do vista do Servidor, provisória porque, quando o Servidor a gerou, ainda não tinha a certeza de que a chave pública de Diffie–Hellman que recebeu e usou era do Cliente. De igual forma, o Cliente também considerará provisória a chave acordada que irá gerar com a suposta chave pública do Servidor, até que este responda, com sucesso, ao desafio que o Cliente lhe irá colocar.

4.2.1 Requisição da Chave Pública do Servidor (C_RKEY_S)

A iniciativa da execução do Skeyx por parte de um Cliente resulta, em primeira instância, na assemblagem e envio, ao Servidor, de uma mensagem do tipo *Client_RequestKey_to_Server* (C_RKEY_S), cuja estrutura pode ser observada na figura 4.3.

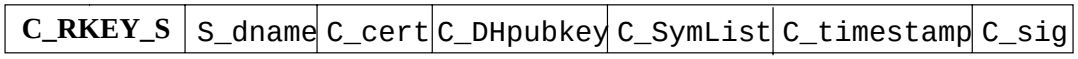

Figura 4.3: Estrutura de uma mensagem C_RKEY_S.

Os campos constituintes de uma mensagem C_RKEY_S são:

- C_RKEY_S: cabeçalho, com o identificador do tipo da mensagem;
- S_dname: nome distinto X.500 do Servidor;
- C_cert: certificado X.509 da chave pública RSA do Cliente;
- C_DHpubkey: chave pública de Diffie–Hellman do Cliente;
- C_SymList: lista dos algoritmos simétricos de encriptação suportados pelo Cliente;
- C_timestamp: marca temporal gerada pelo Cliente;
- C_sig: assinatura digital que cobre a totalidade da mensagem.

O nome distinto S_dname identifica, inequivocamente, o destino da mensagem C_RKEY_S, sendo por isso usado pelo Servidor para fazer *Autenticação do Destino*. A identificação explícita do destinatário é feita também nas restantes mensagens do protocolo Skeyx e justifica–se, primariamente, pelo facto de que as mensagens do Skeyx não são cifradas. Havendo encriptação das mensagens (e supondo que ocorresse posteriormente à sua assinatura), então, no receptor, a confirmação da assinatura após a desencriptação confirmaria, sem margem para dúvidas, que o receptor era o destinatário originalmente pretendido[17].

Uma mensagem C_RKEY_S não é cifrada porque nenhum dos campos constituintes transporta informação suficientemente sensível[18] para justificar uma encriptação (lenta) com a chave pública RSA do destinatário. Por seu turno, a disponibilidade da chave pública RSA do destinatário implicaria a resolução do problema da sua aquisição/distribuição, previamente à (primeira) invocação do Skeyx. Pela troca de informação de domínio público nas sua mensagens, o Skeyx evita esse problema. A ausência de encriptação verifica–se também nos outros tipos de mensagens do Skeyx[19], pelos mesmos motivos apontados para as mensagens C_RKEY_S.

Uma vez que o Cliente espera que as mensagens a receber sejam originadas no Servidor cujo nome distinto é S_dname, convém conservar esse nome para ser utilizado posteriormente nessa verificação (bem como na indicação do destino da mensagem C_CHALL_S). Esta e outra informação de estado de uma conexão Skeyx entre um Cliente e um Servidor é preservada numa estrutura SkeyxSessionInfo (ver figura 4.4), que é progressivamente

[17] O segundo protocolo apresentado na secção 2.5.3 traduz esta situação.
[18] Por exemplo, campos como C_cert e C_DHpubkey são, obviamente, de domínio público.
[19] Exceptuando os campos com os desafios que, no entanto, são cifrados com chaves simétricas.

preenchida por ambos os extremos da conexão, ao longo da execução do protocolo (ver figura 4.8). No caso do nome distinto `S_dname`, é o campo `loc_destiny` que assume o seu valor.

```
typedef struct {

 /* informacao relativa a entidade local */

 Certificates *loc_certs;       /* certificado X.509 da entidade local */
 OctetString  *loc_dhpubkey;     /* chave publica DH da entidade local */
 char   *loc_symlist;       /* algoritmos simetricos da entidade local */
 char   *loc_symalg;       /* algoritmo simetrico escolhido localmente */
 char   *loc_destiny; /* nome distinto do destino das mensagens locais */
 struct timeval loc_time;   /* ultima marca temporal gerada localmente */
 struct timeval loc_time_echoed; /* marca temporal local retornada
                                    pela entidade remota */
 char *loc_challenge;       /* desafio local enviado a entidade remota */
 char *loc_challenge_echoed; /* desafio local retornado pela entidade
                                remota */

 /* informacao relativa a entidade remota */

 char         rem_ipaddr[16];        /* endereco IP da entidade remota */
 Certificates *rem_certs;      /* certificado X.509 da entidade remota */
 OctetString  *rem_dhpubkey;    /* chave publica DH da entidade remota */
 char   *rem_symlist;      /* algoritmos simetricos da entidade remota */
 char   *rem_symalg;      /* algoritmo simetrico escolhido remotamente */
 char   *rem_destiny; /*nome distinto do destino das mensagens remotas */
 struct timeval rem_time_1;    /* 1a marca temporal da entidade remota */
 struct timeval rem_time_2;    /* 2a marca temporal da entidade remota */
 char   *rem_challenge;        /* desafio enviado pela entidade remota */
 char   *rem_entry;      /* entrada da entidade remota, na cache local */

 /* informacao relativa a ambas as entidades */

 BitString  *dhagreedkey;         /* chave acordada de Diffie--Hellman */

} SKeyxSessionInfo;
```

Figura 4.4: Informação de estado de uma sessão Skeyx.

O certificado `C_cert` serve dois propósitos: o Servidor poderá usá–lo para confirmar a assinatura digital `C_sig`; adicionalmente, o certificado transporta a identificação (nome distinto X.500) do Cliente. Note–se que ao receber este certificado com a própria mensagem, o Servidor não necessita de o requisitar a outra fonte.

Uma vez que é provável que o Servidor não disponha da chave pública de Diffie–Hellman do Cliente (a menos que o Skeyx tenha sido executado anteriormente), o Cliente aproveita o pedido da chave do Servidor para lhe fornecer a própria chave (`C_DHpubkey`). Desta forma, garante–se que, no fim da execução do Skeyx, ambas as partes estarão em condições de calcular a chave acordada.

Por outro lado, não é obrigatório que o Cliente e o Servidor suportem o mesmo conjunto de algoritmos simétricos de encriptação. Assim, paralelamente à troca de chaves públicas de Diffie–Hellman, ocorre uma negociação dessas "qualidades de serviço". O objectivo é fazer com que qualquer contacto futuro entre as entidades associadas ao Cliente e ao Servidor seja protegido com um algoritmo simétrico retirado da intersecção dos conjuntos suportados por ambos. Os frutos desta negociação serão visíveis ainda durante o próprio Skeyx, dado que os desafios lançados entre o Cliente e o Servidor serão cifrados com um algoritmo retirado dessa intersecção.

A marca temporal `C_timestamp`, gerada pelo Cliente, serve dois objectivos: identifica a mensagem `C_RKEY_S` como única, prevenindo *ataques–de–repetição* sobre o Servidor (o qual mantém um registo, por origem, da última marca temporal válida — ver secção 4.5.1); adicionalmente, faz a associação entre uma mensagem `C_RKEY_S` e a resposta `S_CHALL_C` respectiva[20], onde se espera que essa marca temporal seja devolvida, pelo que convém guardar uma cópia de `C_timestamp`, a fim de fazer, posteriormente, essa verificação. Assim, a marca temporal `C_timestamp` é preservada no campo `loc_time` da estrutura `SKeyxSessionInfo`.

A assinatura digital `C_sig` é produzida pela encriptação de um *hash* (do tipo MD5) da totalidade da mensagem, com a chave privada RSA do Cliente. Uma vez que a chave pública correspondente viaja no certificado `C_cert`, o Servidor poderá verificar facilmente a assinatura. De acordo com uma classificação apresentada em 2.5.3, o Skeyx é pois um protocolo de Assinatura Digital Directa, dado que apenas os originadores e receptores directos das mensagens estão envolvidos nos processos de assinatura e de verificação.

4.2.2 Recepção de Mensagens `C_RKEY_S`

No Servidor, a recepção de uma mensagem `C_RKEY_S` é seguida da sua Autenticação e verificação da Integridade. Na realidade, ambos os processos se aplicam, em primeiro lugar, ao certificado `C_cert`, recorrendo–se para o efeito à chave pública da autoridade de certificação, que é suposto o Servidor possuir[21]. A partir do momento em que o Servidor pode confiar no certificado `C_cert`, então utiliza a chave pública contida nesse certificado para verificar a assinatura `C_sig`. Se essa verificação for bem sucedida, então conclui–se que:

1. o originador da mensagem é o sujeito do certificado X.509 `C_cert`. Só ele dispõe da chave privada RSA capaz de gerar a assinatura `C_sig` de forma a que esta seja verificada pela chave pública contida no certificado. Assim, garante–se Não–Repudiação da mensagem por parte do seu originador;

2. a mensagem não foi adulterada em trânsito, ou seja, comprova–se a sua Integridade.

A garantia de Não–Repudiação é suficiente, em termos de *Autenticação da Origem*, para um Servidor, no contexto da recepção de uma mensagem `C_RKEY_S`. Com efeito, um Servidor não espera essa categoria de mensagens de um determinado Cliente específico, logo não necessita de comparar a origem dessas mensagens com uma (hipotética) origem desejada. O mesmo não se passa com as restantes mensagens do protocolo Skeyx, em que

[20] Neste contexto, as marcas temporais são usadas como números de sequência.

[21] Desde que os procedimentos relativos à sua aquisição, descritos na secção A.2.1 do apêndice 8, tenham sido previamente executados.

um destinatário espera uma mensagem de um originador bem definido e a verificação desse facto passa a ser uma componente necessária do processo de *Autenticação da Origem*.

O Servidor verifica ainda, através da comparação do campo `S_dname` com o seu nome distinto X.500, se constitui o destino original da mensagem (ou seja, faz *Autenticação do Destino*), ignorando–a se isso não acontecer.

Uma vez aceite a mensagem `C_RKEY_S`, o Servidor actualiza a sua instância da estrutura `SKeyxSessionInfo`, que armazena o estado da conexão Skeyx com o Cliente. A informação contida em `S_dname`, `C_cert`, `C_DHpubkey`, `C_SymList` e `C_timestamp` é transferida para os campos `rem_destiny`, `rem_certs`, `rem_dhpubkey`, `rem_symlist` e `rem_time_1`, respectivamente (rever figura 4.4).

4.2.3 Desafio do Servidor ao Cliente (`S_CHALL_C`)

A assemblagem da resposta do Servidor ao Cliente tem então lugar. Essa resposta consiste numa mensagem *Server_Challenge_Client* (`S_CHALL_C`) através da qual, para além de fornecer a sua chave pública de Diffie–Hellman e os seus algoritmos simétricos, o Servidor desafia o Cliente a demonstrar que a chave pública de Diffie–Hellman não certificada, `C_DHpubkey`, é verdadeiramente sua.

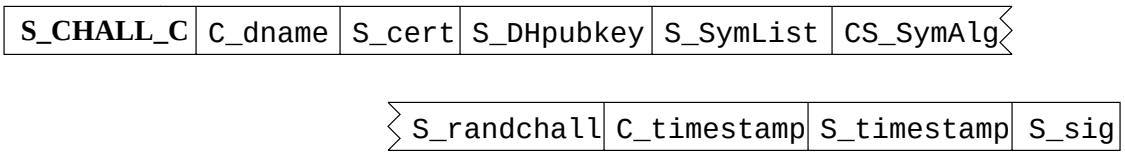

Figura 4.5: Estrutura de uma mensagem `S_CHALL_C`.

Os campos de uma mensagem `S_CHALL_C`, para a qual a figura 4.5 oferece uma representação, são:

- `S_CHALL_C`: cabeçalho, com o identificador do tipo da mensagem;
- `C_dname`: nome distinto X.500 do Cliente;
- `S_cert`: certificado X.509 da chave pública RSA do Servidor;
- `S_DHpubkey`: chave pública de Diffie–Hellman do Servidor;
- `S_SymList`: lista dos algoritmos simétricos de encriptação suportados pelo Servidor;
- `CS_SymAlg`: um algoritmo simétrico comum ao Cliente e ao Servidor;
- `S_randchall`: um desafio aleatório, cifrado com uma chave simétrica, segundo o algoritmo `CS_SymAlg`;
- `C_timestamp`: marca temporal do Cliente, oriunda da mensagem `C_RKEY_S`;
- `S_timestamp`: marca temporal gerada pelo Servidor;
- `S_sig`: assinatura digital que cobre a totalidade da mensagem.

O papel dos campos `C_dname`, `S_cert`, `S_DHpubkey`, `S_SymList` e `S_sig` é análogo ao dos campos homónimos da mensagem `C_RKEY_S`.

O campo `CS_SymAlg` representa um algoritmo extraído da intersecção dos conjuntos de algoritmos simétricos suportados pelo Cliente e pelo Servidor. Os algoritmos `IDEA`, `desCBC3` e `DES-CBC`[22] são, por esta ordem de prioridade, preferencialmente escolhidos. Caso nenhum deles seja suportado, outro algoritmo comum é escolhido. Se a intersecção for vazia, o protocolo Skeyx termina[23] por ter sido impossível a "negociação" de um conjunto mínimo de qualidades de serviço.

O desafio `S_randchall` consiste num valor aleatório, de dimensão fixa[24], que o Servidor cifra com uma determinada chave simétrica, sob o algoritmo `CS_SymAlg`. Essa chave simétrica é derivada de um subconjunto dos bits da chave acordada de Diffie–Hellman, a qual se gera, no Servidor, pela combinação da sua chave privada de Diffie–Hellman com a pretensa chave pública `C_DHpubkey` do Cliente. Se o Cliente for capaz de devolver o desafio decifrado ao Servidor, então prova a posse da chave privada correspondente a `C_DHpubkey` e portanto o Servidor pode confiar na associação da chave `C_DHpubkey` ao Cliente.

O campo `C_timestamp` é preenchido com a marca temporal guardada no campo `rem_time_1` do estado do Servidor e que é precisamente a marca temporal `C_timestamp` proveniente na mensagem `C_RKEY_S`. Desta forma, o Cliente pode não só associar univocamente a resposta `S_CHALL_C` ao seu pedido `C_RKEY_S`, como detectar *ataques–de–repetição*.

A marca temporal `S_timestamp`, gerada pelo Servidor, cumpre o mesmo papel que `C_timestamp` para o Cliente. O Servidor espera–a de volta, na resposta `C_CHALL_S` ao seu desafio `S_CHALL_C`.

Em resultado da assemblagem da mensagem `S_CHALL_C`, os campos `loc_destiny`, `loc_symalg`, `loc_challenge`, `loc_time` e `dhagreedkey` do estado do Servidor são actualizados. Para os quatro primeiros, os valores correspondem, respectivamente, aos dos campos `C_dname`, `CS_SymAlg`, `S_randchall` (versão cifrada) e `S_timestamp` da mensagem `S_CHALL_C`.

4.2.4 Recepção de Mensagens `S_CHALL_C`

No Cliente, a Autenticação e a verificação da Integridade das mensagens `S_CHALL_C` processam–se em moldes semelhantes aos que ocorrem para as mensagens `C_RKEY_S` recebidas no Servidor.

Assim, em primeiro lugar, comprova–se a Autenticidade do certificado `S_cert`[25]. Tal permitirá a utilização da chave pública aí contida na verificação da assinatura digital `S_sig`, processo que permite assegurar a Não–Repudiação da mensagem e concluir da sua Integridade.

Contudo, para o Cliente não basta saber que a mensagem `S_CHALL_C` recebida é originada no sujeito do certificado `S_cert` e que não foi adulterada em trânsito. O Cliente espera uma resposta `S_CHALL_C`, de um Servidor bem definido, precisamente aquele a quem enviou anteriormente uma mensagem `C_RKEY_S` e cujo nome distinto ficou preservado em `loc_destiny`. Assim, é necessário comprovar se o sujeito de `S_cert` coincide com o Ser-

[22] Apresentados aqui recorrendo aos identificadores usados pelo SecuDE.
[23] O Servidor fecha imediatamente o *socket* e o Cliente termina, ao aperceber–se disso, ou por *timeout*.
[24] Correntemente, 16 bytes.
[25] O que supõe, naturalmente, que o Cliente possui a chave pública da autoridade de certificação.

vidor originador esperado, o que se faz pela comparação do nome distinto encerrado no certificado `S_cert` com o nome distinto `loc_destiny`.

À *Autenticação da Origem*, segue–se a *Autenticação do Destino*: o Cliente verifica se é mesmo o destinatário da mensagem, pela comparação do campo `C_dname` desta, com o seu nome distinto X.500, e ignora a mensagem caso não lhe seja dirigida[26].

O processo de Autenticação termina com a comparação do campo `C_timestamp` com a marca temporal `loc_time` da estrutura `SkeyxSessionInfo`[27]. Se forem iguais, então considera–se estabelecida a associação entre a mensagem `C_RKEY_S` enviada anteriormente e a mensagem `S_CHALL_C` recebida.

Durante o processo de Autenticação, o estado da conexão Skeyx do Cliente com o Servidor é actualizado de forma a que os campos `rem_destiny`, `rem_certs`, `rem_dhpubkey`, `rem_symlist`, `rem_symalg`, `loc_time_echoed` e `rem_time_1`, da estrutura `SkeyxSessionInfo` (rever figura 4.4), coincidam, respectivamente, com os campos `C_dname`, `S_cert`, `S_DHpubkey`, `S_SymList`, `CS_SymAlg`, `C_timestamp` e `S_timestamp`, da mensagem `S_CHALL_C` (rever figura 4.5).

Adicionalmente, o campo `rem_challenge` de `SkeyxSessionInfo` irá receber a versão decifrada do desafio `S_randchall`. A chave simétrica usada na desencriptação desse desafio consiste num subconjunto dos bits de uma chave acordada de Diffie–Hellman[28]. Esta chave é gerada pela combinação da chave privada de Diffie–Hellman do Cliente com a (pretensa) chave pública `S_DHpubkey` do Servidor, sendo também guardada em `SkeyxSessionInfo`, no campo `dhagreedkey`.

4.2.5 Desafio do Cliente ao Servidor (`C_CHALL_S`)

Ocorre então a assemblagem da resposta do Cliente ao Servidor, a qual se designa por *Client_Challenge_Server* (`C_CHALL_S`). Através de uma mensagem `C_CHALL_S`, o Cliente pretende responder ao desafio do Servidor, devolvendo–lhe `S_randchall` decifrado e, simultaneamente, lançando–lhe o seu próprio desafio, a fim de que o Servidor prove que a chave pública de Diffie–Hellman não certificada, `S_DHpubkey`, é realmente sua.

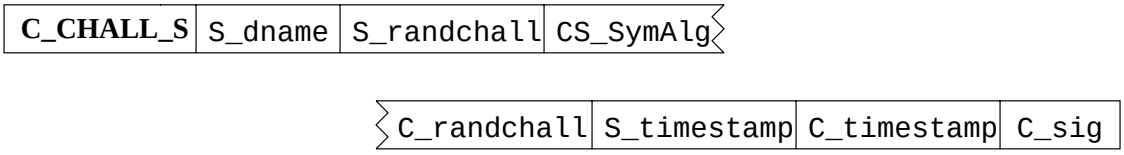

Figura 4.6: Estrutura de uma mensagem `C_SHALL_S`.

A estrutura de uma mensagem `C_CHALL_S` (ver figura 4.6) baseia–se nos seguintes campos:

- `C_CHALL_S`: cabeçalho, com o identificador do tipo da mensagem;

[26] O Cliente termina e o Servidor detecta a quebra da conexão TCP, desistindo então da troca Skeyx em curso.

[27] Recorde–se que o Cliente preservou em `loc_time` a marca temporal enviada em `C_RKEY_S`.

[28] Obviamente, esse subconjunto deverá ser extraído da chave acordada, da mesma forma por ambas as partes, a fim de que as chaves simétricas sejam coincidentes se, efectivamente, ambas as partes trocaram chaves públicas de Diffie–Hellman genuínas.

- `S_dname`: nome distinto X.500 do Servidor;
- `S_randchall`: resposta decifrada ao desafio do Servidor;
- `CS_SymAlg`: um algoritmo simétrico comum ao Cliente e ao Servidor;
- `C_randchall`: um desafio aleatório, cifrado com uma chave simétrica, segundo o algoritmo `CS_SymAlg`;
- `S_timestamp`: marca temporal do Servidor, oriunda da mensagem `S_CHALL_C`;
- `C_timestamp`: marca temporal gerada pelo Cliente;
- `C_sig`: assinatura digital que cobre a totalidade da mensagem.

O campo `S_randchall` contém a versão decifrada do desafio originalmente enviado pelo Servidor na mensagem `S_CHALL_C`.

Em relação ao desafio `C_randchall`, que o Cliente agora lança ao Servidor, ele foi cifrado com uma chave simétrica derivada da chave acordada de Diffie–Hellman (já calculada e preservada em `dhagreedkey`, recorde–se).

O algoritmo `CS_SymAlg`, sob o qual `C_randchall` é cifrado, não coincide necessariamente com o escolhido pelo Servidor para tarefa idêntica. Ou seja, Cliente e Servidor podem ter critérios diferentes na escolha de uma cifra simétrica a partir da intersecção dos algoritmos suportados[29], daí a conveniência deste campo na mensagem `C_CHALL_S`.

Os campos `S_dname`, `S_timestamp`, `C_timestamp` e `C_sig` cumprem o mesmo papel que os seus equivalentes nas mensagens `S_CHALL_C`. Note–se que `C_timestamp` é uma reutilização da marca temporal enviada com `C_RKEY_S` e não uma marca temporal específica (e logo mais recente) para `C_CHALL_S`, dado que a sessão Skeyx continua a mesma e, para o Cliente, `C_timestamp` identifica univocamente a sessão com o Servidor.

Durante a construção de uma mensagem `C_CHALL_S`, os campos `loc_symalg` e `loc_challenge` (versão não cifrada) do estado do Cliente são actualizados, respectivamente, com os mesmos valores que os campos `CS_SymAlg` e `C_randchall` daquela mensagem.

4.2.6 Recepção de Mensagens `C_CHALL_S`

No Servidor, a confirmação da assinatura digital `C_sig` de uma mensagem `C_CHALL_S` é suficiente para assegurar *Autenticação da Origem* dado que se faz com base na chave pública RSA contida no certificado X.509 preservado no campo `rem_certs` da estrutura `SkeyxSessionInfo`. Recorde–se que o campo `rem_certs` foi actualizado, no Servidor, com base no campo `C_cert` da mensagem `C_RKEY_S` e que o sujeito daquele certificado é precisamente o Cliente de quem se espera, nesta fase do protocolo, uma mensagem `C_CHALL_S`. Logo, a confirmação da assinatura digital `C_sig` não só oferece Não–Repudiação da origem e comprova a Integridade da mensagem, como também assegura que o Cliente originador da mensagem é o pretendido.

À semelhança do que acontecia para a mensagem `C_RKEY_S`, a *Autenticação do Destino* para a mensagem `C_CHALL_S` recebida é também baseada no confronto de `S_dname` com o nome distinto X.500 do Servidor.

[29] Na prática, porém, a implementação actual do Skeyx faz uso dos mesmos critérios de selecção no Cliente e no Servidor e por isso os algoritmos simétricos usados na protecção dos desafios coincidem.

A Autenticação de `C_CHALL_S` termina com a verificação de que, para o Cliente e para o Servidor, a troca Skeyx ainda é a mesma, o que se pode concluir pela comparação de `C_timestamp` com `rem_time_1` e de `S_timestamp` com `loc_time`, respectivamente. Pela primeira comparação, garante–se que `C_CHALL_S` é uma mensagem enviada pelo Cliente para concluir a troca Skeyx iniciada com a mensagem `C_RKEY_S` de onde `rem_time_1` foi retirado. Pela última comparação, o Servidor pode associar a resposta `C_CHALL_S` à mensagem `S_CHALL_C` enviada anteriormente.

Seguidamente, o Servidor confronta o desafio `loc_challenge` que lançou ao Cliente, com a resposta `S_randchall` deste. Se forem iguais, o Servidor pode associar a chave pública de Diffie–Hellman `rem_dhpubkey` ao Cliente, já que este provou a posse da respectiva chave privada, sem a qual não teria conseguido responder com sucesso ao desafio.

O Servidor encontra–se então em condições de actualizar a sua *cache* de chaves públicas de Diffie–Hellman, com a chave pública do Cliente. O processo de actualização da *cache* será alvo de discussão na secção 4.5.4.

A recepção de uma mensagem `C_CHALL_S` resulta, para o Servidor, na actualização de `SkeyxSessionInfo` pela inicialização dos campos `rem_destiny`, `loc_challenge_echoed`, `rem_symalg`, `rem_challenge` (versão decifrada), `loc_time_echoed` e `rem_time_2` com o valor dos campos `S_dname`, `S_randchall`, `CS_SymAlg`, `C_randchall`, `S_timestamp` e `C_timestamp`, respectivamente, da mensagem `C_CHALL_S`.

4.2.7 Terminação do Skeyx (`S_ACK_C`)

O Servidor termina o protocolo Skeyx enviando uma mensagem do tipo *Server_Acknowledge_Client* (`S_ACK_S`) ao Cliente, através da qual confirma a aceitação da chave pública de Diffie–Hellman deste e, simultaneamente, responde ao seu desafio, devolvendo–lhe `C_randchall` decifrado.

S_ACK_C	C_dname	C_randchall	C_timestamp	S_timestamp	S_sig

Figura 4.7: Estrutura de uma mensagem `S_ACK_C`.

Uma mensagem `S_ACK_C` compreende os seguintes campos (ver figura 4.7):

- `S_ACK_C`: cabeçalho, com o identificador do tipo da mensagem;
- `C_dname`: nome distinto X.500 do Cliente;
- `C_randchall`: resposta decifrada ao desafio do Cliente;
- `C_timestamp`: marca temporal do Cliente;
- `S_timestamp`: marca temporal do Servidor, oriunda da mensagem `S_CHALL_C`;
- `S_sig`: assinatura digital que cobre a totalidade da mensagem.

Estes campos desempenham as mesmas funções que os campos análogos dos outros tipos de mensagens do Skeyx.

O desafio `C_randchall` obteve–se desencriptando a versão cifrada que acompanha `C_CHALL_S` com uma chave simétrica do tipo `rem_symalg`, derivada da chave acordada

`dhagreedkey`[30]. Recorde–se que esta chave acordada foi gerada pelo Servidor logo após a recepção de `C_RKEY_S`.

A marca temporal `C_timestamp` tem o mesmo valor das marcas que o Cliente depositou nas mensagens `C_RKEY_S` e `C_CHALL_S`. Analogamente, a marca temporal `S_timestamp` é uma reutilização da marca gerada pelo Servidor a propósito da mensagem `S_CHALL_C`. Tal como para as outras mensagens, ambas as marcas situam `S_ACK_C` numa sessão Skeyx específica.

4.2.8 Recepção de Mensagens `S_ACK_C`

No Cliente, o processamento de uma mensagem `S_ACK_C` é em tudo idêntico ao processamento de uma mensagem `C_CHALL_S` no Servidor.

Assim, a *Autenticação da Origem* e a verificação da Integridade de uma mensagem `S_ACK_S` resulta da verificação da assinatura `S_Sig` com base na chave pública integrada no certificado do Servidor preservado em `rem_certs`. A *Autenticação do Destino* é mais uma vez feita pela comparação do campo `C_dname` com o nome distinto do próprio Cliente. As marcas temporais `C_timestamp` e `S_timestamp` devem ser coincidentes com os valores preservados, respectivamente, em `loc_timestamp` e `rem_time_1`, de `SkeyxSessionInfo`, para que a Autenticação se dê por concluída.

De seguida, o desafio `C_randchall` retornado é comparado com o original preservado em `loc_challenge`, a fim de concluir da validade da associação da chave pública de Diffie-Hellman `rem_dhpubkey` ao Servidor.

O Cliente actualiza então a sua *cache* de chaves pública de Diffie–Hellman, com a chave pública do Servidor.

A recepção de uma mensagem `S_ACK_C` resulta, para o Cliente, na actualização de `SkeyxSessionInfo`, pela inicialização dos campos `loc_challenge_echoed`, `loc_time_echoed`, `rem_time_2` e `rem_destiny` com o valor dos campos `C_randchall`, `C_timestamp`, `S_timestamp` e `C_dname`, respectivamente, da mensagem `S_ACK_C`.

4.3 Comunicação Orientada–à–Conexão

O Skeyx é um protocolo que faz evoluir um estado partilhado entre um Cliente e um Servidor. A figura 4.8 mostra a evolução desse estado, apresentando os campos de `SkeyxSessionInfo` que são actualizados em cada passo do protocolo.

Ora, a manutenção (bem como a evolução) de um estado partilhado entre dois extremos de uma conexão é mais fácil quando o protocolo de comunicação utilizado é *orientado–à–conexão*. Por um lado, não é necessário confirmar explicitamente a recepção das mensagens (introduzindo mensagens adicionais para o efeito, no protocolo[31]), nem proceder à sua retransmissão (explícita) em caso de *timeout*. Por outro, o risco de manter estados parciais por períodos de tempo indefinidos deixa de existir, já que a recepção das mensagens que fazem evoluir o estado é síncrona e são dadas "garantias de entrega".

[30] Processo em tudo semelhante ao realizado pelo Cliente sobre o desafio `S_randchall` do Servidor.

[31] Embora se pudesse admitir a hipótese de recorrer a um mecanismo de *piggy–backing* que, todavia, nunca resolveria o problema satisfatoriamente (reflicta–se, por exemplo, na questão das confirmações de confirmações...).

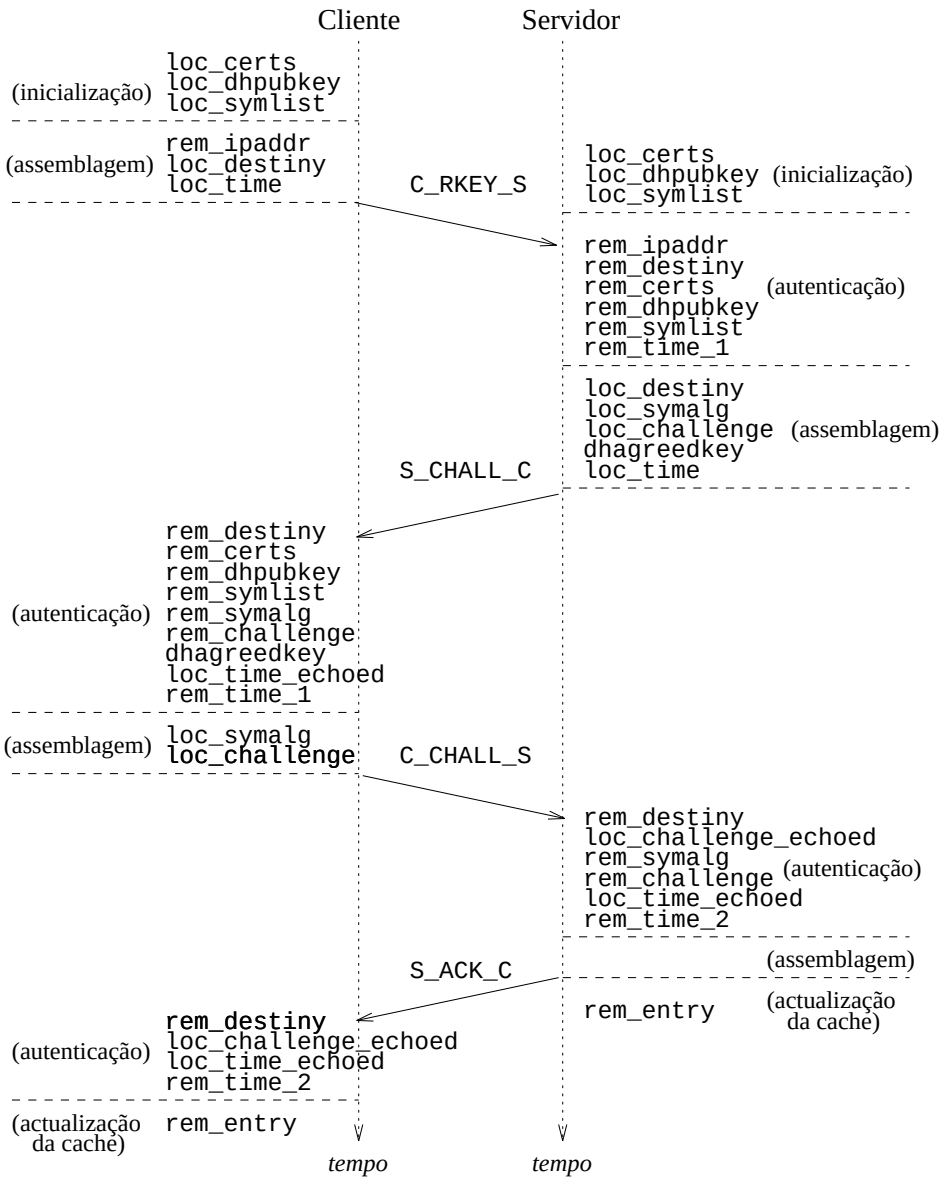

Figura 4.8: Evolução do estado `SkeyxSessionInfo`.

Assim sendo, compreende–se a escolha do TCP como base de comunicação para o Skeyx, contribuindo para o seu carácter "semi–transaccional" (ver secção 4.5.4).

4.4 Reencaminhamento de Mensagens `C_RKEY_S`

Para que um Cliente possa estabelecer uma conexão TCP com um Servidor, necessita, naturalmente, de saber o endereço IP da máquina onde este reside, bem como o porto onde o serviço é disponibilizado[32].

[32] O Cliente necessita de saber também o nome distinto da entidade cujo Servidor vai contactar, mas esse conhecimento é usado pelo processo de Autenticação do Skeyx, não sendo de forma alguma um requisito

Porém, o conhecimento antecipado do porto de destino pode não ser viável. Por um lado, a atribuição de um porto fixo a cada um dos Servidores remotos é impraticável (excepto se o número de Servidores for diminuto), dado que a dimensão do espaço dos portos é limitada, mas o número de entidades é imprevisível (e a cada entidade que "reside" na máquina remota estará associado pelo menos um Servidor). Por outro, admitindo a utilização de portos variáveis, coloca–se imediatamente o problema de fazer chegar esse conhecimento às partes (Clientes) interessadas: essa comunicação deveria ser segura[33], teria que ocorrer sempre que um Servidor arrancasse e este deveria saber antecipadamente que Clientes avisar.

Alternativamente, optou–se pela disponibilização de um serviço de reencaminhamento num porto bem conhecido, presentemente o 7571[34]. A integração desse serviço no Skeyx está esquematizada na figura 4.9.

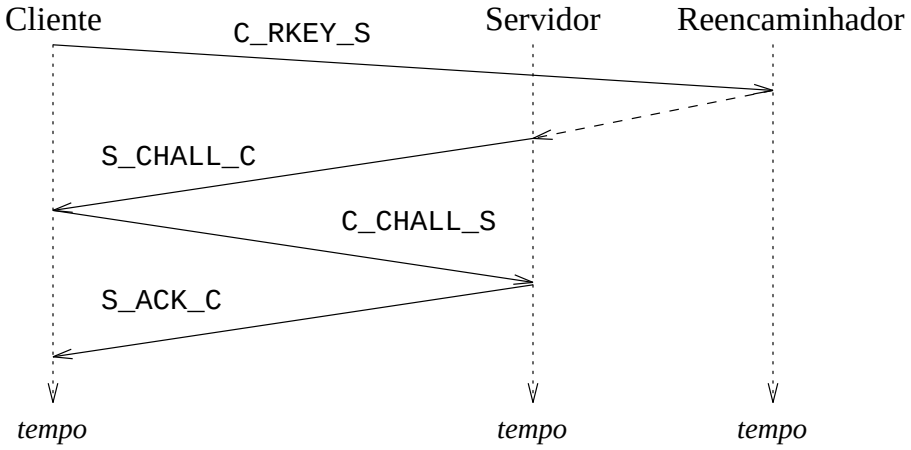

Figura 4.9: Reencaminhamento de uma mensagem `C_RKEY_S`.

Assim, quando um Cliente pretende enviar uma mensagem do tipo `C_RKEY_S` a um Servidor, fá–lo sempre para o porto 7571 da máquina de destino. O Reencaminhador, que escuta permanentemente o porto 7571, analisa a mensagem[35] e consegue determinar, pelo campo `S_dname`, qual a entidade destinatária.

Identificado o Servidor de destino[36], o próximo passo é fazer com que este obtenha um descritor relativo à mesma conexão TCP que o Reencaminhador mantém com o Cliente. Como é suposto os processos Reencaminhador e Servidor não terem nenhum grau de parentesco[37], tal partilha é feita com base no mecanismo descrito em [Ste90][38]. Assim, o Reencaminhador transmite ao Servidor uma mensagem de controlo, específica para estas situações, invocando a primitiva `sendmsg` sobre um *socket* do domínio Unix. No Servidor,

do TCP.

[33] E havendo um canal seguro, põe–se em causa a necessidade de estabelecer outro, com o Skeyx.

[34] Este número sintetiza a data de nascimento do autor. Segundo o registo [RP94] da *Internet Assigned Numbers Authority* (IANA), não está ainda atribuído a nenhum serviço.

[35] Que, recorde–se, não está cifrada.

[36] Implicitamente, estamos já a admitir a existência de, no máximo, um só Servidor, por máquina, associado a uma entidade, o que será justificado na secção 4.6.

[37] À excepção de ambos serem descentes do processo *init*, obviamente.

[38] E que é, segundo o autor, a única forma de passar descritores (de ficheiros, *sockets*, etc.) entre processos não relacionados, na maioria dos sistemas Unix actuais.

o processamento dessa mensagem resulta na obtenção de um descritor da mesma conexão TCP que o Reencaminhador mantém com o Cliente. Entretanto, o Reencaminhador fecha o seu descritor da conexão, concluindo–se o processo de redireccionamento.

A comunicação através de um *socket* do domínio Unix corresponde, na prática, à utilização de um *pipeline* como canal de comunicação entre os dois extremos. Esse canal reside, no nosso caso, por opção, na directoria `/tmp` e por isso convém assegurar que cada Servidor recebe as mensagens de reencaminhamento através de um canal específico, não partilhado, e de preferência sem qualquer associação explícita com o Servidor respectivo. Um *hash* MD5 do nome distinto da entidade associada ao Servidor é suficiente para gerar um identificador com estas propriedades[39]. Note–se que o Reencaminhador consegue gerar este identificador com base no campo `S_dname` da mensagem `C_RKEY_S`, ou seja, sem qualquer contacto prévio entre o Reencaminhador e o Servidor, o primeiro sabe como encontrar o segundo. Esta abordagem contrasta, por exemplo, com a do mecanismo de RPCs da Sun, onde se exige aos servidores o registo prévio junto do processo `portmap`, antes que os clientes estabeleçam qualquer contacto.

O papel desempenhado pelo Reencaminhador é semelhante ao do super–servidor `inetd`. Contudo, esta última aproximação não é adequada ao contexto da interacção Reencaminhador–Servidor(es). Seria possível conceber o registo de um novo serviço em `/etc/inetd.conf` (*e.g.*, `skeyx`) associado ao porto 7571. O `inetd` faria então o `exec` do servidor respectivo (*e.g.*, `/etc/skeyxd`) sempre que uma conexão fosse estabelecida com aquele porto. É precisamente neste ponto que a abordagem do `inetd` se revela inadequada: a cada entidade está associado um (e um só) Servidor (por máquina) que só ela poderá executar, uma vez que para o fazer é necessária a introdução manual de uma palavra–chave (o *Personal Identification Number* (PIN)) de acesso ao seu *Personal Security Environment* (PSE)[40].

Finalmente, e a título de curiosidade, refira–se que a inspecção que o Reencaminhador faz à mensagem `C_RKEY_S` (para determinar o seu destino `S_dname`) não deve fazer avançar a posição de leitura do descritor TCP. Essa posição é preservada numa tabela do núcleo do sistema operativo, que será partilhada com o Servidor[41]. Consequentemente, a leitura do Servidor teria início na posição onde o Reencaminhador tivesse terminado. Para que o Servidor tenha acesso à totalidade da mensagem pendente no descritor TCP que partihar com o Reencaminhador, as operações de leitura deste sobre as mensagens que recebe do Cliente baseiam–se na utilização da primitiva `recv` com a opção `MSG_PEEK`.

[39] Como o *hash* pode originar um identificador não válido no sistema de ficheiros, não se utiliza o *hash* directamente mas sim a sequência de dígitos hexadecimais correspondente. Um esquema semelhante é também utilizado na identificação da entrada de uma entidade remota na *cache*, com base no seu nome distinto (ver secção 4.5.1).

[40] O PSE, recorde–se (rever secção 3.3.6), constitui uma espécie de contentor seguro, fornecido de base pelo SecuDE e onde residem as chaves de uma entidade, entre outra informação sensível (na secção A.2.1 do apêndice A, o papel do PIN e o significado do PSE no contexto do SecuDE são mais bem esclarecidos). O S3L adiciona ao PSE de uma entidade a sua *cache* de chaves públicas de Diffie–Hellman (subdirectoria `.dhcache`) e as listas de controle de acesso usadas na variante multiponto (subdirectoria `.acls`).

[41] Na realidade, a mensagem de controlo enviada pelo Reencaminhador ao Servidor através do *socket* de domínio Unix, tem como resultado a criação de um descritor, no Servidor, com direitos de acesso a essa tabela.

4.5 Gestão da Cache de Chaves Públicas

O Skeyx reflecte–se na actualização de uma *cache* em ambas as entidades Cliente e Servidor. De seguida, apresenta–se e justifica–se a estrutura adoptada para essa *cache*, bem como se descrevem os procedimentos relativos à sua actualização.

4.5.1 Estrutura da Cache

A estrutura da *cache* de uma entidade assenta, basicamente, num conjunto de subdirectorias, cada qual específica para cada entidade remota que executou o Skeyx com a entidade local.

A adopção da *cache* no sistema de ficheiros (e não em memória) justifica–se com base nos seguintes argumentos:

- uma *cache* de uma entidade deve poder ser acedida em simultâneo por vários processos lançados no contexto dessa entidade[42]. Dadas as dificuldades inerentes à partilha de memória e à comunicação inter–processos entre processos sem grau de parentesco directo (*e.g.*, pai–filho), a utilização de um mecanismo de exclusão baseado em trincos[43] sobre o sistema de ficheiros constitui uma solução óbvia (ver secção 4.5.3);

- uma *cache* no sistema de ficheiros é mais duradoura e, potencialmente, mais resistente à morte prematura dos processos que lhe acedem;

- uma vez que a execução do Skeyx constitui um evento relativamente raro, em geral executado uma só vez entre um determinado par de entidades, então a degradação do desempenho do Skeyx resultante da operação directa sobre o sistema de ficheiros é perfeitamente aceitável;

- em caso de necessidade, a mobilidade da *cache* de uma entidade é facilmente assegurada, bem como a produção de cópias de segurança.

A figura 4.10 apresenta, esquematicamente, a estrutura da *cache* para um utilizador genérico, `user_1`.

Em cada entrada[44] da *cache*, encontram–se a chave pública de Diffie–Hellman da entidade remota (ficheiro `DHpubkey`[45]), outra informação relativa à entidade remota (ficheiro `DHpubkey.info`), obtida durante o Skeyx, e a chave acordada de Diffie–Hellman (ficheiro `DHagreedkey`[46]).

Correntemente, apenas se achou útil preservar em `DHpubkey.info` a lista de algoritmos simétricos suportados pela entidade remota[47], para utilização em futuros contactos com essa entidade, *via* S3L (ver capítulo 5). Em futuras evoluções do protocolo Skeyx, outra informação que se ache relevante poderá ser armazenada neste ficheiro.

[42] Concretamente, por uma (e uma só) instância do Servidor e, eventualmente, por vários Clientes. A replicação de Clientes e Servidores será devidamente explorada na secção 4.6.

[43] Do inglês *locks*.

[44] Ou seja, em cada subdirectoria do tipo `hex(md5(nome_distinto_n))`.

[45] Criado com o conteúdo do campo `rem_dhpubkey` de `SKeyxSessionInfo`.

[46] Criado com o conteúdo do campo `dhagreedkey` de `SkeyxSessionInfo`.

[47] Lista essa que, recorde–se, corresponde ao campo `rem_symalgs` de `SkeyxSessionInfo`.

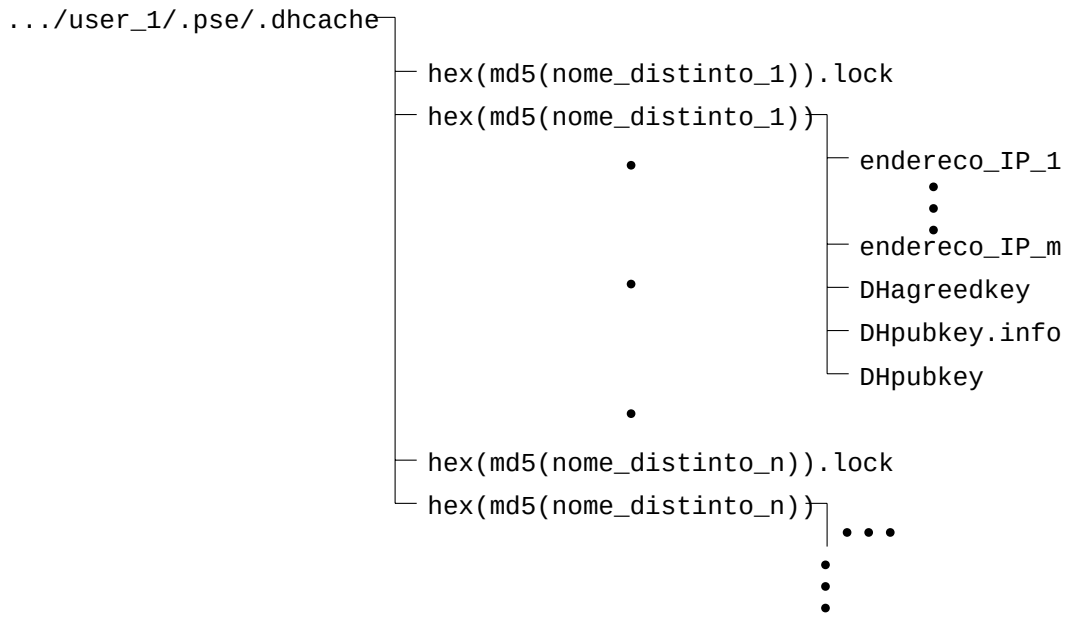

Figura 4.10: Estrutura genérica de uma *cache* de chaves públicas de Diffie–Hellman.

Relativamente à chave acordada, em princípio, poder–se–ia dispensar o seu armazenamento na *cache*, já que a disponibilidade da chave pública de Diffie–Hellman da entidade remota, `DHpubkey`, seria suficiente para gerar a chave acordada através da combinação com a chave privada de Diffie–Hellman da entidade local. Todavia, esse processo envolve operações de chave pública, lentas por natureza. Consequentemente, a fim de acelerar as operações do S3L (cujas chaves individuais de cada mensagem derivam da chave acordada), é conveniente o pré–cálculo da chave acordada, que assim se encontra permanentemente disponível.

A possibilidade de replicação de uma entidade (ver secção 4.6) tem uma correspondência directa na estrutura das entradas da *cache*. Assim, em cada entrada encontram–se um ou mais ficheiros (`endereco_IP_1, ..., endereco_IP_m`) cujo nome é dado pelo endereço IP[48] de uma máquina a partir da qual a entidade, à qual diz respeito a entrada, realizou o Skeyx com a entidade local, dona da *cache*. Cada um desses ficheiros (doravante designados por "ficheiros IP") contém uma marca temporal[49] relativa à última troca Skeyx (bem sucedida) com a instância da entidade remota que reside na máquina identificada pelo nome do ficheiro. As marcas temporais destinam–se não só a detectar *ataques–de–repetição*, como previnem a actualização da *cache* em resultado de trocas Skeyx para as quais as marcas temporais das mensagens se tornaram, entretanto, obsoletas. Este último caso é particularmente bem ilustrado na secção 4.5.4, onde se demonstra como o algoritmo de actualização da *cache* reage satisfatoriamente numa situação concreta de "actualizações cruzadas".

Note–se, porém, que da existência de vários ficheiros IP não se pode concluir da existência, em simultâneo, de um número equivalente de instâncias da entidade remo-

[48] Correspondente ao campo `rem_ipaddr` de `SkeyxSessionInfo`.

[49] Correspondente ao campo `rem_time_1` — ou `rem_time_2`, dado que no fim do Skeyx ambos os campos deverão iguais — de `SkeyxSessionInfo`.

ta, cada qual na sua máquina[50], dada pela designação de cada ficheiro. Algumas dessas instâncias poderão já não existir. Um caso ilustrativo desta situação é dado por uma entidade nómada, para a qual existe, em qualquer instante, uma única instância que, porém, não está vinculada a uma máquina fixa[51]. Neste caso, o "trajecto" dessa entidade ficaria registado na *cache*, admitindo que a entidade executaria o Skeyx, de cada vez que migrasse.

Ainda em relação à estrutura da *cache*, note–se que, apesar de a chave pública de Diffie-Hellman da entidade remota ter sido fornecida *via* Skeyx, pelo menos tantas vezes quanto o número de ficheiros IP presentes na *cache*[52], conserva–se apenas um ficheiro `DHpubkey` cuja chave pública é sempre proveniente da última troca Skeyx efectuada com sucesso. Obviamente, isto pressupõe que todas as réplicas da entidade remota fornecem a mesma chave pública de Diffie–Hellman à entidade local, durante a execução do Skeyx[53].

4.5.2 Identificação das Entradas da Cache

A identificação das entradas da *cache* obedece a um esquema destinado a preservar o anonimato da entidade remota à qual cada entrada se refere. Este esquema é semelhante ao usado pelo Reencaminhador a fim de obter um identificador de um ficheiro através do qual contacta cada Servidor. Assim, é com base no *hash* MD5 do nome distinto da entidade remota que se define o nome a atribuir à sua subdirectoria na *cache*[54]. A notação `hex(...)` da figura 4.10 significa que o *hash*, possivelmente não traduzível num identificador válido no contexto do sistema de ficheiros, foi convertido para uma sequência equivalente de dígitos hexadecimais, como é o caso do identificador `A3DC7FE19F93D4E2B67D6F93C1A1164C` na figura 4.11, a qual contém uma listagem recursiva da *cache* de um utilizador concreto, `ruf`, com apenas uma entrada, dada precisamente pela subdirectoria `A3DC7FE19F93D4E2B67D6F93C1A1164C`.

Contudo, é possível conceber um *ataque–de–dicionário* que, com base numa lista de nomes distintos, produza o *hash* MD5 de cada um e o compare com o identificador das entradas na *cache* a fim de determinar a entidade respectiva[55]. Obviamente que o espaço dos nomes distintos é incomensurável, mas uma análise cuidada do ambiente local (por exemplo, do tráfego de mensagens Skeyx, onde viajam nomes distintos integrados em certificados X.509) poderá proporcionar uma lista de nomes distintos suficientemente reduzida e viável para um *ataque–de–dicionário*. À partida, só o compromisso da chave acordada `DHagreedkey` seria preocupante, caso um ataque deste género fosse bem sucedido, já que é com base naquela chave que o S3L deriva as chaves individuais de protecção das suas mensagens. A restante informação preservada em *cache* é de domínio público e não estabelece qualquer associação explícita com a entidade remota. Assim, por precaução, a chave

[50] Recorde–se que no máximo se admite uma instância de uma entidade por máquina.

[51] Saliente–se que a salvaguarda e transporte da *cache* de cada vez que a entidade migra é da sua responsabilidade.

[52] "Pelo menos" porque o Skeyx pode ter sido executado mais que uma vez com a instância da entidade remota numa determinada máquina.

[53] Mais uma vez se insiste no facto de que a responsabilidade na coerência das chaves preservadas nos PSEs das réplicas não é da responsabilidade do Skeyx.

[54] O caminho completo para essa subdirectoria corresponde ao campo `rem_entry` de `SkeyxSessionInfo`.

[55] O que pressupõe, naturalmente, que o atacante ultrapasse as permissões do sistema de ficheiros, que limitam o acesso à *cache*.

```
/home/ruf/.pse/.dhcache$ ls -laR
total 4
drwxr-xr-x 3 ruf users 1024 Jan 14 19:46 ./
drwx------ 3 ruf users 1024 Jan 19 16:53 ../
drwx------ 2 ruf users 1024 Jan 20 22:47 A3DC7FE19F93D4E2B67D6F93C1A1164C/
-rw------- 1 ruf users    5 Jan 19 16:51 A3DC7FE19F93D4E2B67D6F93C1A1164C.lock

A3DC7FE19F93D4E2B67D6F93C1A1164C:
total 7
drwx------ 2 ruf users 1024 Jan 20 22:47 ./
drwxr-xr-x 3 ruf users 1024 Jan 14 19:46 ../
-rw-r--r-- 1 ruf users   10 Jan 15 04:54 193.136.20.130
-rw-r--r-- 1 ruf users   10 Jan 19 16:51 193.136.8.7
-rw-r--r-- 1 ruf users   68 Jan 19 16:51 DHagreedkey
-rw-r--r-- 1 ruf users  223 Jan 19 16:51 DHpubkey
-rw-r--r-- 1 ruf users  114 Jan 19 16:51 DHpubkey.info
/home/ruf/.pse/.dhcache$
```

Figura 4.11: Exemplo de uma cache `.dhcache`.

acordada, `DHagreedkey`, é cifrada com uma chave simétrica DES, a qual corresponde à mesma chave de protecção do PSE.

4.5.3 Acesso Concorrente à Cache

Uma vez que se admite a coexistência de vários processos que necessitam de ler e escrever na *cache*, é necessário assegurar, em qualquer instante, a sua consistência, o que pode ser conseguido à custa de um mecanismo de exclusão mútua.

Decidida a concretização da *cache* directamente em disco, a utilização de ficheiros de trinco constitui uma solução possível. Assim, a cada entrada da cache é associado um ficheiro de trinco, com a mesma designação da entrada mas com a extensão `.lock` (*e.g.*, `A3DC7FE19F93D4E2B67D6F93C1A1164C.lock` na figura 4.11). Todos os acessos à entrada respectiva são serializados através do seu trinco específico.

O ficheiro de trinco é criado durante a primeira tentativa de acesso (seja ela de escrita ou leitura) à entrada respectiva. Contudo, pode haver outros processos que, concorrentemente, estejam a fazer também o seu primeiro acesso precisamente a essa entrada. A fim de assegurar que só um deles irá criar o ficheiro de trinco, é invocada a primitiva `open` com ambas as opções `O_CREAT` e `O_EXCL`. Segundo o *standard* POSIX [IEE88], tal garante que o teste relativo à existência do ficheiro de trinco e a sua criação em caso negativo ocorram de forma atómica, relativamente a outros processos que tentem fazer o mesmo.

Criado o ficheiro de trinco, sucede–se a "trincagem"[56] propriamente dita sobre esse ficheiro. Como os únicos processos que acedem à *cache* resultam da execução do protocolo Skeyx e uma vez que em tais circunstâncias qualquer acesso de leitura ou escrita é condicionado à aquisição prévia do trinco da entrada pretendida, então é suficiente que o mecanismo de "trincagem" usado seja do tipo *advisory*, com trincos exclusivos[57].

O identificador do último processo que adquiriu o trinco é guardado no próprio ficheiro do trinco, a fim de ser possível terminá–lo caso se detecte uma posse do trinco

[56] Do inglês *locking*.
[57] Ver manual da primitiva `flock`.

anormalmente longa por parte desse processo.

4.5.4 Actualização da Cache

Apesar de o Servidor possuir toda a informação necessária à actualização da sua *cache* logo após a Autenticação de uma mensagem `C_CHALL_S`, essa actualização só é realizada depois de a mensagem `S_ACK_C` ter sido confirmadamente entregue ao Cliente. Essa "garantia" é viável graças à utilização de uma conexão TCP entre o Cliente e o Servidor[58] ao longo de toda a operação do Skeyx (recorde–se a secção 4.3).

Assim sendo, é possível assegurar que ambos os extremos de uma conexão Skeyx atingem o estado final antes de actualizarem a sua *cache*. Sob este ponto de vista, poderíamos classificar a semântica do Skeyx como transaccional: ou ambos as partes atingem o estado final, ou não se procede à actualização das respectivas *caches*.

O Skeyx, porém, não garante que a actualização das *caches* seja bem sucedida em ambas as partes. Com efeito, após a entrega (confirmada) da mensagem `S_ACK_C` do Servidor ao Cliente, a conexão TCP é encerrada e a actualização das *caches* prossegue de forma independente no Cliente e no Servidor, eventualmente com diferentes graus de sucesso[59]. Por este motivo, designamos a semântica do Skeyx de "semi–transaccional".

Todavia, a actualização da entrada de uma *cache* já constitui em si uma transacção: ou todas as modificações são bem sucedidas, ou então o estado antigo da entrada é reposto[60]. A opção por uma semântica deste género na actualização da *cache* justifica–se pelo facto de que entre os elementos de uma entrada (rever secção 4.5.1) existe uma relação indissociável. Logo, não são admissíveis alterações parciais.

O algoritmo de actualização de uma entrada da *cache* é apresentado na figura 4.12. No seu contexto, `rem_entry`, `rem_ipaddr` e `rem_time_1` correspondem aos campos do estado do Skeyx (`SkeyxSessionInfo`) com a mesma designação. São também evidentes o fecho do trinco, no início, e a sua abertura, no fim, bem como as operações de salvaguarda (*guardar*) dos elementos (ficheiros) da entrada e sua recuperação (*recuperar*) em caso de necessidade (operações estas que conferem o carácter "transaccional" à actualização). O algoritmo parte do princípio de que desde que uma operação de *guardar* seja bem sucedida, então a respectiva operação de *recuperar* também o será.

4.6 Replicação no Contexto do Skeyx

No contexto do Skeyx (e por consequência, do S3L) uma entidade define–se pela sua identidade, a qual é dada por um nome distinto X.500, precisamente aquele que corresponde ao sujeito do certificado X.509 da chave pública RSA da entidade. A identidade considera–se

[58] Neste contexto, a designação "garantia de entrega" pode ser enganadora. Convém lembrar que o TCP "garante" entrega à custa de retransmissões, mecanismo que pode não ser suficiente em determinadas situações limite (*e.g.*, sistemas originador e/ou destinatários desligados, ligações físicas quebradas ou saturadas, nós intermédios inoperacionais ou saturados, etc.). Entretanto, a implementação actual do Skeyx tenta lidar com estas contingências permitindo configurar o *timeout* de uma conexão TCP.

[59] Obviamente, esta última situação é indesejável, pelo que foram desenvolvidos mecanismos apropriados para lidar com ela — ver secção 5.4.

[60] Admitindo, obviamente, que o sistema não fica inoperacional antes da recuperação completa do estado antigo. Se tal acontecer, é ainda possível a recuperação "manual" do estado antigo, dado que este só é eliminado quando o novo estado é criado com sucesso.

```
fechar_trinco

SE nao_existe_entrada(rem_entry)
ENTAO /* nunca fizemos Skeyx com a entidade remota */
        criar_e_actualizar_entrada(rem_entry);
        SE (algo_correu_mal) ENTAO eliminar_entrada(rem_entry) FIMSE
SENAO
        SE nao_existe_ficheiro(rem_ipaddr)
        ENTAO /* nunca fizemos Skeyx com a entidade remota na maquina rem_ipaddr */
                guardar(DHpubfile, DHpubfile.info, DHagreedkey);
                actualizar_entrada(rem_entry);
                SE (algo_correu_mal)
                ENTAO
                        recuperar(DHpubfile, DHpubfile.info, DHagreedkey);
                        eliminar_ficheiro(rem_ipaddr);
                FIMSE
        SENAO /* ja fizemos Skeyx com a entidade remota na maquina rem_ipaddr */
                marca_temporal ← ler_ficheiro(rem_ipaddr);
                SE (rem_time_1 > marca_temporal)
                ENTAO /* marca de C_RKEY_S mais recente que marca da cache */
                        guardar(DHpubfile, DHpubfile.info, DHagreedkey);
                        guardar(rem_ipaddr);
                        actualizar_entrada(rem_entry);
                        SE (algo_correu_mal)
                        ENTAO
                                recuperar(DHpubfile, DHpubfile.info, DHagreedkey);
                                recuperar(rem_ipaddr);
                        FIMSE
                FIMSE
        FIMSE
FIMSE

abrir_trinco
```

Figura 4.12: Algoritmo de actualização da *cache*.

imutável, ao contrário das chaves RSA, das quais se espera, no entanto, que tenham um tempo de vida razoável. Adicionalmente, dado que o nome distinto X.500 da entidade integra o certificado X.509 da sua chave pública RSA e esse certificado se encontra preservado num PSE, então estabelece–se também uma relação unívoca entre cada entidade e o seu PSE (ou melhor, entre cada instância de uma entidade[61] e o seu PSE).

Assim, um utilizador que mantém um PSE numa conta de um sistema Unix é, claramente, uma entidade. Um programa cliente de um serviço de transacções seguras também pode ser visto como uma entidade uma vez que, sendo primariamente identificado pelo seu número de série, tal número poderá integrar um nome distinto X.500 que identifica, alternativamente, a cópia em questão do programa. Neste último caso, o fabricante pode até ter–lhe associado um par de chaves RSA as quais poderão ser preservadas num PSE, numa conta criada expressamente para o programa.

Em face do conceito de entidade apresentado, o qual se baseia, fundamentalmente, num nome distinto X.500 único e que, em termos práticos, corresponde à existência de

[61] Tacitamente, estamos já a admitir a possibilidade de replicação de uma entidade, o que só será estabelecido posteriormente.

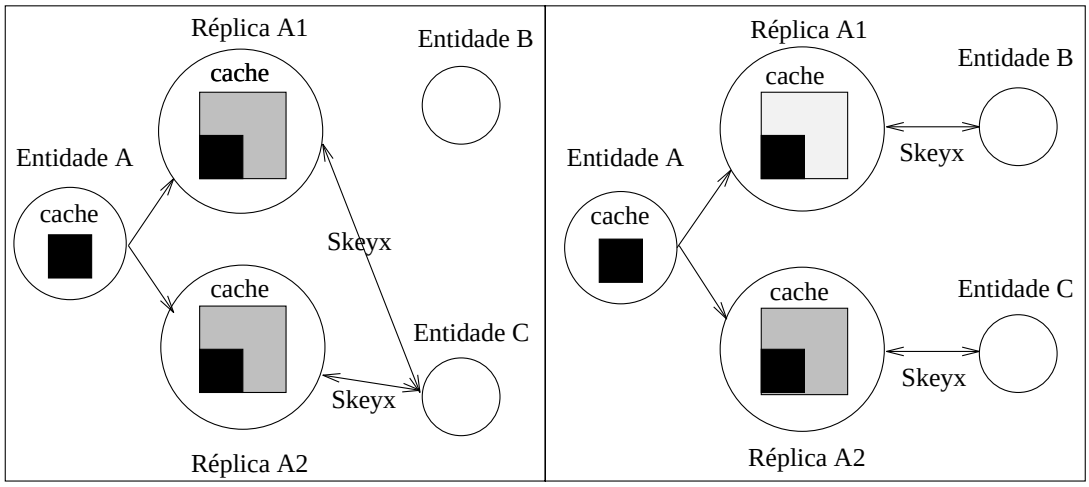

Figura 4.13: Convergência e divergência de *caches* em réplicas de entidades.

um PSE, a replicação de uma entidade poderá ou não fazer sentido, conforme o caso. Consideremos os dois exemplos apresentados no parágrafo anterior. No primeiro caso, é perfeitamente admissível a posse de várias contas (eventualmente em máquinas diferentes) por parte de um utilizador, sem que tal implique a adopção de um nome distinto X.500 alternativo. Adicionalmente, este utilizador, pode achar conveniente a utilização das mesmas chaves RSA e Diffie–Hellman em cada um dos PSEs encerrados nas várias contas. Consequentemente, a replicação do PSE desse utilizador faz sentido. Todavia, no segundo caso (programa cliente de transacções seguras), a viabilidade (e, sobretudo, a legitimidade) da replicação do PSE dependerá, em princípio, da política de licenciamento seguida para o *software* em questão.

Conclui–se, portanto, que a replicação de um entidade é não só admissível, como pode até ser conveniente (caso do exemplo do utilizador com várias contas). Assim sendo, importa averiguar quais os limites que a implementação actual do S3L (em particular, do Skeyx) impõe a essa replicação, uma vez que não faz parte dos objectivos desta dissertação o desenvolvimento de mecanismos de gestão de réplicas.

Relativamente ao Skeyx, o principal problema que a replicação de uma entidade introduz diz respeito à coerência da *cache* de chaves públicas de Diffie–Hellman, em cada réplica. O Skeyx não oferece, actualmente, mecanismos explícitos para lidar com este problema. Se as réplicas seguirem um padrão idêntico de contactos com outras entidades, então as suas *caches* irão tendencialmente convergir para um estado comum. Mesmo que isso não aconteça, partilharão, pelo menos, um estado inicial comum, uma vez que não é de esperar que as chaves públicas de Diffie–Hellman que constituíam a *cache* à data da replicação fiquem obsoletas a curto prazo. A evolução da *cache* nestas duas situações está esquematizada na figura 4.13.

Note–se que, no caso de o par de chaves RSA ou Diffie–Hellman de uma entidade sofrer alterações, a entidade é responsável pela sua propagação às eventuais réplicas. Espera–se, todavia, que este evento seja suficientemente raro para que a distribuição manual das novas versões não seja impraticável.

A replicação de uma entidade está ainda sujeita a uma restrição, derivada da forma

como o Reencaminhador contacta um Servidor associado àquela entidade. De acordo com o que foi dito na secção 4.4, quando o Reencaminhador recebe uma mensagem `C_RKEY_S`, determina o nome distinto da entidade de destino e com base nele deriva o caminho de um canal único, ao qual um Servidor se supõe associado através de um *socket* de domínio Unix. Como a opção `SO_REUSEADDR` da primitiva `setsockopt` se aplica apenas a *sockets* do domínio `AF_INET` (*i.e.*, Internet) [Ste90], então não é possível associar mais do que um Servidor àquele canal. Dado que cada réplica da entidade necessita de pelo menos um Servidor próprio, então conclui–se que, na mesma máquina, só pode haver uma instância (e, equivalentemente, um PSE) de uma entidade. Dito de outra forma: a replicação de uma entidade implica a disposição das réplicas em máquinas diferentes, dado que a cada entidade só pode estar associado um e um só Servidor, por máquina.

A questão da replicação também se coloca sob o ponto de vista dos Clientes. Neste caso, qualquer entidade admite a execução de Clientes em número arbitrário por máquina. A razão principal que está na base desta política algo liberal (quando comparada com a restrição de um só Servidor, em cada máquina, por entidade) é a possibilidade de a componente Cliente do Skeyx poder ser executada não só *por antecipação* (explicitamente, através da invocação de uma aplicação específica — o `s3lkeyxclient` —, fornecida na distribuição do S3L), como também *por necessidade*, de uma forma implícita e automática (concretamente, através da invocação da função `Skeyx`[62]), imediatamente antes do envio ou após a recepção de uma mensagem S3L. Consequentemente, é imprevisível o número de Clientes activos, num dado instante, tudo dependendo do número de aplicações baseadas no S3L, em execução[63]. Por oposição, um Servidor é um (e um só) demónio (correspondente à aplicação `s3lkeyxserver`, fornecida na distribuição do S3L) cuja execução é iniciada sempre de forma explícita.

Finalmente, note–se que a possibilidade da execução simultânea, no contexto de uma instância de uma entidade, de vários Clientes e um Servidor (o qual se desdobra em processos filhos dedicados ao atendimento de cada Cliente remoto), todos efectuando acessos à *cache*, torna evidente a necessidade de mecanismos de exclusão mútua no acesso a esse recurso partilhado. Os mecanismos concretos a usar a este nível foram previamente descritos na secção 4.5.3.

4.7 Trocas Skeyx Cruzadas

Será interessante observar o comportamento do algoritmo da figura 4.12 numa situação que convencionamos designar por "trocas Skeyx cruzadas" e que ocorre quando duas entidades executam, cada uma, (pelo menos) um Cliente, com o objectivo de contactar um Servidor da outra, e de forma a que tal suceda suficientemente próximo no tempo para que as mensagens Skeyx se cruzem e se verifique, em cada entidade, competição entre a sua componente Servidor e a sua componente Cliente no acesso à *cache*. Esta competição pode ainda ser maior dado que, apesar de limitada à execução de um único Servidor por máquina, já estabelecemos (ver secção anterior) que uma entidade poderá lançar vários

[62] Acessível através do módulo `skeyx.c`, que também acompanha a distribuição do S3L.

[63] As circunstâncias especiais que determinam uma invocação implícita do Skeyx no contexto de uma troca de mensagens S3L serão objecto de análise na secção 5.4.

Clientes, cujos destinos podem ser, eventualmente, o mesmo Servidor remoto[64].

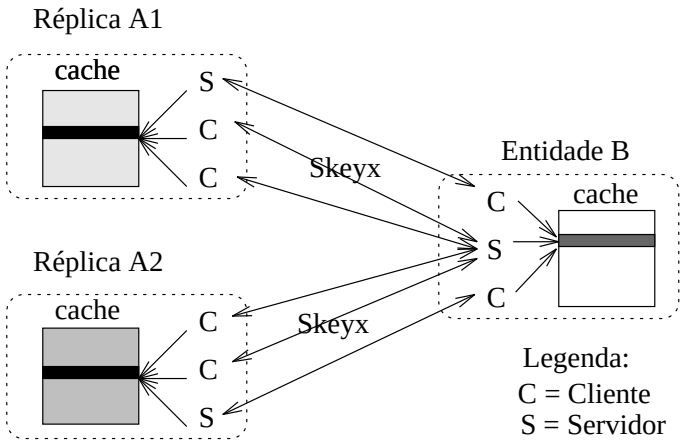

Figura 4.14: Trocas Skeyx cruzadas, com replicação de uma entidade.

Note–se que a execução quase em simultâneo de vários Clientes de uma entidade, dirigidos a um mesmo Servidor remoto de outra, não pressupõe que esses Clientes sejam originados todos na mesma máquina, dado que uma entidade poderá encontrar–se disseminada por várias máquinas. A figura 4.14 é bem representativa desta situação que, contudo, representa um caso limite[65].

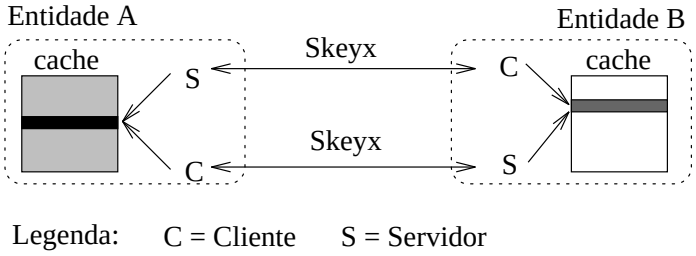

Figura 4.15: Trocas Skeyx cruzadas, sem replicação de nenhuma entidade.

Em geral, as trocas Skeyx cruzadas ocorrerão segundo o modelo da figura 4.15, na qual não se consideraram réplicas para nenhuma das entidades envolvidas.

Os atrasos variáveis na rede conjuntamente com alguma imprevisibilidade no escalonamento de processos podem fazer inverter a ordem "natural" pela qual a actualização das *caches* se daria, numa situação de trocas Skeyx cruzadas. Esta situação é tanto mais provável quanto mais apertadas forem as margens temporais envolvidas, isto é, quanto mais "próximas no tempo" forem iniciadas as trocas Skeyx em ambas as partes.

Pode então acontecer que uma actualização da *cache* não se verifique porque, entretanto, uma troca mais recente (iniciada portanto posteriormente) já conseguiu fazer a sua actualização, tornando obsoleta uma troca iniciada antes, que se atrasou.

[64] Pelo que todos os Clientes, após terem trocado chaves com o Servidor remoto, procurarão aceder à mesma entrada da *cache*, para proceder à sua actualização.

[65] Em abono da verdade, um caso mais complexo seria ainda aquele em que a Entidade B se apresentasse também replicada.

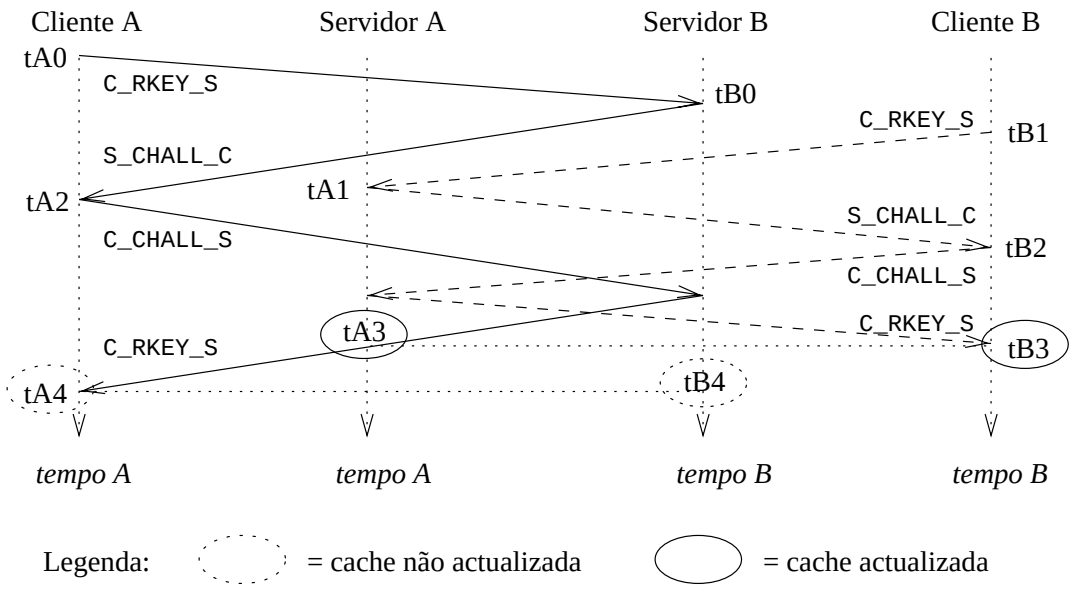

Figura 4.16: Exemplo de transacções cruzadas com obsolescência antecipada.

A figura 4.16 representa um caso particular de trocas Skeyx cruzadas onde se podem observar diferentes resultados relativamente à actualização das *caches*. Os instantes em que essas actualizações têm lugar são assinalados por elipses.

No instante `tA3`, o Servidor A actualiza a *cache* com a chave que recebeu do Cliente B, no instante `tA1`, marcada no instante `tB1`. Essa actualização é bem sucedida porque a troca iniciada pelo Cliente A ainda não se reflectiu na *cache* de A.

O Cliente A inicia, no instante `tA4`, a actualização da *cache* com a chave que recebeu do Servidor B, no instante `tA2`, marcada no instante `tB0`. Entretanto, na *cache* já se encontra a chave introduzida anteriormente pelo Servidor A, com a marca `tB1`. Como `tB1` $>$ `tB0`, ou seja, a chave que o Servidor A recebeu do Cliente B é mais recente que a chave que o Cliente A recebeu do Servidor B, então a actualização da *cache* no instante `tA4` não se concretiza.

No instante `tB3`, o Cliente B actualiza a *cache* com a chave que recebeu do Servidor A, no instante `tB2`, marcada no instante `tA1`. Essa actualização é bem sucedida porque a troca iniciada pelo Cliente A ainda não se reflectiu na *cache* de B.

Contudo, quando o Servidor B inicia, no instante `tB4`, a actualização da *cache* com a chave que recebeu do Cliente A, no instante `tB0`, marcada no instante `tA0`, já essa chave se tornou obsoleta dado que o Cliente B conseguiu antecipar–se no instante `tB3`, introduzindo na *cache* uma chave mais recente, marcada no instante `tA1`, pelo que a actualização da *cache* no instante `tB4` não se concretiza.

A utilidade das marcas temporais preservadas nos ficheiros IP (`endereco_IP_1, ..., endereco_IP_m`) de uma entrada e que são produzidas pelo relógio das máquinas remotas correspondentes, deverá agora ser óbvia. Só através delas o algoritmo de actualização da *cache* consegue resolver coerentemente situações de *ataques–de–repetição* e trocas Skeyx cruzadas.

Capítulo 5

Comunicação Segura Ponto–a–Ponto via S3L

Este capítulo prossegue a apresentação detalhada da arquitectura do S3L, iniciada no capítulo anterior com a descrição do protocolo Skeyx. Concretamente, a variante ponto–a-ponto do S3L é analisada, reservando-se para o próximo capítulo a descrição da modalidade multiponto.

A relação do S3L com o SKIP é clarificada, assumindo–se os pontos de convergência entre ambos os protocolos e evidenciando–se as diferenças que os individualizam.

Em termos práticos, o principal resultado da conjugação do S3L com a gestão de chaves fornecida pelo Skeyx traduz-se num conjunto de funções cuja sintaxe e semântica são muito semelhantes às primitivas de Berkeley de escrita e leitura sobre *sockets* mas fornecendo novas qualidades de serviço sobre os dados trocados, nomeadamente: Privacidade, Autenticação (incluindo Não–Repudiação), Integridade, Sequenciação (contra *ataques–de-repetição*) e Compressão.

5.1 Mensagens SKIP Ponto–a–Ponto

A operação do S3L coincide, em pontos chave, com o modelo adoptado pelo SKIP. Assim sendo, antes de descrever o S3L, justifica-se uma apresentação do SKIP por forma a facilitar a posterior exposição e compreensão dos pontos de convergência e divergência entre ambas as abordagens.

Recorde–se, ainda que de uma forma sucinta, a descrição do funcionamente do SKIP apresentada na secção 3.4.2: admitindo que ambas as entidades comunicantes i e j concretizaram um acordo de Diffie–Hellman, então ambas dispõem de uma chave secreta comum, de 512 bits, no mínimo; cada pacote IP trocado entre as duas entidades será protegido com base numa chave simétrica `Kp`, gerada aleatoriamente (e, se necessário, individualmente para cada pacote), com um tempo de vida variável e que é parte integrante do próprio pacote; por sua vez, a protecção de `Kp` é assegurada pela sua encriptação com base numa chave simétrica `Kij`, derivada da chave secreta de longa duração, acordada e partilhada por i e j; tipicamente, a dimensão de `Kij`, em bits, situar-se-á no intervalo [40, 256][1], pelo que os 512 bits da chave acordada são mais que suficientes para serem reutilizados por `Kij`;

[1] A dimensão concreta depende, obviamente, do algoritmo simétrico escolhido para cifrar `Kp`.

limitando a utilização de `Kij` à protecção de `Kp` (em vez de todo o pacote), diminui–se a exposição de `Kij` a ataques baseados na análise do tráfego[2]; complementarmente, fazendo variar `Kij` ao longo do tempo, torna-se mais difícil comprometer a chave `Kp`; assim, em vez de `Kij`, utiliza–se uma chave `Kijn`, correspondente à `n`'ésima versão da chave `Kij`, sendo `n` função do tempo decorrido desde um instante predefinido até ao instante em que um pacote é assemblado; a identificação da versão `n` da chave `Kij` que se usou terá de ser também parte integrante do próprio pacote, para que o receptor saiba que versão de `Kij` deve gerar localmente para decifrar `Kp`.

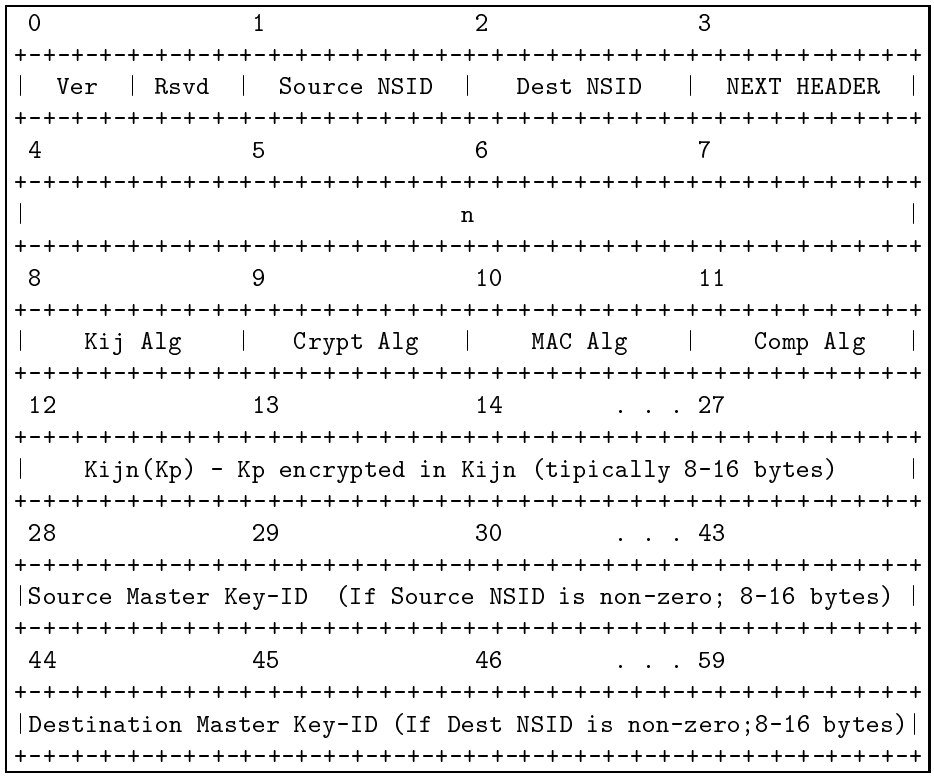

Figura 5.1: Estrutura de um cabeçalho SKIP.

A figura 5.1, adaptada de [AMP95c], reproduz a estrutura de um cabeçalho SKIP. O significado dos seus campos é o seguinte:

- `Ver`: versão do protocolo;
- `Rsvd`: bits reservados para uso futuro;
- `Source NSID`: identificador do espaço de nomes[3] a que pertence o identificador da origem do pacote (ver `Source Master Key-ID`);
- `Dest NSID`: identificador do espaço de nomes a que pertence o identificador do destino do pacote (ver `Destination Master Key-ID`);

[2] E que são tanto mais bem sucedidos quanto mais matéria–prima — *i.e.*, dados cifrados — houver disponível para analisar.

[3] Do inglês *Name Space IDentifier (NSID).*

- `NEXT HEADER`: identificador do protocolo cujo cabeçalho eventualmente se segue ao cabeçalho do SKIP; esse protocolo é, tipicamente, o AH [Atk95b] ou o ESP [Atk95c];
- `n`: contador de 32 bits que define a versão de `Kij` utilizada para cifrar `Kp`; basicamente, este contador é usado para prevenir *ataques–de–repetição* e reutilização de chaves `Kp` comprometidas;
- `Kij Alg`: identificador do algoritmo simétrico usado para cifrar `Kp` com a chave `Kijn`; a especificação do SKIP [AMP95c] recomenda que esse algoritmo seja uma cifra de bloco e a encriptação de `Kp` seja feita no modo CBC[4];
- `Crypt Alg`: identificador do algoritmo simétrico usado para assegurar Privacidade sobre o corpo (dados do utilizador) do pacote;
- `MAC Alg`: identificador do algoritmo usado para assegurar Autenticação e Integridade sobre o corpo do pacote, o que se consegue à custa da produção e verificação de um MAC[5];
- `Comp Alg`: identificador do algoritmo de compressão usado sobre o corpo do pacote;
- `Kijn(Kp)`: chave `Kp` cifrada com base no algoritmo `Kij Alg` e utilizando a `n`'ésima versão da chave `Kij`;
- `Source Master Key-ID`: identificador da origem do pacote;
- `Destination Master Key-ID`: identificador do destino do pacote.

Os campos `Source NSID`, `Dest NSID`, `Source Master Key-ID` e `Destination Master Key-ID` permitem a utilização de outros identificadores que não endereços IP, para identificar a origem e o destino dos pacotes IP protegidos pelo SKIP. Se o campo `Source NSID` tiver o valor zero, então assume-se o endereço IP de origem como identificador da entidade originadora do pacote (e nesse caso o campo `Source Master Key-ID` está ausente). Caso contrário, o campo `Source NSID` contém o identificador de um espaço de nomes ao qual pertence o identificador da origem, contido no campo `Source Master Key-ID`. Analogamente se define o papel dos campos `Dest NSID` e `Destination Master Key-ID`, desta feita relativos à identificação do destino.

Por exemplo, se o campo `Source NSID` indicasse a opção pela utilização de nomes distintos X.500, então o campo `Source Master Key-ID` poderia conter o nome distinto correspondente ao sujeito do certificado X.509 da chave pública de Diffie–Hellman do originador (ou até um *hash*/síntese MD5 daquele nome distinto). A versão 1 do SKIP prevê vários tipos de identificadores para além de endereços IPv4, entre os quais: endereços IPv6, endereços MAC 802.x, sínteses MD5 de endereços RFC-822, sínteses MD5 de nomes NIS, etc. A utilização de esquemas de identificação independentes do endereço IP de origem (ou destino) tem vantagens evidentes, nomeadamente a flexibilidade que confere às entidades participantes na comunicação ao não vinculá-las a uma localização específica ou a um único espaço de nomes. Tal pode ser de grande interesse, por exemplo, no domínio

[4] Por exemplo, a variante CBC do DES, referida na secção 2.2.1, adapta–se perfeitamente a estes requisitos.

[5] Do inglês *Message Authentication Code* — recordar a secção 2.5.2.

da Computação Nómada. Adicionalmente, torna–se possível endereçar várias entidades residentes no mesmo endereço IP e cuja esquema de designação é variável.

O campo `n` contém um número de 32 bits que representa o número de horas decorridas desde um instante predefinido em *Tempo Universal Coordenado* (UTC)[6] — `Jan 1, 1995 00:00:00` — ou seja, `n`= $tempo_{actual} - tempo_{inicial}$, normalizado a horas[7]. O receptor de um pacote SKIP recupera `n` e compara–o com um `n` calculado localmente. O pacote SKIP não é aceite se os valores remoto e local de `n` divergirem em mais de uma unidade, ou seja, $\mid n_{local} - n_{remoto} \mid > 1$ é o critério de rejeição. Este mecanismo de protecção contra *ataques–de–repetição* opera, portanto, a um nível de granularidade relativamente alto e, segundo [AMP95c], apenas exige a sincronização dos relógios do emissor e do receptor com o desfazamento máximo de uma hora.

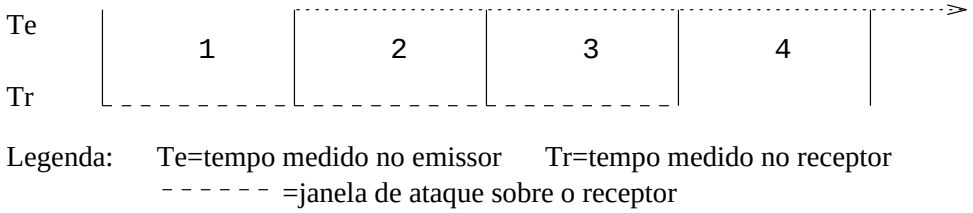

Figura 5.2: Ataque–de–repetição sobre um receptor atrasado.

Todavia, a figura 5.2 demonstra que a janela temporal em que um *ataque–de–repetição* é viável não se encontra limitada a uma hora. Supondo que o relógio do receptor se encontra atrasado em relação ao do emissor, qualquer repetição de uma mensagem originalmente marcada na hora 2 será tida como válida no receptor se recebida durante a hora 1, 2 ou 3. Assim, um *ataque–de–repetição* iniciado na hora 2 pode prolongar–se durante o tempo em que ainda for possível fazer chegar as mensagens repetidas ao receptor sem que este as receba para lá da sua hora 3.

Como se verá na secção 5.2.1, o S3L opera com margens mais estreitas em termos da granularidade de `n` bem como permite parametrizar a diferença entre o seu valor de emissão e de recepção, oferecendo, portanto uma melhoria sensível da protecção pretendida com este mecanismo face a *ataques–de–repetição*.

A geração da `n`'ésima versão da chave `Kij`, designada de `Kijn`, baseia-se na utilização da função de síntese MD5 sobre `Kij` e `n`. Segundo [AMP95c]:

$$\texttt{Kijn}=\text{MD5}(\texttt{Kij} \mid \texttt{n} \mid \texttt{0x01}) \mid \text{MD5}(\texttt{Kij} \mid \texttt{n} \mid \texttt{0x00}),$$

traduzindo "|" a operação de concatenação de bits e sendo `0x01` e `0x00` representações hexadecimais de 1 e 0, respectivamente. Produzem–se assim 256 bits, número suficiente para as chaves dos algoritmos simétricos mais comuns.

O SKIP advoga a separação das chaves usadas no processo de Encriptação e de Autenticação por forma a que o conhecimento de uma delas não comprometa a outra. Consequentemente, a chave `Kp` não é usada directamente em nenhum destes processos. Alternativamente, são geradas as chaves `E_Kp` e `A_Kp`, a utilizar na Encriptação/Desencriptação e na Autenticação, respectivamente. Segundo [AMP95c], estas chaves devem ser geradas

[6] Do inglês *Universal Time Coordinated*.

[7] E com $tempo_{actual}$ e $tempo_{inicial}$ expressos em valores UTC.

pela aplicação da função de síntese MD5 à chave `Kp` e aos valores de `Crypt Alg` e `MAC Alg`, da seguinte forma:

`E_Kp`=MD5(`Kp` | `Crypt Alg` | `0x02`) | MD5(`Kp` | `Crypt Alg` | `0x00`)
`A_Kp`=MD5(`Kp` | `MAC Alg` | `0x03`) | MD5(`Kp` | `MAC Alg` | `0x01`)

Os valores de 256 bits produzidos são, mais uma vez, suficientes para a generalidade dos algoritmos simétricos e de produção de MACs.

5.2 Mensagens S3L ponto–a–ponto

Conceptualmente, o S3L e o SKIP (nas suas variantes ponto–a–ponto[8]), são bastante semelhantes. Contudo, importa desde já assinalar que o S3L e o SKIP operam a níveis distintos em termos da Arquitectura do Sistema Operativo e do Modelo de Comunicações. O SKIP, recorde–se (rever secção 3.4.2), é uma abordagem ao nível da Camada de Rede, integrada directamente no *kernel* e oferecendo, por isso, Privacidade, Autenticação, Integridade, Sequenciação (contra *ataques–de–repetição*) e Compressão de uma forma transparente às camadas superiores. O S3L constitui, *grosso modo*, uma transposição do modelo de operação do SKIP para a Camada de Aplicação, tendo como objectivo fundamental facilitar a utilização daquelas qualidades de serviço pelas aplicações que adoptam os *sockets* de Berkeley como mecanismo de comunicação.

As semelhanças entre o SKIP e o S3L resultam evidentes da comparação da estrutura dos cabeçalhos das respectivas mensagens (figura 5.1 *versus* figura 5.3, respectivamente). Todavia, um cabeçalho S3L consome 93 bytes, ou seja, mais 33 bytes que um cabeçalho SKIP na sua dimensão máxima. A justificação para tal facto encontra-se na introdução dos novos campos `Source IP Address` e `Delta` bem como nos 32 bits adicionais para o campo `n` e nos 24 bytes adicionais consumidos pelo campo `Kp`, em relação aos mesmos campos do cabeçalho SKIP. As razões que ditaram a introdução de novos campos bem como o incremento da dimensão de outros serão esclarecidas, oportunamente, ao longo desta secção.

Os campos `Ver`, `Rsvd`, `n`, `Kij Alg`, `Crypt Alg`, `MAC Alg`, `Comp Alg` e `Kijn(Kp)` desempenham a mesma função que os campos homólogos do cabeçalho SKIP, mas não assumem, necessariamente, os mesmo valores que tomariam no contexto desse protocolo. Situando-se o S3L e o SKIP em níveis hierárquicos completamente distintos, então a questão da interoperabilidade entre implementações de ambos não se coloca. Por consequência, não foi feito nenhum esforço para garantir a compatibilização dos valores assumidos pelos campos comuns. Nomeadamente, a identificação dos diversos algoritmos de encriptação (`Kij Alg` e `Crypt Alg`) e de produção de MACs (`MAC Alg`) seguiu, no caso do S3L, notações importadas do SecuDE, por razões óbvias de conveniência. Para os algoritmos de compressão (`Comp Alg`) usaram-se representações criadas expressamente para o efeito, no âmbito do S3L. Os valores possíveis para estes campos encontram–se discriminados na secção B.2.1 do apêndice B.

Uma diferença fundamental entre o S3L e o SKIP diz respeito ao esquema de identificação da origem e do destino das mensagens. O S3L simplifica o esquema de identificação

[8] A fim de facilitar a exposição, considerar–se–ão, ao longo deste capítulo, apenas as variantes ponto–a–-ponto destas abordagens, salvo indicação explícita em contrário.

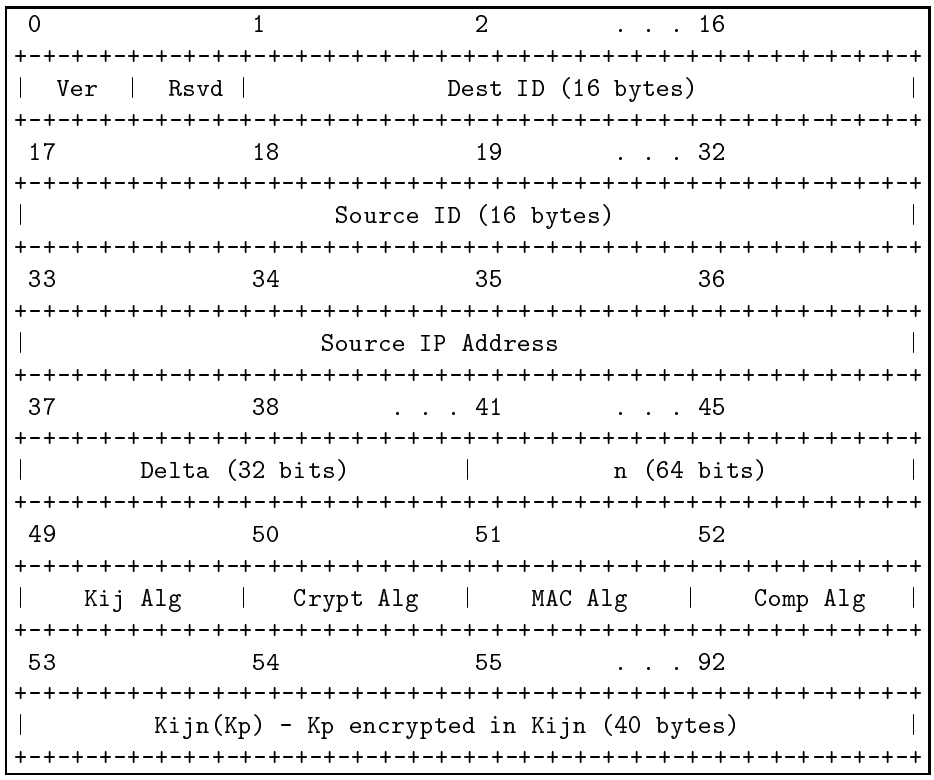

Figura 5.3: Estrutura do cabeçalho de uma mensagem S3L.

originalmente proposto pelo SKIP. Assim, o S3L opta pela utilização de um único tipo de identificadores os quais consistem na síntese MD5 do nome distinto que identifica o sujeito dos certificados X.509 das chaves públicas RSA das entidades comunicantes[9]. Consequentemente, dispensam-se os campos `Source NSID` e `Dest NSID` do SKIP, definindo o S3L os campos `Dest ID` e `Source ID`, equivalentes aos campos `Destination Master Key-ID` e `Source Master Key-ID`, do SKIP. No contexto do presente trabalho, julga-se que esta perda de flexibilidade em termos de possibilidades de identificação não constitui em si uma limitação significativa, já que o esquema de certificação importado do SecuDE (e ao qual o Skeyx e o S3L se encontram, de base, indissociavelmente ligados) implica a identificação das diversas entidades com base nos nomes distintos X.500 encerrados nos respectivos certificados X.509. Assim sendo, questiona-se a necessidade de outro esquema de identificação. Um benefício adicional da utilização de sínteses MD5 consiste na preservação do anonimato das entidades comunicantes caso as mensagens sejam interceptadas, uma vez que o mapeamento reverso das sínteses MD5 é, por definição de função de *hashing* unidireccional, inviável[10].

Pelo exposto, parece não se justificar a introdução de um campo adicional com o endereço IP de origem (`Source IP Address`) na mensagem S3L. Na secção 5.4 a necessidade

[9] Recorde-se que a disponibilidade deste tipo de certificação é fundamental e é, inclusivamente, o único requisito prévio (em termos de infra–estrutura de certificação) para que o Skeyx possa efectuar a troca de chaves públicas de Diffie–Hellman não certificadas.

[10] Desde que o domínio de partida — neste caso o conjunto de todos os nomes distintos X.500 — seja suficientemente numeroso, bem entendido. Caso contrário, um *ataque–de–dicionário* pode ser viável.

deste campo será devidamente fundamentada.

Comparado com o campo `Kp` do cabeçalho SKIP, o mesmo campo no cabeçalho S3L consome, pelo menos, mais 24 bytes. Ora, o SKIP reserva, no máximo, 16 bytes (128 bits) para `Kp`. O S3L reserva o dobro, ou seja, 32 bytes (256 bits). Desta forma fica acautelada a possível utilização futura de chaves simétricas de 256 bits. Quanto aos 8 bytes adicionais que perfazem os 24, eles não fazem parte da chave, resultando da forma particular como o SecuDE implementa a variante CBC dos algoritmos simétricos (como é o caso do DES): são produzidos blocos de 64 bits e, no caso das variantes CBC, é ainda adicionado mais um bloco de 64 bits. Este bloco adicional é absolutamente necessário para permitir decifrar correctamente a chave `Kp` contida nos 32 bytes que o precedem.

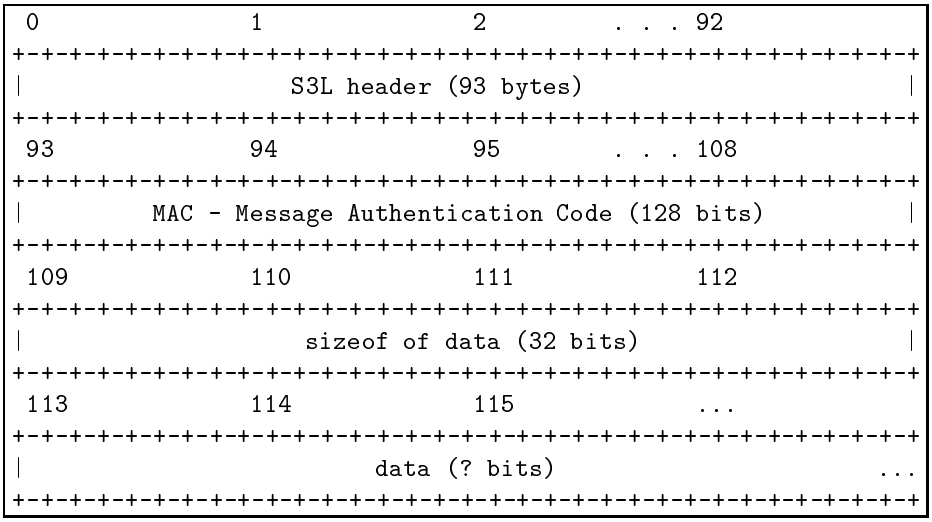

Figura 5.4: Estrutura de uma mensagem S3L.

A especificação do SKIP [AMP95c] não define a localização do MAC, nem tão pouco dos dados a transportar na mensagem, deixando essa tarefa ao critério do protocolo definido no campo `NEXT HEADER`. No caso do S3L, a estrutura completa de uma mensagem pode ser observada na figura 5.4.

O MAC é calculado pela aplicação do algoritmo definido por `MAC Alg` e da chave `A_Kp`[11] sobre a totalidade da mensagem, considerando o campo `MAC` inicializado a zero, uma vez que este campo é o último a ser definido.

O campo `size of data` contém a dimensão do campo de dados `data`. Essa dimensão é especificada em bits por forma a tornar mais expedita a utilização das funções de encriptação e desencriptação do SecuDE, as quais medem as entradas e as saídas em bits. Este campo é, aparentemente, sobredimensionado, dado que o IP reserva apenas 16 bits para a dimensão do campo de dados, sendo essa dimensão especificada em bytes. Todavia, uma mensagem S3L não é produzida forçosamente com o objectivo de ser enviada através da rede. O S3L dispõe de funções na sua API (ver secção 5.3.3) para a produção de mensagens S3L de uma forma independente dos seu métodos de transmissão ou armazenamento. Por exemplo, uma mensagem S3L pode ser depositada num ficheiro e entregue manualmente ao destinatário. Com 32 bits é possível o processamento de dados do utilizador com

[11] Derivada de `Kp` segundo o mesmo método aplicado no SKIP — rever secção 5.1.

dimensão máxima de $(2^{32})/8$ bytes, ou seja, 32768 Mbytes.

Uma mensagem S3L tem, no mínimo 114 bytes, uma vez que o campo de dados `data` não pode ser vazio. Refira–se que o facto de a dimensão do cabeçalho[12] ser de dimensão fixa não é acidental. Tal facilitou bastante a codificação da variante S3L das funções de leitura sobre *sockets* de Berkeley.

5.2.1 Critérios de Rejeição de Mensagens S3L Repetidas

O S3L combate *ataques–de–repetição* através de um mecanismo semelhante ao do SKIP e que dispensa preservar, para cada origem possível, a última marca temporal[13] válida. A utilização de tal mecanismo está, aliás, em plena concordância com o carácter *stateless* que o S3L pretende assumir. Adicionalmente, são evidentes os ganhos de desempenho que se conseguem ao adoptar uma estratégia que evita a pesquisa de marcas temporais (mesmo em *cache*) associadas a um originador, de cada vez que se recebe uma mensagem S3L.

Os campos `Delta` e `n` de uma mensagem S3L sustentam o referido mecanismo de combate a *ataques–de–repetição*. O campo `n` continua a preservar o "instante UTC" em que a mensagem é marcada pelo originador. Porém, relativamente ao mesmo campo do cabeçalho SKIP, a sua dimensão é agora de 64 bits uma vez que a resolução dos valores temporais usados pelo S3L é o microsegundo. O novo campo `Delta` destina–se a parametrizar o desvio máximo admitido entre o tempo instantâneo na recepção e o tempo instantâneo na produção da mensagem (este último dado por `n`). Relembrando que o SKIP fixa o desvio numa hora[14], não só o S3L permite variar esse desvio, como a sua resolução é muito mais fina, sendo, como `n`, da ordem do microsegundo. O valor de `Delta` pode ser encarado como uma espécie de "prazo de validade" da mensagem, por forma a que repetições da mensagem sejam detectadas ao verificar–se que o seu tempo de vida expirou. Quando `Delta` é zero isso significa que a mensagem S3L foi gerada tendo em vista um tempo de vida infinito. Tal pode ser útil em determinadas circunstâncias como por exemplo: quando é completamente imprevisível o atraso na entrega da mensagem S3L ou quando se pretende criar uma mensagem S3L e mantê-la arquivada num ficheiro por tempo indeterminado.

Poderá questionar–se a necessidade de usar uma granularidade tão fina na medição de tempos e desvios entre relógios. Tal opção constitui uma tentativa para contrariar as possíveis janelas de ataque abertas por desempenhos cada vez maiores ao nível das tecnologias de rede e do poder de processamento das máquinas onde residem as entidades comunicantes. Simultaneamente, é importante lembrar que as trocas de mensagens S3L podem ser feitas entre entidades comunicantes na mesma máquina pelo que, nestas circunstâncias, o factor rede é, praticamente, irrelevante, sendo necessário reduzir ao mínimo as oportunidades de ataque.

O cálculo do tempo de vida δ (`Delta`) a atribuir a uma mensagem no acto da sua produção não é trivial. Por um lado, é difícil prever o tempo de propagação através da rede,

[12] Onde, a partir de agora, se consideram também incluídos os campos `MAC` e `size of data`.

[13] Usada, portanto, como número de sequência.

[14] Na prática, esse desvio pode ser quase de duas horas. Admitam-se, por exemplo, tempos UTC, tal que no emissor $n_E = 1000$ horas, resultante da conversão, para horas, do tempo $999 * 3600 + 1$ segundos, e no receptor $n_R = 1001$, resultante da conversão, para horas, do tempo $1001 * 3600$ segundos — *i.e.*, o último segundo da hora 1001 —; neste caso $n_R - n_E = 1$ mas, em segundos, a distância temporal corresponde quase a duas horas.

uma vez que o seu estado é, por natureza, imprevisível. Adicionalmente, os relógios das entidades comunicantes não se podem supor, no caso mais geral, sincronizados. Ainda assim, é possível estabelecer estimativas para δ por forma a minimizar a vulnerabilidade a *ataques-de–repetição*. De seguida, apresentam–se os casos típicos possíveis em termos de sincronização dos relógios do emissor e do receptor, bem como os critérios de aceitação/rejeição das mensagens S3L, incluindo as respectivas estimativas para δ. Todos os tempos em causa se supõem normalizados para o respectivo valor UTC.

Emissor e Receptor Sincronizados

O caso mais simples (e também o menos provável) corresponde à situação em que os relógios do emissor e do receptor estão sincronizados (ver a figura 5.5).

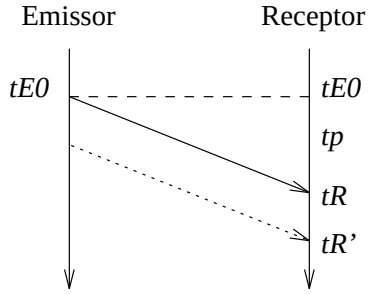

Figura 5.5: Emissor e Receptor sincronizados.

Sendo t_p o tempo de propagação, então o critério de aceitação de uma mensagem originalmente produzida no instante t_{E_0} e recebida no instante t_R será $t_R - t_{E_0} \leq \delta$, com $\delta = t_p$. A repetição recebida no instante $t_{R'}$ não é aceite dado que $t_{R'} - t_{E_0} > \delta$.

Emissor Atrasado Relativamente ao Receptor

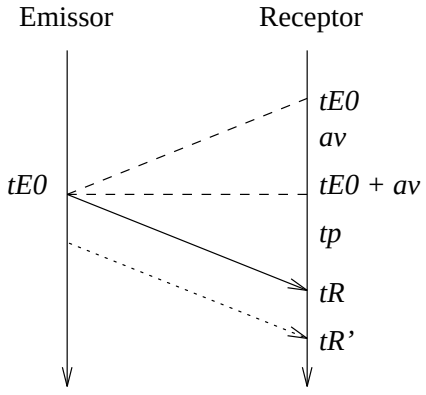

Figura 5.6: Emissor atrasado relativamente ao Receptor.

Neste caso (ver figura 5.6), a aplicação do critério de aceitação $t_R - t_{E_0} \leq \delta$ é válido desde que o emissor tenha estimado $\delta = av + t_p$, sendo av o avanço que o relógio do receptor tem sobre o relógio do emissor.

Emissor Adiantado Sobre o Receptor

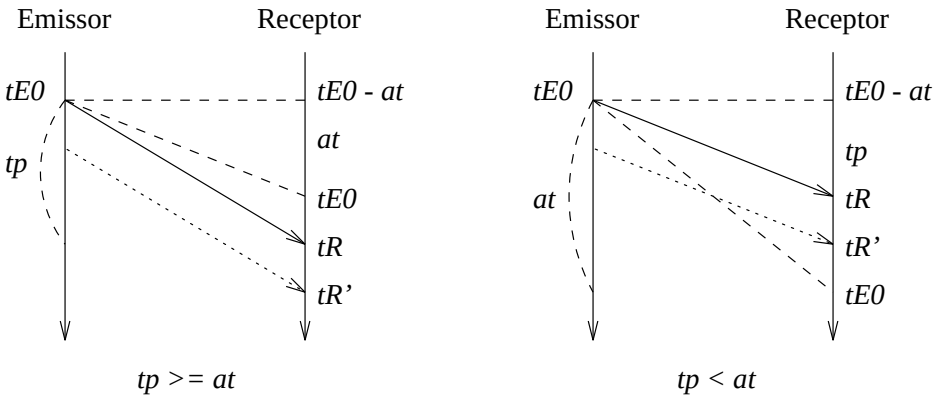

Figura 5.7: Emissor adiantado relativamente ao Receptor.

Existem dois casos possíveis (ver figura 5.7), conforme a relação entre o tempo de propagação e o atraso entre os relógios:

1. tempo de propagação superior ao atraso dos relógios ($t_p \geq at$): a aplicação do critério de aceitação $t_R - t_{E_0} \leq \delta$ ainda é válido desde que o emissor tenha estimado $\delta = t_p - at$, sendo at o atraso que o relógio do receptor tem relativamente ao relógio do emissor;

2. tempo de propagação inferior ao atraso dos relógios ($t_p < at$): o critério de aceitação é agora $t_{E_0} - t_R \geq \delta$ desde que o emissor tenha estimado $\delta = at - t_p$.

Estimativas para δ

Uma vez que a determinação correcta dos desfazamentos av e at e do tempo de propagação t_p é difícil, podem usar–se aproximações para δ tendo, todavia, a consciência de que se abrem janelas de ataque cuja dimensão depende, obviamente, da qualidade da aproximação. Assim, quando o critério de aceitação é $t_R - t_{E_0} \leq \delta$, pode usar–se um majorante de δ. Quando o critério de aceitação é $t_{E_0} - t_R \geq \delta$, fornece–se um minorante de δ.

5.3 A Interface de Programação de Aplicações do S3L

Uma vez estabelecida a estrutura das mensagens S3L (bem como a sua relação com as mensagens SKIP), cabe agora apresentar o conjunto de funcionalidades necessárias à criação, transmissão, recepção e processamento daquelas mensagens.

Sob o ponto de vista do programador de aplicações, o S3L apresenta–se como um conjunto de estruturas de dados e funções, codificadas em C, que lhe permitem continuar a usar a abstracção dos *sockets* de Berkeley, mas com qualidades de serviço acrescidas, concretamente: Privacidade, Autenticação, Integridade, Sequenciação (contra *ataques–de–repetição*) e Compressão, em combinações diversas.

As etapas envolvidas na utilização das funcionalidades S3L por parte de uma entidade são, fundamentalmente, as seguintes:

- tarefas do produtor de mensagens S3L:

  1. obtenção de informação que identifica e caracteriza a entidade produtora;
  2. inicialização de um contexto S3L, especificando a entidade destino da mensagem e as qualidades de serviço pretendidas;
  3. assemblagem da mensagem, com base no contexto seguro definido;
  4. envio pela rede, utilizando as funções de escrita sobre *sockets* providenciadas pelo S3L; alternativamente, outras formas de transmissão poderão ser empregues, como por exemplo o armazenamento em memória secundária tendo em vista a posterior "entrega manual" ao destinatário;
  5. repetição das etapas 2, 3 e 4, com possível reutilização dos contextos produzidos na etapa 2;

- tarefas do consumidor de mensagens S3L:

  1. obtenção de informação que identifica e caracteriza a entidade consumidora;
  2. inicialização de um contexto S3L, preparando–o para acolher a informação contextual produzida pela análise de uma mensagem S3L recebida;
  3. recepção de uma mensagem S3L a partir da rede com base nas funções de leitura sobre *sockets* providenciadas pelo S3L, ou obtenção da mensagem através de outros meios, *c.f.* o seu método original de armazenamento e distribuição (*e.g.*, leitura de um ficheiro);
  4. processamento da mensagem S3L, da qual resulta a recuperação dos dados e o preenchimento do contexto S3L previamente inicializado;
  5. repetição das etapas 3 e 4, com possível reutilização do mesmo contexto (a etapa 2 dispensa–se porque o contexto de recepção é automaticamente sobreposto se for reutilizado).

As próximas secções apresentam as estruturas de dados e funções da Interface de Programação de Aplicações (API)[15] do S3L que são usadas nestas etapas. Todavia, o que se expõe a seguir não pretende substituir, mas antes complementar a secção B.2 do apêndice B onde, sob a forma de "manual de programação", a API do S3L será apresentada com mais detalhe. Em particular, chama–se a atenção para os programas cliente e servidor da secção B.2.8, cuja observação deverá permitir uma melhor compreensão da metodologia a seguir na utilização da API do S3L ponto–a–ponto.

5.3.1 Manipulação de Informação de "Carácter Pessoal"

A figura 5.8 apresenta a estrutura `S3LPartieInfo`, onde é preservada a informação relevante que caracteriza cada entidade comunicante.

O acesso à estrutura `S3LPartieInfo` deve ser feito exclusivamente com base nas seguintes funções[16]:

[15] Do inglês *Application Programming Interface*.

[16] Esta imposição pretende introduzir alguma disciplina no acesso a estruturas de dados críticas para o S3L (ver também a secção 5.3.2, relativa à estrutura `S3LCtx`), semelhante à que se obteria pela utilização de uma classe caso a codificação do S3L tivesse sido feita numa linguagem orientada aos objectos.

```
typedef struct {

char *psedir;        /* caminho completo para a directoria do PSE */
char *dname;         /* nome distinto X.500 da entidade */
char md5dname[16];   /* sintese MD5 do campo dname */
char ipaddr[16];     /* endereco IP onde reside a entidade */
char *symlist;       /* algoritmos simetricos suportados pela entidade */
char *cbclist;       /* subconjunto de symlist com os algoritmos da
                        categoria CBC */
} S3LPartieInfo;
```

Figura 5.8: Estrutura `S3LPartieInfo`.

- `void S3Linit_S3LPartieInfo(S3LPartieInfo *info)`: inicializa os apontadores da estrutura `info` a `NULL`;
- `S3Lbool S3Lfill_S3LPartieInfo(S3LPartieInfo *info)`: preenche os campos da estrutura `info`;
- `void S3Lfree_S3LPartieInfo(S3LPartieInfo *info)`: liberta a memória dinâmica associada a alguns dos campos da estrutura `info`.

A ordem correcta de invocação destas funções corresponde, precisamente, à ordem da sua apresentação (`S3Linit_S3LPartieInfo`, `S3Lfill_S3LPartieInfo` e finalmente `S3Lfree_S3LPartieInfo`) e ocorre, tipicamente, uma só vez, ao longo da execução de um programa.

5.3.2 Manipulação de Contextos Seguros

Toda a informação de contexto necessária à síntese de uma mensagem S3L, ou resultante da sua análise[17], é concentrada na estrutura `S3LCtx`, apresentada na figura 5.9.

À semelhança da estrutura `S3LPartieInfo`, também o acesso à estrutura `S3LCtx` deve ser feito exclusivamente através de funções próprias:

- `void S3Linit_S3LCtx(S3LCtx *ctx, S3LPartieInfo *info)`: inicializa os campos apontadores da estrutura `ctx` a `NULL` excepto o campo `loc_info`, que é inicializado com o valor do parâmetro `info` (desta forma toda a informação relativa à entidade local passa a fazer automaticamente parte do contexto `ctx`);
- `S3Lbool S3Lmake_S3LCtx (S3LCtx *ctx, char *rem_id, char nsid, u_long delta, char *kijalg, char *cryptalg, char *macalg, char *compalg, char *rem_ipaddr, S3Lbool *skeyx_called)`: preenche os diversos campos da estrutura `ctx` por forma a criar um contexto S3L; caso a entidade remota identificada pelo nome distinto[18] `rem_id` careça de uma entrada na *cache* local de chaves públicas de Diffie–Hellman, o protocolo Skeyx é executado automaticamente com a entidade remota, cuja localização é dada por `rem_ipaddr`[19];

[17] Do inglês *parsing*.
[18] Ou síntese MD5 desse nome, *c.f.* indicação da *flag* `nsid`.
[19] O tópico das execuções automáticas (*i.e.*, implícitas) do Skeyx será explorado na secção 5.4.

```
typedef struct {

u_char    version;       /* versao do protocolo S3L */

S3LPartieInfo *loc_info; /* informacao relativa a entidade local */

char rem_md5dname[16]; /* sintese MD5 do nome dist. da entidade remota */
char *rem_entry;       /* entrada da entidade remota, na cache local */
char *rem_symlist;     /* algoritmos simetricos da entidade remota */
char *rem_cbclist;     /* subconjunto de rem_symlist com os algoritmos
                          da categoria CBC */

BitString dhagreedkey;   /* chave acordada de Diffie--Hellman */

struct timeval  rem_n; /* marca temporal remota */
struct timeval  loc_n; /* marca temporal local */
u_long          delta; /* desvio maximo admissivel entre loc_n e rem_n */

char *kijalg;     /* algoritmo de encriptacao de Kp */
char *cryptalg;   /* algoritmo de encriptacao dos dados */
char *macalg;     /* algoritmo de producao do MAC */
char *compalg;    /* algoritmo de compressao dos dados */

} S3LCtx;
```

Figura 5.9: Estrutura `S3LCtx`.

- `void S3Lfree_S3LCtx(S3LCtx *ctx)`: liberta a memória dinâmica associada a alguns dos campos da estrutura `ctx`.

Tipicamente, as funções `S3Linit_S3LCtx` e `S3Lfree_S3LCtx` são invocadas uma única vez, no princípio e no fim de um programa, respectivamente. A função `S3Lmake_S3LCtx` deve ser invocada sempre que se pretende definir um contexto (ou modificar um preexistente) tendo em vista a assemblagem de uma mensagem S3L. Adicionalmente, esta função é invocada automaticamente durante a análise de uma mensagem S3L (ver função `S3Lopen_message`, na secção 5.3.3), definindo–se assim um contexto com a informação coligida durante esse processo.

5.3.3 Síntese e Análise de Mensagens S3L

O processamento de mensagens S3L é levado a cabo por funções específicas, conforme se trate da síntese ou da análise dessas mensagens:

- `S3Lbool S3Lmake_message (char *buf_in, u_long len_buf_in, char *buf_out, u_long *len_buf_out, S3LCtx *ctx)`: assembla uma mensagem S3L, com base nos dados `buf_in` (de dimensão `len_buf_in`) e no contexto `ctx`; a mensagem S3L produzida é depositada em `buf_out`, reflectindo `len_buf_out` a sua dimensão;
- `S3Lbool S3Lopen_message (char *buf_in, u_long len_buf_in, char *buf_out, u_long *len_buf_out, S3LCtx *ctx)`: analisa, campo a campo, uma mensagem

S3L, fornecida em `buf_in`; o contexto `ctx` recebe a informação contextual resultante da análise da mensagem; os dados recuperados são depositados em `buf_out` e a sua dimensão indicada através de `len_buf_out`.

A função `S3Lmake_message` é relativamente linear, dado que toda a informação contextual necessária à produção de uma mensagem S3L se supõe fornecida através do contexto `ctx`, previamente inicializado. Nesse contexto é possível indicar diversas combinações das qualidades de serviço pretendidas, conforme o valor que os campos `delta`, `kijalg`, `cryptalg`, `macalg` e `compalg` assumem. A Encriptação, a Compressão e a Sequenciação (cujas modalidades se indicam, respectivamente, nos campos `cryptalg`, `compalg` e `delta`[20]) dos dados são opcionais, sendo a Encriptação de `Kp` (com o algoritmo indicado em `kijalg`) e a verificação da Autenticidade e da Integridade (com o algoritmo indicado em `macalg`) automaticamente fornecidos. Se houver lugar à compressão de dados, ela ocorrerá antes da eventual encriptação. O cálculo do MAC é, naturalmente, efectuado em último lugar e considerando o respectivo campo `MAC` na mensagem S3L inicializado a zero.

A função `S3Lopen_message` é mais complexa porque tem de ser capaz de lidar com a recepção de mensagens S3L produzidas por entidades para as quais ainda não existe entrada na *cache* local de chaves públicas de Diffie–Hellman ou, existindo essa entrada, se verifique que a chave acordada de Diffie–Hellman aí residente é obsoleta. Estas situações[21] enquadram–se nas possíveis causas de uma ocorrência implícita do protocolo Skeyx, pelo que são desenvolvidas com mais detalhe na secção 5.4. É também a função `S3Lopen_message` que valida a recepção de uma mensagem em termos temporais, usando os critérios apresentados na secção 5.2.1. A Autenticidade e Integridade da mensagem são verificadas pela comparação de um MAC local com o fornecido pela mensagem. Os dados são descomprimidos e decifrados, se esse for o caso.

As funções `S3Lmake_message` e `S3Lopen_message` podem ser usadas "isoladamente", sem uma relação directa com a escrita e leitura sobre *sockets* de Berkeley, a fim de processarem mensagens destinadas ao armazenamento por tempo indefinido ou a outras formas de distribuição.

5.3.4 Escrita e Leitura de Mensagens S3L sobre Sockets

A possibilidade de utilizar o mecanismo de *sockets* com o fim de transmitir mensagens S3L assenta na disponibilização de funções de escrita e leitura semelhantes às primitivas de Berkeley mas que encapsulam os detalhes da assemblagem e desassemblagem das mensagens S3L, recorrendo às primitivas de Berkeley para, em última instância, efectuar a escrita e a leitura das mensagens.

Assim, o S3L oferece as seguintes funções de escrita sobre *sockets* de Berkeley:

- `int S3Lwrite (int fd, char *buf, size_t count, S3LCtx *ctx);`
- `int S3Lsend (int fd, void *msg, int len, unsigned int flags, S3LCtx *ctx);`

[20] Um valor zero significa que não se pretende detecção de *ataques–de–repetição*.

[21] Que, na realidade, são detectadas e resolvidas pela função `S3Lmake_S3LCtx`, invocada por `S3Lopen_message` a fim de sintetizar o contexto resultante da análise da mensagem.

- `int S3Lsendto (int fd, void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen, S3LCtx *ctx);`
- `int S3Lwritev (int fd, struct iovec *vector, size_t count, S3LCtx *ctx);`
- `int S3Lsendmsg (int fd, struct msghdr *msg, unsigned int flags, S3LCtx *ctx).`

Em cada uma destas funções, procede–se à assemblagem (via `S3Lmake_message`) da mensagem S3L onde são encapsulados os dados fornecidos pelo utilizador, seguindo-se, tipicamente, a invocação da função homóloga de Berkeley.

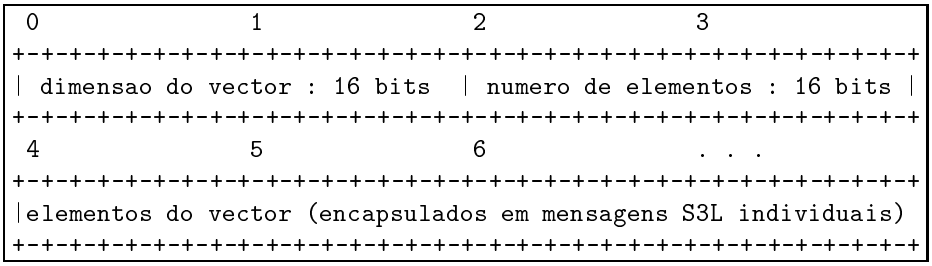

Figura 5.10: Vector `struct iovec` linearizado.

No caso concreto de `S3Lwritev` e de `S3Lsendmsg`, cada mensagem dos vectores `vector` e `msg->msg_iov`, respectivamente, é encapsulada na mensagem S3L correspondente, resultando assim um vector de mensagens S3L. Este vector é então linearizado e o resultado (ver figura 5.10) é enviado *via* `write` no caso de `S3Lwritev`, uma vez que não se justifica a utilização de `writev` para enviar um vector com um único elemento[22]. Quanto a `S3Lsendmsg`, é obrigatória a utilização de `sendmsg` uma vez que esta primitiva pode efectuar o transporte de mensagens de controle entre processos[23], para além de dados do utilizador. A única forma de assegurar que as mensagens de controle eventualmente especificadas em `msg->msg_control` são transmitidas ao destino é, portanto, a utilização da primitiva `sendmsg`.

Relativamente às funções de leitura sobre *sockets*, o S3L disponibiliza as seguintes:

- `int S3Lread (int fd, char *buf, size_t count, S3LCtx *ctx);`
- `int S3Lrecv (int fd, char *buf, int len, unsigned int flags, S3LCtx *ctx);`
- `int S3Lrecvfrom (int fd, char *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen, S3LCtx *ctx);`
- `int S3Lreadv (int fd, struct iovec *vector, size_t count, S3LCtx *ctx);`

[22] O qual consiste numa sucessão de mensagens S3L.

[23] Recorde–se, por exemplo, a sua utilização mencionada na secção 4.4, a fim de efectuar a passagem de descritores TCP entre processos sem grau de parentesco directo.

- `int S3Lrecvmsg (int fd, struct msghdr *msg, unsigned int flags, S3LCtx *ctx)`.

Genericamente, estas funções invocam as funções equivalentes de Berkeley para ler uma mensagem S3L da rede, após o que recuperam os dados do utilizador ali encapsulados, pela invocação da função `S3Lopen_message`.

Mais uma vez, as funções S3L de leitura que operam sobre vectores têm um comportamento ligeiramente diferente das suas congéneres de Berkeley. Assim, `S3Lreadv` e `S3Lrecvmsg` lêem uma só mensagem (um vector linearizado) que consiste numa sequência de mensagens S3L sobre as quais invocam `S3Lopen_message` por forma a reconstruir o vector original. Como é de esperar, `S3Lreadv` não invoca `readv` para proceder à leitura, uma vez que também `S3Lwritev` não invocou `writev`. Em vez disso, `read`, `recv` ou `recvfrom` são suficientes para ler a mensagem transmitida por `S3Lwritev` *via* `write`. Quanto a `S3Lrecvmsg`, a invocação de `recvmsg` garante o processamento de quaisquer mensagens de controle especificadas pelo originador em `msg` aquando da invocação de `S3Lsendmsg`.

Funções de Escrita		Funções de Leitura	
S3L	*Berkeley*	*S3L*	*Berkeley*
`S3Lwrite`	`write`	`S3Lread` → `S3Lrecvfrom`	`recvfrom`
`S3Lsend`	`send`	`S3Lrecv` → `S3Lrecvfrom`	`recvfrom`
`S3Lsendto`	`sendto`	`S3Lrecvfrom`	`recvfrom`
`S3Lwritev`	`writev`[a], `write`	`S3Lreadv`	`readv`[b], `read`, `recvfrom`
`S3Lsendmsg`	`sendmsg`	`S3Lrecvmsg`	`recvmsg`

[a] Só para retorno de erros.
[b] Só para retorno de erros.

Tabela 5.1: Relações de invocação entre as funções S3L e as primitivas de Berkeley.

As relações de invocação entre as funções de leitura e escrita de Berkeley e as respectivas versões S3L podem ser observadas na tabela 5.1.

Note–se que a invocação de `writev` e de `readv` só é feita em determinadas situações de erro em que se pretende retornar valores tão semelhantes o quanto possível aos que a invocação directa dessas funções produziriam.

É também interessante observar a utilização da função `recvfrom` em situações onde seria de esperar a invocação de `read` e `recv`. Tal justifica–se pelo facto de que `recvfrom` é, na prática, equivalente àquelas funções, bastando para isso atribuir a determinados parâmetros o valor `NULL` ou `0` (zero). Adicionalmente, `recvfrom` permite a utilização da *flag* `MSG_PEEK`, o que facilita a leitura das mensagens S3L sobre *sockets* do tipo `SOCK_DGRAM`[24]: o cabeçalho de uma mensagem S3L é inspeccionado a fim de determinar a dimensão da mensagem completa, após o que a leitura efectiva pode acontecer.

À semelhança das primitivas de Berkeley, é possível combinar as funções de leitura e escrita de formas diversas. Para o S3L, as combinações válidas são dadas na tabela 5.2.

[24] Obviamente que `recv` também suporta esta *flag*, mas como `recv` é equivalente a `recvfrom` com `from=NULL` e `fromlen=NULL`, a invocação directa de `S3Lrecvfrom` por parte de `S3Lrecv` é vantajosa, já que constitui uma boa oportunidade de reutilização de código.

Funções de Escrita	Funções de Leitura				
	`S3Lread`	`S3Lrecv`	`S3Lrecvfrom`	`S3Lreadv`	`S3Lrecvmsg`
`S3Lwrite`	√	√	√	×	×
`S3Lsend`	√	√	√	×	×
`S3Lsendto`	√	√	√	×	×
`S3Lwritev`	×	×	×	√	×
`S3Lsendmsg`	×	×	×	×	√

Tabela 5.2: Combinações possíveis entre as primitivas S3L.

Em termos puramente sintácticos, a única diferença visível entre as primitivas de Berkeley e as respectivas funções S3L é o prefixo `S3L` e a utilização de um parâmetro adicional, que é precisamente o contexto `ctx`.

Em termos semânticos, a compatibilidade da implementação actual do S3L é aceitável na maioria dos casos práticos. Todavia, uma vez que as funções S3L apresentadas são, basicamente, *wrappers* das funções homólogas de Berkeley, não se pode esperar das primeiras um comportamento 100% compatível com as últimas. Por exemplo, em relação a algumas primitivas de Berkeley[25] é possível definir o seu comportamento face à ocorrência de sinais[26]. Assim, uma primitiva `read` que seja interrompida pode ser automaticamente recomeçada, se assim se desejar. Caso contrário, retorna prematuramente com `errno=EINTR`, valor cuja ocorrência deve ser sempre testada, por segurança, após a invocação de uma primitiva que não seja recomeçável automaticamente. Actualmente, o S3L ainda não fornece mecanismos capazes de assegurar o recomeço automático das suas primitivas face à ocorrência de sinais. Adicionalmente, a implementação actual do S3L suporta apenas *sockets* do tipo `SOCK_DGRAM` e `SOCK_STREAM`.

5.4 Ocorrências Implícitas do Protocolo Skeyx

É em resultado da execução do protocolo Skeyx, recorde–se (rever capítulo 4), que as entradas da *cache* de chaves públicas de Diffie–Hellman de uma entidade[27] são criadas e sucessivamente actualizadas.

A criação de uma entrada (ou a sua actualização) na *cache* de chaves públicas de Diffie-Hellman pode ocorrer, basicamente, em três circunstâncias[28]:

1. pela execução da aplicação `s3lkeyxclient`, fornecida na distribuição do S3L (ver secção B.1 do apêndice B) e que, *grosso modo*, fornece a possibilidade de, a partir da linha de comando, invocar uma função `Skeyx`, que inicia o protocolo Skeyx sob o ponto de vista da entidade cliente[29];

2. pela invocação da função `Skeyx`, no decorrer da execução de um programa;

[25] [Ste90] refere `flock`, `ioctl`, `read`, `readv`, `wait`, `write` e `writev` para a versão 4.3 do BSD.

[26] Tal definição ocorre por intermédio da primitiva `siginterrupt`.

[27] Ou melhor, de uma instância dessa entidade.

[28] Supondo, em todos os casos, que a necessária infra–estrutura em termos do PSE e dos demónios Reencaminhador (`s3lforwarder`) e Servidor (`s3lkeyxserver`), se encontra operacional.

[29] No que se segue, a designação `Skeyx` refere–se à função correspondente ao protocolo homónimo, que se continuará a designar de Skeyx.

3. por um demónio Servidor (`s3lkeyxserver`), em resultado de um pedido que lhe foi reencaminhado pelo demónio Reencaminhador (`s3lforwarder`).

A primeira hipótese, à qual corresponde uma execução explícita do protocolo Skeyx, traduz, claramente, uma política *de antecipação*, pela definição prévia de um estado que se prevê ser útil num futuro próximo. Adicionalmente, e se assim o desejar, um programador de aplicações pode também invocar, explícitamente, a função `Skeyx` nos seus programas[30], tal como prevê a segunda hipótese.

Todavia, para o programador de aplicações que recorre à API do S3L, a adopção de invocações explícitas de `Skeyx` (seja indirectamente pela execução de `s3lkeyxclient`, seja directamente no seio de um programa) raramente será desejável. O protocolo Skeyx é, fundamentalmente, um pilar do S3L ponto–a–ponto e multiponto, que, para o programador de aplicações, deve passar despercebido o tanto quanto possível.

É nesta última perspectiva que se encaixa o mecanismo de invocações implícitas/automáticas de `Skeyx` e que, basicamente, concretiza uma política de actualização das *caches por necessidade*. A utilidade deste mecanismo revela–se no acto da definição de contextos S3L (via `S3Lmake_S3LCtx`), seja previamente à assemblagem de uma mensagem S3L, seja após a recepção de tais mensagens:

- pretendendo–se enviar uma mensagem S3L para uma entidade ausente da nossa *cache*, o protocolo Skeyx é executado automaticamente na definição do contexto S3L, no seio da função `S3Lmake_S3LCtx`, pelo que a invocação subsequente de uma função de escrita S3L (`S3Lwrite`, `S3Lsend`, etc.) já se processa com um contexto `ctx` válido, resultante da actualização da *cache*. Tomando como exemplo o cliente da figura B.6 (ver secção B.2.8 do apêndice B), a definição de um contexto na linha 19 pode implicar a execução do protocolo Skeyx, sem que o utilizador se aperceba de tal facto, para além da latência introduzida por essa execução;

- recebendo–se uma mensagem S3L de uma entidade ausente da nossa *cache*[31], então, para conseguir processar essa mensagem é necessário executar o protocolo Skeyx com tal entidade. Ou seja, sempre que a função de desassemblagem `S3Lopen_message` (chamada por todas funções S3L de leitura — `S3Lread`, `S3Lrecv`, etc. —) invoca `S3Lmake_S3LCtx` para definir um contexto S3L relativo à mensagem S3L recebida, `S3Lmake_S3LCtx` pode concluir da necessidade de executar o protocolo Skeyx[32], para o que deverá invocar a função `Skeyx`.

  Esta sucessão de acontecimentos, esquematizada na figura 5.11, pode ocorrer, por exemplo, na linha 39 do servidor da figura B.7 (ver secção B.2.8 do apêndice B).

  Para invocar a função `Skeyx`, `S3Lmake_S3LCtx` necessita de lhe fornecer, entre outros parâmetros, a identificação X.500 da entidade remota e o endereço IP dessa entidade, valores que, por sua vez, terão de ser fornecidos a `S3Lmake_ctx` por

[30] Apesar de, para todos os efeitos, a função `Skeyx` se considerar não documentada, uma vez que executa operações consideradas de carácter privado face ao programador de aplicações que utiliza a API do S3L.

[31] Tal situação é perfeitamente possível: se a entidade originadora conseguiu construir a mensagem S3L é porque dispõe, na sua *cache*, de uma entrada relativa à entidade receptora, mas criada após a execução do protocolo Skeyx com uma outra réplica desta.

[32] Como se verá ainda nesta secção, essa conclusão deriva de dois factos possíveis: ausência da entrada pretendida na *cache* ou presença de uma entrada obsoleta.

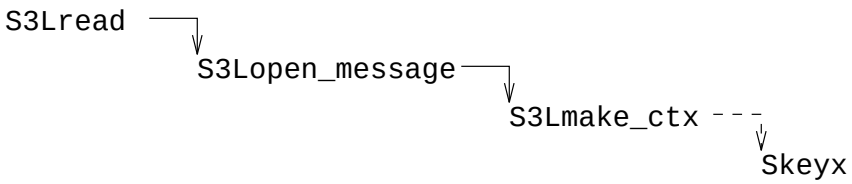

Figura 5.11: Exemplo de execução implícita do protocolo Skeyx.

`S3Lopen_message`. Todavia, `S3Lopen_message` conhece a síntese MD5 do nome distinto do originador (recordar a estrutura do cabeçalho de uma mensagem S3L na figura 5.3) e não o nome distinto original, pelo que esse facto é indicado a `S3Lmake_S3LCtx` através do valor `NSID_MD5X500` para o parâmetro `nsid` daquela função. Recorde–se que a estrutura das mensagens do protocolo Skeyx, tal como definidas no capítulo 4, contemplam a identificação do Servidor apenas com base no seu nome distinto X.500. Contudo, verifica–se agora, no contexto de uma execução implícita de Skeyx, a necessidade de identificá–lo também pela síntese MD5 do seu nome distinto. Esta forma alternativa de identificação mantém, praticamente inalterada, a estrutura das mensagens do Skeyx e, como a sua utilização se considera do foro interno do S3L (e não propriamente destinada ao programador comum), optou–se por conservá-la no anonimato e relativamente indocumentada.

Finalmente, falta ainda esclarecer a forma através da qual `S3Lopen_message` obtém o endereço IP da entidade originadora da mensagem. Esse endereço IP corresponde ao campo `Source IP Address` de uma mensagem S3L (recordar a figura 5.3). Num contexto invocador da função `S3Lopen_message` em que estão presentes *sockets* de Berkeley (como é o caso das funções `S3Lread`, `S3Lrecv`, etc.), o endereço IP de origem poderia ser obtido facilmente através da primitiva `getpeername`[33]. Porém, o uso da função `S3Lopen_message` não se restringe à desassemblagem de mensagens recebidas da rede (recorde–se o que foi dito na secção 5.3.3), o que aconselha a utilização de um mecanismo genérico de especificação do endereço IP da entidade originadora de uma mensagem S3L. No caso concreto, optou–se pela utilização de um campo `Source IP Address` da mensagem S3L. A presença deste campo tem ainda outro benefício adicional: a prevenção de ataques do tipo *ip–spoofing*[34].

Até agora, definiu–se como causa de uma ocorrência implícita do protocolo Skeyx a ausência de uma determinada entrada na *cache* `.dhcache` do PSE. Existem, porém, outros estados em que uma entrada se pode encontrar e cuja detecção pode também desencadear a execução do protocolo Skeyx com a entidade correspondente.

Na secção 4.5.4, o protocolo Skeyx foi caracterizado como "semi–transaccional" no que diz respeito à actualização das *caches* das entidades participantes. Concretamente, o Skeyx garante (tanto quanto o TCP pode garantir) que ambas as entidades atinjam um

[33] Ou ainda de outras formas em determinados casos específicos: por exemplo, através do parâmetro `struct sockaddr *addr` da primitiva `accept` para *sockets* conectados ou do parâmetro `struct sockaddr *from` da primitiva `recvfrom` para *sockets* conectados e não–conectados.

[34] Já em [Bel89] se faz referência a esta categoria de ataques, nos quais o endereço IP de origem de um pacote é modificado *en–route*. Mais recentemente, o *CERT Advisory* CA-96.21 (`ftp://info.cert.org/pub/cert_advisories/CA-96.21.tcp_syn_flooding`) descreve outro — o *TCP SYN Flooding* — que frequentemente ocorre em forma combinada com o *ip–spoofing*.

estado em que dispõem de toda a informação necessária à actualização das *caches*, mas não garante o sucesso simultâneo dessas actualizações, uma vez que a conexão TCP é quebrada imediatamente antes de a actualização das *caches* se processar. Consequentemente, e salvaguardando as situações de actualizações parciais[35], uma entrada da *cache* pode encontrar–se nos seguintes "estados totais":

- *inexistente*: nunca houve tentativas destinadas à sua criação ou, imediatamente antes da sua criação, o processo no seio do qual o Skeyx executava terminou;
- *(existente) actual*: o processo de criação (ou actualização, conforme o caso), foi levado, com sucesso, até ao fim;
- *(existente) obsoleto*: uma entrada *actual* foi marcada como *obsoleta*.

Uma entrada *actual* torna–se *obsoleta* quando se passa a designar a chave acordada `DHagreedkey` aí existente por `DHagreedkey.old`. Obviamente, na mesma entrada não podem coexistir uma chave actual e uma chave obsoleta, correspondendo essa situação, para todos os efeitos, a uma corrupção da entrada.

A *script* `s3lmkdhold`, fornecida com a distribuição do S3L (ver secção B.1 do apêndice B), marca como obsoletas todas as chaves acordadas de Diffie–Hellman guardadas na *cache* do invocador. Na prática, isto corresponde a tornar toda a *cache* obsoleta, por forma a forçar, de uma forma implícita, novas execuções do protocolo Skeyx que irão actualizar, progressivamente, a *cache*. A detecção da obsolescência de uma entrada é feita pela função `S3Lmake_S3LCtx` durante a definição de um contexto S3L. Como desse contexto faz parte a chave acordada de Diffie–Hellman[36], a ausência dessa chave (na prática, a ausência da entrada na *cache*) ou a sua obsolescência desencadeiam a execução implícita do protocolo Skeyx.

São várias as causas que podem ditar a obsolescência da totalidade da *cache*:

- modificação dos parâmetros públicos de Diffie–Hellman;
- modificação das chaves (pública e privada) de Diffie–Hellman do dono da *cache*; para "resolver" esta situação, e dispondo cada entrada na *cache* da chave pública de Diffie-Hellman da entidade correspondente, bastaria recombinar a chave privada do dono da *cache* com aquelas chaves públicas para obter as novas chaves acordadas. Todavia, se o dono da *cache* enviasse agora uma mensagem S3L a uma das entidades com entrada na *cache*, essa entidade não conseguiria abrir a mensagem, porque iria usar uma chave acordada desactualizada[37]. Assim, é preferível detectar, na origem, a

[35] Estas situações deverão ser, em princípio, raras, motivadas, por exemplo, por quebras de energia e outros imponderáveis. Nessas circunstâncias, deverá ser feita uma inspecção manual a cada entrada da *cache*. A "corrupção" de uma entrada é detectável pela ausência de uma parte dos ficheiros (o que significa que a criação foi parcial) ou pela presença de cópias suas de segurança (o que significa que o processo de actualização não foi levado até ao fim). O procedimento a seguir em ambos os casos deve ser a eliminação da entrada, na esperança de que uma futura criação (explícita ou implícita) seja bem sucedida.

[36] Recorde–se a definição da estrutura `S3LCtx` na secção 5.3.2.

[37] Na realidade, a função `S3Lopen_message` acautela, por segurança, esta situação: se a definição de um contexto S3L *via* `S3Lmake_S3LCtx` é bem sucedida mas a autenticação da mensagem falha, então `S3Lopen_message` despreza a mensagem recebida e termina executando Skeyx — desde que o parâmetro `skeyx_called` não venha devolvido de `S3Lmake_S3LCtx` a `S3LTRUE`, sinal de que um Skeyx teria ocorrido no seu interior —, garantindo assim que, na recepção das próximas mensagens, a entrada da *cache* já está actualizada.

presença de uma chave acordada obsoleta e executar, automaticamente, o protocolo Skeyx com a entidade remota, na próxima assemblagem de uma mensagem S3L a ela destinada;

- modificação do PIN (induzida, eventualmente, pelo seu compromisso) do dono da *cache*; recorde–se que todas as chaves acordadas de Diffie–Hellman presentes na *cache* estão cifradas com base numa chave DES derivada do PIN; na ausência de uma aplicação especificamente destinada a decifrar, com o PIN antigo, todas as chaves acordadas e a cifrá–las com o novo, uma forma indirecta de conseguir o mesmo efeito é marcar essas chaves como obsoletas e confiar no mecanismo de execuções implícitas do protocolo Skeyx a fim de assegurar a aplicação do novo PIN;
- modificação da identificação (nome distinto) do dono da *cache*; essa identificação corresponde ao sujeito dos certificados das chaves RSA; na prática, isto corresponde à mudança da identidade de uma entidade e deve ser, em princípio, um evento raro;
- revogação ou modificação das chaves (pública e privada) RSA preservadas no PSE; uma vez que estas chaves são essenciais à autenticação das mensagens do Skeyx, é natural que uma entidade opte por reconstruir a sua *cache* com base na utilização das novas versões destas chaves[38].

Atendendo ao que foi dito sobre os "estados totais" de uma entrada na *cache*, a figura 5.12 apresenta todas as combinações possíveis entre um par de entradas, relativas a duas entidades diferentes, A e B.

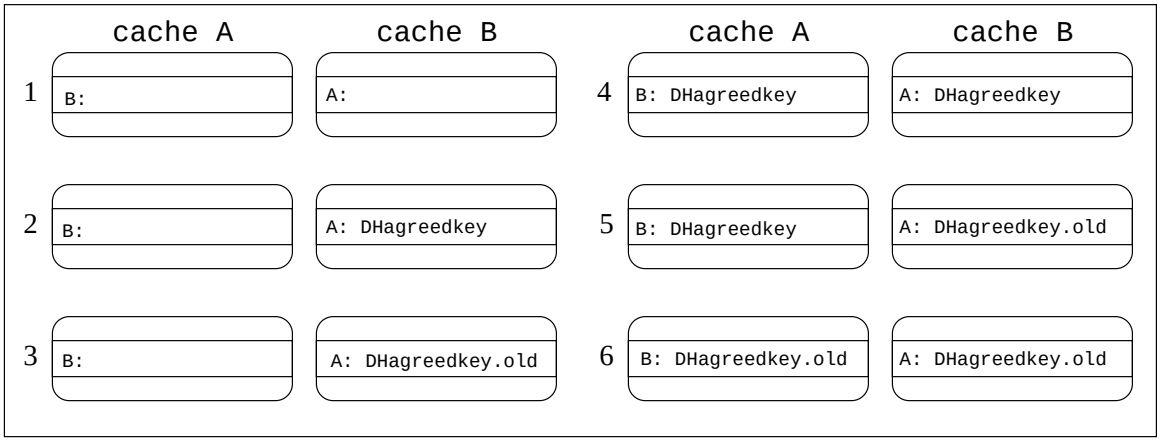

Figura 5.12: Possíveis estados de um par de entradas em duas caches distintas.

A combinação "estável", para a qual todas as outras tendem é, obviamente, a combinação 4, com ambas as *caches* actualizadas. O mecanismo de actualizações implícitas actua da seguinte forma, para as outras combinações:

1 $<A:,B:>$: nem A, nem B possuem entrada recíproca, nas suas *caches*; se um deles decidir enviar uma mensagem S3L ao outro, a função `S3Lmake_S3LCtx` detecta a ausência da entrada e invoca `Skeyx` automaticamente;

[38] A situação de compromisso da chave privada, usada no cálculo das assinaturas `C_sig` e `S_sig` — recordar a estrutura das mensagens do Skeyx em 4.2 — ilustra bem esta preocupação.

2 <*A:,B:DHagreedkey*>: apenas uma das entidades (neste caso B) possui uma entrada e actualizada; se A decide enviar uma mensagem S3L a B, então, em A, `S3Lmake_S3LCtx` executa `Skeyx` automaticamente; se B decide enviar uma mensagem S3L a A, então, em A, `S3Lmake_S3LCtx` também invoca `Skeyx` automaticamente; em ambos os casos a causa da execução de `Skeyx` por A é a ausência de entrada relativa a B;

3 <*A:,B:Dhagreedkey.old*>: A não possui entrada relativa a B, mas B possui entrada relativa a A, embora obsoleta; se A decide enviar uma mensagem S3L a B, então, em A, `S3Lmake_S3LCtx` invoca `Skeyx` automaticamente porque não se detectou a entrada de B; se B decide enviar uma mensagem S3L a A, então, em B, `S3Lmake_S3LCtx` também executa `Skeyx` automaticamente, porque B detecta a obsolescência da sua entrada;

5 <*A:DHagreedkey,B:DHagreedkey.old*>: A possui entrada actualizada relativa a B e B possui uma entrada obsoleta relativa a A; se A envia uma mensagem S3L a B, então, em B, `S3Lmake_S3LCtx` detecta a obsolescência da sua entrada e executa `Skeyx`, após o que continua o processamento da mensagem S3L recebida; se B pretende enviar uma mensagem S3L a A, então, em B, `S3Lmake_S3LCtx` detecta, à partida, a obsolescência da entrada de A e executa `Skeyx` para corrigir essa situação;

6 <*A:DHagreedkey.old,B:DHagreedkey.old*>: A e B possuem ambos entradas recíprocas obsoletas; o primeiro a enviar uma mensagem S3L ao outro provoca a actualização de ambas as entradas.

Resumindo, o S3L possibilita a adopção de duas políticas de actualização da *cache*: *por antecipação*, com base na utilização da aplicação `s3lkeyxclient`, e *por necessidade*, invocando a função `Skeyx` sempre que tal é julgado oportuno pela função de definição de contextos `S3Lmake_S3LCtx`, sendo este último mecanismo o mais discreto e adequado ao programador de aplicações que utiliza a API do S3L.

Capítulo 6

Comunicação Segura Multiponto via S3L

Conclui–se, neste capítulo, a apresentação da arquitectura do S3L, com a descrição das extensões necessárias à sua operação num contexto de comunicação multiponto. As extensões multiponto em questão suportam as mesmas qualidades de serviço oferecidas pelo S3L ponto–a–ponto (Privacidade, Autenticação[1], Integridade, Sequenciação – contra *ataques-de–repetição* – e Compressão) mas assumindo como protocolo base de comunicação a variante multiponto do IP, vulgo *IP Multicast*[2] [Dee89].

Basicamente, o *IP Multicast* assenta na utilização do protocolo *User Datagram Protocol* (UDP) [Pos80a] (protocolo *não–orientado–à–conexão*, que segue uma política de entrega de *melhor esforço* e não garante sequenciação), com o objectivo de fazer chegar um datagrama a um conjunto de sistemas de destino, constituídos num grupo identificado através de um endereço IP de classe D (gama [`224.0.0.0, 239.255.255.255`]).

A utilização do *IP Multicast* como protocolo de comunicação em grupo num ambiente sensível em termos de Segurança não pode ser feita sem a introdução de serviços adicionais. Por um lado, a junção a um determinado grupo *IP Multicast* não encontra obstáculos de natureza selectiva, ou seja, um sistema dispondo de interfaces de rede com capacidades multiponto pode aderir a qualquer grupo, sendo suficiente a execução de um conjunto mínimo de operações sobre *sockets* de Berkeley (ver, por exemplo, o cliente S3L multiponto da secção B.2.8 do apêndice B). Adicionalmente, a junção só é necessária caso se pretenda receber o tráfego específico desse grupo. A geração de tráfego dirigido ao grupo pode ser feita por entidades que não pertencem ao grupo, o que introduz oportunidades óbvias para ataques do tipo *negação–de–serviço*.

A figura 6.1 representa uma possível solução para o problema da permeabilidade dos grupos *IP Multicast*: a constituição de um subgrupo seguro, do grupo mais abrangente identificado por um determinado endereço IP de classe D. O tráfego do subgrupo seguro é sujeito a medidas especiais de protecção (em termos de Privacidade, Autenticação,

[1] Neste caso incluindo, para além da Não–Repudiação, o Controle de Acesso (ver "Controle de acesso ao grupo" na secção 6.2.1).

[2] Refira–se que, ao longo deste capítulo, o *IP Multicast* não será, em si mesmo, objecto de análise aprofundada, uma vez que se assume, sobretudo, como um instrumento de trabalho disponível para, sobre ele, concretizar os protocolos aqui descritos. Referências como [Dee89, Fen96, SM96, SRL96] constituem boas fontes de informação sobre o tema, sendo [Hur97] uma síntese bastante recente do estado da arte.

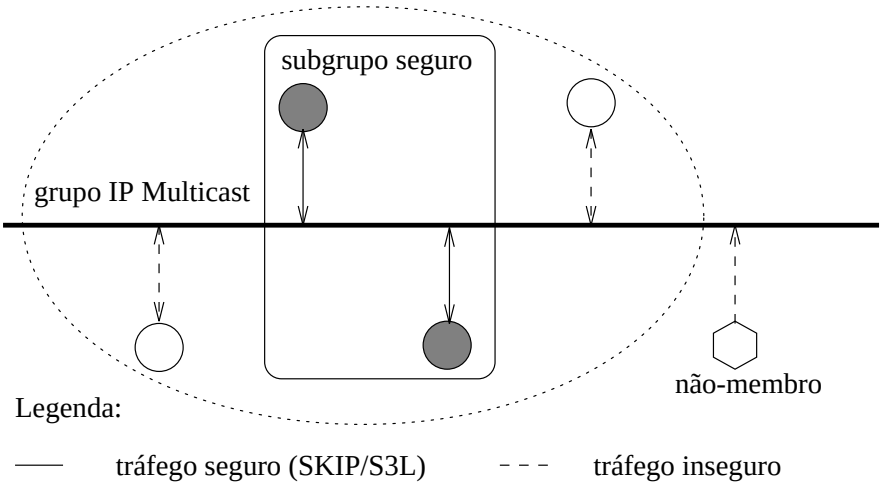

Figura 6.1: Subgrupos seguros e inseguros sobre *IP Multicast*.

etc.) que o "isolam" dos elementos do subgrupo complementar, bem como de tráfego eventualmente gerado por não–membros.

Neste capítulo, são apresentados dois conjuntos de protocolos que governam a criação, gestão e operação de subgrupos seguros de grupos *IP Multicast*. Na linha do capítulo anterior, introduzem–se, em primeiro lugar, as extensões multiponto do SKIP, preparando--se assim o terreno para a definição das extensões multiponto do próprio S3L. Mais uma vez, colocar–se–á ênfase na complementaridade entre as duas abordagens.

Em jeito de antecipação, podemos adiantar que o modelo de programação obtido pela introdução das extensões multiponto do S3L é muito semelhante ao da variante ponto--a–ponto: à inicialização de um "contexto multiponto", segue–se a utilização das "primitivas" `S3Lsendto` e `S3Lrecvfrom` que detectam, automaticamente, a variante multiponto ou ponto–a–ponto do S3L e actuam em conformidade.

6.1 Extensões Multiponto ao SKIP

Tradicionalmente, a distribuição de chaves num contexto de comunicação multiponto baseia–se na existência de um Centro de Distribuição de Chaves (KDC[3]) que fornece, a cada elemento do grupo, a *chave–do–grupo*. Com base nessa chave, cada elemento tem acesso de leitura e escrita ao tráfego do grupo.

Todavia, [AP95] identifica dois problemas fundamentais nesta abordagem:

1. a mudança da *chave–do–grupo* não é escalável relativamente ao número de elementos do grupo: para um grupo numeroso, o KDC poderá ter dificuldades em fornecer, num espaço de tempo útil, a cada elemento, a nova versão da *chave–do–grupo*. Adicionalmente, uma política de mudança de chaves baseada exclusivamente num tempo de vida, sem entrar em linha de conta com outros factores, pode não ser a mais indicada: por exemplo, conexões de alto débito implicam uma taxa de utilização da

[3]Do inglês *Key Distribution Center*.

chave mais elevada, a que corresponde uma obsolescência da mesma mais acelerada, dado que é maior a sua exposição;

2. a reutilização da mesma chave por todos os elementos do grupo exclui a possibilidade de utilizar certas cifras do tipo *stream* (como por exemplo o popular RC4 [Riv92a]), dado que, nessas circunstâncias, tais cifras são vulneráveis a ataques baseados na disponibilidade de pelo menos dois blocos cifrados com a mesma chave[4].

Estes problemas sugerem a necessidade de uma política de gestão das *chaves–do–grupo* que combine, de forma independente para cada elemento do grupo, factores como o tempo de vida e a taxa de utilização da(s) chave(s). Precisamente, o SKIP oferece uma solução [AMP95e] para este problema, através do mecanismo que a seguir se descreve.

Um candidato a membro do grupo `m` deve contactar o KDC (doravante designado por "dono do grupo" e cuja localização e identidade se supõem obtidas de forma segura) a fim de solicitar a junção ao grupo *IP Multicast* de endereço `m`. A troca de mensagens entre candidatos a membros (ou membros efectivos) e o dono do grupo faz–se de forma segura, com base na versão ponto–a–ponto do SKIP.

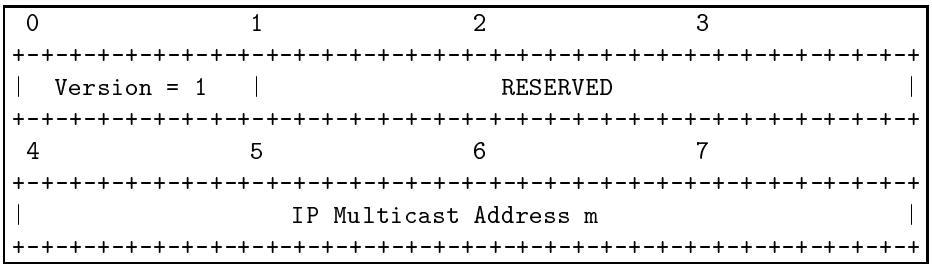

Figura 6.2: Estrutura de um pedido SKIP de junção ao grupo `m`.

A figura 6.2 reproduz o pedido de junção ao grupo `m`. Esse pedido é encapsulado no campo de dados de um pacote SKIP. O dono do grupo analisa o pedido de junção e, mediante a presença (ou ausência) do endereço IP do candidato numa Lista de Controle de Acesso (ACL[5]) para o grupo `m`, confirma (ou ignora) o pedido de junção.

A figura 6.3 representa a confirmação da junção ao grupo, a qual é encapsulada no campo de dados de um pacote SKIP. Da confirmação da junção ao grupo faz parte uma chave simétrica (tipicamente de 256 bits) designada de *Group Interchange Key* (`GIK`). À semelhança das chaves `Kij` do SKIP ponto–a–ponto, a chave `GIK` não será usada directamente na protecção do tráfego que circula no grupo. Alternativamente, `GIK` fornecerá os bits necessários à derivação de uma chave simétrica, usada como *chave–de–encriptação–de–chaves* simétricas `Kp`, eventualmente específicas para cada pacote multiponto. Adicionalmente, o campo `Expiry time` define, em segundos, o instante temporal em que se considera expirada a validade da `GIK`, sendo esse instante relativo a um outro predefinido que, tal como no SKIP ponto–a–ponto, corresponde ao instante UTC[6] `Jan 1, 1995 00:00:00`. Após esse instante, os elementos do grupo entram numa fase de rejunção, destinada a obter a nova

[4]Em [Sch96], págs. 197–198, encontra–se uma descrição elucidativa das permissas e funcionamento deste tipo de ataques a cifras do tipo *stream*.

[5]Do inglês *Access Control List*.

[6]Do inglês *Universal Time Coordinated*.

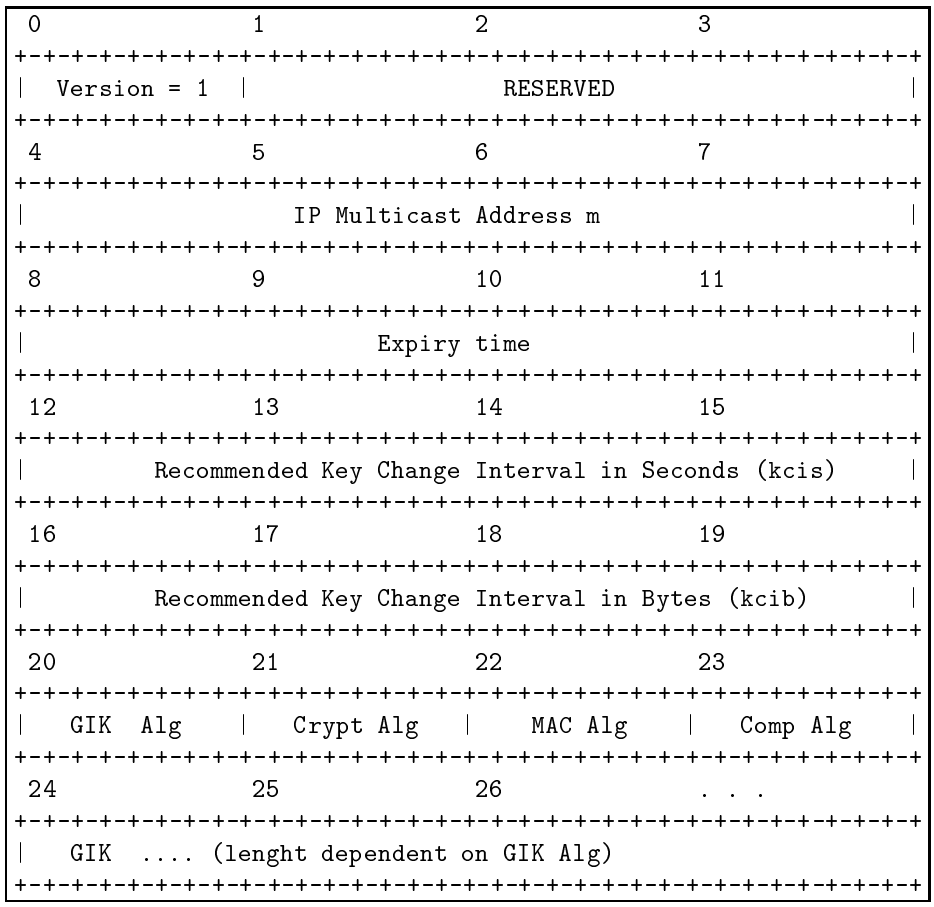

Figura 6.3: Estrutura de uma confirmação SKIP de junção ao grupo `m`.

`GIK` e as qualidades de serviço associadas. Entre essas qualidades de serviço, encontram–se os algoritmos de Encriptação de `Kp` (`GIK Alg`) e dos dados (`Crypt Alg`), de Autenticação (`MAC Alg`) e de Compressão (`Comp Alg`), bem como valores expressos em bytes e segundos (`kcis` e `kcib`, respectivamente) que determinam tempos de vida para as chaves `Kp`. Note–se que todos estes parâmetros são definidos, de uma forma centralizada, pelo dono do grupo sendo partilhados por todos os elementos após a junção.

Uma vez na posse da `GIK` e das qualidades de serviço associadas, um elemento está apto a receber e a gerar tráfego seguro dirigido ao grupo *IP Multicast* de endereço `m`. A figura 6.4 reproduz a estrutura de um cabeçalho SKIP quando os dados se destinam a serem transportados pelo *IP Multicast*. A estrutura apresentada é compatível com o cabeçalho da variante ponto–a–ponto (rever a figura 5.1), determinando o receptor, pela análise do endereço de destino, se deve aplicar procedimentos ponto–a–ponto ou multiponto no processamento do pacote recebido. Note–se que, na vertente multiponto, os campos `Source NSID` e `Dest NSID` são inicializados a zero, tal significando que origem e destino se identificam pelo endereço IP ponto–a–ponto e multiponto, respectivamente, pelo que se encontram ausentes os campos `Source Master Key-ID` e `Destination Master Key-ID`. Adicionalmente, os quatro campos correspondentes à definição de algoritmos de Encriptação, Autenticação e Compressão, estarão também inicializados a zero (indicação

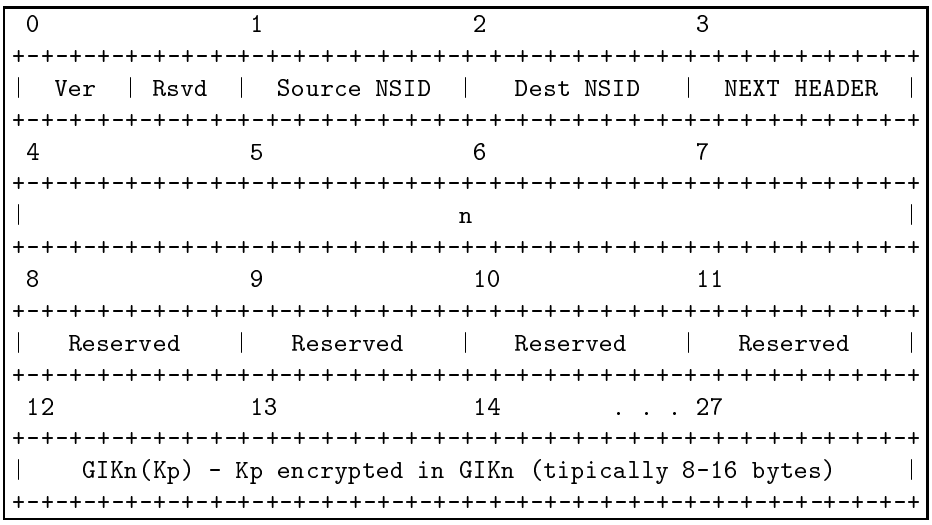

Figura 6.4: Estrutura de um cabeçalho SKIP multiponto.

dada por `Reserved`), uma vez que essas qualidades de serviço se supõem predefinidas pelo dono do grupo e partilhadas por todos os membros. Uma excepção à inicialização a zero por omissão ocorre quando (e, mais uma vez, à semelhança do que acontece na variante ponto–a–ponto do SKIP) esses campos albergam valores necessários a protocolos cujos cabeçalhos se seguem ao do SKIP. Nesse caso, o campo `NEXT HEADER` identifica tais protocolos[7].

O papel do campo `n` continua a ser semelhante ao desempenhado na versão ponto-a–ponto do SKIP ou seja, indica qual a versão[8] da chave `GIK` usada para cifrar `Kp`. Portanto, além de `GIK` ter um tempo de vida limitado, são usadas progressivamente novas versões `GIKn` que, à semelhança de `Kijn`, são dadas por [AMP95e]:

$$\texttt{GIKn}=\text{MD5}(\texttt{GIK} \mid \texttt{n} \mid \texttt{0x01}) \mid \text{MD5}(\texttt{GIKn} \mid \texttt{n} \mid \texttt{0x00})$$

Por sua vez, uma chave `Kp`, que é obtida aleatoriamente, é regenerada sempre que pelo menos um dos limites expressos por `kcis` ou `kcib` é ultrapassado, ou seja, sempre que o tempo de vida determinado por `kcis` expirou ou a quantidade de tráfego processado com origem no nó em questão excede `kcib`. Tal como na variante ponto–a–ponto do SKIP, `Kp` desdobra–se em `A_Kp` e em `E_Kp`, chaves simétricas a usar, respectivamente, para a Autenticação e a Encriptação dos dados, sendo geradas da seguinte forma:

$$\texttt{E\_Kp}=\text{MD5}(\texttt{Kp} \mid \texttt{Crypt Alg} \mid \texttt{0x02}) \mid \text{MD5}(\texttt{Kp} \mid \texttt{Crypt Alg} \mid \texttt{0x00})$$
$$\texttt{A\_Kp}=\text{MD5}(\texttt{Kp} \mid \texttt{MAC Alg} \mid \texttt{0x03}) \mid \text{MD5}(\texttt{Kp} \mid \texttt{MAC Alg} \mid \texttt{0x01})$$

6.2 Extensões Multiponto ao S3L

O S3L multiponto cumpre um dos objectivos fundamentais desta tese, ou seja, o fornecimento de grupos *IP Multicast* seguros. Para isso, recorde–se, foi necessário conceber uma

[7] Tipicamente o AH [Atk95b] e/ou o ESP [Atk95c].

[8] Em termos temporais, dado que `n` representa o número de horas decorridas desde o instante UTC `Jan 1, 1995 00:00:00` até ao instante em que é `n` é definido, no originador.

infra–estrutura de gestão de chaves, baseada no protocolo Skeyx, e sobre a qual foi possível obter comunicação segura ponto–a–ponto, de uma forma semelhante à proporcionada pelo SKIP, mas ao nível da Camada de Aplicação.

As extensões multiponto do S3L constituem assim o último patamar da arquitectura do S3L, apresentada na figura 4.1. Tendo por base as facilidades oferecidas pelo S3L ponto-a–ponto, o S3L multiponto herda, naturalmente, a influência que o SKIP teve na concepção daquele protocolo. Todavia, não só as semelhanças entre o S3L multiponto e o SKIP multiponto serão patentes ao longo deste capítulo como, de igual modo, serão evidenciadas abordagens diferentes em aspectos como: a estrutura das Listas de Controle de Acesso, o mecanismo de detecção de *ataques–de–repetição*, a definição de um tempo de vida do grupo independentemente do tempo de vida da chave `GIK`, o mecanismo de rejunção em face da obsolescência das chaves `GIK` e até a preocupação em evitar a sobreposição do tráfego de dois grupos S3L no mesmo endereço *IP Multicast*.

6.2.1 Processo de Junção ao Grupo

Tal como o SKIP, o S3L pressupõe a existência de uma entidade que desempenha o papel de "dono do grupo", controlando a admissão de novos elementos e definindo as políticas de gestão de chaves e qualidades de serviço que caracterizam o grupo.

O dono do grupo é, na realidade, um demónio, baseado na aplicação `s3lmgowner` (acrónimo de *S3L Multicast Group Owner*), à qual se indica, através da linha de comando, o endereço *IP Multicast* do grupo cujo tráfego se pretende proteger, bem como as diversas qualidades de serviço que definem essa protecção[9].

Função do Reencaminhador

A junção ao grupo socorre–se do S3L ponto–a–ponto a fim de proteger o tráfego gerado nesse processo. As mensagens trocadas durante o processo de junção são, portanto, de natureza ponto–a–ponto, tendo, à partida, como únicos intervenientes o candidato a membro e o dono do grupo. Contudo, essas mensagens não se resumem, necessariamente a mensagens S3L. Convém lembrar que é possível que uma entidade[10] X, que controla um grupo, e uma entidade Y, que se quer juntar ao grupo, nunca tenham realizado o protocolo Skeyx, pelo que a sua execução implícita irá ocorrer, após o que o processo de junção prossegue baseado apenas na troca de mensagens S3L ponto–a–ponto.

A figura 6.5 ilustra bem esta situação, revelando também o novo papel que o demónio Reencaminhador (`s3lforwarder`) adquire no contexto do S3L multiponto. Aquele demónio, recorde–se (rever a secção 4.4), oferece um mecanismo de redireccionamento de pedidos Skeyx, dispensando, nos clientes, o conhecimento antecipado do porto onde é disponibilizado o serviço `s3lkeyxserver` de uma determinada entidade. No contexto do S3L multiponto, cada entidade passa a executar, potencialmente, mais um servidor: o demónio `s3lmgowner`, através do qual administra um determinado grupo *IP Multicast*. Os pedidos de junção serão recebidos, originalmente, por `s3lforwarder` no porto 7571 e redireccionados, através de um *socket* de domínio Unix, localizado em `/tmp`, para o demónio `s3lmgowner` apropriado. Mais uma vez, é garantida a unicidade da designação dos *sockets*

[9] Ver a secção B.1.4 do apêndice B para uma descrição completa das opções do comando `s3lmgowner`.
[10] Ou melhor, uma das suas possíveis instâncias.

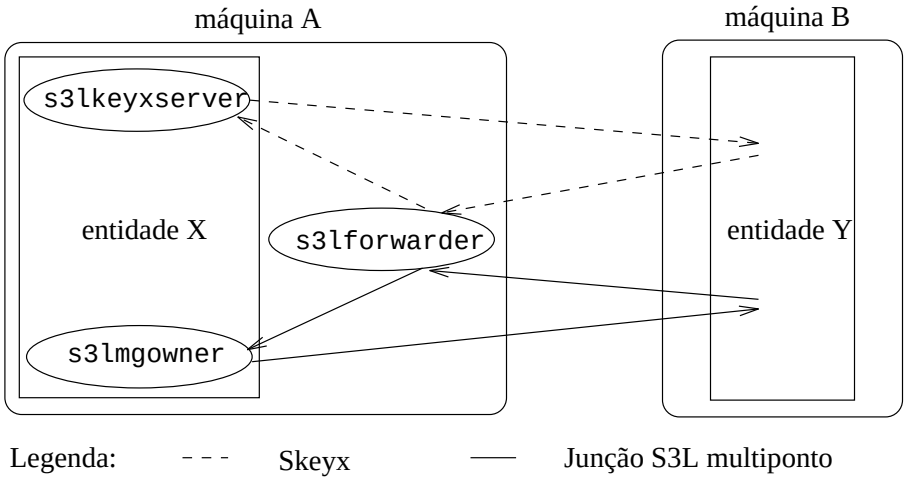

Figura 6.5: Dupla função do demónio `s3lforwarder`.

em `/tmp`, graças à utilização da síntese MD5 do nome distinto da entidade que controla o grupo, desta feita combinada com o endereço do grupo *IP Multicast* em questão. Por exemplo,

```
/tmp# ls -la
...
srwxr-xr-x 1 ... E1D2FE11A5D6FDFB0E662EF198A185A1=
srwxr-xr-x 1 ... E1D2FE11A5D6FDFB0E662EF198A185A1.235.0.0.0=
...
```

revela, em `/tmp`, a existência de dois *sockets* de domínio Unix, através dos quais os serviços `s3lkeyxserver` e `s3lmgowner` (este relativo ao grupo `235.0.0.0`) da entidade "`C=PT, CN=ruf, D=user`"[11] são contactados, durante o redireccionamento efectuado por `s3lforwarder`.

Pedidos de Junção

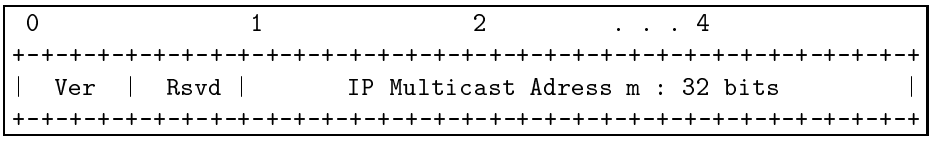

Figura 6.6: Estrutura de um pedido S3L de junção ao grupo `m`.

A figura 6.6 representa a estrutura de um pedido de junção S3L multiponto ao grupo *IP Multicast* de endereço `m`. As semelhanças com o pedido equivalente no contexto do SKIP (rever a figura 6.2) são evidentes. Um pedido de junção é encapsulado no campo de dados de uma mensagem S3L ponto–a–ponto que o candidato a membro envia ao dono

[11] Para a qual a síntese MD5 deste nome distinto é, em formato hexadecimal, precisamente `E1D2FE11A5D6FDFB0E662EF198A185A1`.

do grupo. Para esta mensagem, não é assegurada Privacidade dos dados, mas apenas Autenticação e Integridade. Tal justifica–se pela necessidade de o demónio Reencaminhador (`s3lforwarder`) do sistema de destino determinar o endereço `m`, pela análise do campo de dados da mensagem S3L. Na posse desse endereço e da identificação do destino final (que retira do campo `Dest ID` do cabeçalho da mensagem[12]), o redireccionamento da conexão pode ser efectivado.

Note–se que um pedido de junção ao grupo `m'`, dirigido a uma entidade que não gere esse grupo, não chega a ser redireccionado, dado que o necessário *socket* de domínio Unix não será detectado em `/tmp`. Todavia, não é de excluir a possibilidade de serem directamente "injectados" nesses *sockets* (por outras vias — ilícitas, obviamente — que não o processo `s3lforwarder`) pedidos de junção fictícios, dirigidos inclusivamente a um grupo diferente daquele que o respectivo `s3lmgowner` controla. Neste último caso, o problema é ainda resolvido através da verificação, em `s3lmgowner`, de que o endereço `m` que consta de um pedido de junção coincide, efectivamente, com o endereço do grupo gerido[13].

Controle de Acesso ao Grupo

Face a um pedido de junção legítimo, o dono do grupo deve verificar se a identidade do candidato a membro consta da Lista de Controle de Acesso (ACL) para o grupo em questão (`m`). A fim de albergar as ACLs dos vários grupos que uma entidade pode gerir, bem como informação acerca da evolução da composição dos grupos, foi concebida uma estrutura simples de directorias e ficheiros, subjacentes à directoria `.acls`, por sua vez subdirectoria de `.pse`[14]. Por exemplo,

```
~/.pse# ls -laR .acls
total 4
drwxr-xr-x   5 ruf      users        1024 Jun  7 19:39 ./
drwx------   4 ruf      users        1024 Jun  7 19:13 ../
drwx------   2 ruf      users        1024 Jun  1 16:17 235.0.0.0/
drwx------   2 ruf      users        1024 May 29 03:29 235.1.2.3/

.acls/235.0.0.0:
total 17
drwx------   2 ruf      users        1024 Jun  1 16:17 ./
drwxr-xr-x   5 ruf      users        1024 Jun  7 19:39 ../
-rw-r--r--   1 ruf      users          41 May 29 03:33 acl
-rw-rw-rw-   1 ruf      users       12643 Jun  1 16:17 members

.acls/235.1.2.3:
total 35
drwx------   2 ruf      users        1024 May 29 03:29 ./
drwxr-xr-x   5 ruf      users        1024 Jun  7 19:39 ../
-rw-r--r--   1 ruf      users          29 May 29 03:29 acl
-rw-r--r--   1 ruf      users       30946 May 29 03:29 members
```

[12] Rever a estrutura de uma mensagem S3L ponto–a–ponto, dada pela figura 5.3.

[13] Um problema semelhante pode ocorrer num servidor de pedidos Skeyx (`s3lkeyxserver`). Analogamente, o destino de um pedido Skeyx que um servidor `s3lkeyxserver` recebe é comparado com a identidade desse receptor e ignorado em caso de divergência (*Autenticação do Destino*).

[14] Recorde–se que é o conteúdo desta directoria que suporta o *Personal Security Environment* (PSE) de uma entidade, tal como definido pelo SecuDE [Sch95a].

revela a estrutura de controle de acesso que governa, para a entidade em causa no exemplo, a admissão de novos elementos aos grupos `235.0.0.0` e `235.1.2.3`. Assim, tal entidade (ou melhor, a sua instância particular em causa), pode executar dois demónios `s3lmgowner`, destinados à gestão dos grupos `235.0.0.0` e `235.1.2.3`, respectivamente. Cada ficheiro designado de `acl`, preserva a Lista de Controle de Acesso do respectivo grupo. Por exemplo,

```
~/.pse/.acls/235.0.0.0# cat acl
C=PT, CN=ruf, D=user
C=PT, CN=ze, D=user
```

revela que, para o grupo `235.0.0.0`, sob a gestão do `s3lmgowner` da entidade em causa, só são admissíveis junções dos candidatos cujo nome distinto é "`C=PT, CN=ruf, D=user`" ou "`C=PT, CN=ze, D=user`". É também possível saber o historial das junções ao grupo, bastanto para isso consultar o seu ficheiro `members`:

```
~/.pse/.acls/235.0.0.0# cat members
C=PT, CN=ruf, D=user : 193.136.8.7 : Sun Jun  1 16:17:11 1997
C=PT, CN=ruf, D=user : 193.136.8.7 : Sun Jun  1 16:17:43 1997
C=PT, CN=ze, D=user : 193.136.8.7 : Sun Jun  1 16:17:05 1997
C=PT, CN=ze, D=user : 193.136.8.7 : Sun Jun  1 16:17:38 1997
```

Em `members` conserva–se a identificação dos elementos, o endereço IP de origem do pedido de junção e o instante temporal em que a admissão teve lugar[15].

Na secção B.1.4 do apêndice B são descritas as aplicações `s3lmgaclsmaint` e `s3lmglist` que se destinam, respectivamente, a gerir as ACLs de uma entidade, bem como a consultar o historial dos grupos. O recurso a aplicações exclusivamente dedicadas a estas tarefas[16] possibilita, graças à utilização de um mecanismo de trincos, a execução de demónios `s3lmgowner` em simultâneo com acessos aos ficheiros `acl` e `members`, sem que tal coloque em causa a consistência desses ficheiros e a coerência do estado interno dos demónios `s3lmgowner`.

Confirmações de Junção

Verificada a presença do candidato a membro na Lista de Controle de Acesso do grupo, o dono do grupo deverá terminar o processo de junção, devolvendo ao candidato uma mensagem de confirmação, através da qual lhe comunica quais as qualidades de serviço que caracterizam o tráfego do grupo.

A figura 6.7 apresenta a estrutura de uma confirmação S3L de junção a um grupo. Os campos `kcis`, `kcib`, `GIK Alg`, `Crypt Alg`, `MAC Alg` e `CompAlg` desempenham a mesma função que os campos análogos da confirmação SKIP (rever a figura 6.3). Os restantes campos suportam uma política de gestão de *chaves–do–grupo* diferente da adoptada pelo SKIP. A explicação para o significado desse campos que a seguir se apresenta é devidamente complementada na secção 6.2.3, onde todos os elementos dessa política serão devidamente enquadrados e a opção pela sua utilização devidamente justificada.

O campo `Group Remaining Lifetime` (`GRL`) transporta o tempo de vida (em segundos) que restava ao grupo no instante em que o seu dono assemblou a confirmação de

[15] Não é registado o momento do abandono do grupo porque, para grupos *IP Multicast*, não é gerado, por parte de um elemento em fase de abandono, nenhuma mensagem indicativa desse evento.

[16] Em vez da utilização de comandos *ad–hoc*, como o `cat` dos exemplos apresentados.

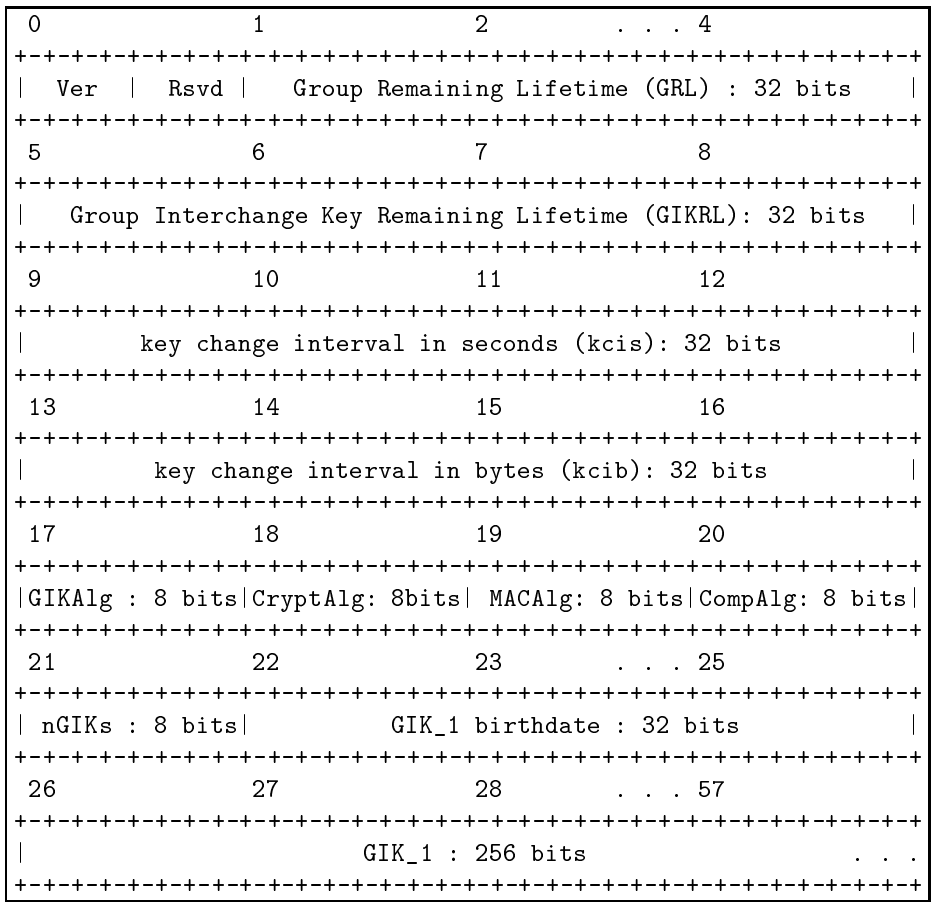

Figura 6.7: Estrutura de uma confirmação S3L de junção ao grupo `m`.

junção. Esgotado esse tempo de vida, a actividade do grupo considera–se encerrada: o dono do grupo termina (não sendo possíveis, portanto, mais junções) e os restantes elementos também cessarão, progressivamente, a sua actividade relacionada com o grupo, à medida que detectam o seu término. Um valor zero caracteriza o grupo como permanente, enquanto um valor positivo indica que o grupo é transiente. Note–se que a definição de um tempo de vida para o grupo por parte do seu dono não é contemplada no SKIP.

O campo `GIK Remaining Lifetime` (`GIKRL`) indica, também em segundos[17], o tempo de vida que restava à chave `GIK` mais recente[18] (`GIK_j`), aquando da assemblagem da mensagem de confirmação. Um valor zero indica que a chave é *permanente*[19], enquanto um valor positivo indica que a chave é *efémera*, ou seja, esgotado o tempo de vida `GIKRL`, e caso o tempo que resta da vida do grupo o permita[20], é efectuada uma rejunção ao grupo.

Alternativamente à indicação de tempos de vida, seria possível fornecer instantes de

[17] Quer `GRL`, quer `GIKRL`, têm granularidade da ordem do segundo o que pressupõe, portanto, que a duração mínima de um grupo ou de uma chave `GIK` seja, precisamente, de um segundo. A utilidade de tempos de vida inferiores é discutível...

[18] Do conjunto das chaves fornecidas nesta confirmação.

[19] Recordem–se, todavia, as contra–indicações relativamente à utilização de chaves de grupo permanentes, apresentadas na secção 6.1.

[20] Isto é, se ainda não decorreram `GRL` segundos no membro.

término[21]. Essa é a abordagem do SKIP relativamente à (única) chave `GIK` que envia na confirmação. Todavia, mesmo assumindo que, em todo o grupo, o tempo medido é convertido para o respectivo tempo UTC, a adopção de tempos de término requer a sincronização dos relógios de todos os elementos do grupo. O S3L contorna esse problema, pela utilização de "tempos de vida restantes" em vez de instantes de término. Naturalmente, há que levar em conta a adição dos atrasos de propagação das mensagens de confirmação a esses "tempos de vida restantes". Como se verá na secção 6.2.3, o mecanismo adoptado pelo S3L, contempla esse problema, resolvendo ainda outros que permaneceriam mesmo com a utilização de relógios sincronizados.

A figura 6.7 revela ainda que uma confirmação S3L de junção pode incluir mais do que uma chave `GIK`. O campo `nGIKs` define o número dessas chaves, seguindo–se uma sequência de pares da forma (`GIK_i birthdate`, `GIK_i`) — com `i=1,2,...,j` —, em que `GIK_i birthdate` é o instante temporal, em segundos, em que a chave `GIK_i` foi gerada, no dono do grupo. A chave mais recente desta sequência é a última (`GIK_j`) e será essa a chave a usar na transmissão de mensagens S3L multiponto para o grupo, desde que o tempo decorrido desde a recepção dessa chave até à sua utilização não seja superior ao tempo definido pelo campo `GIKRL`[22]. As outras chaves da sequência constituem chaves `GIK` obsoletas, mas que poderão ser úteis na leitura de mensagens S3L multiponto que foram geradas com base nessas chaves e que, devido aos atrasos de propagação, serão recebidas já durante o período de vigência de uma nova `GIK` (a `GIK_j`). Em 6.2.3, este aspecto será retomado.

Finalmente, ao contrário do pedido de junção, a sua confirmação necessita de Privacidade, dado que transporta consigo indicações temporais que governam o comportamento do novo elemento do grupo e, sobretudo, a sequência de chaves `GIK` que devem, obviamente, permanecer secretas. Consequentemente, a confirmação de junção viaja, devidamente cifrada, no campo de dados de uma mensagem S3L ponto–a–ponto.

6.2.2 Mensagens S3L Multiponto

Concluída a fase de junção ao grupo, um membro encontra–se apto a enviar e receber mensagens S3L multiponto.

O cabeçalho dessas mensagens (ver a figura 6.8) encontra–se consideravelmente simplificado face ao cabeçalho das mensagens SKIP multiponto (ver a figura 6.4) e mesmo das mensagens S3L ponto–a–ponto (rever a figura 5.3).

A identificação das entidades comunicantes limita–se, nas variantes multiponto do SKIP e do S3L, à utilização de endereços IP (ponto–a–ponto para a origem e multiponto para o destino). A variante multiponto do SKIP traduz esse facto, recorde–se (rever a secção 6.1), pela inicialização a zero dos campos `Source NSID` e `Dest NSID`, justificando assim a ausência dos campos `Source Master Key-ID` e `Destination Master Key-ID`. A presença dos campos `Source NSID` e `Dest NSID` na estrutura de uma mensagem SKIP multiponto destina–se apenas a fornecer compatibilidade estrutural com os cabeçalhos do SKIP ponto–a–ponto. Já na variante multiponto do S3L, foram colocadas de lado preocupações de compatibilidade estrutural com o cabeçalho ponto–a–ponto, uma vez que

[21] Do inglês *expiry times*.

[22] Obviamente, caso `GIKRL` seja zero, esperamos encontrar nesta sequência uma única chave que é permanente.

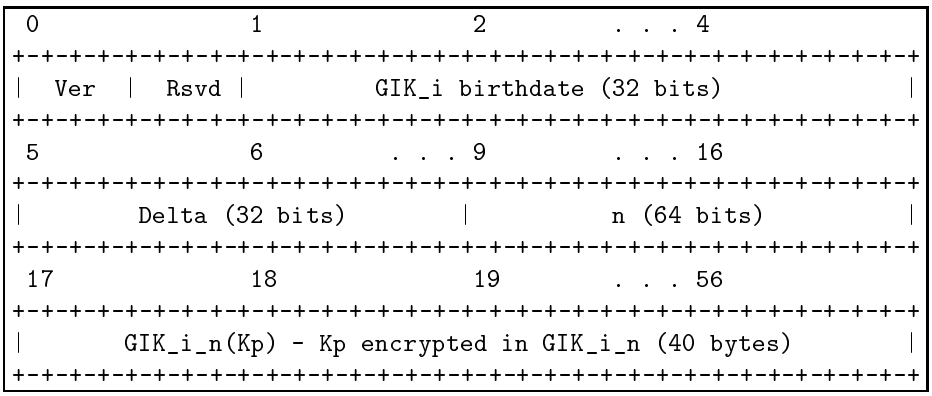

Figura 6.8: Estrutura do cabeçalho de uma mensagem S3L multiponto.

o processamento das mensagens S3L de ambas as variantes foi claramente demarcado, em termos de implementação. Acresce ainda que os campos `Dest ID` e `Source ID` que o S3L ponto–a–ponto utiliza como identificadores (sínteses MD5 de nomes distintos X.500) das entidades comunicantes não são agora necessários, dado que a identificação se faz com base em endereços IP.

O campo `Source IP Address` das mensagens S3L ponto–a–ponto também se encontra ausente duma mensagem multiponto uma vez que o seu papel só faz sentido no contexto da variante ponto–a–ponto (recorde–se a secção 5.4 do capítulo anterior).

Os algoritmos de Encriptação (`GIK Alg`[23] e `Crypt Alg`), Autenticação (`MAC Alg`) e Compressão (`Comp Alg`) para o tráfego intra–grupo são definidos pelo dono do grupo e comunicados aos novos membros na fase de junção. Esses algoritmos definem qualidades de serviço comuns a todo o grupo, pelo que não há necessidade de se fazerem comunicar nas mensagens S3L multiponto. Mais uma vez, por razões de estrita compatibilidade estrutural, o SKIP multiponto reserva espaço no cabeçalho para esses campos, sem deles fazer uso.

O campo `Delta`, em conjunção com o campo `n`, cumpre um papel idêntico ao desempenhado na versão ponto–a–ponto, ou seja, a detecção de *ataques–de–repetição* com base na especificação de um tempo de vida limitado `Delta` para a mensagem, o qual constitui um desvio máximo admissível entre o tempo instantâneo na recepção e na produção da mensagem (este último dado por `n`). Porém, como veremos na secção 6.2.4, a utilidade de `Delta` é limitada num contexto de comunicação multiponto.

O campo `n` continua a definir uma versão temporal para a *chave–de–encriptação–-de–chaves* (`GIK_i`) que protege `Kp`. O instante temporal em que a mensagem S3L multiponto é assemblada é convertido a uma escala UTC com origem no instante `Jan 1, 1995 00:00:00`, obtendo–se assim o valor de `n`.

Note–se que, ao contrário do SKIP multiponto, em que se admite apenas uma `GIK` em circulação (a `GIK` mais recente gerada pelo dono do grupo), o S3L multiponto também processa mensagens geradas com base em `GIK`s que, entretanto, ficaram obsoletas. A utilização do campo `GIK_i birthdate` permite identificar diferentes gerações de `GIK`s. Este campo transporta o instante temporal em que o dono do grupo gerou a `GIK` que presidiu à protecção da mensagem. Concretamente, terá sido utilizada a versão `n` da

[23] De certa forma equivalente ao `Kij Alg` do S3L ponto–a–ponto.

`GIK` nascida a `GIK_i birthdate`. Assim, a notação `GIK_i_n(Kp)` refere–se à utilização da versão `n` da `GIK` gerada no instante `GIK_i birthdate`, como chave simétrica de encriptação da chave `Kp`[24].

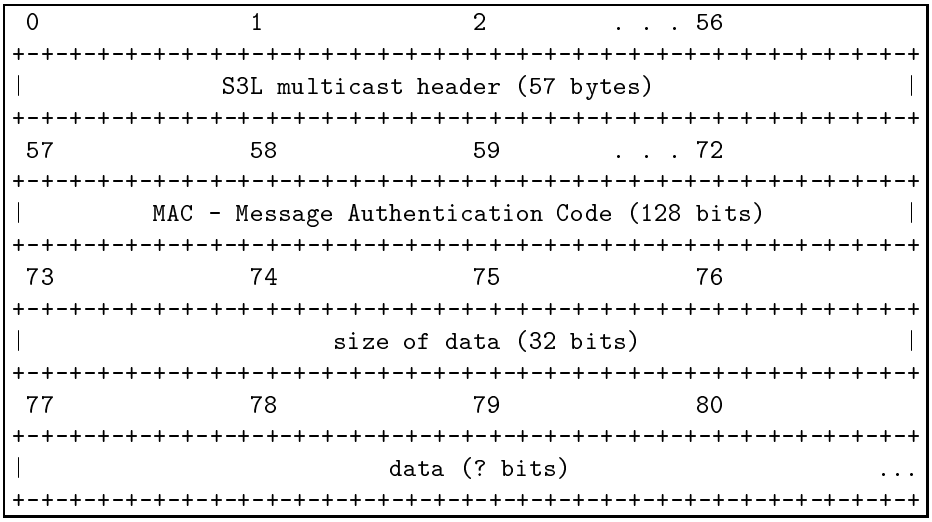

Figura 6.9: Estrutura de uma mensagem S3L multiponto.

A figura 6.9 apresenta a estrutura completa de uma mensagem S3L multiponto, contemplando o cabeçalho da figura 6.8, bem como os campos `MAC`, `size of data` e `data`, análogos aos campos homólogos da mensagem S3L ponto–a–ponto.

6.2.3 Gestão de Chaves de Grupo no S3L Multiponto

Nesta secção, descreve–se com mais detalhe a política de gestão de chaves seguida pelo S3L multiponto, procurando justificá–la como alternativa a algumas insuficiências detectadas no mecanismo originalmente proposto pelo SKIP.

No SKIP multiponto, uma mensagem de confirmação de junção transporta consigo um campo designado de `Expiry time`. Este campo consiste num instante temporal, normalizado a uma escala UTC com origem em `Jan 1 1995 00:00:00`, calculado pelo dono do grupo como o instante em que a validade da `GIK` actual cessará e uma nova `GIK` será gerada. `Expiry time` é também o instante em que uma "vaga de rejunções" se iniciará, a partir dos actuais membros.

Este modelo é irrealista, nomeadamente pelas seguintes razões:

1. pressupõe que a totalidade do grupo (dono incluído) tem os relógios sincronizados, o que implica a execução de mais um protocolo suplementar para a sincronização de relógios entre todos os elementos do grupo (*e.g.*, *Network Time Protocol* (NTP) [Mil92]). Esse protocolo deverá ser seguro[25], sob pena de a gestão de chaves `GIK` ser

[24] Por sua vez gerada aleatoriamente com o propósito de proteger a mensagem S3L multiponto em questão.

[25] O *Simple Network Time Protocol* (SNTP) [Mil96] que constitui, basicamente, uma versão simplificada do NTP, suporta uma série de extensões que permitirão a Autenticação das suas transacções em modo multiponto, prevendo–se que esteja para breve o aparecimento de uma especificação dessas extensões.

comprometida[26]. Todavia, é precisamente o problema da comunicação segura em grupo que estamos a tentar resolver. Adicionalmente, num contexto *IP Multicast*, em que os elementos do grupo estão potencialmente dispersos de tal forma que os atrasos de propagação são muito variáveis, torna–se difícil a sincronização de relógios com margens de desvio mínimas[27];

2. mesmo admitindo a sincronização dos relógios do grupo, então, numa situação em que a taxa de mudança da chave `GIK` é elevada (justificada, por exemplo, por uma taxa de utilização também elevada), a carga da máquina onde executa o dono do grupo aumenta nos períodos de rejunção o que, em conjunção com atrasos de propagação variáveis, conduz, facilmente, a situações em que, após a rejunção, a `GIK` obtida é já obsoleta (atente–se, por exemplo, na figura 6.10).

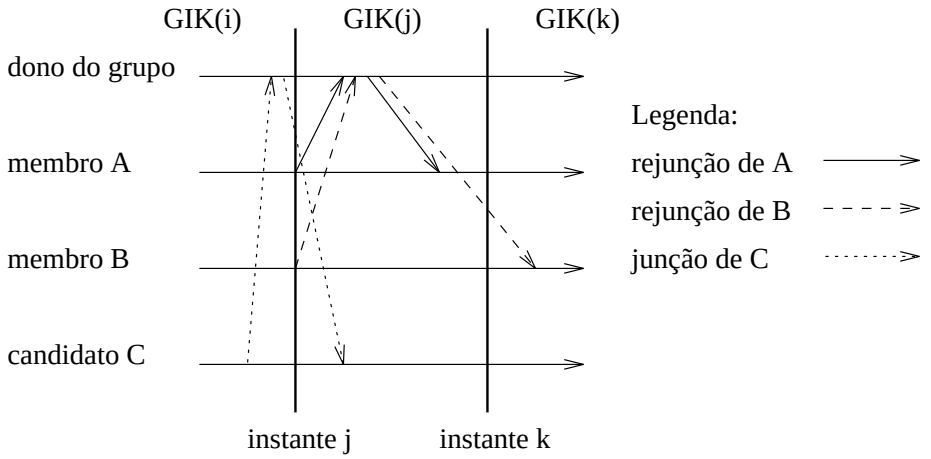

Figura 6.10: Obsolescência da `GIK` após a junção ao grupo.

Esta situação pode também acontecer durante a primeira junção ao grupo (uma vez que esta ocorre assincronamente) e se não for detectada, resulta na produção de mensagens baseadas numa `GIK` obsoleta e na impossibilidade de ler mensagens geradas com base na nova. Obviamente, sendo conhecido o tempo de vida da `GIK` (tempo comum para todas as `GIK`s), seria possível, a um membro do grupo, lançar a rejunção relativa à próxima `GIK` se, à "data de nascimento" desta, a obtenção da anterior não se tivesse ainda concluído. Tal pressupõe um mecanismo assíncrono de actualização de `GIK`s o que, por si só, também é problemático (ver ponto seguinte). Note–se ainda que, enquanto a nova `GIK` não é obtida, o membro não pode gerar tráfego, nem deve processar tráfego;

3. tomado isoladamente, o processo de rejunção aqui em causa é síncrono, uma vez que ocorre periodicamente, em instantes bem definidos. Contudo, no contexto de

[26] Por exemplo: se o relógio de um membro for antecipado de forma a fazer a rejunção um pouco antes da geração da nova `GIK`, então a velha `GIK` ficará na sua posse durante um intervalo de tempo em que já é a nova `GIK` que está em vigor.

[27] Embora o NTP assegure, teoricamente, sincronizações da ordem dos nanosegundos, os resultados conseguidos, na maioria dos casos práticos, rondam o intervalo [1, 50] milisegundos. São também da ordem dos milisegundos as sincronizações que, teoricamente, o SNTP se propõe assegurar.

aplicações programadas com base em funcionalidades S3L multiponto, a adopção deste mecanismo de rejunção introduz algumas dificuldades adicionais, pelo carácter assíncrono que a rejunção então adquire nesse contexto, já que uma aplicação deverá ser programada de forma a que a interrupção do processamento de mensagens S3L multiponto e a sua retoma (já com uma nova `GIK`) não coloque em causa o processamento previamente efectuado dessas mensagens. Adicionalmente, a aplicação não pode fazer uso de alguns mecanismos de interrupção por *software*, em número limitado (como por exemplo o sinal `SIGALRM`) uma vez que estes, provavelmente, já se encontram reservados para o despoletamento assíncrono do processo de rejunção.

Alternativamente ao modelo do SKIP, o S3L propõe um mecanismo independente da sincronização dos relógios, tolerante à operação com `GIK`s obsoletas (mas de forma parametrizável, por forma a controlar reutilizações abusivas) e em que as rejunções ocorrem *por necessidade* (em vez de *por antecipação*).

Neste novo modelo, o dono do grupo mantém uma lista circular, de dimensão parametrizável, contendo pelo menos as últimas `S3LM_MIN_GIKS` chaves `GIK` geradas, ordenadas por idade. Essa lista é gerida segundo uma filosofia *Last In First Out* (LIFO), por forma a que chaves mais recentes se sobreponham, progressivamente, às chaves mais antigas. Num dado instante, o conteúdo desta lista pode ser entendido como o conjunto das `GIK`s autorizadas a "circular" pelo dono do grupo, recomendando–se a utilização da `GIK` da lista o mais recente possível (o que os membros do grupo tentam garantir através do processo de rejunção).

Quando o dono do grupo recebe um pedido de (re)junção e este é aceite, devolve, na mensagem de confirmação, a lista de `GIK`s. Associada a cada `GIK`, encontra–se também o instante temporal, medido em segundos, em que o dono do grupo efectuou a sua geração. Os campos `nGIKs`, `GIK_1 birthdate`, `GIK_1`, ..., `GIK_j birthdate`, `GIK_j` materializam o transporte da lista de `GIK`s numa mensagem S3L de confirmação de junção (rever a figura 6.7). No receptor da mensagem, a lista de `GIK`s antiga é sempre sobreposta pela nova lista.

Os membros obrigam–se a transmitir mensagens S3L multiponto baseadas na utilização da chave `GIK` mais recente da sua lista. Assim, antes de assemblar uma mensagem S3L multiponto, um membro deve averiguar se a `GIK` mais recente da sua lista (`GIK_j`) já expirou, confrontando uma "data de nascimento local", gerada aquando da recepção da lista na última confirmação de junção, com o prazo de validade estipulado pelo campo `GIKRL` dessa confirmação. Se `GIK_j` não expirou, a mensagem S3L é gerada e o seu campo `GIK_i birthdate` preenchido com a "data de nascimento remota" `GIK_j birthdate`, medida no dono do grupo aquando da geração de `GIK_j`. O campo `GIK_i birthdate` de uma mensagem S3L multiponto identifica assim, perante qualquer elemento do grupo, a `GIK` de base usada na construção dessa mensagem, definindo o campo `n` uma sua versão temporal usada especificamente pelo originador da mensagem. Por outro lado, se `GIK_j` expirou, tem lugar o processo de rejunção, só após o qual a assemblagem da mensagem S3L multiponto prossegue, desta feita, com uma `GIK_j` mais recente. Note–se que esta `GIK_j` não é, necessariamente, a última gerada pelo dono do grupo, porque, durante o trânsito da confirmação de junção, a `GIK` pode já ter sido regenerada no dono do grupo. Todavia, espera–se que os destinatários de mensagens S3L multiponto sintetizadas com base na `GIK_j` recebida ainda contenham essa `GIK_j` na sua lista ou então que consigam obtê–la do dono do grupo através de um processo de rejunção (que só tem lugar se essa

chave for recente quando comparada com as restantes da lista).

É portanto possível (e admissível, dentro do limite mínimo dado por `S3LM_MIN_GIKS`) que um membro receba uma mensagem S3L multiponto gerada com base numa `GIK` que não é a mais recente da sua lista. Tal chave pode, inclusivamente, não fazer parte dessa lista. Se for demasiado obsoleta (o que se conclui comparando o campo `GIK_i birthdate` da mensagem S3L multiponto com a "data de nascimento remota" da chave mais velha da lista), a mensagem S3L é ignorada. Se for mais recente, indicia que, entretanto, a lista do receptor ficou obsoleta e é necessário actualizá–la através de uma rejunção, após a qual se espera encontrar na lista uma `GIK` com "data de nascimento" correspondente ao campo `GIK_i birthdate` da mensagem recebida. Se tal não acontecer, a mensagem é ignorada. Caso contrário, processada. Note–se que, ao contrário do que se passa na emissão, durante a recepção de uma mensagem S3L multiponto a rejunção não é motivada pelo término do tempo de vida da chave mais recente da lista, mas sim pela constatação de que se recebeu uma mensagem gerada com uma chave mais recente do que qualquer uma da lista.

A parametrização correcta da dimensão `S3LM_MIN_GIKS` da lista de chaves é, porventura, o aspecto mais problemático desta abordagem. Uma lista generosa de `GIK`s permite explorar chaves comprometidas durante mais tempo, mas é útil quando a geração de novas `GIK`s é tão rápida que, devido aos atrasos variáveis que os elementos de um grupo disperso sofrem durante o processo de rejunção, estes correm o risco de receber chaves obsoletas e gerar tráfego com base em `GIK`s menos recentes. Precisamente porque pode haver tráfego gerado com base em `GIK`s recentes, mas que não são a última versão, a dimensão da lista de `GIK`s deve ser tal que contemple essas chaves. Por outro lado, se a taxa de regeneração de `GIK`s é elevada, as janelas de ataque com `GIK`s comprometidas são também mais reduzidas, o que ajuda a contrabalançar um maior número de chaves na lista.

Por oposição, se o tempo de vida de uma chave `GIK` for prolongado, então pode ser possível reduzir a dimensão da lista ao mínimo admissível (5). Por tempo de vida prolongado entende–se um período de tempo capaz de absorver os maiores tempos de propagação possíveis entre quaisquer dois elementos do grupo (incluindo o dono). A figura 6.11 ilustra este raciocínio, nomeadamente o estabelecimento do valor 5 para a constante `S3LM_MIN_GIKS`.

6.2.4 Sequenciação de Mensagens S3L Multiponto

À semelhança da variante ponto–a–ponto, o S3L multiponto também contempla, nas suas mensagens, um campo `Delta` e um campo `n` (rever a figura 6.8).

O campo `n` é essencial, independentemente do valor de `Delta`, uma vez que define a variante temporal da `GIK` usada que foi originalmente gerada no dono do grupo no instante dado pelo campo `GIK_i birthdate`.

Quanto a `Delta`, recorde–se[28] que o S3L ponto–a–ponto o define como o desvio máximo admissível entre o tempo UTC instantâneo na recepção e o tempo UTC `n` instantâneo da emissão. `Delta` é, portanto, como que um tempo de vida limitado para a mensagem, definido na origem e que entra em linha de conta não só com o tempo de propagação desde a origem ao destino, como também com o desfazamento entre os relógios do emissor e do receptor.

[28] Rever a secção 5.2.1.

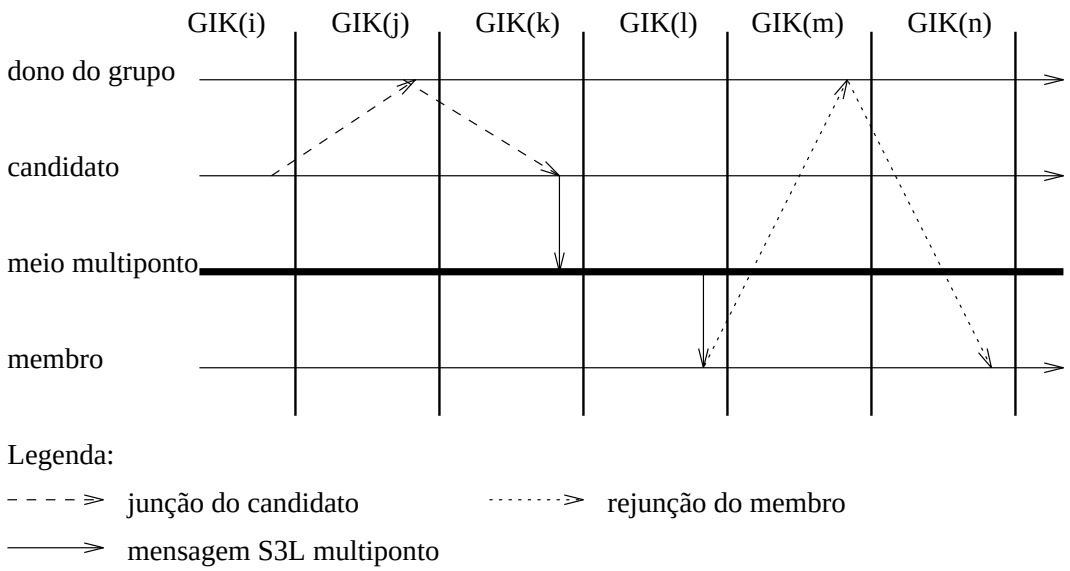

Figura 6.11: Necessidade de uma lista mínima de 5 `GIK`s.

Porém, as orientações para o cálculo de `Delta` fornecidas na secção 5.2.1 não são directamente extensíveis para um contexto de comunicação multiponto dado que, para cada destino possível (qualquer membro do grupo), os atrasos de propagação e o desfazamento de relógios são, invariavelmente, diferentes. Mesmo considerando que os elementos do grupo se localizam na mesma rede (e não dispersos, como seria o caso mais geral), o que permite considerar tempos de propagação, na prática, equivalentes, não é possível determinar um `Delta` que não acabe por introduzir janelas de ataque em um ou mais membros, devido à dessincronização de relógios.

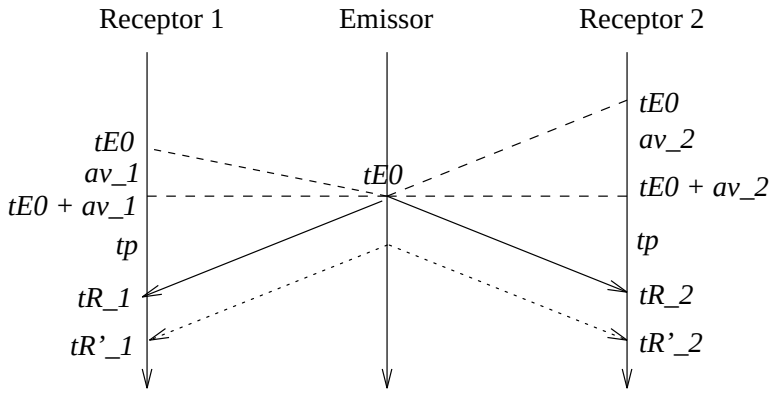

Figura 6.12: Emissor atrasado relativamente a dois Receptores.

A figura 6.12 apresenta um exemplo desta situação em que, por simplificação, se considerou o emissor atrasado relativamente a ambos os receptores. Se o emissor definir `Delta`$= av_2 + t_p$, então o Receptor2 consegue detectar a repetição da mensagem, recebida no instante $t_{R'_2}$. Tal não acontece no Receptor1 para a mesma cópia recebida no seu instante $t_{R'_1}$ dado que, pelo critério definido na secção 5.2.1, $t_{R'_1} - t_{E_0} \leq$`Delta` e portanto a

cópia da mensagem é aceite pelo Receptor1.

Assim, a utilidade do campo `Delta` reduz–se a um cenário próximo do ideal: tempos de propagação iguais e relógios sincronizados. Uma situação que se aproxima destes requisitos corresponde a um grupo em que todos os elementos se localizam na mesma máquina. Numa rede local onde se execute um protocolo de sincronização de relógios e em que os tempos de propagação são muito semelhantes, é também viável a utilização de `Delta`. Em ambos estes casos, `Delta` acaba por corresponder apenas ao atraso de propagação (comum), uma vez que se consideram os relógios sincronizados. Esta situação é ilustrada pela figura 5.5.

As dificuldades apresentadas na utilização de `Delta` sugerem a necessidade de outro mecanismo de Sequenciação contra *ataques–de–repetição*. Idealmente, se um receptor mantiver um registo, para cada origem possível, da última marca temporal que recebeu numa mensagem, então é possível controlar a recepção de mensagens repetidas e, inclusivamente, efectuar a sua reordenação. No contexto do S3L multiponto estes dois aspectos assumem particular importância, porquanto o *IP Multicast*, baseando–se no UDP, não oferece transmissão fiável e sequenciada. Tipicamente, a ausência de garantia de entrega é combatida, pelo originador, através da retransmissão das mensagens[29]. No âmbito do S3L multiponto, isso equivale à retransmissão da mesma mensagem S3L multiponto em pacotes UDP diferentes. Ora, o campo `n` de uma mensagem S3L multiponto pode ser usado como mecanismo base de ordenação uma vez que define uma marca temporal única, para cada mensagem, que pode ser interpretada como número de sequência. Não é difícil imaginar um cenário onde tal seja útil. Por exemplo: num grupo S3L multiponto circulam cotações da bolsa, originadas num membro servidor (que não é, necessariamente, o dono do grupo); mediante a assinatura do serviço (a que corresponde a presença na Lista de Controle de Acesso ao grupo), os membros recebem as cotações em mensagens S3L multiponto; a aplicação responsável, em cada membro, por visualizar as cotações recebidas é capaz de rejeitar cotações repetidas (relativas ao mesmo instante temporal de outras previamente recebidas) bem como de actualizar o historial de cotações com valores menos recentes, mas que foram recebidos após valores mais actuais.

Na próxima secção, veremos que o valor do campo `n` de uma mensagem S3L multiponto recebida é acessível através de um campo `rem_n` (*remote n*) da estrutura de dados `S3LMCtx`, que define o contexto de emissão/recepção de uma mensagem S3L multiponto. Desta forma, as aplicações que necessitem de Sequenciação de mensagens S3L multiponto têm à sua disposição a referida marca temporal.

6.3 A Interface de Programação de Aplicações do S3L Multiponto

Nesta secção descrevem–se as extensões multiponto à Interface de Programação de Aplicações do S3L ponto–a–ponto, inicialmente apresentada na secção 5.3. As estruturas de dados e funções aqui introduzidas devem ser aplicadas em conjunção com os procedimentos necessários à utilização de *sockets* de Berkeley sobre *IP Multicast*. As extensões multiponto do S3L são também descritas na secção B.2 do apêndice B, apresentando–se

[29] No protocolo *Internet Group Management Protocol* (IGMP) [Fen96], por exemplo, adopta–se um comportamento semelhante, quando um *multicast router* em fase de arranque envia, para o grupo 224.0.0.1, duas *membership queries* praticamente seguidas.

na secção B.2.8 do mesmo apêndice exemplos concretos de um servidor e de um cliente S3L multiponto.

Tal como acontece no S3L ponto–a–ponto, a utilização das funcionalidades proporcionadas pelas extensões multiponto segue um roteiro bem definido, que se divide em dois conjuntos de tarefas, em função do papel a desempenhar:

- tarefas do produtor de mensagens S3L multiponto:

  1. obtenção de informação que identifica e caracteriza a entidade produtora;
  2. inicialização de um contexto S3L multiponto, especificando–se o endereço *IP Multicast* do grupo e identificando–se o seu dono;
  3. assemblagem da mensagem S3L multiponto com base no contexto inicializado em 2 (e que acaba por ser completamente definido, em resultado de uma junção implícita junto do dono do grupo S3L);
  4. envio da mensagem assemblada para o grupo S3L, subgrupo de um determinado grupo *IP Multicast*, através da função `S3Lsendto`;
  5. repetição das etapas 2, 3 e 4, com possível reutilização dos contextos inicializados na etapa 2 sempre que o grupo em causa e o seu dono se mantenham os mesmos;

- tarefas do consumidor de mensagens S3L multiponto:

  1. obtenção de informação que identifica e caracteriza a entidade consumidora;
  2. inicialização de um contexto S3L multiponto, especificando–se o endereço *IP Multicast* do grupo e identificando–se o seu dono;
  3. junção explícita de um *socket* de Berkeley ao grupo *IP Multicast*[30];
  4. recepção de uma mensagem S3L multiponto *via* `S3Lrecvfrom`;
  5. processamento da mensagem recebida com base no contexto inicializado em 2 (e que acaba por ser completamente definido, em resultado de uma junção implícita junto do dono do grupo S3L);
  6. repetição das etapas 2, 3, 4 e 5, com possível reutilização dos contextos inicializados na etapa 2 sempre que o grupo em causa e o seu dono se mantenham os mesmos, em cujo caso se dispensa também a repetição da junção *IP Multicast* efectuada em 3.

Das tarefas apresentadas, ressaltam algumas semelhanças com o modelo de utilização da API do S3L ponto–a–ponto. Nomeadamente, a obtenção de informação de "carácter pessoal" que caracteriza a entidade produtora ou consumidora de mensagens continua a constituir uma tarefa (a primeira) obrigatória. As estruturas de dados e funções relacionadas com a manipulação desse tipo de informação foram já apresentadas na secção 5.3.1 do capítulo anterior. As restantes tarefas lidam com alguns pormenores específicos do S3L multiponto, pelo que as respectivas estruturas de dados e funções serão objecto de discussão nas próximas secções.

[30] Dispensável para um emissor, mas necessária num receptor.

6.3.1 Manipulação de Contextos Multiponto Seguros

Um contexto multiponto define–se como uma estrutura de dados onde se concentra toda a informação relevante à operação de uma entidade como membro de um grupo S3L. É com base na informação guardada nessa estrutura que uma entidade pode produzir mensagens S3L multiponto de acordo com as qualidades de serviço definidas pelo dono do grupo.

```
typedef struct {
        time_t  birthdate;    /* ``data de nascimento'' da GIK */
        char    gik[32];      /* GIK */
} S3LMGikInfo;                /* informacao relativa a uma GIK */

typedef struct {
u_char version; /* versao das extensoes multiponto do S3L */

char   mipaddr[16];     /* endereco IP multicast do grupo */
time_t group_lifetime;  /* tempo de vida do grupo */
time_t group_birthdate; /* ``data de nascimento'' do grupo */

char gik_file[UNIX_PATH_MAX]; /* ficheiro com a proxima versao da GIK */
S3LMGikInfo gik_array[S3LM_MAX_GIKS]; /* array circular de GIKs */
int     gik_array_front; /* 1a posicao livre do array */
int     gik_array_back;  /* 1a posicao ocupada do array */
S3Lbool gik_array_full;  /* estado (cheio/nao cheio) do array */
time_t  gik_lifetime;    /* tempo de vida de uma GIK */

time_t  kp_kcis; /* intervalo de mudanca de Kp em segundos */
size_t  kp_kcib; /* intervalo de mudanca de Kp em bytes */

char    gikalg[64];  /* algoritmo de encriptacao de Kp */
char    cryptalg[64];/* algoritmo de encriptacao dos dados */
car     macalg[64];  /* algoritmo de producao do MAC */
char    compalg[64]; /* algoritmo de compressao dos dados */
} S3LMGroupInfo;
```

Figura 6.13: Estruturas auxiliares `S3LMGikInfo` e `S3LMGroupInfo`.

A figura 6.14 apresenta a definição, em C, do tipo `S3LMCtx`, relativo a um contexto S3L multiponto. A figura 6.13 apresenta outros tipos de dados auxiliares, entre os quais o tipo `S3LMGroupInfo`. O dono do grupo mantém, numa estrutura deste tipo, a informação que caracteriza o grupo. É precisamente esta informação que é devolvida a um candidato ou membro numa confirmação de (re)junção, ficando acessível através do campo `group_info` do contexto multiponto `S3LMCtx`. Portanto, a definição de um contexto só se encontra concluída após uma junção ao grupo S3L. Todavia, antes dessa junção, que ocorre implicitamente durante a síntese ou análise de mensagens S3L multiponto (ver funções `S3LMmake_message` e `S3LMopen_message`), é necessário colocar no contexto um conjunto mínimo de informação que permita levar a cabo, posteriormente, o referido processo de junção. É esse o papel da função `S3LMinit_S3LMCtx`:

- `void S3LMinit_S3LMCtx (S3LMCtx *mctx, S3LPartieInfo *loc_info, char *mipaddr, char *gowner_dname, char *gowner_ipaddr)`: inicializa um con-

```
typedef struct {
S3LPartieInfo *loc_info;  /* informacao relativa `a entidade local */

char gowner_dname[SBUF_SIZE]; /* nome distinto X.500 do dono do grupo */
char gowner_ipaddr[16];       /* endereco IP do dono do grupo */
S3LMGroupInfo group_info;     /* caracterizacao do grupo */
time_t loc_gik_birthdate; /* ``data de nascimento'' local da GIK mais
                             recente */

size_t kp_counter;   /* numero de bytes enviados com a ultima Kp */
time_t kp_birthdate; /* ``data de nascimento'' local da ultima Kp */
char   kp[32];       /* ultima Kp gerada aleatoriamente */

struct timeval  loc_n; /* marca temporal local */
struct timeval  rem_n; /* marca temporal remota */
u_long          delta; /* desvio maximo admissivel entre loc_n e rem_n */
} S3LMCtx;
```

Figura 6.14: Estrutura `S3LMCtx`.

texto multiponto `mctx`, actualizando os seus campos `loc_info`, `group_info.mipaddr`, `gowner_dname` e `gowner_ipaddr` com o valor dos parâmetros homólogos da função.

A API do S3L não oferece mais funções de manipulação de contextos multiponto. A inicialização dos restantes campos fica a cargo dos mecanismos implícitos de (re)junção ao grupo. A utilização de memória estática dispensa também uma função de libertação específica.

Existem, no entanto, dois campos cuja manipulação directa poderá ter interesse. A leitura do campo `rem_n` após a recepção de uma mensagem permitirá aceder ao valor do campo `n` dessa mensagem a fim de proceder, por exemplo, à sua sequenciação ou à detecção de repetições. A escrita no campo `delta` de um valor positivo, antes do envio de uma mensagem, constitui indicação de que no destino esse campo deverá ser usado na detecção de *ataques–de–repetição*, de uma forma análoga à das mensagens S3L ponto–a–ponto.

A reutilização de um contexto `S3LMCtx` é possível desde que o grupo e o seu dono se mantenham inalterados, uma vez que a reactualização do contexto ocorre, implícita e progressivamente, *por necessidade*.

6.3.2 Síntese e Análise de Mensagens S3L Multiponto

À semelhança das mensagens do S3L ponto–a–ponto, o processamento de mensagens S3L multiponto é efectuado por funções específicas, conforme se trate da assemblagem ou da desassemblagem dessas mensagens:

- `S3Lbool S3LMmake_message (char *buf_in, u_long len_buf_in, char *buf_out, u_long *len_buf_out, S3LMCtx *mctx)`: sintetiza uma mensagem S3L multiponto, com base nos dados `buf_in` (de dimensão `len_buf_in`) e no contexto `mctx`; a mensagem produzida é depositada em `buf_out`, reflectindo `len_buf_out` a sua dimensão;

- `S3Lbool S3LMopen_message (char *buf_in, u_long len_buf_in, char *buf_out, u_long *len_buf_out, S3LCtx *mctx)`: analisa, campo a campo, uma mensagem S3L, fornecida em `buf_in`; o contexto `mctx` recebe informação contextual resultante da análise da mensagem e de uma eventual (re)junção ao grupo durante esse processo; os dados recuperados da mensagem são depositados em `buf_out` e a sua dimensão indicada através de `len_buf_out`.

A estrutura das mensagens S3L multiponto foi previamente introduzida na secção 6.2.2. Aparte o trabalho de assemblagem e desassemblagem de mensagens desse tipo, estas funções levam a cabo execuções do processo de (re)junção ao grupo. As condições que determinam, nestas funções, a execução implícita de (re)junções ao grupo foram discutidas previamente na secção 6.2.3.

6.3.3 Escrita e Leitura de Mensagens S3L Multiponto sobre Sockets

As funções `S3Lsendto` e `S3Lrecvfrom` foram apresentadas, em primeira instância, na secção 5.3.4 do capítulo anterior, optando–se, nesse contexto, por não antecipar a sua relação com o S3L multiponto. A utilização destas funções para enviar e receber mensagens S3L multiponto encapsuladas no campo de dados de pacotes UDP deriva do facto de que, no modelo de programação de *sockets* de Berkeley, são precisamente as primitivas `sendto` e `recvfrom` as únicas primitivas de escrita e leitura que operam com *sockets IP Multicast*. A utilização das funções `S3Lrecvfrom` e `S3Lsendto` assegura, assim, semelhanças sintácticas e semânticas com as respectivas primitivas de Berkeley.

O protótipo final que a seguir se apresenta para as funções `S3Lsendto` e `S3Lrecvfrom` apenas difere do apresentado anteriormente na secção 5.3.4 pela utilização do tipo `void *` na tipificação do parâmetro do contexto, sendo internamente feitas conversões para `S3LCtx *` ou `S3LMCtx *`, conforme o caso:

- `int S3Lsendto (int fd, void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen, void *ctx);`
- `int S3Lrecvfrom (int fd, char *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen, void *ctx);`

Internamente, cada uma desta funções invoca procedimentos ponto–a–ponto (`S3Lmake_message`, `S3Lopen_message`) ou multiponto (`S3LMmake_message`, `S3LMopen_message`), em função do tipo do endereço IP encerrado no campo `sin_addr` dos parâmetros `to` ou `from`, *c.f.* o caso.

6.4 Detecção de Grupos Pré–Activos

Genericamente, os grupos *IP Multicast* dividem–se em duas categorias: permanentes e transientes. Ambas as categorias utilizam endereços IP de classe D (gama [`224.0.0.0, 239.255.255.255`]).

Os grupos permanentes utilizam endereços bem conhecidos, tipicamente na zona baixa da gama [`224.0.0.0, 224.255.255.255`] e registados pela *Internet Assigned Numbers Authority* (IANA), sendo divulgados através de documentos como o RFC 1700 [RP94].

Informação ainda mais actualizada sobre endereços permanentes pode também ser obtida através do comando "`host -l mcast.net`".

Por sua vez, os grupos transientes são criados, dinamicamente, *por necessidade*, pelas aplicações que usufruem, esporadicamente, das capacidade multiponto do IP, não havendo, actualmente, nenhum mecanismo em operação capaz de assegurar uma reserva de endereços mutuamente exclusivos[31]. Nestas circunstâncias, é perfeitamente possível que o tráfego multiponto gerado por dois conjuntos distintos de aplicações se "misture", caso o endereço IP de classe D escolhido seja o mesmo.

Ao nível da variante multiponto do S3L, foram tomadas medidas no sentido de resolver, o tanto quanto possível, o problema da colisão de endereços: opcionalmente, pode ser solicitada à aplicação `s3lmgowner` que detecte a existência de um grupo *IP Multicast* em actividade, com o mesmo endereço do grupo que se pretende gerir.

A coexistência de mais do que um dono do grupo para o mesmo endereço *IP Multicast* deve ser evitada, por razões óbvias: basta uma diferença mínima nas chaves `GIK`, nos algoritmos ou até na temporização da `GIK` para que seja incompatível o tráfego gerado sob ambos os domínios de qualidades de serviço. Adicionalmente, pode não existir mais nenhum dono do grupo para o endereço pretendido, mas é perfeitamente admissível que esse grupo esteja em actividade, com tráfego *IP Multicast* de outra natureza que não S3L.

6.4.1 Detecção na Própria Máquina

A detecção de outro dono S3L, do mesmo grupo, localizado na mesma máquina que o novo dono, é trivial, porquanto os demónios `s3lmgowner` reservam recursos de uma forma que os identifica facilmente. Recordando o que foi dito na secção 6.2.1, os pedidos de junção são recebidos através de um *socket* de domínio Unix, localizado em `/tmp`, e designado segundo um esquema semelhante ao referido na secção 4.4 para os demónios `s3lkeyxserver`, mas que leva também em conta o endereço *IP Multicast*. Pela análise do conteúdo de `/tmp`, um demónio `s3lmgowner` em arranque, fica imediatamente a saber da existência de outro que controla o mesmo grupo. Todavia, a não existência de outro demónio `s3lmgowner` para esse grupo não é conclusiva, dado que a máquina pode já pertencer ao grupo em causa, por via de outra aplicação que tenha efectuado a junção independentemente de procedimentos S3L[32]. Neste caso, a detecção deve dirigir–se ao perímetro definido pela rede local (LAN), contemplando também a própria máquina.

6.4.2 Detecção na Rede Local

A detecção de elementos do grupo pretendido na LAN da qual faz parte a máquina onde o novo dono quer executar é possível, desde que exista pelo menos um "encaminhador multiponto[33]" em actividade nessa LAN. Esse encaminhador pode existir implementado quer em *hardware* (embora a base instalada de encaminhadores com capacidades multiponto

[31] A este propósito, em [BZ93] sugere–se a criação de uma *Multicast Group Authority* (MGA), organizada hierarquicamente numa estrutura semelhante ao DNS, possibilitando a distribuição dinâmica de pacotes de endereços multiponto e mantendo informação sobre a composição dos grupos.

[32] A junção aqui em causa refere–se à realização de uma operação `IP_ADD_MEMBERSHIP`, através da primitiva `setsockopt`, sobre um *socket* do tipo `SOCK_DGRAM`, e pode ser observada no código que exemplifica um cliente S3L multiponto, na secção B.2.8 do apêndice B.

[33] Do inglês *multicast router*.

seja ainda reduzida), quer sob a forma de um demónio — o `mrouted`. O encaminhador e as máquinas da LAN com "capacidades *IP Multicast*"[34] executam o protocolo *Internet Group Management Protocol* (IGMP) [Fen96], através do qual o encaminhador mantém um registo dos grupos *IP Multicast* que têm elementos na LAN. Desta forma, o encaminhador pede aos encaminhadores vizinhos que lhe passem o tráfego *IP Multicast* dirigido apenas aos grupos activos na sua LAN. Periodicamente[35], um encaminhador multiponto solicita a todas as máquinas da LAN que lhe enviem informações acerca dos grupos *IP Multicast* a que pertencem. Estes pedidos (*membership queries*), bem como as respectivas respostas (*membership reports*), são dirigidos ao grupo `224.0.0.1`, ao qual pertencem, por omissão, todas as máquinas de uma LAN com capacidades *IP Multicast*.

Assim, para que uma máquina da LAN fique a saber se um determinado endereço *IP Multicast* tem membros associados nessa LAN, bastar–lhe–á auscultar o tráfego IGMP, nomeadamente os relatórios de pertença (*membership reports*). Precisamente, o demónio `s3lmgowner` permite, sob pedido expresso (opção `-b`), executar essa detecção. Porém, a análise de tráfego IGMP só é possível através de um *socket* de tipo `SOCK_RAW`, associado ao protocolo `IPPROTO_IGMP`, estando a criação deste tipo de *sockets* limitada, por razões óbvias de segurança[36], a utilizadores com privilégios de super–utilizador. Consequentemente, a aplicação `s3lmgowner` é gerada com *setuid root*, a fim de permitir a detecção de grupos activos por utilizadores normais. Caso se prescinda desta funcionalidade, é sempre possível gerar novamente `s3lmgowner` sem privilégios especiais (ver a secção A.1 do apêndice A).

6.4.3 Detecção num Contexto Global

A preocupação com a pré–activação do mesmo grupo fora do âmbito da rede local justifica–se apenas quando se pretende criar um grupo cujos elementos irão, possivelmente, dispersar–se por redes distintas. Caso contrário, os procedimentos descritos anteriormente, em conjunção com a manutenção do parâmetro `ttl` (*time–to–live*) dos pacotes *IP Multicast* com o valor 1 (a fim de não serem nunca encaminhados para o exterior da sua LAN de origem) são suficientes para uma operação sem problemas.

Quanto à detecção de elementos do grupo fora das fronteiras da rede local, não existe, actualmente, um método genérico e universal para o efeito. Tal deriva, sobretudo, da conjugação dos seguintes factores:

1. nem todos os encaminhadores têm capacidades multiponto. Protocolos de encaminhamento multiponto como o *Distance Vector Multicast Routing Protocol* (DVMRP) [WPD88], o *Multicast Open Shortest Path First* (MOSPF) [Moy94] ou o *Protocol Independent Multicast* (PIM) [EFHT97] só recentemente foram introduzidos nos encaminhadores comercialmente mais bem sucedidos, pelo que a base instalada destes dispositivos é, ainda, pouco significativa;

2. consequentemente, a própria experiência na utilização intensiva desses protocolos é também diminuta, pelo que são de prever correcções ao *firmware*/*software* dos encaminhadores, à medida que eventuais problemas sejam detectados. Naturalmente,

[34] A título de curiosidade, refira–se que o comando `ping -t 1 -c 2 224.0.0.1` fornece a lista das máquinas da LAN com capacidade de processar tráfego *IP Multicast*.

[35] De 125 segundos em 125 segundos, de acordo com [Fen96].

[36] Basta lembrar que um *socket* deste tipo pode ser usado para forjar ou analisar o tráfego IP.

isto gera alguma desconfiança relativamente à viabilidade de utilizar o *IP Multicast* em aplicações de grande escala e onde, eventualmente, o transporte de informação crítica (sobre um protocolo que já é, por natureza, "infiável" e "não–sequenciado") seja necessário;

3. mesmo dispondo de encaminhadores multiponto em *hardware*, ou recorrendo, na sua ausência, à utilização de túneis IPIP entre demónios `mrouted`, os próprios protocolos de encaminhamento multiponto carecem de mecanismos de base, que permitam a detecção de grupos activos a partir de pontos que não pertencem a uma rede com elementos nesses grupos. Estes protocolos estabelecem uma espécia de árvore/grafo de encaminhamento, para cada grupo, que liga os encaminhadores das redes que têm pelo menos um elemento nesse grupo. Consequentemente, noutras redes sem elementos nesse grupo não existe a hipótese de saber se o grupo está activo algures, porque os encaminhadores só mantêm informação de pertença sobre os grupos activos na sua rede, bem como se dispõem a receber tráfego apenas para esses grupos, "podando"[37] os ramos da árvore de encaminhamento que não lhes interessam;

4. os mecanismos de *firewall*[38] mais populares, baseados em *gateways* de aplicação, não suportam protocolos *não–orientados–à–conexão*, como o UDP (que transporta os datagramas *IP Multicast*), limitando a utilização do *IP Multicast* ao interior da *firewall*. Neste caso, a desvantagem derivada do facto de não ser possível aceder a tráfego *IP Multicast* originado no exterior é, de certa forma, compensada pela garantia de Privacidade do tráfego que circula no interior da *firewall*, bem como pela imunidade à "mistura" com tráfego proveniente de fontes externas e dirigido também ao mesmo grupo.

Em face deste cenário, a criação de grupos transientes à escala de toda a Internet, garantindo a exclusividade do endereço IP de classe D usado é, naturalmente, problemática. Geralmente, a solução passa pelo anúncio prévio de que determinado endereço vai ser usado, "oficialmente", a partir de determinada data, durante um intervalo de tempo bem definido. Existem inclusivamente ferramentas capazes de anunciar e detectar anúncios da utilização de um determinado endereço *IP Multicast*. Em `http://www.merit.edu/net-research/mbone/index/titles.html` pode ser encontrada uma lista de aplicações dessa categoria, genericamente designadas de *Session Announcement Utilities*, bem como de inúmeras outras que tiram partido do *IP Multicast*.

[37] Do inglês *pruning*.
[38] Consultar, por exemplo, [CZ95], que ainda constitui uma obra de referência na matéria.

Capítulo 7

Análise de Desempenho

A fim de averiguar o impacto das qualidades de serviço disponibilizadas pelo S3L relativamente à utilização simples e directa dos *sockets* de Berkeley, foi efectuado um estudo comparativo baseado na medição de tempos resultantes da adopção de cada uma dessas abordagens. Acima de tudo, pretendia–se averiguar qual o preço a pagar pela Privacidade, Autenticação, Integridade e Compressão[1], em ambas as modalidades ponto–a–ponto e multiponto do S3L.

7.1 Material de Apoio

Para a realização dos testes de desempenho, foram seleccionadas duas máquinas, interligadas através de uma rede Ethernet de 10 Mbps e configuradas de forma idêntica:

- um microprocessador Intel Pentium a 166 MHz;
- placa–mãe com *chipset* Intel 430 HX e 512Kb de *cache pipelined burst*;
- 32Mb de memória EDO e 2Gb de disco EIDE de 10ms;
- sistema operativo Linux (versão 2.0.30 do *kernel*, distribuição *Slackware 3.2*).

Uma vez que os testes realizados tinham em vista, sobretudo, a comparação, em termos relativos, da "abordagem Berkeley" e da "abordagem S3L" e não tanto a medição de tempos de desempenho absolutos, a adopção destas máquinas em detrimento de outras mais rápidas é perfeitamente aceitável. Adicionalmente, registe–se que tais máquinas constituem, actualmente, o nível de entrada nas configurações disponibilizadas pelo mercado de PCs, pelo que os tempos obtidos serão sempre susceptíveis de melhoramento em configurações de melhor nível.

7.2 Metodologia

A medição de tempos de desempenho de operações que se iniciam numa máquina e se concluem noutra é, à partida, problemática, uma vez que estão envolvidos dois relógios diferentes, em princípio não sincronizados.

[1] Nos testes efectuados não se solicitou detecção de *ataques–de–repetição*, uma vez que o impacto desse requisito é, em termos temporais, negligenciável.

Todavia, uma operação de escrita suportada por um protocolo que oferece confirmações de entrega, como é o caso do TCP, não apresenta dificuldades de maior: a diferença entre o instante imediatamente posterior ao retorno da função de escrita e o instante imediatamente anterior à sua invocação constitui um tempo que entra não só em linha de conta com o processamento local e remoto dos dados, mas também com o tempo de propagação dos mesmos (e da respectiva confirmação de entrega). O tempo de propagação é um factor importante na medição do desempenho do S3L dado o ganho potencial que a utilização da Compressão introduz, atenuando os tempos consumidos em operações de Encriptação e Autenticação.

Já no contexto de um protocolo *não–orientado–à–conexão*, como é o caso do UDP, a aplicação do método anterior à medição do tempo consumido numa função de escrita revela–se inadequada, uma vez que tal função retorna assim que a mensagem é entregue à rede, confiando–se nos "melhores esforços" desta para fazer chegar a mensagem ao destino final.

É portanto necessária uma metodologia de medida de desempenho que contemple os tempos de propagação em ambos os cenários (TCP e UDP) e que seja independente da sincronização dos relógios das máquinas envolvidas. Assim, relativamente às diversas operações de escrita e leitura (variantes Berkeley e S3L), optou–se pela medição de "tempos de ida e volta", o que permitiu, como benefício adicional, integrar numa mesma medida o tempo de escrita e o tempo de leitura. Um "tempo de ida e volta" consiste, portanto, no tempo que decorre desde a invocação de uma função de escrita até ao retorno da respectiva função de leitura, supondo que uma escrita é seguida, de imediato, da leitura dos mesmos dados, devolvidos por um servidor remoto que actua como reflector.

A resolução escolhida para os valores temporais foi o microsegundo, uma vez que era essa a resolução do relógio da máquina onde foram registados os tempos. Tais tempos devem ser entendidos como majorantes para cada cenário em causa, uma vez que o código do S3L contém ainda elementos auxiliares de depuração.

Em alguns casos, apresentam–se também os valores da carga da máquina (obtidos através da execução do comando `w`) anteriores e posteriores à execução de cada teste, pretendendo–se com isso fornecer elementos que ajudem a avaliar o impacto de cada tipo de operação realizada.

7.3 Resultados

7.3.1 Actualizações de Contextos S3L Ponto–a–Ponto

O primeiro teste realizado destinou–se a determinar o tempo consumido na actualização de um contexto S3L ponto–a–ponto, com e sem execução implícita do protocolo Skeyx. A execução do protocolo Skeyx efectuou–se sempre entre duas entidades situadas em máquinas distintas a fim de entrar em linha de conta com tempos de propagação. O tempo medido consiste no tempo médio consumido pela execução da função `S3Lmake_S3LCtx` numa amostra de dimensão 10.

A tabela 7.1 apresenta os tempos registados. O tempo consumido quando ocorre a execução do protocolo Skeyx é notoriamente elevado (aproximadamente 2 segundos), dada a relativa complexidade das transacções realizadas. Afortunadamente, a criação de estado partilhado entre duas entidades ocorre uma só vez, aquando do primeiro contacto

`S3Lmake_S3LCtx`	tempo consumido	carga anterior	carga posterior
com Skeyx	1953320 μs	0.07, 0.09, 0.04	0.29, 0.14, 0.06
sem Skeyx	7737 μs	0.04, 0.09, 0.06	0.03, 0.08, 0.06

Tabela 7.1: Tempo consumido na função `S3Lmake_S3LCtx`.

S3L ponto–a–ponto entre ambas. Posteriomente, o tempo dispendido na recuperação do contexto em *cache* (ainda que em disco) é bastante inferior, como seria de esperar.

7.3.2 Primitivas Ponto–a–Ponto

De seguida, procurou-se confrontar todas as primitivas de escrita e leitura de Berkeley com as funções análogas do S3L, nas suas variantes ponto–a–ponto. Para o efeito, a dimensão escolhida para o bloco de dados foi de 1K (para as primitivas de transporte de vectores — `(S3L)writev`, `(S3L)readv`, `(S3L)sendmsg` e `(S3L)recvmsg` —, assumiram-se vectores com dois blocos de 1K). Adicionalmente, definiu-se o cenário previsivelmente mais favorável (em termos de desempenho) para o S3L com blocos de tal dimensão: sem Compressão[2] e sem Encriptação dos dados. A Autenticação, obrigatória, foi efectuada à custa de MACs resultantes da aplicação dos algoritmos MD5 e IDEA. A Encriptação das chaves `Kp`, também obrigatória, realizou-se com base no algoritmo DES-CBC. A utilização destes algoritmos manteve–se constante em todos os testes ao desempenho do S3L (ponto–a–ponto e multiponto).

Primitivas	tempo consumido	carga anterior	carga posterior
`write+read`	4832 μs	0.04, 0.13, 0.13	0.04, 0.13, 0.13
`S3Lwrite+S3Lread`	50963 μs	0.01, 0.03, 0.06	0.01, 0.02, 0.06
`send+recv`	4821 μs	0.06, 0.14, 0.14	0.06, 0.14, 0.14
`S3Lsend+S3Lrecv`	50850 μs	0.02, 0.03, 0.04	0.02, 0.03, 0.04
`sendto+recvfrom`	4764 μs	0.00, 0.08, 0.11	0.00, 0.08, 0.11
`S3Lsendto+S3Lrecvfrom`	51684 μs	0.00, 0.02, 0.03	0.08, 0.04, 0.04
`writev+readv`	6634 μs	0.12, 0.08, 0.09	0.12, 0.08, 0.09
`S3Lwritev+S3Lreadv`	99628 μs	0.06, 0.07, 0.04	0.20, 0.10, 0.05
`sendmsg+recvmsg`	9058 μs	0.05, 0.07, 0.09	0.05, 0.07, 0.08
`S3Lsendmsg+S3Lrecmsg`	99964 μs	0.23, 0.10, 0.05	0.19, 0.09, 0.05

Tabela 7.2: Tempo consumido pelas primitivas de Berkeley e S3L sobre TCP.

A tabela 7.2 apresenta os tempos médios registados para cada grupo de funções, resultantes de uma amostra de dimensão 100. Neste caso, apenas foi testada a transmissão de dados ponto–a–ponto sobre TCP. Os tempos assim obtidos são, previsivelmente, superiores aos resultantes da utilização do UDP, uma vez que se consome tempo em confirmações de entrega. Todavia, em termos relativos, a diferença entre as funções S3L e as primitivas de Berkeley deverá ser semelhante sobre UDP, suspeitas confirmadas posteriormente

[2]Com blocos de dimensão reduzida, o tempo consumido na compressão acaba por não compensar a economia de tempo de propagação.

através de testes realizados às primitivas `(S3L)write` e `(S3L)read` (ver tabela 7.3 *versus* tabela 7.4).

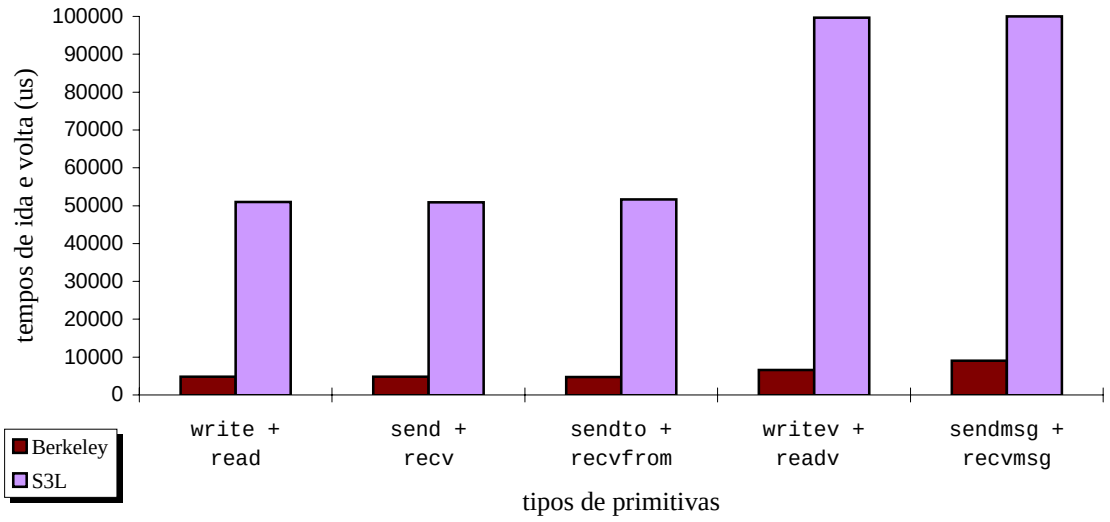

Figura 7.1: Desempenho das primitivas de Berkeley e S3L sobre TCP.

O gráfico da figura 7.1 ajuda a compreender melhor a diferença, em termos absolutos, entre os tempos consumidos pelas primitivas de Berkeley e pelas funções S3L, nas condições específicas em que foi realizado o teste em causa. Essa diferença é, de forma evidente, considerável. A obtenção de tempos superiores era, aliás, de esperar, uma vez que o próprio S3L assenta na utilização das primitivas de Berkeley, assim como faz uma utilização intensiva das funcionalidades criptográficas de base oferecidas pelo SecuDE, as quais determinam, em boa parte, o grau de desempenho do S3L. Na prática, o S3L "limita–se" a empacotar e desempacotar os dados do utilizador, num formato bem definido pelos seus protocolos. Ainda assim, o preço a pagar pela Autenticação (já que não se efectuou Encriptação nem Compressão) é, para o exemplo em questão, bastante elevado.

Da análise do gráfico, conclui–se ainda que os tempos consumidos pelas primitivas de Berkeley (e, por consequência, pelas funções S3L que as invocam) dividem–se, claramente, em dois grupos: um relativo às primitivas de transporte de vectores, que consomem um tempo aproximadamente proporcional à dimensão do vector (dois blocos de 1K, no nosso caso), e o grupo complementar. Em ambos os grupos, os tempos são muito semelhantes, sugerindo, ao nível do *kernel*, funções de base comuns[3]. Adicionalmente, recorde–se (rever a tabela 5.1) que algumas funções S3L de leitura se baseiam na invocação da mesma primitiva de Berkeley (concretamente, `recvfrom`).

Por si só, estes resultados não são conclusivos, uma vez que é necessário determinar a influência que a dimensão do bloco de dados tem, em conjunção com a sua Encriptação e/ou Compressão. Para o efeito, foi realizada nova amostragem, de dimensão 100, fazendo variar a dimensão do bloco de dados de uma forma exponencial (segundo potências de 2). Optou–se por fazer esse estudo apenas com as primitivas `(S3L)write` e `(S3L)read`, dada a semelhança de tempos com as restantes primitivas do seu grupo, bem como a proporcionalidade de tais tempos com o grupo de primitivas de transporte de vectores.

[3] Ver, por exemplo, [WS96], para uma explicação detalhada do código dessas funções.

Analogamente ao estudo da figura 7.1, escolheu–se o TCP como protocolo de comunicação de base.

Foram então considerados quatro cenários possíveis: `write+read` (variante Berkeley), `S3Lwrite+S3Lread` (variante S3L ponto–a–ponto simples), `S3Lwrite+S3Lread` com Compressão dos dados e `S3Lwrite+S3Lread` com Compressão e Encriptação (segundo o algoritmo IDEA) dos dados.

dimensão dos dados	`write` + `read`	`S3Lwrite` + `S3Lread`	`S3Lwrite+S3Lread` com Compressão	`S3Lwrite+S3Lread` com Compressão e Encriptação
1K	4832 μs	50963 μs	70072 μs	75085 μs
2K	7983 μs	55323 μs	74458 μs	79998 μs
4K	18898 μs	68971 μs	83284 μs	92823 μs
8K	39014 μs	99877 μs	104491 μs	120232 μs
16K	69057 μs	139262 μs	127128 μs	150390 μs
32K	107469 μs	198960 μs	153234 μs	177698 μs
64K	171994 μs	308222 μs	220422 μs	257133 μs
128K	289874 μs	497191 μs	331855 μs	375222 μs
256K	511010 μs	905416 μs	561276 μs	604575 μs
512K	983132 μs	1714814 μs	1023028 μs	1090544 μs
1024K	1890444 μs	3316010 μs	1932694 μs	2041695 μs

Tabela 7.3: Tempo consumido pelas primitivas `(S3L)write` e `(S3L)read` sobre TCP.

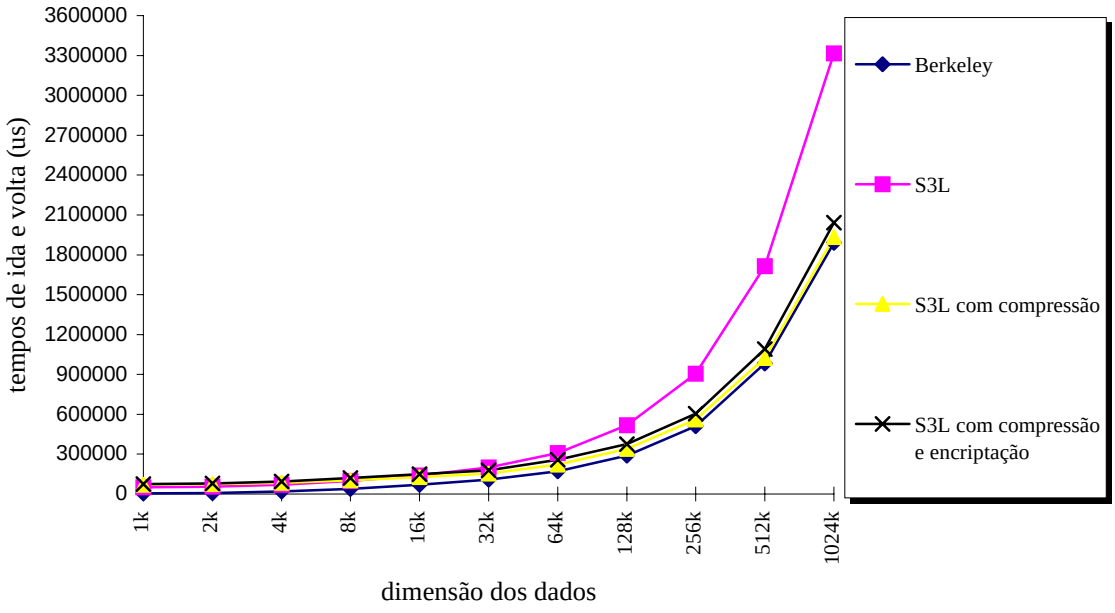

Figura 7.2: Desempenho das primitivas `(S3L)write` e `(S3L)read` sobre TCP.

A tabela 7.3 condensa os tempos obtidos em cada uma dessas situações, e o gráfico da figura 7.2 ilustra a evolução desses valores com a variação exponencial da dimensão do

bloco de dados. Como era de prever, o desempenho do S3L melhora significativamente com a introdução de Compressão, sendo essa melhoria tanto mais evidente quanto maior for a dimensão dos dados. Para blocos de grandes dimensões, o ganho de desempenho permitido pela Compressão permite obter tempos da mesma ordem de grandeza para as primitivas de Berkeley e S3L, com a vantagem adicional de estas fornecerem Autenticação e, se solicitada, detecção de *ataques–de–repetição*. Naturalmente, a Encriptação continua a representar um "encargo" adicional, mas quando combinada com a Compressão, continuam a obter–se tempos bastante razoáveis.

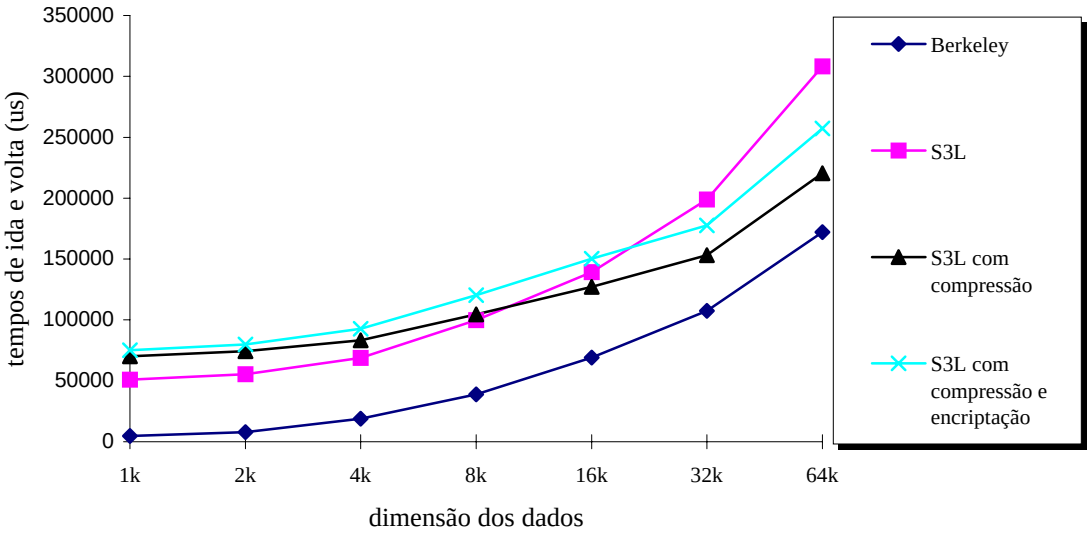

Figura 7.3: Ampliação da figura 7.2 para blocos no intervalo [1K, 64K].

Para blocos de dados de dimensão mais modesta, a figura 7.3 apresenta, com mais detalhe, as diferenças entre os cenários da figura 7.2, deixando, por exemplo, perceber que, sobre TCP, e para o bloco de dados usado nos testes, o tempo consumido pela Compressão é compensado pela diminuição do tempo de propagação, a partir de uma dimensão de bloco algures entre 8K e 16 K.

A fim de comprovar o previsível ganho de desempenho resultante da utilização do UDP, foi realizado um teste similar ao reproduzido pela figura 7.3, desta feita com uma dimensão máxima para o bloco de dados de 63K, dado que, ao contrário do TCP, o UDP não segmenta os dados antes destes serem enviados pelo IP. O IP reserva 16 bits no seu cabeçalho para especificar a dimensão dos dados [Pos80b]. Como o cabeçalho IP pode ter, no máximo, 60 bytes, sobram 65535-60=65475 bytes, dos quais 8 são para o cabeçalho UDP [Pos80a]. Os restantes 65467 bytes são, portanto, a dimensão máxima do campo de dados de um datagrama UDP, superior ainda à dimensão usada (63K=64512) neste teste. Embora segundo [Pos80b] um *host* tenha apenas a obrigação de suportar a recepção de datagramas de 576 bytes, [Ste96] refere que a maioria das implementações actuais asssegura, pelo menos, a recepção de datagramas de 8192. Nos testes realizados sobre UDP com o sistema operativo Linux, a utilização de 63K para a dimensão do bloco de dados não constituiu problema.

A tabela 7.4 regista os tempos de ida e volta obtidos e o gráfico 7.4 ilustra a sua evolução com a variação da dimensão do bloco de dados. Da comparação dos valores

dimensão dos dados	`write` + `read`	`S3Lwrite` + `S3Lread`	`S3Lwrite+S3Lread` com Compressão	`S3Lwrite+S3Lread` com Compressão e Encriptação
1K	4232 μs	50616 μs	68145 μs	72646 μs
2K	6684 μs	55077 μs	72503 μs	77546 μs
4K	10281 μs	60582 μs	80615 μs	90788 μs
8K	17400 μs	73445 μs	102536 μs	118745 μs
16K	32455 μs	99494 μs	124887 μs	148751 μs
32K	62168 μs	150401 μs	150573 μs	175625 μs
63K	117840 μs	248010 μs	202169 μs	230719 μs

Tabela 7.4: Tempo consumido pelas primitivas `(S3L)write` e `(S3L)read` sobre UDP.

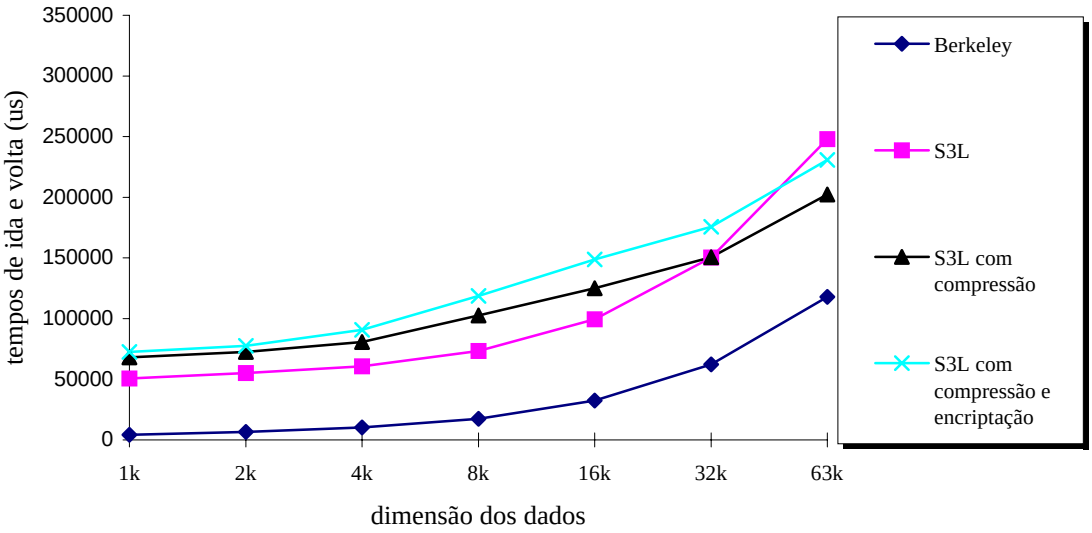

Figura 7.4: Desempenho das primitivas `(S3L)write` e `(S3L)read` sobre UDP.

das tabelas 7.3 e 7.4, bem como dos respectivos gráficos, conclui–se, de imediato, que a utilização do UDP, como era previsto, representa tempos de ida e volta menores, derivados da ausência de confirmações de recepção. Adicionalmente, verifica–se um deslocamento, para a direita, dos cruzamentos entre as curvas do S3L, os quais definem os pontos a partir dos quais a Compressão (eventualmente combinada com Encriptação) compensa os tempos de propagação consumidos pela utilização simples do S3L.

Em termos qualitativos, o facto de os benefícios da Compressão se revelarem mais tardiamente é facilmente explicável: se considerarmos o tempo consumido na confirmação de entrega pelo TCP como integrado no tempo de propagação desde a origem ao destino, então o UDP goza de tempos de propagação menores; com menores tempos de propagação, só uma Compressão mais efectiva dos dados poderá trazer benefícios; como a Compressão se revela mais eficaz à medida que a dimensão dos dados aumenta[4], compreende–se facilmente o deslocamento em causa.

[4]Para o mesmo tipo — binário ou texto — de dados, bem entendido e que foi fixado para todos os testes realizados.

A conclusão óbvia destas observações é a de que compensa usar as funcionalidades S3L ponto–a–ponto para blocos de dados suficientemente grandes para que a Compressão consiga atenuar, em termos de um menor tempo de propagação, o tempo consumido nos processos de Encriptação e Autenticação[5].

7.3.3 Algoritmos Simétricos

Nos testes até agora apresentados, a Encriptação dos dados baseou–se, invariavelmente, na utilização do algoritmo simétrico IDEA. Tal como demonstra o gráfico da figura 7.5, baseado na tabela 7.5, este algoritmo apresenta um desempenho superior aos restantes algoritmos simétricos oferecidos pelo SecuDE, pelo que se compreende a sua utilização, por omissão, no contexto do S3L[6].

`S3Lwrite+S3Lread`	tempo consumido	carga anterior	carga posterior
`NO_CRYPT`	50963 μs	0.01, 0.03, 0.06	0.01, 0.02, 0.06
`IDEA`	62327 μs	0.15, 0.17, 0.15	0.14, 0.16, 0.15
`DES-ECB`	99459 μs	0.16, 0.12, 0.11	0.21, 0.14, 0.12
`desCBC3`	169549 μs	0.14, 0.13, 0.11	0.25, 0.16, 0.12

Tabela 7.5: Tempo consumido por vários algoritmos simétricos.

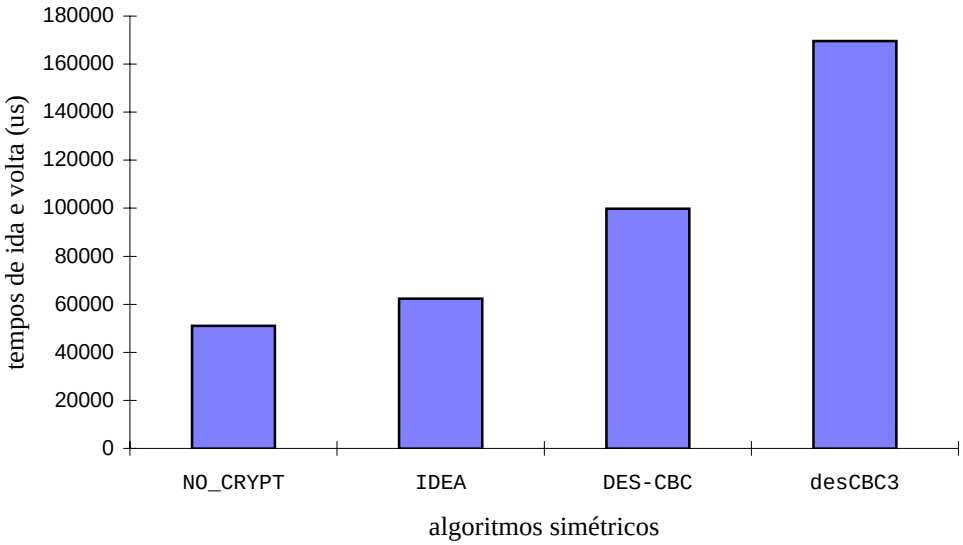

Figura 7.5: Desempenho de vários algoritmos simétricos.

Os valores da tabela 7.5 foram obtidos com base na média dos valores de uma amostra de dimensão 100, para as primitivas `S3Lwrite+S3Lread`, sem Compressão e com blocos de dados de 1K. A designação `NO_CRYPT` refere–se à não utilização de Encriptação sobre os dados. A pequena diferença entre o tempo consumido pelo IDEA e o tempo consumido sem

[5]Note–se que este tempo é, tendencialmente, maior, com o incremento da dimensão dos dados.

[6]Embora seja sempre possível optar pela utilização de qualquer outro, se tal for explicitamente solicitado durante a definição de contextos S3L (ponto–a–ponto e multiponto).

Encriptação, bem como a evolução registada da carga da máquina em ambas as situações, encoraja a utilização do IDEA.

7.3.4 Junções a Grupos S3L Multiponto

O último conjunto de testes realizados visou a variante multiponto do S3L. Em primeiro lugar, foi determinado o tempo consumido no processo de junção a um grupo S3L, supondo o dono do grupo e o candidato a membro situados em máquinas distintas. Para uma amostra de 100 junções, o tempo médio consumido pelo candidato na execução do protocolo de junção[7] foi de 123826 μs. O tempo obtido contrasta com o valor relativo à execução do protocolo Skeyx durante a definição de um contexto ponto–a–ponto (1953320.4 μs, segundo a tabela 7.1), sendo bastante menor. Note–se, todavia, que é um tempo superior ao consumido (7737.7 μs, segundo a tabela 7.1) na definição de um contexto S3L ponto–a–ponto com base em informação preservada em *cache* após a execução prévia do Skeyx.

Resumindo: no S3L ponto–a–ponto consomem–se, em média, aproximadamente 2 s, na criação de uma entrada na *cache* de chaves públicas de Diffie–Hellman; posteriormente, a definição de um contexto S3L ponto–a–ponto consumirá, tão somente, aproximadamente 8 ms (em média), correspondentes à recuperação do contexto da *cache* em disco; no S3L multiponto consomem–se, em média, aproximadamente 124 ms sempre que ocorre (re)junção ao grupo; uma vez efectuada a junção, o contexto multiponto, já em memória, é reutilizável, não havendo, portanto, penalizações de registo na sua definição.

7.3.5 Primitivas Multiponto

Finalmente, efectuou–se a avaliação do desempenho das primitivas de comunicação multiponto. Concretamente, as combinações `sendto+recvfrom` por parte de Berkeley e `S3Lsendto+S3Lrecvfrom` por parte do S3L foram confrontadas, tendo mais uma vez como pano de fundo a variação da dimensão do bloco de dados e a introdução de Compressão e Encriptação. Analogamente a outros testes realizados, a dimensão das amostras foi 100. O protocolo de comunicação usado foi o UDP[8], pelo que a dimensão máxima do bloco de dados foi limitada a 63K.

Para a realização deste teste, executou–se um dono de grupo S3L numa das duas máquinas disponíveis. Nessa máquina, instalou–se também um processo reflector de tráfego *IP Multicast*. Na outra máquina, executou–se o cliente produtor do tráfego original. Ambos os processos inibiram a recepção de tráfego *IP Multicast* por eles gerado a fim de garantir a recepção de datagramas originados apenas no outro processo.

Os tempos medidos apresentam–se na tabela 7.6 e o gráfico correspondente é dado pela figura 7.6. Os resultados obtidos são idênticos aos produzidos sobre UDP para tráfego ponto–a–ponto (ver a tabela 7.4 e a figura 7.4), dado que o grupo constituído representa uma situação simplificada (elementos na mesma LAN) e que, em termos de tempos de propagação e processamento protocolar, não apresenta variações de registo.

[7] Correspondente à execução da função `S3LMjoin_group`.

[8] Para o domínio `AF_INET` apenas *sockets* do tipo `SOCK_DGRAM` e `SOCK_RAW` suportam *IP Multicast*, sendo a utilização de *sockets* de tipo `SOCK_RAW` restrita, como é sabido, ao super–utilizador.

dimensão dos dados	`sendto` + `recvfrom`	`S3Lsendto` + `S3Lrecvfrom`	`S3Lsendto` + `S3Lrecvfrom` c/ Compressão	`S3Lsendto` + `S3Lrecvfrom` c/ Comp. e Encriptação
1K	4280 μs	39853 μs	55885 μs	62191 μs
2K	6721 μs	42490 μs	60058 μs	67144 μs
4K	10373 μs	49337 μs	70036 μs	79802 μs
8K	18500 μs	61752 μs	91238 μs	109176 μs
16K	32551 μs	88389 μs	113891 μs	137746 μs
32K	67629 μs	137215 μs	137564 μs	163986 μs
63K	127389 μs	234435 μs	198559 μs	218086 μs

Tabela 7.6: Latência das primitivas `(S3L)sendto` e `(S3L)recvfrom` multiponto.

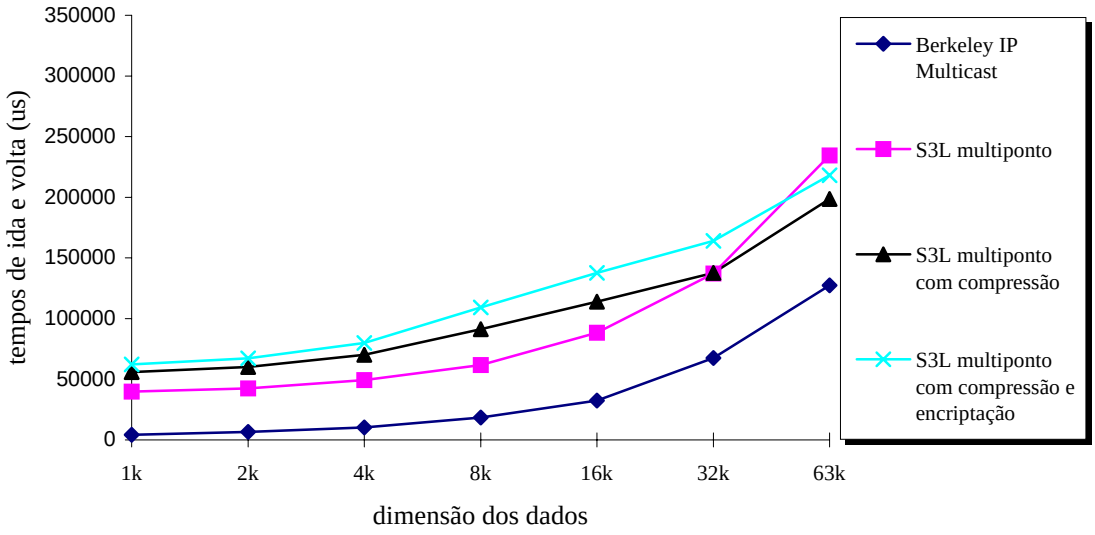

Figura 7.6: Desempenho das primitivas `(S3L)sendto` e `(S3L)recvfrom` multiponto.

As conclusões são, portanto, análogas às produzidas anteriormente num contexto de comunicação ponto–a–ponto: a carga adicional introduzida pelas qualidades de serviço do S3L multiponto torna–se menos significativa, relativamente à utilização das funcionalidades de Berkeley normais, na medida em que a Compressão dos dados cumprir eficazmente o seu papel de reduzir (em termos de tempos de propagação) o tempo adicional consumido nos processos de Autenticação e Encriptação.

Capítulo 8

Conclusões

O desenvolvimento do S3L considera–se oportuno num contexto onde, apesar de já serem conhecidas abordagens à comunicação segura ponto–a–ponto semelhantes[1] (como por exemplo o *Secure Sockets Layer* (SSL) [FKK96]), são poucas as que, em simultâneo, oferecem operação segura de grupos *IP Multicast*.

Precisamente, uma dessas abordagens é o *Simple Key Management for Internet Protocols* (SKIP) [AP95, AMP95c] da Sun, que oferece uma solução transparente para o problema da comunicação segura ponto–a–ponto e multiponto no âmbito da pilha TCP/IP. Conceptualmente, o S3L assimilou alguns elementos fundamentais da operação do SKIP, nomeadamente a utilização do acordo de Diffie–Hellman como ponto de partida para a obtenção de um modelo *stateless* de operação. Todavia, enquanto o SKIP se situa a um nível arquitectural relativamente baixo (na Camada de Rede e integrado no núcleo do Sistema Operativo), o S3L localiza–se acima da interface de programação dos *sockets* de Berkeley. Consequentemente, os mecanismos de operação do S3L acabam por divergir do SKIP, não só em resultado do diferente posicionamento arquitectural de ambos, como também pelo facto de que o S3L assenta em serviços criptográficos de base oferecidos pelo *Secure Development Environment* (SecuDE) [Sch95a].

A face visível do S3L é uma API através da qual os protocolos de comunicação segura ponto–a–ponto e multiponto do S3L são acessíveis. Essa API permite uma migração relativamente expedita de aplicações codificadas segundo o modelo tradicional Cliente–Servidor dos *sockets* de Berkeley para um modelo muito semelhante (na realidade assente sobre o primeiro) capaz de fornecer novas qualidades de serviço no âmbito de uma comunicação segura: Privacidade, Autenticação (incluindo Não–Repudiação), Integridade, Compressão, Sequenciação (contra *ataques–de–repetição*) e Controlo de Acesso (na variante multiponto).

O esquema de operação do S3L apresenta–se, em geral, *stateless*, a menos da necessária execução, pelo menos uma vez, do Skeyx, um protocolo auxiliar de troca de chaves públicas de Diffie–Hellman não certificadas. Este protocolo dispensa a certificação das chaves públicas de Diffie–Hellman, embora dependa da certificação das chaves públicas RSA dos intervenientes de forma a garantir a segurança da troca das chaves públicas de Diffie-Hellman. Essa dependência é, contudo, minimizada: uma entidade comunicante precisa

[1] Relativamente ao posicionamento em termos da Arquitectura do Sistema Operativo e do Modelo de Comunicações e não tanto no que diz respeito aos protocolos e mecanismos próprios de operação.

apenas do certificado da sua chave pública RSA, não havendo necessidade de recuperar previamente, de algum repositório (*e.g.*, a Directoria X.500), o certificado da entidade com quem vai executar o protocolo Skeyx. Evitou–se, assim, um confronto directo com o problema da distribuição de chaves certificadas, mas exige–se que essas chaves sejam certificadas dentro da mesma hierarquia X.509.

Outro aspecto do Skeyx é a necessidade da manutenção de servidores individuais por cada entidade, o que constitui uma abordagem reconhecidamente pouco escalável. Contudo, tal opção deveu–se, fundamentalmente, à disposição da *cache* sob o *Personal Security Environment* (PSE) do SecuDE, auferindo assim das restrições de acesso que exigem a utilização de um *Personal Identification Number* (PIN) de conhecimento restrito a cada entidade e que, portanto, não pode ser usado por um único servidor de chaves públicas de Diffie–Hellman para todas as entidades de uma máquina.

Como aspectos mais positivos do Skeyx poderemos apontar, entre outros:

- o mecanismo de actualizações "semi–transaccional" destinado a minimizar a hipótese de actualizações com diferentes graus de sucesso nas *caches* de duas entidades comunicantes; esse mecanismo é ainda complementado pelas actualizações implícitas;
- a designação "anónima" das entradas da *cache*, baseada em sínteses MD5 dos nomes distintos X.500 das entidades correspondentes;
- o mecanismo de acesso concorrente (com trincos do tipo *advisory*) à *cache*, partilhado por todos os processos que, no contexto do S3L, necessitam de lhe aceder; este mecanismo, combinado com o algorimto de actualização das *caches*, possibilita a manutenção de um estado coerente das mesmas;
- suporte à replicação das *caches*, *i.e.*, na eventualidade da replicação de uma entidade, o mecanismo de actualizações implícitas efectua a convergência das réplicas desde que o seu padrão de contactos seja semelhante.

Porventura, a maior dificuldade na implementação da versão ponto–a–ponto do S3L foi a resolução do problema da Sequenciação de mensagens sem recurso à preservação, para cada origem possível, da última marca temporal/número de sequência válido. Tal era indispensável no sentido de assegurar o carácter *stateless* do S3L. Adicionalmente, perseguia–se uma solução independente da sincronização de relógios dos intervenientes, a qual teria, necessariamente, que assentar num protocolo já de si seguro. Apesar de a solução obtida ser, de longe, mais efectiva que a preconizada pelo SKIP no combate a *ataques–de–repetição* (mais não seja pela reduzida granularidade oferecida pelo S3L), reconhece–se que, na prática, o problema não ficou completamente resolvido porque os critérios de rejeição de mensagens repetidas servem–se de elementos (nomeadamente, tempos de propagação) cuja medição rigorosa é difícil. O recurso a estimativas é, portanto, inevitável, abrindo–se, eventualmente, janelas de ataque.

Na implementação do S3L ponto–a–ponto importa também registar algumas opções cujos efeitos se consideram positivos. Por exemplo, o S3L simplifica o esquema de identificação do SKIP, utilizando apenas sínteses MD5 de nomes distintos para identificar as entidades comunicantes. Para além da sua simplicidade, o anonimato das entidades e a independência da sua localização são algumas das vantagens deste esquema. Adicionalmente, no contexto do S3L é possível gerar mensagens destinadas a armazenamento ou a

outra forma de entrega que não a rede. Essas mensagens são rigorosamente iguais, em termos de aplicação de qualidades de serviço (Privacidade, Autenticação, etc.), às mensagens entregues à rede. Resultante de um pormenor de implementação, a presença do endereço IP de origem numa mensagem S3L ponto–a–ponto (o que não acontece no SKIP) oferece protecção contra ataques do tipo *ip–spoofing*.

Os problemas com a temporalidade e o carácter imprevisível dos atrasos de propagação voltaram a ser sentidos no desenvolvimento do S3L multiponto onde, de novo, o preço a pagar pela manutenção de uma operação essencialmente *stateless* (a menos da (re)junção ao grupo) e pela não necessidade de sincronização de relógios entre membros (potencialmente muito dispersos) de um grupo foi a admissão da operação do grupo com chaves que não constituem, necessariamente, a última *chave–do–grupo* gerada pelo seu dono. A opção assumida (que, basicamente, consiste na manutenção de um *array* circular, de dimensão parametrizável, com um conjunto de chaves obsoletas admissíveis, para além da chave mais recente) revela–se mais resistente a estrangulamentos na actualização da *chave–do–grupo* junto do dono, evitando a paragem da operação do grupo durante o processo de actualização dessa chave. Melhor ainda, livra–se do problema que a utilização de chaves marginalmente obsoletas (*i.e.*, chaves cuja validade expirou quando a transmissão de mensagens que protegiam estava em curso) constitui.

Recordam–se ainda outros aspectos que orientaram a concepção do S3L multiponto:

- o processo de (re)junção é automático, ocorrendo transparentemente, durante a execução de uma "primitiva S3L multiponto" de escrita ou leitura;
- ao contrário do que acontece no SKIP, as Listas de Controle de Acesso operam sobre entidades e não sobre endereços IP;
- é possível, nalguns casos, prevenir a criação de um grupo com base num endereço *IP Multicast* transiente previamente "reservado".

Apesar de submetida a numerosos testes durante o seu desenvolvimento, a implementação actual do S3L carece ainda de uma utilização suficientemente alargada para que se possam produzir aqui conclusões imparciais sobre a sua segurança, robustez e facilidade de utilização. Todavia, em termos de desempenho, os testes realizados permitem afirmar com alguma propriedade que os benefícios obtidos com as qualidades de serviço extra (Privacidade, Autenticação, etc.) ultrapassam a inevitável sobrecarga introduzida pela dependência do S3L de criptografia por *software* (fornecida pelo SecuDE).

Em termos de trabalho futuro, ficam abertas algumas vias cuja concretização permitirá, por um lado, simplificar o modelo de operação do S3L e, por outro, actualizar a sua implementação. Fundamentalmente, perspectiva–se:

1. adopção de um esquema de certificação de chaves públicas de Diffie–Hellman (efectuando, se necessário, extensões ao SecuDE com esse objectivo) com o fim de simplificar o protocolo Skeyx, já que o esquema de *desafio–resposta* usado pretende compensar a ausência dessa certificação; note–se que, a menos da negociação de algoritmos, o Skeyx poderia ser dispensado caso a certificação de chaves públicas de Diffie-Hellman se baseasse numa infra–estrutura distribuída, onde a aquisição (segura) da chave pública de Diffie–Hellman do destinatário teria lugar antes da transmissão

de qualquer pacote S3L; este cenário corresponderia a um mecanismo verdadeiramente *stateless*, sob o ponto de vista das entidades comunicantes, já que não seria difícil conseguir garantias de suporte comum de um conjunto mínimo de algoritmos *standard* (*e.g.* DES, 3DES, IDEA, RC4, MD5, etc.);

2. alternativamente à opção anterior, a evolução do Skeyx para um modelo de operação que, dispensando certificação RSA, permita ainda a troca segura de chaves públicas de Diffie–Hellman; em tal modelo, poderia também ser contemplada a hipótese de concentrar num único servidor por máquina a responsabilidade das trocas Skeyx em nome de todas as entidades dessa máquina (a atribuição de privilégios especiais a este servidor teria, no entanto, que ser bem ponderada, já que se constitui como *single–point–of–failure*);

3. utilização de versões mais recente do SecuDE. Estas versões suportam um leque mais alargado de algoritmos, oferecem certificação X.509v3 e viabilizam a utilização de *threads*;

4. migração da implementação actual do S3L para outra plataformas. A restrição actual ao sistema operativo Linux resulta, fundamentalmente, de ter sido esse o ambiente de desenvolvimento original do S3L. A extensão a outros sistemas operativos da família Unix deverá ser relativamente expedita. Já a migração para a plataforma Windows 95/NT se afigura mais problemática, dado que determinados pormenores de implementação tiram partido de características específicas do UNIX[2].

Para terminar, podemos ainda concluir que a construção de protocolos do tipo dos preconizados pelo S3L é um verdadeiro jogo de compromissos, onde cada opção de desenho e implementação deve ser ponderada, na mira do equilíbrio sempre difícil entre o objectivo prioritário da Segurança e as limitações à sua implementação que inevitavelmente surgem, como bem exemplificam os problemas levantados pelo tratamento, sempre complicado, da Temporalidade.

[2]Neste contexto, é animadora a recente proposta OpenNT da Softway (`http://www.softway.comp/OpenNT/`) que, basicamente, oferece um subsistema Unix sobre o *kernel* do Windows NT.

Apêndice A

Instalação e configuração do S3L

A.1 Instalação

A distribuição actual do S3L (versão *beta* 1.0) contém quatro pacotes de *software*:

1. `s3l-b01.tgz` – ambiente de desenvolvimento S3L;
2. `secude-4.4b0.tgz`[1] – ambiente de desenvolvimento SecuDE;
3. `zlib-1.0.3.tgz` – biblioteca de compressão;
4. `mrouted-3.8.tgz` – demónio `mrouted` e outros utilitários de suporte[2], para *IP Multicast*.

O pacote `secude-4.4b0.tgz` contém uma versão do ambiente de desenvolvimento SecuDE, especialmente modificada[3] a fim de ser possível a distribuição manual dos parâmetros de Diffie–Hellman. Concretamente, essa modificação operou–se sobre o utilitário `secude-4.4.b0/src/util/psemaint.c`, e é visível na figura A.1. A informação fornecida neste manual não dispensa a consulta da documentação que acompanha a distribuição do SecuDE 4.4b0, nomeadamente do volume 2 [Sch95d].

A compilação e a instalação dos pacotes contidos na distribuição do S3L devem ser feitas pelo super–utilizador. Assumindo a directoria `/tmp` como directoria de trabalho, os passos a seguir são:

```
/tmp# tar zxvf s3l-b01-dist.tgz
/tmp# cd s3l-b01-dist
/tmp/s3l-b01-dist# ./make
/tmp/s3l-b01-dist# ./make install
```

A desinstalação e a eliminação do produto da compilação faz–se também através da `Makefile` principal:

[1] A versão 5.0, introduzida muito recentemente, apresenta algumas modificações que a tornam incompatível com a versão 4.4b0, nomeadamente em termos de APIs, utilitários e estrutura de directorias. Essas modificações são, nalguns casos, bastante importantes (*e.g.*, a "eliminação" das variáveis globais, o que permitirá escrever código *multithreaded*), pelo que se prevê a migração futura do S3L para o ambiente SecuDE 5.0.

[2] Ambos pré–compilados para o sistema operativo Linux.

[3] O original pode ser encontrado em `ftp://ftp.darmstadt.gmd.de/pub/secude`.

```
.../secude-4.4.b0/src/util# diff psemaint.c.original psemaint.c.patched
4010a4011,4017
> /* begin DH PATCH */
>                         else if ((dh_parm = d_AlgId(ostr))) {
>                                 oid = af_get_objoid(DHparam_name);
>                                 aux_free_AlgId(&dh_parm);
>                         }
> /* end DH PATCH */
```

Figura A.1: Modificação ao `psemaint.c` original.

```
/tmp/s3l-b01-dist# ./make uninstall
/tmp/s3l-b01-dist# ./make clean
```

Qualquer pacote poderá ser individualmente compilado, instalado, removido ou depurado do produto da sua compilação, através de `Makefile`s específicas localizadas nas directorias de topo de cada um.

A instalação processa–se, por omissão, na subárvore `/usr/local`. Este comportamento pode ser modificado através da configuração dos ficheiros `Makefile` da distribuição. O processo de instalação actualiza as seguintes directorias com os ficheiros discriminados:

1. `/usr/local/lib`: `libs3l.a`, `libsecude.a` e `libz.a`;
2. `/usr/local/include`: `zconf.h` e `zlib.h`;
3. `/usr/local/include/s3l`: `common.h`, `keyx.h`, `skeyx.h` e `s3lsocket.h`;
4. `/usr/local/include/secude`: cópia de `secude-4.4.b0/src/include`;
5. `/usr/local/bin`: algumas aplicações do SecuDE (`cacreate, getpkroot, certify, psecreate, instpkroot, getkey, instcert, pem` e `psemaint`); aplicações do S3L (`s3lforwarder, s3lkeyxserver, s3lkeyxclient, s3ltest, s3lmkdhold, s3lmgowner, s3lmgaclsmaint, s3lmglist` e `s3lmtest`);
6. `/usr/local/sbin`: demónio `mrouted` e utilitários `map-mbone, mrinfo` e `mtrace`;
7. `/usr/local/man/man8`: manuais de suporte aos utilitários *IP Multicast*.

Determinadas aplicações podem ser apenas executadas pelo super–utilizador, como é o caso de `cacreate, getpkroot, certify` e `s3lforwarder`, pelo que as suas permissões são devidamente modificadas. Por omissão, o comando `s3lmgowner` é gerado com *setuid root* a fim de que ainda seja possível a detecção de grupos *IP Multicast* "pré–activos" quando o invocador desse comando não é o super–utilizador. Este comportamento pode ser alterado editando o ficheiro `.../s3l-b01-dist/s3l-b01/src/Makefile` e comentando a linha "`chmod u+s s3lmgowner`".

A.2 Configuração

A.2.1 Configuração do SecuDE

Criação da Root CA

O primeiro passo da configuração relativa ao SecuDE é a criação da Root CA[4], o que deverá ser feito pelo super–utilizador, de preferência na sua conta (ou noutro local do sistema de ficheiros, de acesso igualmente restrito), através da invocação do comando `cacreate`:

```
/root# cacreate -r "C=PT, CN=RootCA, D=CA"
Enter PIN for .ca/.capse:
Reenter PIN for .ca/.capse:
```

O significado dos parâmetros fornecidos ao comando `cacreate` é o seguinte:

- `-r`: a CA a criar é a Root CA (e não uma "CA normal");
- `"C=PT, CN=rootCA, D=ca"`: o nome distinto X.500 da Root CA.

Note–se que é solicitada a introdução (e confirmação) de um PIN[5], com base no qual será derivada uma chave DES usada em futuros acessos à informação sensível preservada no PSE[6] da Root CA. Opcionalmente (opção `-q`) seria possível solicitar a criação de dois pares de chaves assimétricas RSA. Um destinar–se–ia à produção e verificação de assinaturas digitais (par de Assinatura). O outro seria usado exclusivamente em operações de encriptação e desencriptação (par de Encriptação). No contexto do S3L (concretamente do protocolo Skeyx) apenas é necessário um par de chaves RSA, e que desempenhará funções de Assinatura[7]. Por omissão, sempre que se gera um PSE, seja ele de uma CA[8] ou de um utilizador normal, é criado um só par de chaves públicas RSA.

Em relação aos *nomes relativamente distintos* que constituem um *nome distinto* X.501 de um emissor (CA) ou sujeito (utilizador) de um certificado, o SecuDE admite apenas a utilização de um subconjunto dos atributos X.520 válidos: countryName (C), organizationalName (O), organizationalUnitName (OU), surname (S), commonName (CN), localityName (L), stateOrProvinceName (SP), streetAddress (ST), title (T), serialNumber (SN), businessCategory (BC) e description (D), sendo os restantes atributos ignorados. Neste e nos próximos exemplos, os nomes distintos usados são puramente circunstanciais, não tendo qualquer relação com nomes efectivamente registados na Directoria. Adicionalmente, e por questões de ordem prática, usa–se um número mínimo de atributos. Saliente–se porém que o SecuDE contempla a utilização de um mecanismo de aliases caso se opte por nomes distintos bem recheados de atributos.

Com o comando `cacreate`, é também possível criar outras CAs (que não a Root CA), noutras contas ou até noutras máquinas onde o SecuDE também tenha sido instalado[9]. Para simplificar os exemplos fornecidos neste manual, assume–se a existência de uma só

[4] Do inglês *Root Certification Authority*.
[5] Do inglês *Personal Identification Number*.
[6] Do inglês *Personal Security Environment*.
[7] Embora pudesse, se tal se desejasse, efectuar também operações de Encriptação.
[8] O PSE de uma CA é gerado automaticamente durante a execução do comando `cacreate`.
[9] [Sch95d] refere ainda a utilização alternativa dos comandos `gen_ca` e `inst_ca` para CAs geradas directamente por uma Root CA.

CA, que é precisamente a Root CA. As entidades a serem certificadas poderão, eventualmente, residir em outras máquinas que não a da Root CA.

Criação do PSE dos Utilizadores

Uma vez criada a Root CA, é necessário criar o PSE dos utilizadores normais. Na realidade, o SecuDE prevê a possibilidade de haver mais do que um PSE por utilizador, o que se poderia traduzir na associação de diferentes nomes distintos ou pares de chaves assimétricas (RSA ou Diffie–Hellman) a um mesmo utilizador, ou seja, um utilizador poderia usufruir de várias identidades (conferidas pela unicidade dos nomes distintos ou das chaves assimétricas) e portanto, sob esse ponto de vista, um utilizador poderia "albergar" várias entidades (cada qual no seu PSE). Nesta linha, o S3L retira significado ao termo utilizador e assume a entidade como protagonista dos seus protocolos, sendo puramente "acidental" o facto de que a entidade seja suportada, na realidade, por um PSE na conta de um determinado utilizador. Mais uma vez, a fim de simplificar os exemplos de operação a apresentar, optou–se por associar um só PSE a cada utilizador.

A criação de um PSE é feita recorrendo ao comando `psecreate`. Por exemplo, para o utilizador `ruf`, teríamos:

```
/home/ruf# psecreate "C=PT, CN=ruf, D=user"
Enter PIN for .pse:
Reenter PIN for .pse:
```

Mais uma vez, é solicitada a introdução e confirmação de um PIN de protecção do PSE.

Após a criação do PSE de um utilizador, é possível lá encontrar os seguintes objectos (ficheiros) relevantes:

- `SKnew.sf`: chave privada RSA;
- `Cert.sf`: chave pública RSA;
- `PKRoot.sf`: contém, provisoriamente, a mesma chave que `Cert.sf`.

A chave pública `Cert.sf` é criada sob a forma de um protótipo de certificado auto–assinado, que terá de ser submetido a uma CA (à Root CA, no nosso caso), para ser devidamente certificado.

As chaves RSA criadas pelo `psecreate` têm, por omissão, 512 bits. Um outro valor pode ser especificado através da opção `-k nbits`.

Certificação das Chaves Públicas RSA

Para substituir o protótipo auto–assinado da chave pública de um utilizador (`Cert.sf`), por uma versão devidamente certificada pela Root CA, há duas possibilidades, adequadas à coexistência ou não da Root CA e do(s) utilizador(es) a certificar, na mesma máquina.

Certificação na Mesma Máquina Neste caso, as tarefas de certificação devem ser antecedidas da actualização da `PKRoot.sf`. A PKRoot é, recorde–se, a chave pública RSA da Root CA. Para substituir a `PKRoot.sf` no PSE de um utilizador pela genuína `PKRoot.sf` da Root CA é preciso:

1. extraí–la do PSE da Root CA, invocando, como `root`, o comando `getpkroot`:

   ```
   /root# getpkroot pkroot
   ```

2. fazer chegar a chave extraída, `pkroot`, ao utilizador de destino; por exemplo, se o destino fosse o utilizador `ruf`, seria suficiente que ele fizesse:

   ```
   /home/ruf# cp ~root/pkroot .
   ```

3. o utilizador faria então a instalação de `pkroot` no seu PSE, pela invocação do comando `instpkroot`:

   ```
   /home/ruf# instpkroot pkroot
   ```

Uma vez que no exemplo presente simplificado os utilizadores estão directamente subordinados à Root CA, então não são necessários Caminhos de Certificação Directos (FCPaths)[10]. Se assim não fosse, os comandos `getfcpath` e `instfcpath` fariam a extracção e instalação, respectivamente, dos FCPaths das CAs (que não a Root CA) às quais os utilizadores estariam subordinados.

Actualizada a `PKRoot.sf`, seria então possível, ao utilizador, prosseguir com a certificação da sua chave pública. Os passos a seguir deveriam ser:

1. o utilizador (*e.g.*, `ruf`) faz a extracção do protótipo da chave pública `Cert.sf` usando o comando `getkey`:

   ```
   /home/ruf# getkey proto
   ```

2. de seguida, a Root CA adquire o protótipo, fazendo, por exemplo:

   ```
   /root# cp ~ruf/proto .
   ```

3. a Root CA procede então à certificação do protótipo `proto`, com o comando `certify`, gerando o certificado `cert`:

   ```
   /root# certify proto cert
   ```

4. o utilizador adquire o certificado `cert`, fazendo, por exemplo:

   ```
   /home/ruf# cp ~root/cert .
   ```

5. finalmente, o utilizador instala o certificado no seu PSE, substituindo o ficheiro `Cert.sf` por uma nova versão, através da invocação do comando `instcert`:

   ```
   /home/ruf# instcert -H cert
   ```

[10] Recorde–se, do inglês *Forward Certification Paths*.

Certificação via PEM O SecuDE fornece uma implementação do standard *Privacy Enhanced Mail* (PEM) [Lin93a, Ken93, Bal93, Kal93], que permite a troca de certificados através de correio electrónico, adequada portanto quando o emissor de certificados e o(s) sujeito(s) não reside(m) na mesma máquina. A actualização da `PKRoot.sf` (e eventualmente do `FCPath.sf`) acontece no decorrer do próprio processo de certificação, pelo que se dispensa um procedimento prévio de actualização específico.

Os passos essenciais a seguir, neste contexto, são:

1. o utilizador (*e.g.*, `ze`) gera o protótipo do certificado, `certreq.pem`, fazendo:

   ```
   /home/ze# echo "Certification Request" > certreq
   /home/ze# pem mic-clear -C -i certreq -o certreq.pem
   ```

   após o que o envia, por correio electrónico, à autoridade de certificação, Root CA;

2. a Root CA procede à certificação do protótipo recebido, gerando o certificado `cert`:

   ```
   /root# pem certify -i certreq.pem -o cert -c .ca
   ```

   após o que o envia, por correio electrónico, ao utilizador;

3. finalmente, o utilizador procede à instalação do certificado `cert` (e da `PKRoot.sf`) no seu PSE, fazendo:

   ```
   /home/ze# pem -i cert -o dummyfile
   ```

Obviamente, o processo de certificação não se pode basear exclusivamente em trocas de correio electrónico, dada a vulnerabilidade desse método a ataques, por exemplo, do tipo *homem–no–meio*. Assim, adicionalmente à transmissão de um pedido de certificação PEM *via* correio electrónico, é comum o envio de uma segunda cópia, *fora–de–banda* (através de fax ou carta, por exemplo), na qual conste, impressa, pelo menos a assinatura do sujeito do certificado. A reposta, contendo o certificado e a chave pública da Root CA, é também objecto dos mesmos cuidados.

Geração das Chaves de Diffie–Hellman

O Skeyx é um protocolo de suporte do S3L que efectua a troca de chaves públicas de Diffie-Hellman (não certificadas) entre duas entidades. Essa chaves supõem–se geradas com base nos mesmos parâmetros públicos[11] de Diffie–Hellman. Caso contrário, o acordo de Diffie-Hellman, que produz uma chave acordada na qual se baseiam os protocolos do S3L, não é possível.

Assim, é necessário providenciar para que os parâmetros públicos de Diffie–Hellman sejam comunicados às entidades interessadas antes de estas gerarem as suas chaves pública e privada de Diffie–Hellman.

Suponha–se que os parâmetros foram gerados na Root CA e depositados no ficheiro `DHparam.sf` do seu PSE, para o que foi necessário fazer:

[11] Um número primo grande p e um número pequeno g tal que g é uma raiz primitiva de p.

```
/root# psemaint -c .ca
Enter PIN for .ca/.capse:
PSE .ca/.capse> dhinit
Length of prime modulus p in bits: 512
Length of private values x in bits: 512
Generating prime p and base g ...
PSE .ca/.capse> chpin
Enter New PIN:
Reenter New PIN:
PSE .ca/.capse> quit
```

O último passo destinou–se a remover o PIN de acesso ao PSE a fim de que seja agora possível extrair uma versão não cifrada de `DHparam.sf`:

```
/root# psemaint -c .ca
PSE .ca/.capse> read DHparam DHparam.root
PSE .ca/.capse> chpin
Enter New PIN:
Reenter New PIN:
PSE .ca/.capse> quit
```

Note–se que após extrair os parâmetros para o ficheiro `DHparam.root`, teve–se o cuidado de repôr de novo um PIN não vazio de acesso ao PSE da Root CA. De seguida, o ficheiro `DHparam.root` será distribuído pelos utilizadores interessados, o que deve ser feito recorrendo às facilidades PEM do SecuDE a fim de que os destinatários possam comprovar a autenticidade dos parâmetros recebidos:

```
/root# uuencode DHparam.root DHparam.root > DHparam.root.uue
/root# pem -c .ca mic-clear -C -i DHparam.root.uue -o DHparam.root.uue.pem

... DHparam.root.uue.pem enviado por correio electrónico para ze ...

/home/ze# pem -i DHparam.root.uue.pem -o DHparam.root.uue
Enter PIN for .pse:


Message signed by <C=PT, CN=RootCA, D=CA>
MIC Validated with new PKRoot
A new valid signature key of <C=PT, CN=RootCA, D=CA>
with fingerprint 'DA36 91EA C547 E3F9 3BDB 8CA7 4C6A B029 ' found, signed by
<C=PT, CN=RootCA, D=CA>.
Save into PKList (y/n) ? n
/home/ze# uudecode DHparam.root.uue
```

Neste ponto, o utilizador `ze` possui o ficheiro `DHparam.root`, que deve integrar no seu PSE:

```
/home/ze/# psemaint
Enter PIN for .pse:
PSE .pse> write
Enter name of object on .pse: DHparam
File: DHparam.root
PSE .pse> quit
```

As chaves de Diffie–Hellman (`DHprvkey` e `DHpubkey`) podem ser agora geradas, com base nos parâmetros recém–adquiridos. Ainda para o utilizador `ze` teríamos:

```
/home/ze# cd ~/.pse
/home/ze/.pse# psemaint
PSE .pse> dh1
Use DH Parameter of default PSE object, peers public value or create
temporary one? (d,p,t): d
Use dhWithCommonModulus OID (without DH parameters)? (y/n): n
Name or ref of private DH key: DHprvkey
PSE object DHprvkey created
File for own public DH key y: DHpubkey
PSE .pse> quit
```

Note–se que o Skeyx espera encontrar `DHpubkey` no PSE. Daí o cuidado (`cd ~/.pse`) em garantir que a invocação do `psemaint` seja feita no interior do PSE, para que a gravação de `DHpubkey` aí ocorra.

A.2.2 Configuração do S3L

Instalação do Reencaminhador

O demónio Reencaminhador, baseado na aplicação `s3lforwarder`, deverá estar permanentemente em execução, o que pode ser garantido, por exemplo, pela sua inclusão nas *scripts* de arranque da máquina. Em algumas versões do Linux (*e.g.*, Slackware), bastará fazer:

```
/root# echo "/usr/local/bin/s3lforwarder" >> /etc/rc.d/rc.local
```

Criação da Cache `.dhcache`

Cada utilizador é responsável por criar, no seu PSE, uma subdirectoria `.dhcache`. Nessa directoria, residirá a *cache* de chaves públicas de Diffie–Hellman de outros utilizadores, obtidas pela execução do protocolo Skeyx. Por exemplo, o utilizador `ze`, deverá, após a criação do seu PSE, fazer:

```
/home/ze/.pse# mkdir .dhcache
```

Criação da Directoria `.acls`

As extensões multiponto do S3L exigem a criação de mais uma subdirectoria do PSE, para o caso de a entidade/utilizador em questão ser o "dono" de um (ou mais) grupos. Na subdirectoria `.acls` residirão as listas de controle de acesso para um ou mais grupos *IP Multicast*. Por exemplo, o utilizador `ruf`, deverá, após a criação do seu PSE e da subdirectoria `.dhcache`, fazer:

```
/home/ruf/.pse# mkdir .acls
```

A.2.3 Configuração do IP Multicast

Nesta secção descrevem–se os passos necessários à configuração do suporte para *IP Multicast* numa máquina com o sistema operativo Linux[12]. À excepção dos detalhes de compilação do *kernel*, os restantes passos aplicam–se, de forma geral, a outros sistemas operativos da família Unix.

1. Compilar uma versão estável do *kernel* do Linux com suporte para *IP Multicast* (recomenda–se a versão 2.0.30). Durante a configuração do *kernel*, devem ser activadas as seguintes opções:

   ```
   Code maturity level options
   [*] Prompt for development and/or incomplete code/drivers
   ...
   Networking options
   [*] IP: forwarding/gatewaying
   [*] IP: multicasting
   <*> IP: tunneling
   [*] IP: multicast routing (EXPERIMENTAL)
   ```

   As opções `forwarding/gatewaying, tunneling` e `multicast routing` devem ser activadas independentemente do facto de na máquina em questão se pretender executar o demónio `mrouted`. Caso contrário, a aplicação `s3lmgowner` não poderá verificar se o grupo *IP Multicast* fornecido como parâmetro já se encontra activo.

   Na fase de arranque, o novo *kernel* deverá produzir a mensagem:

   ```
   IP Protocols: IGMP, ICMP, UDP, TCP, IPIP
   Linux IP multicast router 0.06.
   ```

2. Adicionar a rota, por omissão, para o tráfego *IP Multicast*. Por exemplo, para a interface `eth0`, teremos:

   ```
   /root# route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0
   ```

   A adição deste comando à *script* `/etc/rc.d/rc.local` garante a sua execução na fase de arranque da máquina.

3. A configuração anterior (passos 1 e 2) é suficiente para permitir a formação e operação de grupos locais, *i.e.*, restritos à mesma LAN[13].

   Para grupos cujos elementos se dispersam, potencialmente, através de várias LANs, é necessário garantir o encaminhamento do tráfego *IP Multicast* correspondente a esses grupos. Na ausência de um encaminhador[14] em *hardware* com suporte para *IP Multicast*, elege–se uma máquina, em cada LAN, para executar o demónio `mrouted` e encaminhar o tráfego *IP Multicast* através de túneis IPIP.

   A especificação dos túneis e de outros parâmetros necessários à operação do `mrouted` faz–se através do ficheiro `/etc/mrouted.conf`, cuja estrutura se encontra descrita no manual do `mrouted`. Por exemplo, a linha

[12] Para uma explicação mais detalhada, consultar `http://www.teksouth.com/linux/multicast`.
[13] Neste contexto, LAN é um conjunto de máquinas que partilham o mesmo endereço IP de rede.
[14] Do inglês *router*.

```
tunnel 193.136.8.7 193.136.195.220 threshold 1 rate_limit 64
```

descreve, em `/etc/mrouted.conf`, um túnel IPIP da máquina local `193.136.8.7` para a máquina remota `193.136.195.220`, sendo `threshold 1` o `ttl` (*time–to–live*) mínimo dos pacotes para poderem ser encaminhados pelo túnel, a uma taxa não superior a 64 Kbits/s (`rate_limit 64`). Reciprocamente, espera–se que na máquina `193.136.195.220`, o túnel esteja configurado de forma idêntica, mas com `193.136.8.7` como extremo remoto do túnel.

Obviamente, numa máquina destinada a executar o `mrouted`, recomenda–se a sua adição às *scripts* de arranque do sistema operativo. No caso concreto do Linux,

```
/root# echo "/usr/local/sbin/mrouted &" >> /etc/rc.d/rc.local
```

seria suficiente para resolver o problema.

Apêndice B

Utilização e programação do S3L

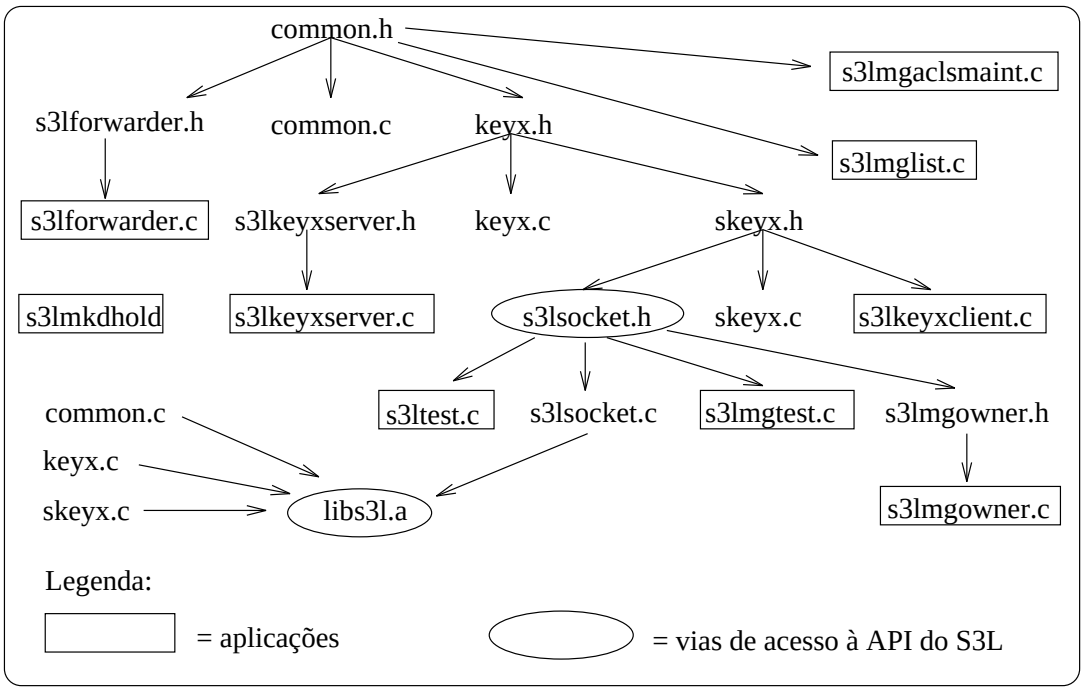

Figura B.1: Hierarquia de dependências para o S3L *beta* 1.

A figura B.1 representa a hierarquia de dependências entre os vários ficheiros que constituem o código fonte do pacote `s3l-b01.tgz`. Sob o ponto de vista do utilizador, o produto da sua compilação divide–se em duas categorias fundamentais:

1. aplicações (demónios servidores e diversos comandos de gestão);
2. *header files* e bibliotecas de acesso à API do S3L.

B.1 Aplicações

O processo de instalação do S3L disponibiliza (em `/usr/local/bin`, por omissão) um conjunto básico de comandos e aplicações, que se apresentam nesta secção, num formato similar ao dos manuais *on–line* dos sistemas Unix.

NOTA IMPORTANTE: A fim de assegurar a consistência da sua *cache* `.dhcache`, cada utilizador deve respeitar algumas regras fundamentais relativas à execução simultânea de determinadas actividades/aplicações:

- a execução de operações de manutenção do PSE (através do comando `psemaint` fornecido pelo SecuDE, por exemplo) é incompatível com a execução, em paralelo, das aplicações `s3lkeyxserver`, `s3lkeyxclient` e `s3lmkdhold` e de qualquer outra que aceda à *cache*;

- uma operação de manutenção do PSE que se traduza na mudança do nome distinto, de qualquer chave pública ou privada — RSA ou Diffie–Hellman —, dos parâmetros de Diffie–Hellman, do PIN, ou de outros elementos específicos da entidade em questão, implica a execução posterior do comando `s3lmkdhold`, antes de executar quaisquer outras aplicações que necessitem de aceder à *cache* (*e.g.*, `s3lkeyxserver`,
  `s3lkeyxclient`, etc.); a obsolescência da *cache* será automática e progressivamente eliminada, pelo mecanismo de execuções implícitas do protocolo Skeyx de que o S3L dispõe;

- a execução do comando `s3lmkdhold` é incompatível com a execução simultânea de qualquer outra aplicação que necessite de acesso à *cache* (*e.g.*, `s3lkeyxserver`, `s3lkeyxclient`, etc.).

B.1.1 Aplicações Genéricas

Nome: `s3lforwarder` – demónio reencaminhador de pedidos Skeyx e de pedidos de
`s3lforwarder`
junção a um subgrupo S3L

Sinopse:

```
s3lforwarder [-d]
```

Descrição:

O demónio `s3lforwarder` é um reencaminhador de pedidos de execução do protocolo Skeyx dirigidos a um determinado servidor `s3lkeyxserver` na máquina local. Complementarmente, o demónio `s3lforwarder` também efectua o redireccionamento de pedidos de junção a um subgrupo S3L para o demónio `s3lmgowner` local apropriado.

A recepção de pedidos Skeyx e de junção faz–se no porto 7571 e o seu reencaminhamento efectua–se através de um *socket* de domínio Unix, específico para cada servidor, e localizado na directoria `/tmp`.

Sem quaisquer opções de invocação, o demónio `s3lforwarder` registará quaisquer mensagens de erro em `/var/adm/syslog` e quaisquer mensagens de informação em `/var/adm/messages`.

A fim de terminar graciosamente, o demónio `s3lforwarder` deve ser eliminado através dos sinais `SIGTERM` ou `SIGINT`.

Opções: `-d` (*debug*):
o demónio `s3lforwarder` permanece ligado ao terminal de invocação, o qual receberá todas as mensagens produzidas internamente.

Ver também:
`s3lkeyxserver`, `s3lkeyxclient`, `s3lmgowner`

B.1.2 Aplicações Skeyx

Nome: `s3lkeyxserver` – demónio servidor de pedidos Skeyx `s3lkeyxserver`

Sinopse:
`s3lkeyxserver [-d]`

Descrição:
O demónio `s3lkeyxserver` é um servidor que atende pedidos de troca de chaves públicas de Diffie–Hellman, baseados na execução do protocolo Skeyx. Os pedidos são originalmente recebidos pelo demónio `s3lforwarder` que efectua o seu reencaminhamento para o demónio `s3lkeyxserver` apropriado.

Invocado sem quaisquer parâmetros, o demónio `s3lkeyxserver` registará eventuais mensagens de erro em `/var/adm/syslog` e eventuais mensagens de informação em `/var/adm/messages`.

Cada utilizador é responsável pela manutenção em execução do seu demónio `s3lkeyxserver`, após a criação da *cache* `.dhcache` no seu PSE.

A terminação graciosa do demónio `s3lkeyxserver` faz–se à custa dos sinais `SIGTERM` ou `SIGINT`.

Opções: `-d` (*debug*):
o demónio `s3lkeyxserver` permanece ligado ao terminal de invocação, o qual receberá todas as mensagens produzidas internamente.

Ver também:
`s3lforwarder`, `s3lkeyxclient`, `s3lmkdhold`

Nome: `s3lkeyxclient` – cliente de pedidos Skeyx `s3lkeyxclient`

Sinopse:
`s3lkeyxclient server_dname server_addr [-d]`

Descrição:
O comando `s3lkeyxclient` permite executar explicitamente o protocolo Skeyx com uma outra entidade, resultando na actualização das *caches* `.dhcache` de ambas com as versões mais recentes das chaves públicas de Diffie-Hellman respectivas.

Opções: `server_dname`:
nome distinto X.500 da entidade cujo servidor se vai contactar; exemplo: "`C=pt, CN=ruf, D=user`";
`server_addr`:
endereço IP da máquina onde reside o servidor a contactar;
`-d` (*debug*):
modo de depuração.

Ver também:
`s3lforwarder`, `s3lkeyxserver`, `s3lmkdhold`

Nome: `s3lmkdhold` – *script* de invalidação da *cache* `.dhcache` `s3lmkdhold`

Sinopse:

```
s3lmkdhold
```

Descrição:
A *script* `s3lmkdhold` torna obsoletas (pela adição da extensão `.old`) todas as chaves acordadas de Diffie–Hellman (ficheiros designados de `DHagreedkey`) mantidas na *cache* `.dhcache` do utilizador que a invoca.

Ver também:
`s3lforwarder`, `s3lkeyxserver`

B.1.3 Aplicações S3L Ponto–a–Ponto

Name: `s3ltest` – aplicação de teste do S3L ponto–a–ponto `s3ltest`

Sinopse:

```
s3ltest -s write_type proto_type file receiver_dname receiver_addr
          receiver_port [-d] [-z] [-D delta] [-cbc alg] [-sym alg]
          [-mac alg]

s3ltest -r read_type proto_type file receiver_port [-d]
```

Descrição:
A aplicação `s3ltest` transfere um ficheiro entre um cliente e um servidor, permitindo testar todas as combinações possíveis entre as funções de escrita e leitura do S3L ponto–a–ponto, conjuntamente com todas as combinações possíveis das diversas qualidades de serviço.

A execução de `s3ltest` só deve ser feita após a criação do PSE e da *cache* `.dhcache` das entidades que constituem o cliente e o servidor, bem como do lançamento dos respectivos demónios `s3lkeyxserver`. O demónio `s3lforwarder` deve estar também em execução nas máquinas do cliente e do servidor.

Opções: `-s` (*sender*):
cliente;
`-r` (*receiver*):
servidor;

`write_type`:
: `write`, `send`, `sendto`, `writev`, `sendmsg` (primitivas de escrita de Berkeley cuja versão S3L se deve usar);

`read_type`:
: `read`, `recv`, `recvfrom`, `readv`, `recvmsg` (primitivas de leitura de Berkeley cuja versão S3L se deve usar);

`proto_type`:
: `tcp` ou `udp` (determina a utilização de *sockets* do tipo `SOCK_STREAM` ou `SOCK_DGRAM`);

`file`: nome do ficheiro a transferir ou a gravar, conforme o caso;

`receiver_addr`:
: endereço IP da máquina do servidor;

`receiver_port`:
: porto onde o servidor fornece o serviço;

`-d`(*debug*):
: modo de depuração.

`-z`: não aplicar compressão; por omissão aplica–se compressão fornecida pela biblioteca `zlib`;

`-D delta`:
: indicação do tempo de vida `delta`, em microsegundos, da mensagem S3L, para efeitos de protecção contra *ataques–de–repetição*; por omissão, o tempo de vida é infinito;

`-cbc alg`:
: algoritmo simétrico da categoria CBC a usar para cifrar a chave `Kp` com a chave `Kijn`; valores possíveis para `alg` são: `DES-CBC`, `desCBC_pad`, `desCBC3_pad`, `DES-CBC-ISO0`, `DES-CBC-ISO1`, `DES3-CBC-ISO0` e `DES3-CBC-ISO1`;

`-sym alg`:
: algoritmo simétrico a usar para cifrar os dados com a chave `Kp`; por omissão utiliza–se o `IDEA`; valores possíveis para `alg` são: `DES-ECB`, `DES-CBC`, `DES-EDE`, `desCBC_pad`, `desCBC3`, `desCBC3_pad`, `IDEA`, `DES-CBC-ISO0`, `DES-CBC-ISO1`, `DES3-CBC-ISO0` e `DES3-CBC-ISO1`;

`-mac alg`:
: algoritmo simétrico a usar em conjunção com o algoritmo de produção de sínteses `MD5` a fim de gerar o MAC dos dados; por omissão utiliza–se o `IDEA`; os valores possíveis para `alg` são os mesmos que para a opção `-sym`.

Ver também:
: `s3lforwarder, s3lkeyxserver, s3lkeyxclient`

B.1.4 Aplicações S3L Multiponto

Nome: `s3lmgowner` – demónio dono de um grupo S3L `s3lmgowner`

Sinopse:

```
s3lmgowner -mip mipaddr [-gt nsec] [-gikt nsec] [-gikf file]
                        [-kcis nsecs] [-kcib nbytes] [-cbc alg]
                        [-sym alg] [-mac alg] [-z] [-d] [-b] [-h]
```

Descrição:

O demónio `s3lmgowner` (acrónimo de *S3L Multicast Group Owner*) é um servidor que atende pedidos de junção a um subgrupo S3L de um grupo *IP Multicast*. Os pedidos são originalmente recebidos pelo demónio `s3lforwarder` que efectua o seu reencaminhamento para o demónio `s3lmgowner` apropriado.

O demónio `s3lmgowner` define as qualidades de serviço que todos os membros devem respeitar enquanto produtores e consumidores de tráfego relativo ao subgrupo S3L. Essas qualidades de serviço são indicadas a `s3lmgowner` através da linha de comando.

Cada utilizador pode executar quantos demónios `s3lmgowner` desejar, desde que cada instância tenha a seu cargo a gestão de um grupo S3L com um endereço *IP Multicast* diferente. Esta restrição de base pode ser complementada com a utilização da opção `-b`.

Previamente à execução de um demónio `s3lmgowner`, deve ser criada, através da aplicação `s3lmgaclsmaint`, a Lista de Controle de Acesso relativa ao grupo que se vai gerir.

A terminação graciosa do demónio `s3lmgowner` pode ser conseguida à custa dos sinais `SIGTERM` ou `SIGINT`.

Opções:

`-mip mipaddr`:
: endereço *IP Multicast* sobre o qual será constituído o subgrupo S3L; `mipaddr` deverá ser um endereço transiente, na gama [`225.0.0.0`, `239.255.255.255`];

`-gt nsec`:
: tempo de vida do demónio `s3lmgowner`, em segundos; um valor zero (0) para `nsec` indica um tempo de vida infinito, sendo esse o valor utilizado por omissão;

`-gikt nsec`:
: tempo de vida de uma chave `GIK` (*Group Interchange Key*), em segundos, após o qual será gerada uma nova chave; um valor zero (0) para `nsec` indica um tempo de vida infinito, sendo esse o valor utilizado por omissão;

`-gikf file`:
: ficheiro onde se encontra a próxima `GIK` a usar; constitui uma alternativa à geração aleatória das `GIK`s pelo próprio demónio `s3lmgowner`;

`-kcis nsec`:
: tempo de vida, em segundos, de qualquer chave `Kp` usada, por um membro, para cifrar os dados enviados para o grupo; após `nsec`, deve ser

gerada, aleatoriamente, uma nova chave `Kp`; um valor zero (0) para `nsec` indica um tempo de vida infinito, sendo esse o valor utilizado por omissão;

`-kcib nbytes:`
tempo de vida, em bytes, de qualquer chave `Kp` usada, por um membro, para cifrar os dados enviados para o grupo; após `nbytes` cifrados com a `Kp` actual, deve ser gerada, aleatoriamente, uma nova chave `Kp`; um valor zero (0) para `nbytes` indica um tempo de vida infinito, sendo esse o valor utilizado por omissão; é gerada uma nova `Kp` sempre que o mínimo dos valores indicados nas opções `-kcis` e `-kcib` é atingido;

`-cbc alg:`
algoritmo simétrico da categoria CBC a usar, nos membros do grupo, para cifrar a chave `Kp` com a `n`'ésima versão da `GIK` actualmente em uso (`n` é um campo de 32 bits, das mensagens S3L multiponto); valores possíveis para `alg` são: `DES-CBC`, `desCBC_pad`, `desCBC3_pad`, `DES-CBC-ISO0`,
`DES-CBC-ISO1`, `DES3-CBC-ISO0` e `DES3-CBC-ISO1`; por omissão, utiliza-se o `DES-CBC`;

`-sym alg:`
algoritmo simétrico a usar, nos membros do grupo para cifrar os dados de uma mensagem S3L multiponto com a chave `Kp`; valores possíveis para `alg` são: `DES-ECB`, `DES-CBC`, `DES-EDE`, `desCBC_pad`, `desCBC3`, `desCBC3_pad`, `IDEA`, `DES-CBC-ISO0`, `DES-CBC-ISO1`, `DES3-CBC-ISO0` e `DES3-CBC-ISO1`; por omissão utiliza–se o `IDEA`;

`-mac alg:`
algoritmo simétrico a usar, nos membros do grupo, em conjunção com o algoritmo de produção de sínteses `MD5`, a fim de gerar o MAC de uma mensagem S3L multiponto; os valores possíveis para `alg` são os mesmos que para a opção `-sym`; por omissão utiliza–se o `IDEA`;

`-z:` não aplicar compressão; por omissão aplica–se compressão fornecida pela biblioteca `zlib`;

`-d` (*debug*):
o demónio `s3lmgowner` permanece ligado ao terminal de invocação, o qual receberá todas as mensagens produzidas internamente; sem esta opção, eventuais mensagens de erro serão registadas em `/var/adm/syslog` e eventuais mensagens de informação em `/var/adm/messages`;

`-b:` efectuar, no contexto da rede local, a detecção da utilização prévia do endereço *IP Multicast* fornecido na opção `-mip`;

`-h:` apresentar um resumo do significado das opções.

Ver também:
`s3lforwarder`, `s3lmgaclsmaint`, `s3lmglist`, `s3lmtest`

Nome: `s3lmgaclsmaint` – aplicação de gestão de listas de controlo de acesso `s3lmgaclsmaint`

Sinopse:

```
s3lmgaclsmaint [-a mipaddr [-m X.500id | -f file]] |
               [-r mipaddr [-m X.500Id | -f file]] |
               [-l [mipaddr [-m X.500id | -f file]]] |
               [-h]
```

Descrição:

A aplicação `s3lmgaclsmaint` (acrónimo de *S3L Multicast Group Access Control Lists Maintenance*) possibilita a execução de tarefas de manutenção sobre a estrutura e conteúdo das várias listas de controlo de acesso que uma entidade possui com o propósito de filtrar a junção de novos membros aos grupos S3L que gere (através dos demónios `s3lmgowner` respectivos).

A aplicação `s3lmgaclsmaint` pode executar em simultâneo com a aplicação `s3lmglist` e com os vários demónios `s3lmgowner` eventualmente em execução. A coerência interna das listas de controle de acesso é preservada graças à utilização, por parte destas aplicações, de um mecanismo de trincos no acesso às listas de controle de acesso. Todavia, num determinado instante, um utilizador pode executar apenas uma instância da aplicação `s3lmgaclsmaint`.

Opções:

`-a mipaddr`:
: cria uma directoria vazia, identificada pelo endereço *IP Multicast* `mipaddr`, na subárvore `.acls` do PSE; nessa directoria residirá a ACL (ficheiro `acl`) do grupo `mipaddr` e o seu historial de junção (ficheiro `members`);

`-a mipaddr -m X.500id`:
: adiciona o nome distinto `X.500id` à ACL do grupo `mipaddr`; a directoria `mipaddr` e a ACL são criadas se não existirem;

`-a mipaddr -f file`:
: adiciona a lista de nomes distintos contida no ficheiro `file` (um nome por linha) à ACL do grupo `mipaddr`; a directoria `mipaddr` e a ACL são criadas se não existirem;

`-r mipaddr`:
: remove a subdirectoria `mipaddr` de `.acls`, e com ela a ACL e o historial do grupo `mipaddr`;

`-r mipaddr -m X.500id`:
: remove o nome distinto `X.500id` da ACL do grupo `mipaddr`;

`-r mipaddr -m X.500id`:
: remove todos os nomes distintos que constam do ficheiro `file` (um nome por linha) da ACL do grupo `mipaddr`;

`-l`:
: listagem do conteúdo de todas as ACLs preservadas na subárvore `.acls` do PSE;

`-l mipaddr`:
: listagem do conteúdo da ACL do grupo `mipaddr`;

`-l mipaddr -m X.500id`:
: verifica se o nome distinto `X.500id` consta da ACL do grupo `mipaddr`;

`-l mipaddr -f file`:
: verifica se todos os nomes distintos contidos no ficheiro `file` constam da ACL do grupo `mipaddr`;

`-h`:
: apresenta um resumo do significado das opções.

Ver também:
: `s3lmglist`, `s3lmgowner`

Nome: `s3lmglist` – aplicação de consulta ao historial de junção `s3lmglist`

Sinopse:

```
s3lmglist [mipaddr [-m X.500Id | -f file]] |
          [-h]
```

Descrição:

A aplicação `s3lmglist` (acrónimo de *S3L Multicast Group List*) permite consultar o historial de junção aos grupos com entrada (subdirectoria) na subárvore `.acls` do PSE.

Cada entrada de um grupo em `.acls` conserva dois ficheiros: `acl`, que é a ACL para esse grupo e `members`, onde reside o historial de junção ao grupo. O abandono do grupo S3L não é registado, efectuando–se silenciosamente, à semelhança do abandono de um grupo *IP Multicast*.

A aplicação `s3lmglist` pode executar em simultâneo com a aplicação `s3lmgaclsmaint` e com os vários demónios `s3lmgowner` eventualmente em execução. Num determinado instante, um utilizador pode executar várias instâncias da aplicação `s3lmglist`, uma vez que esta só efectua operações de leitura.

Sem opções de invocação, `s3lmglist` fornece o historial de junção de todos os grupos com entrada na subárvore `.acls` do PSE.

Opções: `mipaddr`:
: listagem do historial de junção do grupo `mipaddr`;

`mipaddr -m X.500id`:
: verifica se o nome distinto `X.500id` consta do historial de junção do grupo `mipaddr`;

`mipaddr -f file`:
: verifica quais os nomes distintos contidos no ficheiro `file` que constam do historial de junção do grupo `mipaddr`;

`-h`:
: apresenta um resumo do significado das opções.

Ver também:
: `s3lmgaclsmaint`, `s3lmgowner`

Name: `s3lmtest` – aplicação de teste do S3L multiponto `s3lmtest`

Sinopse:

```
s3lmtest -s gowner_dname gowner_ipaddr mipaddr port [-ttl n]
            [-D delta][-l] [-d]

s3lmtest -r gowner_dname gowner_ipaddr mipaddr port [-s] [-d]
```

Descrição:

A aplicação `s3lmtest` demonstra as funcionalidades do S3L multiponto, com base num pequeno exemplo que se divide em duas componentes: um servidor de tempo e o respectivo cliente. O servidor transmite a data e hora actuais encapsulados em mensagens S3L multiponto que os clientes recuperam auscultando o tráfego *IP Multicast* do grupo criado para o efeito.

A execução da aplicação `s3lmtest` só deve ser efectuada depois da definição e execução de um dono para o grupo (ver o manual do demónio `s3lmgowner`). As entidades subjacentes ao demónio `s3lmgowner`, à variante servidor e à variante cliente da aplicação `s3lmtest` são mutuamente independentes.

Opções: `-s` (*sender*):
servidor;

`-r` (*receiver*):
cliente;

`gowner_dname`:
nome distinto X.500 do dono do grupo;

`gowner_ipaddr`:
endereço IP da máquina onde reside o dono do grupo;

`mipaddr`:
endereço *IP Multicast* do grupo que o dono <`gowner_dname`, `gowner_ipaddr`> gere;

`port`: porto através do qual o servidor dissemina a informação e onde os clientes a receberão, pelo que <`mipaddr`, `port`> identifica, univocamente, o ponto de acesso ao serviço fornecido pelo servidor;

`-ttl n`:
tempo de vida (*time–to–live*), do tipo *hop–count*, dos pacotes *IP Multicast* emitidos pelo servidor ; por omissão, o valor de `n` é 1, ou seja, os pacotes *IP Multicast* não são encaminhados para fora da rede local; dimensionando `n`, pode–se controlar o "raio de propagação" dos pacotes *IP Multicast* gerados pelo servidor;

`-D delta`:
indicação do tempo de vida `delta`, em microsegundos, das mensagens S3L multiponto, para efeitos de protecção contra *ataques–de–repetição*; por omissão, `delta` é zero, ou seja, o tempo de vida é infinito; recomenda–se a utilização de `delta` com valores diferentes de zero (positivos) apenas quando servidor e clientes se localizam na mesma máquina ou na mesma rede local e desde que os relógios das máquinas intervenientes se encontrem "sincronizados";

`-l` (*loopback off*):
impede o *loopback* de pacotes *IP Multicast* gerados pelo servidor; não se deve usar se, na máquina do servidor, existem outros processos (clientes, por exemplo) associados ao grupo *IP Multicast* `mipaddr`;

`-s` (*share on*):
activa a partilha do porto `port` com o fim de permitir que outros processos, na mesma máquina que o cliente, possam também receber tráfego

do grupo `mipaddr`; a partilha explícita do porto `port` é obrigatória em todos os processos que, numa dada máquina, escutam o tráfego do grupo `mipaddr` nesse porto;

`-d`(*debug on*):
modo de depuração.

Ver também:
`s3ltest, s3lmgowner`

B.2 Programação de Aplicações com a API do S3L

As estruturas de dados e funções disponibilizadas pela API do S3L organizam–se de uma forma natural em grupos lógicos, determinados pelo tipo de tarefas específicas que desempenham.

Esses grupos são apresentados de seguida, terminando a descrição da API com a apresentação de exemplos, devidamente comentados, que demonstram a metodologia a seguir na programação de aplicações com a API do S3L.

B.2.1 Constantes e Tipos Fundamentais

Ao longo da hierarquia apresentada na figura B.1, são definidas diversas constantes e tipos. Apesar de não se encontrarem explicitamente em `s3lsocket.h` (uma vez que são necessárias previamente), algumas dessas definições devem ser do conhecimento do utilizador da API do S3L:

```
/*********************************
definicoes extraidas de common.h
*********************************/

/* tipo de dados ``booleano''*/
typedef int S3Lbool;

#define S3LTRUE 1
#define S3LFALSE 0
#define S3LABORT -1

/* valores possiveis de um "name space identifier" */
#define NSID_X500 0x00
#define NSID_MD5X500 0x01

/* sobrecarga induzida pelo cabecalho de uma mensagem S3L */
#define S3L_HEADER_SIZE 113

/* variavel global que controla a activacao do debug do SecuDE */
S3Lbool verbose;

/*********************************
definicoes extraidas de keyx.h
*********************************/
```

```
/* identificadores validos para os algoritmos simetricos admitidos
pelo SecuDE e usados em diversas funcoes do S3L*/
#define SYM_ENC_ALGS "DES-ECB,DES-CBC,DES-EDE,desCBC_pad,desCBC3,desCBC3_pad,\
                  IDEA,DES-CBC-ISO0,DES-CBC-ISO1,DES3-CBC-ISO0,DES3-CBC-ISO1"
```

Relativamente a `s3lsocket.h`, destaca–se:

```
/**********************************
definicoes extraidas de s3lsocket.h
**********************************/

/* sobrecarga maxima adicional eventualmente induzida pela encriptacao e/ou
   compressao dos dados numa mensagem S3L ponto-a-ponto ou multiponto */
#define S3L_DATA_OVERHEAD 32

/* valor de delta por omissao: 0 micro-segundos */
#define S3L_DEFAULT_DELTA (0L)

/* algoritmo simetrico do tipo CBC por omissao: DES-CBC */
#define S3L_DEFAULT_CBC_ALG "DES-CBC"

/* algoritmo simetrico generico por omissao */
#define S3L_DEFAULT_SYM_ALG "IDEA"

/* ausencia de encriptacao*/
#define S3L_NO_CRYPT "NO_CRYPT"

/* algoritmo de geracao de MACs por omissao: MD5(IDEA) */
#define S3L_DEFAULT_MAC_ALG "IDEA"

/* algoritmo de compressao por omissao (e unico) */
#define S3L_DEFAULT_COMPRESS_ALG "ZLIB-1.0.3"

/* ausencia de compressao */
#define S_NO_COMPRESS "NO_Z"

/* informacao de ``caracter pessoal'', que caracteriza uma entidade */
typedef struct {
...
} S3LPartieInfo;

/* informacao que caracteriza um contexto S3L ponto-a-ponto*/
typedef struct {
...
} S3LCtx;

/* sobrecarga induzida pelo cabecalho de uma mensagem S3L multiponto */
#define S3LM_HEADER_SIZE 77

/* numero minimo de celulas num array circular de GIKs:  */
#define S3LM_MIN_GIKS 5+1
```

```
/* informacao que caracteriza uma chave GIK */
typedef struct{
...
} S3LMGikInfo;

/* informacao que caracteriza um grupo S3L */
typedef struct {
...
} S3LMGroupInfo;

/* informacao que caracteriza um contexto S3L multiponto */
typedef struct {
...
} S3LMCtx;
```

A apresentação completa dos tipos `S3LPartieInfo`, `S3LCtx`, `S3LMGikInfo`, `S3LMGroupInfo` e `S3LMCtx`, bem como das diversas funções afins, reserva–se para secções próprias deste manual.

B.2.2 Manipulação de Informação de "Carácter Pessoal"

Cada entidade tem associada determinada informação que a caracteriza e que se espera que se mantenha constante ao longo da execução de um programa (por parte dessa entidade) que utilize as funcionalidades da API do S3L.

```
typedef struct {

char *psedir;        /* caminho completo para a directoria do PSE */
char *dname;         /* nome distinto X.500 da entidade */
char md5dname[16];   /* sintese MD5 do campo dname */
char ipaddr[16];     /* endereco IP onde reside a entidade */
char *symlist;       /* algoritmos simetricos suportados pela entidade */
char *cbclist;       /* subconjunto de symlist com os algoritmos da
                        categoria CBC */
} S3LPartieInfo;
```

Figura B.2: Estrutura `S3LPartieInfo`.

Essa informação é preservada numa estrutura do tipo `S3LPartieInfo`, cuja declaração, extraída de `s3lsocket.h`, se apresenta na figura B.2.

A API do S3L disponibiliza algumas funções destinadas à inicialização, preenchimento e libertação de estruturas deste tipo: `S3Linit_S3LPartieInfo`, `S3Lfill_S3LPartieInfo` e `S3Lfree_S3LPartieInfo`, respectivamente.

Tipicamente, estas funções são invocadas uma única vez ao longo da execução de um programa que utilize a API do S3L. A função `S3Lfill_S3LPartieInfo` deve ser invocada o mais cedo possível pois bloqueia, aguardando a introdução de um PIN[1] que o SecuDE exige para aceder ao PSE[2], onde reside a informação necessária para preencher os campos `dname`, `symlist` e `cbclist` da estrutura `S3LPartieInfo`.

[1] Do inglês *Personal Identification Number*.
[2] Do inglês *Personal Security Environment*.

Nome: `S3Linit_S3LPartieInfo` – inicializa uma estrutura `S3LPartieInfo` `S3Linit_S3LPartieInfo`

Sinopse:

```
#include <s3lsocket.h>
void S3Linit_S3LPartieInfo(S3LPartieInfo *info);
```

Descrição:

A função `S3Linit_S3LPartieInfo` inicializa os "campos apontadores" da estrutura `info` a `NULL`, preparando–a para uma posterior invocação à função `S3Lfill_S3LPartieInfo`.

Ver também:

`S3Lfill_S3LPartieInfo, S3Lfree_S3LPartieInfo`

Nome: `S3Lfill_S3LPartieInfo` – preenche uma estrutura `S3LPartieInfo` `S3Lfill_S3LPartieInfo`

Sinopse:

```
#include <s3lsocket.h>
S3Lbool S3Lfill_S3LPartieInfo(S3LPartieInfo *info);
```

Descrição:

A função `S3Lfill_S3LPartieInfo` consulta o PSE do utilizador que a invoca a fim de obter a informação necessária ao preenchimento dos diversos campos da estrutura `info`.

A consulta ao PSE é precedida de um pedido de introdução do respectivo PIN de acesso, definido durante a criação do PSE.

A memória dinâmica necessária para determinados campos da estrutura `info` é reservada dentro da função.

Valores de retorno:

A função retorna `S3LTRUE` se bem sucedida. Caso contrário retorna `S3LABORT`.

Ver também:

`S3Linit_S3LPartieInfo, S3Lfree_S3LPartieInfo`

Nome: `S3Lfree_S3LPartieInfo` – liberta memória dinâmica numa estrutura `S3LPartieInfo` `S3Lfree_S3LPartieInfo`

Sinopse:

```
#include <s3lsocket.h>
void S3Lfree_S3LPartieInfo(S3LPartieInfo *info);
```

Descrição:

A função `S3Lfree_S3LPartieInfo` liberta a memória dinâmica associada a alguns dos campos da estrutura `info` invocando, por fim, a função `S3Linit_S3LPartieInfo` de forma a que esses campos fiquem novamente inicializados a `NULL`.

Ver também:

`S3Linit_S3LPartieInfo, S3Lfill_S3LPartieInfo`

B.2.3 Manipulação de Contextos S3L Ponto–a–Ponto

Um contexto S3L ponto–a–ponto condensa toda a informação necessária a uma troca de mensagens S3L ponto–a–ponto[3]. Assim, previamente à assemblagem de uma mensagem S3L é necessário definir um destino bem como as qualidades de serviço pretendidas, ou seja, define–se um contexto S3L. Correspondentemente, da análise[4] de uma mensagem S3L resulta a recuperação não só da mensagem original como também do contexto que assistiu à sua geração.

```
typedef struct {
u_char    version;       /* versao do protocolo S3L */
S3LPartieInfo *loc_info; /* informacao relativa a entidade local */

char rem_md5dname[16]; /* sintese MD5 do nome dist. da entidade remota */
char *rem_entry;       /* entrada da entidade remota, na cache local */
char *rem_symlist;     /* algoritmos simetricos da entidade remota */
char *rem_cbclist;     /* subconjunto de rem_symlist com os algoritmos
                          da categoria CBC */

BitString dhagreedkey;   /* chave acordada de Diffie--Hellman */

struct timeval  rem_n; /* marca temporal remota */
struct timeval  loc_n; /* marca temporal local */
u_long          delta; /* desvio maximo admissivel entre loc_n e rem_n */

char *kijalg;     /* algoritmo de encriptacao de Kp */
char *cryptalg;   /* algoritmo de encriptacao dos dados */
char *macalg;     /* algoritmo de producao do MAC */
char *compalg;    /* algoritmo de compressao dos dados */
} S3LCtx;
```

Figura B.3: Estrutura `S3LCtx`.

A estrutura `S3LCtx`, definida em `s3lsocket.h` e apresentada na figura B.3, constitui o tipo de dados no qual assenta um contexto S3L.

Note–se que, de um contexto S3L faz sempre parte a informação relevante sobre a entidade local, acessível através do campo `loc_info`, o qual se supõe previamente inicializado e preenchido pelas funções de manipulação de informação de "carácter pessoal" (rever a secção B.2.2).

O tipo `Bitstring` é originário do SecuDE, sendo originalmente definido em [Sch95d] como:

```
struct BitString {
        unsigned int    nbits;
        char            *bits;
}
```

[3] A bem da legibilidade do resto do manual, assume–se, doravante, que as designações "contexto(s) S3L" e "mensagem(ens) S3L" se referem, por omissão, a objectos do domínio S3L ponto–a–ponto. As correspondentes designações da variante multiponto do S3L serão, porém, explicitamente usadas (*e.g.*, "contexto(s) S3L multiponto" e "mensagem(ens) S3L multiponto").

[4] Do inglês *parsing*.

A inicialização, preenchimento e libertação de memória associada a contextos S3L faz-se à custa das funções `S3Linit_S3LCtx`, `S3Lmake_S3LCtx` e `S3Lfree_S3LCtx`, respectivamente. Em geral, chamadas às funções `S3Linit_S3LCtx` e `S3Lfree_S3LCtx` ocorrem uma só vez ao longo da vida de um programa. Quanto a `S3Lmake_S3LCtx`, é invocada explicitamente sempre que se pretende definir um "contexto de emissão" e é invocada implicitamente no seio de `S3Lopen_message` (ver B.2.4) a fim de definir o "contexto de recepção". Este último poderá ser imediatamente reutilizado, por exemplo, para enviar uma mensagem S3L de resposta à mesma entidade originadora.

Nome: `S3Linit_S3LCtx` – inicializa uma estrutura `S3LCtx` `S3Linit_S3LCtx`

Sinopse:

```
#include <s3lsocket.h>
void S3Linit_S3LCtx(S3LCtx *ctx, S3LPartieInfo *info);
```

Descrição:

A função `S3Linit_S3LCtx` inicializa os "campos apontadores" da estrutura `ctx` a `NULL`, preparando–a para uma posterior invocação à função `S3Lmake_S3LCtx`.

Todavia, o campo `loc_info` é inicializado com o valor do parâmetro `info`, fazendo com que toda a informação relativa à entidade local passe a fazer automaticamente parte do contexto `ctx`.

Ver também:

`S3Lmake_S3LCtx`, `S3Lfree_S3LCtx`

Nome: `S3Lmake_S3LCtx` – define contextos `S3LCtx` `S3Lmake_S3LCtx`

Sinopse:

```
#include <s3lsocket.h>
S3Lbool   S3Lmake_S3LCtx(S3LCtx *ctx,   char *rem_id,   char nsid,
u_long delta,    char *kijalg,    char *cryptalg,    char *macalg,
char *compalg, char *rem_ipaddr, S3Lbool *skeyx_called);
```

Descrição:

A função `S3Lmake_S3LCtx` preenche os diversos campos da estrutura `ctx` definindo assim um contexto S3L. No entanto, por razões de ordem prática, alguns dos campos de `ctx` não são definidos através desta função: `version` e `loc_info` são definidos em `S3Linit_S3LCtx`; as marcas temporais `rem_n` e `loc_n` são definidas em `S3Lopen_message` e `S3Lmake_message`, respectivamente.

A definição dos campos `rem_symlist`, `rem_cbclist` e `dhagreedkey` faz–se com base em informação preservada na *cache* `.dhcache`, concretamente na entrada `rem_entry`. Este campo corresponde a um caminho, no sistema de ficheiros, para uma subdirectoria de `.dhcache` onde, supostamente, se encontra a entrada da entidade remota identificada pelo parâmetro `rem_id` de `S3Lmake_S3LCtx`. Na ausência dessa entrada, o protocolo Skeyx é automaticamente executado, após o que, uma vez criada a entrada da entidade remota na cache, os campos `rem_symlist`, `rem_cbclist` e `dhagreedkey` podem ser finalmente preenchidos.

O significado dos parâmetros de `S3Lmake_S3LCtx` é o seguinte:

`ctx`: contexto S3L a (re)definir;

`rem_id`:
identifica a "entidade remota" a que, conjuntamente com a entidade local (dada pelo campo `ctx->loc_info`), o contexto se refere;

`nsid` (*Name Space IDentifier*):
tipo do identificador `rem_id`; um valor `NSID_X500` corresponde à presença de um nome distinto X.500 em `rem_id`; um valor `NSID_MD5X500` corresponde à presença de uma síntese MD5 de um nome distinto X.500 em `rem_id`; no primeiro caso, o campo `rem_md5dname` de `ctx` é definido pela síntese MD5 de `rem_id`, enquanto que no segundo, `rem_md5dname` é uma cópia directa de `rem_id`; os valores `NSID_X500` e `NSID_MD5X500` são definidos em `common.h`;

`delta`: tempo de vida, em microsegundos, da mensagem S3L associada a este contexto; o campo `delta` de `ctx` é inicializado deste valor; um valor `S3L_DEFAULT_DELTA` (definido em `s3lsocket.h` como `0L`) corresponde a um tempo de vida infinito ou seja, não se pretende qualquer protecção contra *ataques–de–repetição*;

`kijalg`:
algoritmo simétrico da família CBC com que, em conjunção com a `n`'ésima versão da chave `dhagreedkey`, se cifrará/cifrou a chave específica (`Kp`) de cada mensagem; `n` será dado pelos campos `loc_n` num contexto de emissão e por `rem_n` num contexto de recepção; os valores de `kijalg` devem pertencer, primariamente, ao "subconjunto CBC" de `SYM_ENC_ALGS` (definido em `keyx.h`) e, em última instância, à intersecção dos conjuntos `ctx->rem_cbclist` e `ctx->loc_info->cbclist`; caso contrário, não serão aceites, e o contexto não será definido; `s3lsocket.h` define `S3L_DEFAULT_CBC_ALG` como `DES-CBC`, valor a indicar, em caso de indecisão;

`cryptalg`:
algoritmo simétrico a usar, em conjunção com a chave `Kp`, na encriptação dos dados do utilizador integrados na mensagem S3L; os valores de `kijalg` devem pertencer, primariamente, ao conjunto `SYM_ENC_ALGS` (definido em `keyx.h`) e, em última instância, à intersecção dos conjuntos `ctx->rem_symlist` e `ctx->loc_info->symlist`; caso contrário, não serão aceites, e o contexto não será estabelecido; o valor `S3L_DEFAULT_SYM_ALG` (definido como `IDEA`) pode ser usado em caso de indecisão e o valor `S3L_NO_CRYPT` indica que se dispensa encriptação dos dados do utilizador; ambos os valores são definidos em `s3lsocket.h`;

`macalg`:
algoritmo simétrico a usar, em conjunção com a chave `Kp` e o algoritmo de síntese MD5, para produzir o MAC (*Message Authetication Code*) da mensagem; os valores potenciais de `macalg` e `cryptalg` coincidem, bem como o seu método de validação; o valor `S3L_DEFAULT_MAC_ALG` a indicar em caso de decisão é definido em `s3lsocket.h` como `IDEA`; a autenticação das mensagens S3L é obrigatória;

`compalg:`
: algoritmo de compressão a usar sobre os dados do utilizador; na versão actual do S3L é suportado apenas o algoritmo `S3L_DEFAULT_COMPRESS_ALG`, definido em `s3lsocket.h` como `ZLIB-1.0.3`; a compressão é facultativa, o que se pode exprimir através do valor `S3L_NO_COMPRESS`;

`rem_ipaddr:`
: endereço IP onde reside a entidade remota;

`skeyx_called:`
: indicador da ocorrência do protocolo Skeyx; normalmente tem o valor `S3LFALSE`; se antes da invocação a `S3Lmake_S3LCtx`, o protocolo Skeyx tiver sido executado recentemente com a entidade `rem_id` na máquina `rem_ipaddr` então `skeyx_called` deverá ser fornecido com o valor `S3LTRUE`, evitando assim uma eventual execução automática de Skeyx dentro de `S3Lmake_S3LCtx`; a variável que suporta `skeyx_called` deve ser reutilizada quando `rem_id` e `rem_ipaddr` são invariantes entre sucessivas invocações de `S3Lmake_S3LCtx`;

Valores de retorno:
: A função retorna `S3LTRUE` se bem sucedida. Caso contrário retorna `S3LABORT`.

Ver também:
: `S3Linit_S3LCtx`, `S3Lfree_S3LCtx`, `S3Lopen_message`

Nome: `S3Lfree_S3LCtx` – liberta memória dinâmica numa estrutura `S3LCtx` `S3Lfree_S3LCtx`

Sinopse:
```
#include <s3lsocket.h>
void S3Lfree_S3LCtx(S3LCtx *ctx);
```

Descrição:
: A função `S3Lfree_S3LCtx` liberta a memória dinâmica associada a alguns dos campos da estrutura `info` invocando, por fim, a função `S3Linit_S3LCtx` de forma a que esses campos fiquem novamente inicializados a `NULL`.

Ver também:
: `S3Linit_S3LCtx, S3Lmake_S3LCtx`

B.2.4 Manipulação de Mensagens S3L Ponto–a–Ponto

Uma mensagem S3L constitui uma forma de encapsulamento de dados do utilizador através da qual é possível oferecer certas qualidades de serviço: Privacidade, Autenticação, Integridade, Compressão e tempo de vida limitado. Estas qualidades de serviço são primariamente estabelecidas na definição do contexto que irá presidir à criação dessas mensagens (rever `S3Lmake_S3LCtx`).

As mensagens S3L não se destinam exclusivamente à sua transmissão através da rede, sendo aliás independentes do seu veículo de propagação. É inclusivamente possível utilizá–las como "contentores" seguros de informação sob a forma de ficheiros. Nesta última aplicação revela–se de bastante utilidade a possibilidade de parametrizar o tempo de vida

dessas mensagens (através do campo `delta` do contexto), estabelecendo, por assim dizer, uma espécie de "prazo de validade" das mesmas.

A API do S3L oferece funcionalidades de assemblagem e desassemblagem de mensagens S3L e que são suficientemente genéricas para serem utilizadas em qualquer contexto que exija as qualidades de serviço acima discriminadas. Essas funcionalidades apresentam–se de seguida.

Nome: `S3Lmake_message` – assembla uma mensagem S3L `S3Lmake_message`

Sinopse:

```
#include <s3lsocket.h>
S3Lbool     S3Lmake_message     (char *buf_in,     size_t len_buf_in,
char *buf_out, size_t *len_buf_out, S3LCtx *ctx);
```

Descrição:

A função `S3Lmake_message` assembla uma mensagem S3L cujo destinatário e qualidades de serviço pretendidas se supõem especificadas no contexto `ctx` (previamente inicializado e preenchido *via* `S3Linit_S3LCtx` e `S3Lmake_S3LCtx`, respectivamente).

A mensagem a encapsular é fornecida em `buf_in` e a sua dimensão é dada por `len_buf_in`. A mensagem S3L produzida é depositada em `buf_out`, reflectindo `len_buf_out` a sua dimensão final.

À partida, a dimensão do *array* `buf_out` dever ser, no mínimo, de `len_buf_in` + `S3L_HEADER_SIZE` + `S3L_DATA_OVERHEAD`. As constantes `S3L_HEADER_SIZE` e `S3L_DATA_OVERHEAD` estão definidas em `s3lsocket.h` e representam, respectivamente, a "sobrecarga" introduzida pelo cabeçalho da mensagem S3L e a "sobrecarga" resultante da aplicação de encriptação e/ou compressão sobre mensagens pequenas, uma vez que neste último caso pode suceder a expansão das mensagens.

Valores de retorno:

A função retorna `S3LTRUE` se bem sucedida. Caso contrário retorna `S3LABORT`.

Ver também:

`S3Lopen_message`, `S3Linit_S3LCtx`, `S3Lmake_S3LCtx`

Nome: `S3Lopen_message` – desassembla uma mensagem S3L `S3Lopen_message`

Sinopse:

```
#include <s3lsocket.h>
S3Lbool     S3Lopen_message     (char *buf_in,     size_t len_buf_in,
char *buf_out, size_t *len_buf_out, S3LCtx *ctx);
```

Descrição:

A função `S3Lopen_message` analisa uma mensagem S3L, recuperando os dados aí preservados e o contexto que presidiu à sua geração.

A mensagem S3L é fornecida em `buf_in` e a sua dimensão é dada por `len_buf_in`. Os dados recuperados são depositados em `buf_out`, reflectindo `len_buf_out` a sua dimensão. A memória relativa ao *array* `buf_out` supõe–se previamente reservada.

O contexto `ctx` assume–se, pelo menos inicializado (via `S3Linit_S3LCtx`), embora possa ser reutilizado um contexto preenchido, em cujo caso será sobreposto pelos novos valores coligidos durante a análise da mensagem S3L.

O receptor de uma mensagem S3L poderá usar o contexto `ctx` resultante da análise dessa mensagem para assemblar uma segunda mensagem destinada ao originador da primeira. Para isso, basta fornecer `ctx` como parâmetro em `S3Lmake_message`. Tal pressupõe que as mesmas qualidades de serviço exigidas pelo emissor original serão agora escolhidas pelo receptor na assemblagem da sua resposta.

Valores de retorno:

A função retorna `S3LTRUE` se bem sucedida. Caso contrário retorna `S3LABORT`.

Ver também:

`S3Lmake_message`, `S3Linit_S3LCtx`, `S3Lmake_S3LCtx`

B.2.5 Manipulação de Contextos S3L Multiponto

O S3L permite a constituição de subgrupos seguros sobre os grupos de base oferecidos pelo *IP Multicast*. Ao contrário do S3L ponto–a–ponto, não é necessário indicar, explicitamente, para cada transacção entre duas entidades comunicantes, quais as qualidades de serviço que vão ditar a segurança da transacção. Uma espécie de "dono do grupo", materializado pela aplicação `s3lmgowner`, define, para todo o grupo, a política de gestão de chaves e os diversos algoritmos de Encriptação, Autenticação e Compressão a usar.

A figura B.4 apresenta as estruturas de dados usadas, ao nível do dono de um grupo, a fim de manter a informação que caracteriza o grupo. Essas estruturas de dados são preenchidas com base em informação fornecida ao demónio `s3lmgowner`, através da sua linha de comando (ver o manual de `s3lmgowner` na secção B.1.4).

Os membros ficam a conhecer as qualidades de serviço que se obrigam a usar, enquanto membros do grupo, após a execução de um processo de junção que ocorre, de forma implícita, *por necessidade*, durante a assemblagem ou desassemblagem de uma mensagem S3L multiponto (ver funções `S3LMmake_message e S3LMopen_message` em B.2.6).

A informação recebida do dono do grupo em resultado do processo de junção é mantida, nos membros, em estruturas de dados do mesmo tipo que as usadas pelo dono do grupo. Adicionalmente, é necessário preservar a identidade e a localização do dono do grupo, uma vez que, dependendo da política de gestão de chaves GIK[5] definida pelo dono do grupo, pode ser necessário repetir o processo de junção, posteriormente. Como o processo de (re)junção se baseia na utilização de mensagens S3L ponto–a–ponto, a chamada informação de "carácter pessoal" (rever B.2.2) relativa ao membro, também deve estar facilmente acessível.

Assim, um contexto S3L multiponto define–se como uma estrutura de dados onde se concentra toda a informação relevante à operação de uma entidade como membro de um grupo S3L. A figura B.5 apresenta a definição, em C, do tipo `S3LMCtx`, relativo a um contexto S3L multiponto.

A inicialização de um contexto `S3LMCtx` por forma a permitir a realização posterior de (re)junções ao grupo é efectuada através da função `S3LMinit_S3LMCtx` que, tipicamente, é invocada uma única vez, no início de um programa que utiliza as funcionalidades S3L

[5]Do inglês *Group Interchange Key*.

```
typedef struct {
        time_t  birthdate;    /* ``data de nascimento'' da GIK */
        char    gik[32];      /* GIK */
} S3LMGikInfo;                /* informacao relativa a uma GIK */

typedef struct {
u_char version; /* versao das extensoes multiponto do S3L */

char   mipaddr[16];     /* endereco IP multicast do grupo */
time_t group_lifetime;  /* tempo de vida do grupo */
time_t group_birthdate; /* ``data de nascimento'' do grupo */

char gik_file[UNIX_PATH_MAX]; /* ficheiro com a proxima versao da GIK */
S3LMGikInfo gik_array[S3LM_MAX_GIKS]; /* array circular de GIKs */
int     gik_array_front; /* 1a posicao livre do array */
int     gik_array_back;  /* 1a posicao ocupada do array */
S3Lbool gik_array_full;  /* estado (cheio/nao cheio) do array */
time_t  gik_lifetime;    /* tempo de vida de uma GIK */

time_t  kp_kcis; /* intervalo de mudanca de Kp em segundos */
size_t  kp_kcib; /* intervalo de mudanca de Kp em bytes */

char    gikalg[64];  /* algoritmo de encriptacao de Kp */
char    cryptalg[64];/* algoritmo de encriptacao dos dados */
car     macalg[64];  /* algoritmo de producao do MAC */
char    compalg[64]; /* algoritmo de compressao dos dados */
} S3LMGroupInfo;
```

Figura B.4: Estruturas `S3LMGikInfo` e `S3LMGroupInfo`.

multiponto. A reutilização de um contexto `S3LMCtx` é possível desde que o grupo e o seu dono se mantenham inalterados, uma vez que a reactualização do contexto ocorre, implícita e progressivamente, por necessidade, no interior das funções `S3LMmake_message` e `S3LMopen_message`.

A API do S3L não oferece mais funções de manuseamento de contextos S3L multiponto. A inicialização dos restantes campos fica a cargo dos mecanismos implícitos de (re)junção ao grupo. A utilização de memória estática dispensa também uma função de libertação específica.

Contudo, existem dois campos cuja manipulação directa poderá ter interesse. A leitura do campo `rem_n` após a recepção de uma mensagem S3L multiponto permitirá aceder ao valor do campo `n` dessa mensagem e utilizá–lo, por exemplo, num mecanismo de sequenciação ou de detecção de repetições. A escrita no campo `delta` de um valor positivo, antes do envio de uma mensagem, constitui indicação de que no destino esse campo deverá ser usado na detecção de *ataques–de–repetição* de uma forma análoga à sua utilização em mensagens S3L ponto–a–ponto.

```
typedef struct {
S3LPartieInfo *loc_info;  /* informacao relativa `a entidade local */

char gowner_dname[SBUF_SIZE]; /* nome distinto X.500 do dono do grupo */
char gowner_ipaddr[16];       /* endereco IP do dono do grupo */
S3LMGroupInfo group_info;     /* caracterizacao do grupo */
time_t loc_gik_birthdate; /* ``data de nascimento'' local da GIK mais
                             recente */

size_t kp_counter;   /* numero de bytes enviados com a ultima Kp */
time_t kp_birthdate; /* ``data de nascimento'' local da ultima Kp */
char   kp[32];       /* ultima Kp gerada aleatoriamente */

struct timeval  loc_n; /* marca temporal local */
struct timeval  rem_n; /* marca temporal remota */
u_long          delta; /* desvio maximo admissivel entre loc_n e rem_n */
} S3LMCtx;
```

Figura B.5: Estrutura `S3LMCtx`.

Nome: `S3LMinit_S3LMCtx` – inicializa uma estrutura `S3LMCtx` `S3LMinit_S3LMCtx`

Sinopse:

```
#include <s3lsocket.h>
void   S3LMinit_S3LMCtx   (S3LMCtx *mctx,   S3LPartieInfo *loc_info,
char *mipaddr, char *gowner_dname, char *gowner_ipaddr);
```

Descrição:

A função `S3LMinit_S3LMCtx` inicializa um contexto multiponto `mctx`, actualizando os seus campos `loc_info`, `group_info.mipaddr`, `gowner_dname` e `gowner_ipaddr` com o valor dos parâmetros homólogos da função.

Ver também:

`S3LMmake_message`, `S3LMopen_message`

B.2.6 Manipulação de Mensagens S3L Multiponto

Uma mensagem S3L multiponto encapsula os dados do utilizador de forma a que só um membro do mesmo grupo que o assemblador da mensagem pode validar a estrutura e interpretar correctamente o conteúdo dessa mensagem.

O manuseamento de mensagens S3L multiponto implica, frequentemente, a execução implícita de processos de (re)junção ao grupo S3L, através dos quais uma mensagem a enviar poderá gozar das últimas qualidades de serviço definidas pelo dono do grupo, bem como uma mensagem recebida poderá ser processada com base nas qualidades de serviço originalmente em vigor aquando da sua produção[6].

À semelhança das mensagens do S3L ponto–a–ponto, o processamento de mensagens S3L multiponto é efectuado por funções específicas, conforme se trate da assemblagem ou da desassemblagem dessas mensagens:

[6] E desde que, essas qualidades de serviço — nomeadamente chaves — não se encontrem demasiadamente obsoletas.

Nome: `S3LMmake_message` – assembla uma mensagem S3L multiponto `S3LMmake_message`

Sinopse:

```
#include <s3lsocket.h>
S3Lbool    S3LMmake_message    (char *buf_in,    size_t len_buf_in,
char *buf_out, size_t *len_buf_out, S3LMCtx *mctx);
```

Descrição:

A função `S3LMmake_message` assembla uma mensagem S3L multiponto cujo grupo destinatário e qualidades de serviço pretendidas se supõem especificadas no contexto `mctx` (previamente inicializado via `S3LMinit_S3LMCtx`).

A mensagem a encapsular é fornecida em `buf_in` e a sua dimensão é dada por `len_buf_in`. A mensagem S3L multiponto produzida é depositada em `buf_out`, reflectindo `len_buf_out` a sua dimensão final. A actualização do contexto `mctx` através de um processo de (re)junção junto do dono do grupo ocorre, implicitamente, caso a junção nunca tenha ocorrido ou caso se detecte a obsolescência desse contexto.

À partida, a dimensão do *array* `buf_out` dever ser, no mínimo, de `len_buf_in` + `S3LM_HEADER_SIZE` + `S3L_DATA_OVERHEAD`. As constantes `S3LM_HEADER_SIZE` e `S3L_DATA_OVERHEAD` estão definidas em `s3lsocket.h` e representam, respectivamente, a "sobrecarga" introduzida pelo cabeçalho da mensagem S3L multiponto e a "sobrecarga" resultante da aplicação de encriptação e/ou compressão sobre mensagens pequenas, uma vez que neste último caso pode suceder a expansão das mensagens.

O contexto `mctx` poderá ser reutilizado na assemblagem ou desassemblagem de qualquer outra mensagem relativa ao mesmo grupo originalmente definido na inicialização de `mctx` (ver manual da função `S3LMinit_S3LMCtx`).

Valores de retorno:

A função retorna `S3LTRUE` se bem sucedida. Caso contrário retorna `S3LABORT`.

Ver também:

`S3LMopen_message`, `S3LMinit_S3LMCtx`

Nome: `S3LMopen_message` – desassembla uma mensagem S3L multiponto `S3LMopen_message`

Sinopse:

```
#include <s3lsocket.h>
S3Lbool    S3LMopen_message    (char *buf_in,    size_t len_buf_in,
char *buf_out, size_t *len_buf_out, S3LMCtx *mctx);
```

Descrição:

A função `S3LMopen_message` analisa uma mensagem S3L multiponto, supostamente originada no mesmo grupo ao qual o receptor pertence. O receptor processa a mensagem com base nas qualidades de serviço típicas do grupo, mantidas no contexto `mctx`. Se o contexto do receptor for demasiado obsoleto (ou insuficiente) para processar a mensagem, tem lugar uma (re)junção implícita junto do dono do

grupo, na tentativa de obter informação sobre o contexto que presidiu à geração da mensagem.

A mensagem S3L multiponto é fornecida em `buf_in` e a sua dimensão é dada por `len_buf_in`. Os dados recuperados são depositados em `buf_out`, reflectindo `len_buf_out` a sua dimensão. A memória relativa ao *array* `buf_out` supõe–se previamente reservada.

O contexto `mctx` assume–se inicializado *via* `S3LMinit_S3LMCtx` e poderá ser reutilizado na assemblagem ou desassemblagem de qualquer outra mensagem relativa ao mesmo grupo definido nessa inicialização.

Valores de retorno:

A função retorna `S3LTRUE` se bem sucedida. Caso contrário retorna `S3LABORT`.

Ver também:

`S3LMmake_message`, `S3LMinit_S3LMCtx`

B.2.7 Escrita e Leitura de Mensagens S3L em Sockets

A API do S3L oferece um conjunto de funções de escrita e leitura sobre *sockets* bastante semelhantes às funções homólogas de Berkeley. O objectivo dessas funções é proporcionar ao utilizador de *sockets* as qualidades de serviço que advêm da utilização de mensagens S3L (Privacidade, Autenticação, Integridade, Compressão, parametrização do tempo de vida).

Basicamente, as funções do S3L assentam em chamadas às funções equivalentes de Berkeley, mas escondem do invocador os detalhes de assemblagem e desassemblagem das mensagens S3L que irão transportar os dados do utilizador.

As semelhanças sintácticas entre as funções S3L e de Berkeley facilitam bastante a reconversão de aplicações entre os dois grupos de funções. Registe–se, todavia, que, em termos semânticos, a equivalência não é total:

- as funções S3L suportam, por enquanto, apenas *sockets* do tipo `SOCK_STREAM` e `SOCK_DGRAM`;
- não é possível definir as funções S3L como automaticamente recomeçáveis face à ocorrência de sinais, ao contrário do que pode ser feito com as funções de Berkeley através da primitiva `siginterrupt`;
- um eventual parâmetro que indique o número de bytes a ler é desprezado porque uma função S3L de leitura consome uma (ou mais, c.f. o caso) mensagem S3L completa, cuja dimensão é determinada pela análise do seu cabeçalho; de igual modo, um valor de retorno que indique o número de bytes escrito leva em consideração o número total de bytes da mensagem S3L assemblada e não dos dados que o utilizador forneceu.

Funções de Escrita

Nome: `S3Lwrite` – escreve uma mensagem S3L para um descritor de ficheiro `S3Lwrite`

Sinopse:

```
#include <s3lsocket.h>
int S3Lwrite (int fd, char *buf, size_t count, S3LCtx *ctx);
```

Descrição:

A função `S3Lwrite` é análoga à função `write`, excepto que `buf` é encapsulado numa mensagem S3L com base no contexto `ctx`, antes de ser escrita para `fd`.

Valores de retorno:

A função retorna o número de bytes escritos se bem sucedida (esse número corresponde à dimensão da mensagem S3L, sendo, invariavelmente, diferente de `count`). Caso contrário retorna `-1` e `errno` é modificada de acordo com o erro específico em causa.

Erros: Os mesmos valores produzidos por `write` e mais alguns específicos:

`EINVAL`:
: `ctx` inválido;

`EPROTO`:
: assemblagem da mensagem S3L falhou ou escrita da mensagem S3L completa falhou;

Ver também:

`write`, `S3Lread`

Nome: `S3Lsend, S3Lsendto, S3Lsendmsg` – escrevem mensagens S3L para um descritor de *socket* `S3Lsend` `S3Lsendto` `S3Lsendmsg`

Sinopse:

```
#include <s3lsocket.h>

int  S3Lsend (int fd,  void *msg,  int len,  unsigned int flags,
S3LCtx *ctx);

int S3Lsendto  (int fd,  void *msg,  int len,  unsigned int flags,
const struct sockaddr *to, int tolen, void *ctx);

int  S3Lsendmsg (int fd,  struct msghdr *msg,  unsigned int flags,
S3LCtx *ctx);
```

Descrição:

As funções `S3Lsend` e `S3Lsendto` são semelhantes às funções `send` e `sendto`, respectivamente, excepto que a mensagem `msg` é encapsulado numa mensagem S3L, com base no contexto `ctx`, antes de ser escrita em `fd`.

A função `S3Lsendto` pode ser usada para enviar mensagens S3L ponto–a–ponto ou multiponto, conforme o tipo de endereço IP fornecido no campo `sin_addr` do parâmetro `to`, pelo que o parâmetro `ctx` é do tipo `void *`, sendo internamente

convertido para `S3LCtx *` ou `S3LMCtx *`. Note–se ainda que o envio de mensagens S3L multiponto através de `S3Lsendto` está limitado a *sockets* `fd` de domínio `AF_INET` e tipo `SOCK_DGRAM`.

A função `S3Lsendmsg` é semelhante à função `sendmsg`, excepto que o vector `msg->msg_iov` é convertido num vector equivalente de mensagens S3L, assembladas com base no contexto `ctx`.

Valores de retorno:

As funções retornam o número de bytes enviados se bem sucedidas (esse número corresponde à dimensão da mensagem S3L, sendo, invariavelmente, diferente de `len`). Caso contrário retornam `-1` e `errno` é modificada de acordo com o erro específico em causa.

Erros: Os mesmos valores produzidos por `send`, `sendto` e `sendmsg` e mais alguns valores específicos:

`EINVAL`:
: `ctx` inválido;

`EPROTO`:
: assemblagem da mensagem (ou vector de mensagens) S3L falhou ou escrita da mensagem (ou vector de mensagens) S3L completa(o) falhou;

Ver também:

`send`, `sendto`, `sendmsg`, `S3Lrecv`, `S3Lrecvfrom`, `S3Lrecvmsg`

Nome: `S3Lwritev` – escreve um vector de mensagens S3L para um descritor de ficheiro `S3Lwritev`

Sinopse:

```
#include <s3lsocket.h>
int S3Lwritev (int fd, struct iovec *vector, size_t count,
S3LCtx *ctx);
```

Descrição:

A função `S3Lwritev` é semelhante à função `writev`, excepto que o vector `vector` é convertido num vector equivalente de mensagens S3L, assembladas com base no contexto `ctx` e posteriormente escritas em `fd`.

Valores de retorno:

A função retorna o número total de bytes enviados se bem sucedida (esse número corresponde à dimensão do vector de mensagens S3L, sendo, invariavelmente, diferente da soma dos campos `iov_len` do vector original). Caso contrário retorna `-1` e a variável `errno` é modificada de acordo com o erro específico em causa.

Erros: Os mesmos valores produzidos por `writev` e mais alguns valores específicos:

`EINVAL`:
: `ctx` inválido;

`EPROTO`:
: assemblagem do vector de mensagens S3L falhou ou escrita do vector de mensagens S3L completo falhou;

Ver também:
`writev`, `S3Lreadv`

Funções de Leitura

Nome: `S3Lread` – lê uma mensagem S3L de um descritor de ficheiro `S3Lread`

Sinopse:
```
#include <s3lsocket.h>
int S3Lread (int fd, char *buf, size_t count, S3LCtx *ctx);
```

Descrição:

A função `S3Lread` é análoga à função `read`, excepto que em `buf` são colocados os dados do utilizador integrados na mensagem S3L lida de `fd`.

O parâmetro `count` permanece por razões de compatibilidade de sintaxe, não sendo determinante no número de bytes lidos, uma vez que `S3Lread` lê sempre uma mensagem S3L completa. Contudo, espera–se que `buf` tenha pelo menos a dimensão dada por `count`.

O contexto S3L de recepção é gravado em `ctx`, permitindo a sua eventual reutilização numa resposta S3L ao originador.

Valores de retorno:

A função retorna, se bem sucedida, o número de bytes dos dados do utilizador integrados na mensagem S3L (esse número não tem qualquer relação com o parâmetro `count`). Caso contrário retorna `-1` e `errno` é modificada de acordo com o erro específico em causa.

Erros: Os mesmos valores produzidos por `read` e `getsockopt` e mais alguns valores específicos:

`EINVAL`:
: `ctx` nulo;

`EPROTO`:
: a leitura da mensagem S3L completa falhou ou a sua desassemblagem falhou;

Ver também:
`read`, `S3Lwrite`

Nome: `S3Lrecv, S3Lrecvfrom, S3Lrecvmsg` – leêm mensagens S3L de um descritor de *socket* `S3Lrecv` `S3Lrecvfrom` `S3Lrecvmsg`

Sinopse:
```
#include <s3lsocket.h>

int S3Lrecv (int fd, char *buf, int len, unsigned int flags,
S3LCtx *ctx);
```

```
int S3Lrecvfrom (int fd, char *buf, int len, unsigned int
flags, struct sockaddr *from, int *fromlen, void *ctx);

int S3Lrecvmsg (int fd, struct msghdr *msg, unsigned int flags,
S3LCtx *ctx);
```

Descrição:

As funções `S3Lrecv` e `S3Lrecvfrom` são semelhantes às funções `recv` e `recvfrom`, excepto que em `buf` são colocados os dados do utilizador integrados na mensagem S3L lida de `fd`.

A função `S3Lrecvfrom` pode ser usada para receber mensagens S3L ponto–a–ponto ou multiponto, conforme o tipo do endereço IP relativo à origem dessas mensagens. A necessidade dessa distinção implica que tal endereço seja sempre solicitado, internamente, a `recvfrom`, sendo devolvido para o exterior em função do valor dos campos `from` e `fromlen` de `S3Lrecvfrom`. Internamente são também feitas as conversões adequadas de `ctx` para `S3LCtx *` ou `S3LMCtx *`. Note–se ainda que a recepção de mensagens S3L multiponto através de `S3Lrecvfrom` está limitada a *sockets* `fd` de domínio `AF_INET` e tipo `SOCK_DGRAM`.

O parâmetro `len` permanece por razões de compatibilidade de sintaxe, não sendo determinante no número de bytes lidos, uma vez que `S3Lrecv` e `S3Lrecvfrom` lêem sempre uma mensagem S3L completa. Contudo, espera–se que `buf` tenha pelo menos a dimensão dada por `len`.

A função `S3Lrecvmsg` é análoga à função `recvmsg` excepto que o vector `msg->msg_iov` recebe os dados produzidos pela desassemblagem, com base no contexto `ctx`, de um vector de mensagens S3L.

O contexto S3L de recepção é gravado em `ctx`, permitindo, posteriormente, uma eventual reutilização.

Valores de retorno:

As funções retornam, se bem sucedidas, o número de bytes dos dados do utilizador integrados na mensagem S3L ou a soma desses bytes, no caso de ter sido lido um vector (o valor de retorno não tem qualquer relação com o parâmetro `len` ou com o valor inicial dos campos `iov_len` do vector fornecido; estes últimos são modificados, reflectindo o número de bytes recebidos em cada célula). Caso contrário retornam `-1` e `errno` é modificada de acordo com o erro específico em causa.

Erros: Os mesmos valores produzidos por `recvfrom`, `recvmsg`, `getsockopt` e mais alguns valores específicos:

`EINVAL`:
: `ctx` nulo;

`EPROTO`:
: a leitura da mensagem S3L completa (ou do vector de mensagens S3L) falhou ou a sua desassemblagem falhou;

Ver também:

`recv`, `recvfrom`, `recvmsg`, `S3Lsend`, `S3Lsendto`, `S3Lsendmsg`

Nome: `S3Lreadv` – lê um vector de mensagens S3L de um descritor de ficheiro `S3Lreadv`

Sinopse:

```
#include <s3lsocket.h>
int   S3Lreadv   (int fd,   struct iovec *vector,   size_t count,
S3LCtx *ctx);
```

Descrição:

A função `S3Lreadv` é semelhantes à função `readv`, excepto que em `vector` são colocados os dados do utilizador integrados nas mensagens S3L de um vector lido de `fd`.

Os valores originais dos campos `iov_len` do vector `vector` não são determinantes para o número de bytes lidos da mensagem S3L correspondente, uma vez que se lê um vector linearizado de mensagens S3L completas. Contudo, espera–se que cada `iov_base` de `vector` tenha pelo menos a dimensão dada pelo campo `iov_len` respectivo.

O contexto S3L de recepção é gravado em `ctx`, permitindo a sua eventual reutilização numa resposta S3L ao originador.

Valores de retorno:

A função retorna, se bem sucedida, a soma do número de bytes depositados no vector (o valor de retorno não tem qualquer relação com o o valor inicial dos campos `iov_len` do vector fornecido; estes últimos são modificados, reflectindo o número de bytes recebidos em cada célula). Caso contrário retorna `-1` e `errno` é modificada de acordo com o erro específico em causa.

Erros: Os mesmos valores produzidos por `read`, `readv`, `recvfrom`, `getsockopt` e mais alguns valores específicos:

`EINVAL`:
: `ctx` nulo;

`EPROTO`:
: a leitura do vector completo de mensagens S3L falhou ou a sua desassemblagem falhou;

Ver também:

`readv`, `S3Lwritev`

B.2.8 Metodologia e Exemplos de Utilização da API do S3L

Nesta secção sugere–se uma metodologia genérica a seguir na programação de aplicações com as funcionalidades oferecidas pela API do S3L. Os passos a seguir nas variantes ponto-a–ponto e multiponto do S3L são semelhantes. Todavia, existem diferenças suficientes que justificam uma apresentação separada. A metodologia apresentada é consubstanciada com exemplos comentados do tipo cliente–servidor.

S3L Ponto–a–Ponto

As etapas envolvidas na utilização das funcionalidades ponto–a–ponto da API do S3L são, fundamentalmente, as seguintes:

- tarefas do produtor de mensagens S3L ponto–a–ponto:

  1. obtenção de informação que identifica e caracteriza a entidade produtora;
  2. inicialização de um contexto S3L ponto–a–ponto, especificando a entidade destino da mensagem e as qualidades de serviço pretendidas;
  3. assemblagem da mensagem S3L ponto–a–ponto, com base no contexto seguro definido;
  4. envio pela rede, utilizando as funções de escrita sobre *sockets* providenciadas pelo S3L; alternativamente, outras formas de transmissão poderão ser empregues, como por exemplo o armazenamento em memória secundária tendo em vista a posterior "entrega manual" ao destinatário;
  5. repetição das etapas 2, 3 e 4, com possível reutilização dos contextos produzidos na etapa 2;

- tarefas do consumidor de mensagens S3L ponto–a–ponto:

  1. obtenção de informação que identifica e caracteriza a entidade consumidora;
  2. inicialização de um contexto S3L ponto–a–ponto, preparando–o para acolher a informação contextual produzida pela análise de uma mensagem S3L ponto–a–-ponto recebida;
  3. recepção de uma mensagem S3L ponto–a–ponto a partir da rede com base nas funções de leitura sobre *sockets* providenciadas pelo S3L, ou obtenção da mensagem através de outros meios, *c.f.* o seu método original de armazenamento e distribuição (*e.g.*, leitura de um ficheiro);
  4. processamento da mensagem S3L ponto–a–ponto, da qual resulta a recuperação dos dados e o preenchimento do contexto S3L ponto–a–ponto previamente inicializado;
  5. repetição das etapas 3 e 4, com possível reutilização do mesmo contexto (a etapa 2 dispensa–se porque o contexto de recepção é automaticamente sobreposto se for reutilizado).

A figura B.6 encerra o código C relativo a um produtor de mensagens S3L ponto–a–-ponto. As linhas de código mais relevantes, em termos de funcionalidades próprias do S3L são, de seguida, comentadas:

- linha 1: inclusão do ficheiro `s3lsocket.h`, que define a interface de acesso às funcionalidades S3L (e também às funcionalidades normais sobre *sockets* Berkeley, pelo que se dispensa a inclusão explícita das interfaces específicas para esse efeito);
- linha 6: declaração de uma estrutura `personal_info` do tipo `S3LPartieInfo` onde será preservada informação específica do produtor;
- linha 7: declaração de uma estrutura do tipo `S3LCtx` destinada à definição de contextos S3L ponto–a–ponto;

```
1   #include "s3lsocket.h"
2   #define MENSAGEM "Ola mundo!"
3
4   main()
5   {
6       S3LPartieInfo      personal_info;
7       S3LCtx             ctx;
8       S3Lbool            skeyx_called=S3LFALSE;
9       int                ret, sockfd;
10      struct sockaddr_in srvaddrin;
11
12      /******* etapa 1: obtencao de informacao pessoal guardada no PSE *******/
13      S3Linit_S3LPartieInfo(&personal_info);
14      if (S3Lfill_S3LPartieInfo(&personal_info)!=S3LTRUE)
15          exit(1);
16
17      /******* etapa 2: inicializacao e preenchimento do contexto S3L *******/
18      S3Linit_S3LCtx(&ctx, &personal_info);
19      if ((ret=S3Lmake_S3LCtx (&ctx, "C=PT, CN=ruf, D=user", NSID_X500, 0L
20                               "DES-CBC", "IDEA", "IDEA", "NO_Z",
21                               "193.136.8.7", &skeyx_called))!=S3LTRUE)
22          exit(1);
23
24      /* procedimentos usuais de instalacao de um cliente TCP */
25
26      if ((sockfd=socket(AF_INET, SOCK_STREAM, 0))<0)
27          exit(1);
28
29      bzero(&srvaddrin, sizeof(srvaddrin));
30      srvaddrin.sin_family = AF_INET;
31      srvaddrin.sin_addr.s_addr = inet_addr("193.136.8.7");
32      srvaddrin.sin_port = htons(55555);
33
34      if (connect(sockfd, (struct sockaddr *) &srvaddrin, sizeof(srvaddrin)) < 0)
35          exit(1);
36
37      /******* etapas 3 e 4 : assemblagem e envio da mensagem S3L *******/
38      ret=S3Lwrite(sockfd, MENSAGEM, strlen(MENSAGEM)+1, &ctx);
39
40      /******* eventual repeticao das etapas 2 (so preenchimento), 3 e 4 *******/
41
42      /******* devolucao de memoria dinamica na saida *******/
43      S3Lfree_S3LPartieInfo(&personal_info);
44      S3Lfree_S3LCtx(&ctx);
45      close(sockfd);
46      exit(0);
47  }
```

Figura B.6: Exemplo de um produtor de mensagens S3L ponto–a–ponto.

- linha 8: declaração de uma variável booleana cujo valor lógico representa a ocorrência prévia (no contexto deste produtor), implícita ou explícita, do protocolo Skeyx; por motivos óbvios, a variável é inicializada a `S3LFALSE`; a variável será um dos argumentos de `S3Lmake_S3LCtx` (ver linha 21) e daí a sua presença obrigatória;

- linha 13: inicialização da estrutura `personal_info`, preparando–a para posterior preenchimento;
- linha 14: preenchimento da estrutura `personal_info`, pela consulta de vários elementos de informação "pessoal" preservados no PSE do produtor; para o efeito, será pedida a introdução de um PIN, pelo que é conveniente que a invocação de `S3Lfill_S3LPartieInfo` se faça o mais cedo possível, como é o caso do exemplo em questão;
- linha 18: inicialização da estrutura `ctx` tendo em vista uma posterior definição;
- linhas 19, 20 e 21: definição do contexto S3L ponto–a–ponto, com base no qual será gerada uma ou mais mensagens S3L ponto–a–ponto no produtor; no caso presente, "`C=PT, CN=ruf, D=user`" é o nome distinto X.500 do destino (consumidor), `NSID_X500` indica que se usou o nome distinto (e não a sua síntese MD5) para identificar o destino, `0L` é o valor de δ (ou seja, não se pede a detecção de *ataques–de–repetição*), `DES-CBC` identifica o algoritmo simétrico do tipo CBC com base no qual `Kp` será cifrado, `IDEA` (1ª ocorrência) define o algoritmo com base no qual a mensagem do utilizador será cifrada, `IDEA` (2ª ocorrência) define o algoritmo a usar em conjunção com o MD5 para produzir o MAC, `NO_Z` significa que não se pretende compressão dos dados, `193.136.8.7` é o endereço IP do destinatário (útil caso seja necessário executar Skeyx implicitamente) e `skeyx_called` controla e reflecte a ocorrência do protocolo Skeyx no interior de `S3Lmake_S3LCtx` (se for `S3LTRUE`, então o protocolo Skeyx foi executado recentemente com o consumidor e portanto não deve ser executado novamente; caso contrário, indica que, em caso de necessidade, o protocolo Skeyx deve ser executado);
- linha 38: envio da mensagem `MENSAGEM` com base no contexto S3L ponto–a–ponto `ctx`;
- linhas 43 e 44: devolução da memória dinâmica reservada durante a inicialização ou preenchimento das estruturas `personal_info` e `ctx`.

As tarefas mais importantes correspondem, precisamente, às etapas apresentadas no início desta secção para um produtor de mensagens S3L ponto–a–ponto.

A figura B.7 apresenta o código C do consumidor de mensagens S3L ponto–a–ponto correspondente ao produtor da figura B.6. Seguem–se alguns comentários relativos ao código do consumidor:

- linhas 1, 5, 6, 12, 13, e 17: semelhantes às linhas de código com o mesmo conteúdo, no produtor (ver a figura B.6); saliente–se que, relativamente ao contexto `ctx` há agora apenas lugar à sua inicialização (linha 17), uma vez que o seu preenchimento é feito, automaticamente, durante a análise de uma mensagem S3L ponto–a–ponto recebida (ver linha 39);

```
1   #include "s3lsocket.h"
2
3   main()
4   {
5       S3LPartieInfo      personal_info;
6       S3LCtx             ctx;
7       int                sockfd, newsockfd, ret;
8       struct sockaddr_in srvaddrin;
9       char               buf[1024];
10
11      /******* etapa 1: obtencao de informacao pessoal guardada no PSE *******/
12      S3Linit_S3LPartieInfo(&personal_info);
13      if (S3Lfill_S3LPartieInfo(&personal_info)!=S3LTRUE)
14          exit(1);
15
16      /******* etapa 2: inicializacao do contexto S3L *******/
17      S3Linit_S3LCtx(&ctx, &personal_info);
18
19      /* procedimentos usuais de instalacao de um servidor TCP */
20      if ((sockfd=socket(AF_INET, SOCK_STREAM, 0))<0)
21          exit(1);
22
23      bzero(&srvaddrin, sizeof(srvaddrin));
24      srvaddrin.sin_family = AF_INET;
25      srvaddrin.sin_addr.s_addr = htonl(INADDR_ANY);
26      srvaddrin.sin_port = htons(55555);
27
28      if(bind(sockfd, (struct sockaddr *) &srvaddrin, sizeof(srvaddrin)) < 0)
29          exit(1);
30
31      if (listen(sockfd, 5))
32          exit(1);
33
34      if ((newsockfd=accept(sockfd, 0, 0)) < 0)
35          exit(1);
36
37      /******* etapas 3 e 4: recepcao e desassemblagem da mensagem S3L
38                com actualizacao do contexto ctx ******/
39      ret=S3Lread(newsockfd, buf, 1024, &ctx);
40
41      /******* etapa 5: eventual repeticao das etapas 3 e 4 apos novo accept */
42
43      /******* devolucao de memoria dinamica (‘a saida) *******/
44      S3Lfree_S3LPartieInfo(&personal_info);
45      S3Lfree_S3LCtx(&ctx);
46      close(newsockfd); close(sockfd);
47      exit(0);
48  }
```

Figura B.7: Exemplo de um consumidor de mensagens S3L ponto–a–ponto.

- linha 39: leitura de uma mensagem S3L ponto–a–ponto da rede e seu processamento, daí resultando a obtenção dos dados encapsulados na mensagem e a definição do contexto S3L ponto–a–ponto correspondente às qualidades de serviço especificadas

pelo originador da mensagem; caso o consumidor assuma, de seguida, o papel de produtor de mensagens, este contexto recém definido poderá ser reutilizado se o destino dessas mensagens for o originador que definiu, em primeira instância, esse contexto;

- linha 41: possível recepção de novas mensagens S3L ponto–a–ponto (via `S3Lread`, `S3Lrecv`, etc.) com eventual reutilização do mesmo contexto `ctx` sem necessidade de reinicializá–lo (os antigos valores dos seus campos do contexto são automaticamente sobrepostos pelos novos);
- linhas 44 e 45: devolução da memória dinâmica reservada durante a inicialização ou preenchimento das estruturas `personal_info` e `ctx`.

S3L Multiponto

A utilização das funcionalidades proporcionadas pelas extensões multiponto do S3L segue um roteiro bem definido, que se divide em dois conjunto de tarefas, em função do papel a desempenhar:

- tarefas do produtor de mensagens S3L multiponto:
  1. obtenção de informação que identifica e caracteriza a entidade produtora;
  2. inicialização de um contexto S3L multiponto, especificando–se o endereço *IP Multicast* do grupo e identificando–se o seu dono;
  3. assemblagem da mensagem S3L multiponto com base no contexto inicializado em 2 (e que acaba por ser completamente definido, em resultado de uma junção implícita junto do dono do grupo S3L);
  4. envio da mensagem assemblada para o grupo S3L, subgrupo de um determinado grupo *IP Multicast*, através da função `S3Lsendto`;
  5. repetição das etapas 2, 3 e 4, com possível reutilização dos contextos inicializados na etapa 2 sempre que o grupo em causa e o seu dono se mantenham os mesmos;
- tarefas do consumidor de mensagens S3L multiponto:
  1. obtenção de informação que identifica e caracteriza a entidade consumidora;
  2. inicialização de um contexto S3L multiponto, especificando–se o endereço *IP Multicast* do grupo e identificando–se o seu dono;
  3. junção explícita de um *socket* Berkeley ao grupo *IP Multicast*[7];
  4. recepção de uma mensagem S3L multiponto *via* `S3Lrecvfrom`;
  5. processamento da mensagem recebida com base no contexto inicializado em 2 (e que acaba por ser completamente definido, em resultado de uma junção implícita junto do dono do grupo S3L);

[7] Dispensável para um emissor, mas necessária num receptor.

6. repetição das etapas 2, 3, 4 e 5, com possível reutilização dos contextos inicializados na etapa 2 sempre que o grupo em causa e o seu dono se mantenham os mesmos, em cujo caso se dispensa também a repetição da junção *IP Multicast* efectuada em 3.

A figura B.8 apresenta o código de um produtor de mensagens S3L multiponto. Seguem–se alguns comentários sobre as linhas de código extra relativamente a um "produtor de Berkeley":

- linha 1: inclusão do ficheiro `s3lsocket.h`, que define a interface de acesso às funcionalidades S3L (e também às funcionalidades normais sobre *sockets* Berkeley, pelo que se dispensa a inclusão explícita das interfaces específicas para esse efeito);
- linha 4: declaração de uma estrutura `personal_info` do tipo `S3LPartieInfo` onde será preservada informação específica do produtor;
- linha 5: declaração de uma estrutura do tipo `S3LMCtx` destinada à definição de contextos S3L multiponto;
- linha 13: inicialização da estrutura `personal_info`, preparando–a para posterior preenchimento;
- linha 14: preenchimento da estrutura `personal_info`, pela consulta de vários elementos de informação "pessoal" preservados no PSE do produtor; para o efeito, será pedida a introdução de um PIN, pelo que é conveniente que a invocação de `S3Lfill_S3LPartieInfo` se faça o mais cedo possível, como é o caso do exemplo em questão;
- linhas 18 a 20: inicialização da estrutura `mctx` tendo em vista a sua definição completa, através do processo de junção ao grupo, a ocorrer, posteriormente, de forma implícita (linha 37); para além da "informação pessoal" do produtor (`personal_info`), obtida na linha 14, indica–se o grupo *IP Multicast* (`235.0.0.0`), o nome distinto do dono do subgrupo S3L (`C=PT, CN=ruf, D=user`) bem como a sua localização (`193.136.8.7`); na linha 20, a inicialização do campo `delta` a zero (`0L`) indica que não se pretende detecção de *ataques–de–repetição* com base num tempo de vida das mensagens, dado por `delta`;
- linhas 37 e 38: a mensagem produzida na linha 32 é convertida numa mensagem S3L multiponto e enviada para o subgrupo S3L cujo dono foi definido na linha 18; na primeira iteração do ciclo, ocorrerá o processo de junção junto do dono do subgrupo S3L, o que permitirá completar a definição do contexto S3L multiponto `mctx`, após o que a mensagem S3L é assemblada e enviada para o endereço *IP Multicast* `235.0.0.0`; nas próximas iterações do ciclo, poderão ocorrer rejunções ao subgrupo S3L, desde que a *chave–do–grupo* (`GIK`) seja efémera e o seu tempo de vida tenha, entretanto, expirado; a variável de contexto `mctx` é sucessivamente reutilizada;
- linha 45: exemplifica a devolução da memória dinâmica reservada durante a definição da estrutura `personal_info` na linha 14 e `ctx`; obviamente, esta linha não é executada no programa em questão, devido ao ciclo infinito da linha 30.

```
1   #include "s3lsocket.h"
2   main()
3   {
4       S3LPartieInfo         personal_info;
5       S3LMCtx               mctx;
6       struct sockaddr_in    addr;
7       int                   sockfd;
8       struct ip_mreq        mreq;
9       time_t                tempo;
10      char                  buf[1024];
11
12      /******* etapa 1: obtencao de informacao pessoal guardada no PSE */
13      S3Linit_S3LPartieInfo(&personal_info);
14      if (S3Lfill_S3LPartieInfo(&personal_info)!=S3LTRUE)
15          exit(1);
16
17      /******* etapa 2: inicializacao do contexto S3L multiponto */
18      S3LMinit_S3LMCtx(&mctx, &personal_info, "235.0.0.0", "C=PT, CN=ruf, D=user",
19                       "193.136.8.7");
20      mctx.delta=0L;
21
22      /* procedimentos usuais de instalacao de um servidor IP multicast */
23      if ((sockfd=socket(AF_INET, SOCK_DGRAM,0))<0)
24          exit(1);
25      bzero(&addr, sizeof(addr));
26      addr.sin_family = AF_INET;
27      addr.sin_port = htons(55555);
28      addr.sin_addr.s_addr = inet_addr("235.0.0.0");
29
30      while (1) {
31          tempo=time(0);
32          sprintf(buf, "tempo local no produtor : %s", ctime(&tempo));
33
34          /******* etapas 3, 4 e 5: assemblagem e envio da mensagem S3L
35                   multiponto com actualizacao implicita do contexto mctx que
36                   sera reutilizado dentro deste ciclo */
37          if ((S3Lsendto(sockfd, buf, strlen(buf)+1, 0, (struct sockaddr *)&addr,
38                         sizeof(addr), &mctx))<=0)
39              exit(1);
40
41          sleep(5);
42      }
43
44      /******* devolucao de memoria dinamica (`a saida) *******/
45      S3Lfree_S3LPartieInfo(&personal_info);
46      close(sockfd);
47      exit(0);
48  }
```

Figura B.8: Exemplo de um produtor S3L multiponto.

A figura B.9 apresenta o código C relativo ao consumidor de mensagens S3L multiponto correspondente ao produtor da figura B.8. Seguem–se alguns comentários relativos ao seu código:

```
1   #include "s3lsocket.h"
2
3   main()
4   {
5       S3LPartieInfo         personal_info;
6       S3LMCtx               mctx;
7       struct sockaddr_in    addr;
8       int                   addrlen, sockfd;
9       struct ip_mreq        mreq;
10      int                   share;
11      char                  buf[1024];
12
13      /******* etapa 1: obtencao de informacao pessoal guardada no PSE */
14      S3Linit_S3LPartieInfo(&personal_info);
15      if (S3Lfill_S3LPartieInfo(&personal_info)!=S3LTRUE)
16          exit(1);
17
18      /******* etapa 2: inicializacao do contexto S3L multiponto */
19      S3LMinit_S3LMCtx(&mctx, &personal_info, "235.0.0.0", "C=PT, CN=ruf, D=user",
20                       "193.136.8.7");
21
22      /* procedimentos usuais de instalacao de um cliente IP multicast */
23      if ((sockfd=socket(AF_INET, SOCK_DGRAM,0))<0)
24          exit(1);
25
26      bzero(&addr, sizeof(addr));
27      addr.sin_family = AF_INET;
28      addr.sin_port = htons(55555);
29      addr.sin_addr.s_addr = inet_addr("235.0.0.0");
30      addrlen=sizeof(addr);
31
32      /* partilha do porto usado (55555) com outros processos */
33      share=1;
34      if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char *)&share,
35                     sizeof(share))<0)
36          exit(1);
37
38      if (bind(sockfd, (struct sockaddr*)&addr, sizeof(addr))<0)
39          exit(1);
40
41      /******* etapa 3: juncao explicita ao grupo IP multicast 235.0.0.0 */
42      mreq.imr_multiaddr.s_addr = inet_addr("235.0.0.0");
43      mreq.imr_interface.s_addr = htonl(INADDR_ANY);
44      if (setsockopt(sockfd, IPPROTO_IP, IP_ADD_MEMBERSHIP, (const char*)&mreq,
45                     sizeof(mreq))<0)
46          exit(1);
47
48      while(1) {
49          bzero(buf, SBUF_SIZE);
50          addrlen=sizeof(addr);
51
52          /******* etapas 4, 5 e 6: recepcao e desassemblagem da mensagem S3L
53                   multiponto com actualizacao implicita do contexto mctx que
54                   sera reutilizado dentro deste ciclo */
55          if (S3Lrecvfrom(sockfd, buf, SBUF_SIZE, 0, (struct sockaddr *)&addr,
56                          &addrlen, &mctx)<=0)
```

```
57              exit(1);
58
59          printf ("S3Lrecvfrom: %s\n", buf);
60          addr.sin_addr.s_addr = inet_addr("235.0.0.0");
61      }
62
63      /******* devolucao de memoria dinamica (‘a saida) *******/
64      S3Lfree_S3LPartieInfo(&personal_info);
65      close(sockfd);
66      exit(0);
67  }
```

Figura B.9: Exemplo de um consumidor S3L multiponto.

- linhas 1, 5, 6, 14, 15, 19 e 64: semelhantes às linhas de código com o mesmo conteúdo, no produtor (ver a figura B.8); saliente–se que, relativamente ao campo `delta` do contexto `mctx`, omite–se qualquer inicialização, dado que será sobreposto pelo `delta` definido nas mensagens S3L multiponto recebidas;

- linhas 33 a 35: indicação de que se pretende partilhar, com outros processos, o porto escolhido (`55555`) para receber o tráfego *IP Multicast* (do grupo `235.0.0.0`, neste caso); sem esta operação de partilha, não é possível a coexistência de vários consumidores na mesma máquina;

- linhas 42 a 45: junção explícita ao grupo *IP Multicast* `235.0.0.0`; note–se que o produtor carece desta junção, uma vez que não é necessário ser membro de um grupo *IP Multicast* para gerar tráfego dirigido a esse grupo; todavia, quer no produtor, quer no consumidor, ocorrem junções obrigatórias, embora implícitas, ao subgrupo S3L;

- linha 55: recepção de uma mensagem S3L multiponto, comunicada ao consumidor *via IP Multicast*; a primeira mensagem recebida (correspondente à primeira iteração do ciclo) desencadeia, implicitamente, o processo de junção ao subgrupo S3L, após o que a mensagem poderá ser desassemblada e os seus dados extraídos; a recepção das próximas mensagens poderá implicar rejunções caso a *chave–do–grupo* seja efémera; o contexto `mctx` é sucessivamente reutilizado.

Os exemplos multiponto apresentados (figura B.8 e figura B.9) contêm uma série de passos especificamente destinados à criação, configuração e utilização de *sockets IP Multicast*, não sendo, neste guia, fornecidas explicações adicionais sobre o assunto. Recomenda–se a consulta de [LFJL86] que, apesar de não ser muito recente, constitui ainda um excelente tutorial sobre a matéria.

Adicionalmente, os exemplos multiponto pressupõem a existência de um dono do grupo, identificado por "`C=PT, CN=ruf, D=user`", localizado na máquina `193.136.8.7` e gerindo o grupo `235.0.0.0`. Assim, na máquina `193.136.8.7`, a entidade "`C=PT, CN=ruf, D=user`" seria responsável pela definição das ACLs e execução do demónio `s3lmgowner` para o grupo `235.0.0.0` (ver o manual de `s3lmgaclsmaint` e `s3lmgowner`).

O código dos produtores e consumidores S3L (ponto–a–ponto e multiponto) apresentados pode ser encontrado na directoria `.../s3l-b01-dist/s3l-b01/examples`, uma vez

aberta a distribuição do S3L. A `Makefile` correspondente contém as opções necessárias de compilação e de ligação com bibliotecas (assumindo que a distribuição foi correctamente instalada), pelo que se recomenda a sua reutilização no desenvolvimento de aplicações que recorrem à API do S3L.

Nas aplicações S3L a desenvolver, o valor `S3LTRUE` para a variável global `verbose` dever ser usado apenas para efeitos de depuração, resultando na produção de mensagens indesejáveis nas versões finais.

Bibliografia

[Ama96] J.C. Rufino Amaro. Grupos Seguros sobre Multicast IP. Technical report, Universidade do Minho, Maio 1996.

[AMP95a] Ashar Aziz, Tom Markson, and Hemma Prafullchandra. Internet Draft – Certificate Discovery Protocol, Dezembro 1995. (work in progress).

[AMP95b] Ashar Aziz, Tom Markson, and Hemma Prafullchandra. Internet Draft – Encoding of an Unsigned Diffie–Hellman Public Value, Dezembro 1995. (work in progress).

[AMP95c] Ashar Aziz, Tom Markson, and Hemma Prafullchandra. Internet Draft – Simple Key–Management For Internet Protocols (SKIP), Dezembro 1995. (work in progress).

[AMP95d] Ashar Aziz, Tom Markson, and Hemma Prafullchandra. Internet Draft – SKIP Algorithm Discovery Protocol, Dezembro 1995. (work in progress).

[AMP95e] Ashar Aziz, Tom Markson, and Hemma Prafullchandra. Internet Draft – SKIP Extensions for IP Multicast, Dezembro 1995. (work in progress).

[AN94] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. Research Report 125, Digital – Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301, Junho 1994.

[AN95] R.J. Anderson and R. Needham. Robustness Principles for Public Key Protocols. In *Advances in Cryptology – CRYPTO 95 Proceedings*. Springer–Verlag, 1995.

[AP95] Ashar Aziz and Martin Patterson. Design and Implementation of SKIP. Technical Report ICG–95–004, Sun Microsystems – Internet Commerce Group, M/S PAL01–550, 2550 Garcia Avenue, Mountain View, CA 94043, Junho 1995.

[Atk95a] R. Atkinson. RFC 1825 – Security Architecture for the Internet Protocol, Agosto 1995.

[Atk95b] R. Atkinson. RFC 1826 – IP Authentication Header (AH), Agosto 1995.

[Atk95c] R. Atkinson. RFC 1827 – IP Encapsulating Security Protocol (ESP), Agosto 1995.

[Bal93] D. Balenson. RFC 1423 – Privacy Enhancement for Internet Electronic Mail: Part III – Algorithms, Modes, and Identifiers, Fevereiro 1993.

[Bar93] John Barkley. Comparing Remote Procedure Calls – NISTIR 5277. U.S. Department of Commerce, Outubro 1993.

[BCK+94] Robert Bagwill, Lisa Carnahan, Richard Kuhn, Anastase Nakassis, Michael Ranson, John Barkley, Shu-jen Chang, Paul Markovitz, Karen Olsen, and John Wack. Security in Open Systems – NIST Special Publication 800–7. U.S. Department of Commerce, Julho 1994.

[Bel89] S.M. Bellovin. Security Problems in the TCP/IP Protocol Suite. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey 07974, Abril 1989.

[Bel90] S.M. Bellovin. Pseudo–Network Drivers and Virtual Networks. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey 07974, 1990.

[BM91] S.M. Bellovin and M. Merritt. Limitations of the Kerberos Protocol. In *Winter 1991 USENIX Conference Proceedings*, pages 253–267. USENIX Association, 1991.

[BN84] A.D. Birrel and B.J. Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1):39–59, Fevereiro 1984.

[Bra87] D.K. Branstad. Considerations for Security in the OSI Architecture. *IEEE Network*, 1(2):34–39, Abril 1987.

[BZ93] R. Braudes and S. Zabele. RFC 1458 – Requirements for Multicast Protocols, Maio 1993.

[CCI84] CCITT. Recommendation X.200 – Open Systems Interconnection (OSI): Basic Reference Model. International Telecommunications Union, Geneva, 1984. (also as ISO standard 7498–1).

[CCI88] CCITT. Recommendation X.500 – The Directory – Overview of Concepts, Models, and Service. International Telecommunications Union, Geneva, Dezembro 1988.

[CCI89] CCITT. Recommendation X.509 – The Directory Authentication Framework. International Telecommunications Union, Geneva, 1989.

[CLA+96] Germano Caronni, Hannes Lubich, Ashar Aziz, Tom Markson, and Rich Skrenta. SKIP – Securing the Internet. In *Proceedings of the 5th Workshop on Enabling Technologies*. IEEE Computer Society Press, 1996.

[CZ95] D. Brent Chapman and Elizabeth D. Zwicky. *Building Internet Firewalls*. O'Reilly, Setembro 1995.

[DA97] Tim Dierks and Christopher Allen. Internet Draft – The TLS Protocol (version 1.0), Maio 1997. (work in progress).

[DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. A Strengthened Version of RIPEMD. German Information Security Agency / Katholieke Universiteit Leuven, Abril 1996.

[Dee89] Steve Deering. RFC 1112 – Host Extensions for IP Multicasting, Agosto 1989.

[Dee95] Steve Deering. RFC 1883 – Internet Protocol, Version 6 (IPv6) Specification, Dezembro 1995.

[DH76] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT–22(6):644–654, Novembro 1976.

[Dob96] Hans Dobbertin. Cryptanalysis of MD5 Compress. German Information Security Agency, Maio 1996.

[DOW92] W. Diffie, P.C. van Oorschot, and M.J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[EFHT97] D. Estrin, D. Farinacci, A. Helmy, and D. Thaler. RFC 2117 – Protocol Independent Multicast-Sparse Mode (PIM-SM), Junho 1997.

[EK95] D. Eastlake and C. Kaufman. Internet Draft – Domain Name Security Extensions, 1995. (work in progress).

[Fen96] W. Fenner. Internet Draft – Internet Group Management Protocol, Version 2, Setembro 1996. (work in progress).

[FIG+97] R. Fielding, UC Irvine, J. Gettys, J. Mogul, DEC, H. Frystyk, T. Berners-Lee, and MIT/LCS. RFC 2068 – Hypertext Transfer Protocol – HTTP/1.1, Janeiro 1997.

[FKK96] Alan O. Freier, Philip Karlton, and Paul C. Kocher. Internet Draft – The SSL Protocol (Version 3.0), Março 1996. (work in progress).

[Fou92a] Open Software Foundation. Distributed Computing Environment – An Overview, Janeiro 1992.

[Fou92b] Open Software Foundation. Security in a Distributed Computing Environment – A White Paper, Janeiro 1992.

[Gon93] L. Gong. Variations on the Themes of Message Freshness and Replay. In *Proceedings of the IEEE Computer Security Foundations Workshop*, Junho 1993.

[GS96] Simson Garfinkel and Gene Spafford. *Practical Unix & Internet Security.* O'Reilly & Associates, Inc., 2nd edition, Abril 1996.

[Hur97] Mike Hurwicz. Multicast to the Masses. *Byte*, 22(6):93–97, Junho 1997.

[IB93] John Ioannidis and Matt Blaze. The Architecture and Implementation of Network–Layer Security Under Unix. In *Proceedings of the 4th Usenix Unix Security Workshop*, Santa Clara, California, Outubro 1993.

[IEE88] IEEE. Portable Operating System Interface for Computer Environments (POSIX) – Standard 1003.1, Setembro 1988.

[IM90] C. I'Anson and C. Mitchell. Security Defects in CCITT Recommendation X.509 – the Directory Authentication Framework. *Computer Communications Review*, 20(2):30–34, Abril 1990.

[ISO87a] ISO/IEC. International Standard 8824 – Specification of Abstract Sintax Notation One (ASN.1), 1987.

[ISO87b] ISO/IEC. International Standard 8825 – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), 1987.

[ISO91] ISO/IEC. ISO Remote Procedure Call Specification – ISO/IEC CD 11578 N6561, Novembro 1991.

[ISO92a] ISO/IEC. Draft International Standard 10736 – Transport Layer Security Protocol (TLSP) , Julho 1992.

[ISO92b] ISO/IEC. Draft International Standard 11577 – Network Layer Security Protocol (NLSP) , Novembro 1992.

[IT97] ITU-T. ITU-T Recommendation X.509 (Previously CCITT Recommendation) – The Directory Authentication Framework. International Telecommunications Union, Geneva, Junho 1997.

[Kal93] B.S. Kaliski. RFC 1424 – Privacy Enhancement for Internet Electronic Mail: Part IV – Key Certificates and Related Services, Fevereiro 1993.

[Kau93] C. Kaufman. RFC 1507 – Distributed Authentication Security Service, Setembro 1993.

[Ken93] S.T. Kent. RFC 1422 – Privacy Enhancement for Internet Electronic Mail: Part II – Certificate-Based Key Management, Fevereiro 1993.

[Kle90] D.V. Klein. 'Foiling the Cracker': A Survey of, and Implications to, Password Security. In *Proceedings of the USENIX UNIX Security Workshop*, 1990.

[KN93] J.T. Kohl and B.C. Neuman. RFC 1510 – The Kerberos Network Authentication Service (V5), Setembro 1993.

[KPP94] P. Kaijser, T. Parker, and D. Pinkas. SESAME: The Solution to Security for Open Distributed Systems. *Journal of Computer Communications*, 17(4):501–518, Julho 1994.

[Kra95] Hugo Krawczyk. SKEME: A Versatile Secure Key Exchange Mechanism for Internet. Technical report, IBM T.J. Watson Research Center, New York, Novembro 1995.

[KS] P. Karn and W.A. Simpson. Internet draft – The Photuris Session Key Management Protocol. (work in progress).

[Lev90] W. Leveque. *Elementary Theory of Numbers.* Dover, New York, 1990.

[LFJL86] Samuel J. Leffler, Robert S. Fabry, William N. Joy, and Phil Lapsley. An Advanced 4.4BSD Interprocess Communication Tutorial. Technical report, University of California, Berkeley, 1986.

[Lin90] John Linn. Generic Security Service Application Program Interface (Version C.3). Secure Systems Group, Digital Equipment Corporation, Setembro 1990.

[Lin93a] John Linn. RFC 1421 – Privacy Enhancement for Internet Electronic Mail: Part I – Message Encipherment and and Authentication Procedures, Fevereiro 1993.

[Lin93b] John Linn. RFC 1508 – Generic Security Service Application Program Interface, Setembro 1993.

[Lin93c] John Linn. RFC 1511 – Common Authentication Technology Overview, Setembro 1993.

[Lin96] John Linn. Internet Draft – Generic Security Service Application Program Interface, Version 2, Agosto 1996. (work in progress).

[LKB+94] Susan Landau, Stephen Kent, Clint Brooks, Scott Charney, Dorothy Denning, Whitfield Diffie, Anthony Lauck, Doug Miller, Peter Neumann, and David Sobel. Codes, Keys and Conflicts: Issues in U.S. Crypto Policy. Technical report, Association for Computing Machinery, Inc., Junho 1994.

[LM90] X. Lai and J. L. Massey. A Proposal For a New Block Encryption Standard. In *Advances in Cryptology - Eurocrypt 90 Proceedings*, pages 389–404, Berlin, 1990. Springer Verlag.

[LMKQ89] S.J. Leffler, M.K. McKusick, M.J. Karels, and J.S. Quaterman. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison–Wesley, Reading, Mass., 1989.

[LMS93] Jack B. Lacy, Don P. Mitchell, and W.M. Schell. Cryptolib: Cryptography in software. In *Proceedings of Usenix Unix Security Workshop IV*, pages 1–17, Santa Clara, California, 1993.

[Loc94] Harold W. Lockhart. *OSF DCE: Guide to Developing Distributed Applications.* J. Ranade Workstation Series. McGraw–Hill, 1994.

[Mil92] D. Mills. RFC 1305 – Network Time Protocol (NTP) – version 3, Março 1992.

[Mil96] D. Mills. RFC 2030 – Simple Network Time Protocol (SNTP) – version 4, Outubro 1996.

[MIT91] MIT Project Athena. *Kerberos V5 application programming library (DRAFT)*, Junho 1991.

[MNSS88] S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Slatzer. *Section E.2.1: Kerberos Authentication and Authorization System*. Project Athena Technical Plan, M.I.T., Cambridge, MA, Outubro 1988.

[Moy94] J. Moy. RFC 1584 – Multicast Extensios to OSPF (MOSPF), Março 1994.

[MS87] J.H Moore and G.J. Simmons. Cycle Structure of the DES with Weak and Semi–Weak Keys. In *Advances in Cryptology – CRYPTO 86 Proceedings*, pages 3–32. Springer–Verlag, 1987.

[MTHZ92] R. Molva, G. Tsudik, E. Van Herreweghen, and S. Zatti. KryptoKnight authentication and key distribution system. In *Proceedings of the 2nd European Symposium on Research in Computer Security*, pages 155–174, Toulouse, France, Novembro 1992. Springer Verlag.

[NS78] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, Dezembro 1978.

[NT94] B.C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, 32(9):33–38, Setembro 1994.

[oS77] National Bureau of Standards. Data Encription Standard – NBS FIPS PUB 46. U.S. Department of Commerce, Janeiro 1977.

[oS80] National Bureau of Standards. DES Modes of Operation – NBS FIPS PUB 81. U.S. Department of Commerce, Dezembro 1980.

[oST94a] National Institute of Standards and Technology. Data Signature Standard – NIST FIPS PUB 186. U.S. Department of Commerce, Maio 1994.

[oST94b] National Institute of Standards and Technology. Escrowed Encryption Standard – NIST FIPS PUB 185. Washington, D.C., Fevereiro 1994.

[oST95] National Institute of Standards and Technology. Secure Hash Standard – NIST FIPS PUB 180–1. U.S. Department of Commerce, Abril 1995.

[Per95] C. Perkins. Internet Draft – IP Encapsulation within IP, Outubro 1995.

[Pfl89] C.P. Pfleeger. *Security in Computing*. Prentice–Hall, Englewood Cliffs, N.J., 1989.

[Pos80a] J. Postel. RFC 768 – User Datagram Protocol (UDP), Agosto 1980.

[Pos80b] J. Postel. RFC 791 – Internet Protocol (IP), Setembro 1980.

[Pos80c] J. Postel. RFC 793 – Transmission Control Protocol (TCP), Setembro 1980.

[Pre93] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, Janeiro 1993.

[Ram94] Rick Ramsey. *All about Administrating NIS+*. SunSoft Press, Prentice Hall, 1994.

[Rih94] K. Rihaczek. Data Interchange and Legal Security – Signature Surrogates. *Computers & Security*, 13(4):287–293, Setembro 1994.

[Riv92a] R. L. Rivest. The RC4 Encryption Algorithm. RSA Data Security, Inc., Março 1992.

[Riv92b] R.L. Rivest. RFC 1320 – The MD4 Message–Digest Algorithm, Abril 1992.

[Riv92c] R.L. Rivest. RFC 1321 – The MD5 Message–Digest Algorithm, Abril 1992.

[Rob94] M.J.B. Robshaw. On Pseudo–Collisions in MD5. Technical Report TR–102, RSA Laboratories, Julho 1994.

[RP94] J. Reynolds and J. Postel. RFC 1700 – Assigned Numbers, Outubro 1994.

[RSA78] R.L. Rivest, A. Shamir, and L.M. Adleman. A Method for Obtaining Digital Signatures and Public–Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, Fevereiro 1978.

[Sch95a] Wolfgang Schneider. *SecuDE Overview (version 4.4)*. GMD, Institut für TeleKooperations Technik, Darmstadt, Germany, Março 1995.

[Sch95b] Wolfgang Schneider. *SecuDE: Principles of Security Operations (version 4.4)*. GMD, Institut für TeleKooperations Technik, Darmstadt, Germany, Março 1995.

[Sch95c] Wolfgang Schneider. *SecuDE: Security Applications Guide (version 4.4)*. GMD, Institut für TeleKooperations Technik, Darmstadt, Germany, Março 1995.

[Sch95d] Wolfgang Schneider. *SecuDE: Security Commands, Functions and Interfaces (version 4.4)*. GMD, Institut für TeleKooperations Technik, Darmstadt, Germany, Março 1995.

[Sch96] Bruce Schneier. *Applied Cryptography*. Programming/Security. John Wiley & Sons, Inc., 2nd edition, 1996.

[Sim95] W. Simpson. RFC 1853 – IP in IP Tunneling, Outubro 1995.

[SM96] C. Semeria and T. Maufer. Internet Draft – Introduction to IP Multicast Routing, Maio 1996. (informational).

[SNS88] J.G. Steiner, C. Neuman, and J.I. Schiller. Kerberos: An Authentication Service for Open Networked Systems. In *Proceedings of the Winter 1988 USENIX Conference*, pages 191–202, Fevereiro 1988.

[SRL96] Kevin Savetz, Neil Randall, and Yves Lepage. *MBONE: Multicasting Tomorrow's Internet*. IDG Books Worldwide, Inc., Abril 1996.

[Sta95] William Stallings. *Network And Internetwork Security: principles and practice.* Prentice Hall, 1st edition, 1995.

[Ste90] W. Richard Stevens. *UNIX Networking Programming.* Software Series. Prentice Hall, 1st edition, 1990.

[Ste96] W. Richard Stevens. *TCP/IP Illustrated Volume 1 – The Protocols.* Professional Computing Series. Addison–Wesley, Março 1996.

[Sun87] SunMicrosystems. RFC 1014 – XDR: External Data Representation Standard, Junho 1987.

[Sun88] SunMicrosystems. RFC 1050 – RPC: Remote Procedure Call Protocol Specification, Abril 1988.

[Sun91] SunSoft. Solaris ONC: Design and Implementation of Transport–Independent RPC. Solaris 2.0 White Papers, 1991.

[TA91] J.J. Tardo and K. Alagappan. SPX: Global authentication using public key certificates. In *Proceedings of the 12th IEEE Symposium on Research in Security and Privacy*, pages 232–244, Oakland, California, Maio 1991.

[Tuc79] W. Tuchman. Hellman Presents No Shortcut Solutions to DES. *IEEE Spectrum*, 16(7):40–41, Julho 1979.

[WABL93] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, Ashville, North Carolina, 1993.

[WBSL94] Y.C. Thomas Woo, Raghuram Bindignavle, Shaowen Su, and Simon S. Lam. SNP: An Interface for Secure Network Programming. In *Proceedings of the USENIX Summer Technical Conference, Boston, MA*, Junho 1994.

[Wie93] Michael Wiener. Efficient DES Key Search. In *Proceedings of Crypto 93*, 1993.

[WPD88] D. Waitzman, C. Partridge, and S. Deering. RFC 1075 – Distance Vector Multicast Routing Protocol (DVMRP), Novembro 1988.

[Wra91] John C. Wray. Generic Security Service Application Programming Interface – Overview and C bindings (Draft D.1). Secure Systems Group, Digital Equipment Corporation, Maio 1991.

[Wra93] John C. Wray. RFC 1509 – Generic Security Service API : C–bindings, Setembro 1993.

[WS96] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated Volume 2 – The Implementation.* Professional Computing Series. Addison–Wesley, Março 1996.

[Zim95] P. R. Zimmermann. *The Official PGP Users's Guide.* Mit Press, Boston, 1995.