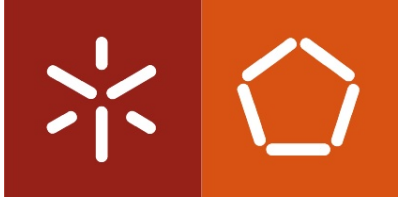
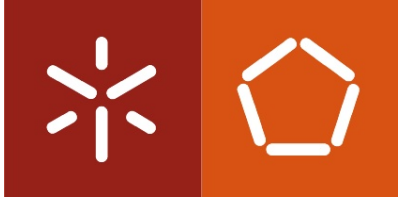


Universidade do Minho
Escola de Engenharia

Bernardo Luís Fernandes Portela

Segurança Criptográfica no Armazenamento e Partilha de Dados em Ambientes Cloud

Outubro de 2013



Universidade do Minho
Escola de Engenharia

Bernardo Luís Fernandes Portela

Segurança Criptográfica no Armazenamento e Partilha de Dados em Ambientes Cloud

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efectuado sob a orientação de
Doutor Manuel Bernardo Barbosa

Outubro de 2013

DECLARAÇÃO

Nome

Bernardo Luís Fernandes Portela

Endereço electrónico: BLFPortela@gmail.com Telefone: 258972509 / 927713000

Número do Bilhete de Identidade: 13742388

Título dissertação ☒ / tese ☐

Segurança Criptográfica no Armazenamento e Partilha de Dados em Ambientes Cloud

Orientador(es):

Manuel Bernardo Barbosa

Ano de conclusão: 2013

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

Mestrado em Engenharia Informática

É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE.

Universidade do Minho, 31 / 10 / 2013

Assinatura: Bernardo Portela

Agradecimentos

Ao meu orientador, professor Manuel Bernardo Barbosa, que, graças ao seu profissionalismo, experiência e conhecimento, me motivou e ajudou durante todo o percurso a desenvolver uma dissertação com qualidade. Um especial agradecimento por toda a dedicação demonstrada nas preciosas orientações, que encaminharam a minha forma de pensar e de escrever para o caminho correto.

Ao professor Rui Carlos Oliveira, que me deu o eventual, mas tão necessário, "puxão de orelhas", para orientar o meu trabalho no melhor sentido. Foi o seu interesse e o aconselhamento na área que me motivaram a seguir este tema.

Ao meu colega João Paulo, que, para além de disponibilizar uma amostra a ser utilizada no meu trabalho, sempre apresentou extrema disponibilidade para atender às dúvidas resultantes da minha inexperiência.

Aos meus pais, que apesar de todas as dificuldades por que passam, nunca duvidaram das minhas capacidades e sempre fizeram o seu melhor para me apoiar durante os momentos mais complicados. Não teria sido capaz de terminar este trabalho sem eles.

Aos meus amigos, que sempre estiveram por perto e nunca falharam quando foi importante. Por todas as brincadeiras, provocações e palhaçadas, que não deixavam espaço para tristeza e desânimo.

Finalmente, a todos os que, apesar de não serem aqui explicitamente mencionados, tenham contribuído para o desenvolvimento desta dissertação, direta ou indiretamente.

Resumo

Os avanços tecnológicos no modelo de $cloud$ têm vindo a possibilitar o aumento exponencial na quantidade de informação armazenável online. Serviços de partilha de ficheiros como Dropbox ou Google Drive já desempenham um papel importante entre os utilizadores, tornando a partilha e acessibilidade de ficheiros uma funcionalidade cada vez mais desejada.

A implementação de sistemas seguros está rigorosamente associada à noção de confiança. Os serviços de partilha utilizados na prática tendem a contemplar um nível de confiança leve, habitualmente considerando apenas adversários que não o provedor, o que inviabiliza estes sistemas para os utilizadores com informação sensível. Por outro lado, não depositar qualquer confiança no provedor é uma solução contraditória ao modelo, uma vez que força a maior parte do processamento a ser realizado localmente e a elevada performance dos servidores remotos mantem-se inutilizada.

Assim, os desafios associados ao desenvolvimento de segurança para $cloud$ consistem em adaptar o grau de confiança exigido, de modo a equilibrar performance, segurança e funcionalidade.

Esta dissertação propõe-se a realizar uma avaliação da segurança atual e das soluções alternativas existentes, bem como a implementação de um sistema de partilha de ficheiros seguro. Os objetivos iniciais passam pela análise das alternativas através de uma abordagem tanto teórica como prática, identificando os problemas, apresentando alternativas e testando a viabilidade das mesmas. Finalmente, foi implementado um sistema que vai aplicar o conhecimento contemplado numa perspetiva prática, seguido de uma validação de segurança que permita comparabilidade com sistemas atuais.

Abstract

Technological developments in the cloud model allow for an exponential growth in the online storage of information. File sharing services such as Dropbox or Google Drive already play an important role among the users, promoting the demand for information availability and data sharing.

The implementation of secure systems is strictly related to the concept of trust. In practice, data sharing systems tend to assume a weak level of trust, usually considering an adversary model that excludes the provider, reducing its usefulness for users with sensible information. On the other hand, not trusting the provider in any way is a solution that contradicts the model itself, as it forces the bulk of processing to be performed locally, and the remote server's high performance remains unused.

Therefore, the challenges in developing cloud security consist in adapting the trust level, balancing performance, security and functionality.

This thesis proposes an analysis of the current security and its alternative solutions, as well as an implementation of a secure file sharing system. These initial objectives consist on both a theoretical and practical approach to the subject, identifying the problems, presenting alternatives and testing their viability. Finally, the implemented system is to apply the solutions previously evaluated on a practical perspective, followed by a security validation that allows for comparison with modern systems.

Conteúdo

Acrónimos

AES Advanced Encryption Standard

CA Certification Authority

CC Common Criteria

CBC Cipher Block Chaining

CCA Chosen-Ciphertext Attack

CE Convergent Encryption

CL-PKC Certificateless Public Key Cryptography

CL-PKE Certificateless Public Key Encryption

CLS Certificateless Signature

CPA Chosen-Plaintext Attack

DES Data Encryption Standard

DoS Denial-of-Service

HCE Hash and Convergent Encryption

IBE Identity-Based Encryption

KGC Key Generation Center

MLE Message-Locked Encryption

PKI Public Key Infrastructure

PP Protection Profile

RCE Randomized Convergent Encryption

SAR Security Assurance Requirements

SFR Security Functional Requirements

ST Security Target

STC Strong Tag Consistency

TC Tag Consistency

TOE Target of Evaluation

UEK User Encryption Key

USK User Signature Key

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

No mundo digital, os avanços tecnológicos possibilitaram o aumento exponencial na quantidade de informação armazenável. Este crescimento leva, por sua vez, a uma necessidade cada vez maior de funcionalidades de partilha de ficheiros. Afinal, se os utilizadores têm cada vez mais dados guardados, mais desses dados vão ser úteis quando partilhados com outras pessoas [23].

A facilidade de partilha veio a aumentar com o melhoramento de redes que permitem a transmissão mais rápida e dinâmica de ficheiros e, inevitavelmente, este foi um ponto que veio a acentuar-se com a evolução da $internet$. A generalização do serviço e as ligações de melhor qualidade levaram os utilizadores a sentir-se constantemente ligados através deste sistema, querendo cada vez mais poder interagir uns com os outros com a mesma facilidade e rapidez que o fariam na vida real [14].

O modelo de $cloud$ consiste na utilização de servidores remotos para processamento e armazenamento. Recorrendo a este paradigma, os clientes são cobrados apenas pelos recursos necessários a cada momento, beneficiando da elasticidade do modelo e de uma redução no investimento inicial em infraestrutura. O interesse em utilizar a $internet$ como um meio de partilha conjuga naturalmente com a noção de $cloud$, que agrega informação de diferentes utilizadores num mesmo $datacenter$.

Serviços de partilha de ficheiros como Dropbox ou Google Drive já desempenham um papel importante entre os utilizadores, tornando a partilha e acessibilidade de ficheiros uma funcionalidade cada vez mais desejada. No entanto, o ignorar de medidas de segurança em prol de melhor performance põe em causa a utilização destes sistemas [42].

A implementação de sistemas seguros está rigorosamente associada à noção de confiança, sendo que esta define as entidades contra as quais se planeia contrariar ameaças. Os serviços de partilha utilizados na prática tendem a contemplar um nível de confiança leve, habitualmente considerando apenas adversários que não o provedor. Atitudes como estas tornam os sistemas inviáveis aos utilizadores com informação sensível, que exigem garantias mais fortes.

Por outro lado, não confiar no provedor e aplicar técnicas criptográficas do lado do cliente é uma solução contraditória ao modelo, uma vez que a maior parte do processamento é realizado localmente e a elevada performance dos servidores remotos mantem-se inutilizada. Esta opção de manter a informação cifrada durante a transmissão e armazenamento tende a drasticamente reduzir a performance de um sistema, bem como a eliminar funcionalidades que exijam processamento remoto.

Assim, os desafios associados ao desenvolvimento de segurança para $cloud$ consistem em adaptar o grau de confiança exigido, de modo a equilibrar performance, segurança e funcionalidade.

1.2 Partilha de ficheiros na $cloud$

O modelo de $cloud$

O modelo $cloud$ define um modelo computacional que recorre à utilização de servidores localizados remotamente, quer para processamento como para armazenamento. Este termo é utilizado para fazer referência tanto às aplicações que são disponibilizadas como serviços online, como ao $hardware$ e aos $datacenters$ que providenciam esses mesmos serviços.

Numa abordagem não orientada à nuvem, os recursos das empresas são limitados pelo investimento inicial. Essa decisão é importante e complexa, uma vez que uma escolha menos adequada pode resultar num excesso de recursos para o serviço oferecido ou numa incapacidade de atender a um fluxo de pedidos maior que o previsto.

A adaptabilidade do modelo de $cloud$ possibilita às empresas um investimento inicial reduzido e uma melhor resposta às flutuações do seu serviço. Através desta forma de escalar $pay\text{-}as\text{-}you\text{-}go$, apenas são utilizados os recursos necessários a cada momento, evitando a problemática do investimento inicial e oferecendo ao sistema capacidades de adaptação a pontos críticos de procura [6, 31]. Esta elasticidade natural da $cloud$ é uma caraterística central no agressivo crescimento deste modelo.

As ideias primordiais do modelo de $cloud$ surgiram nos anos 60. Joseph Licklider visionou uma ligação global, que permitia acesso a programas e informação de qualquer site, a qualquer momento. Segundo Margaret Lewis, diretora de marketing da AMD, esta é uma perspetiva muito semelhante ao que hoje é considerado $cloud\ computing$ [44]. O estado atual da $cloud$ apresenta-se como um modelo com capacidades de processamento aparentemente ilimitadas. Um exemplo do poder destes servidores foi dado pela Amazon que, em 2011, participou na competição de top500 supercomputadores com o seu serviço, alcançando a posição 42 [25]. Isto significou que era possível ter acesso a poder de processamento igualável aos melhores computadores do mundo sem ter de adquirir a maquinaria associada.

Serviços de partilha de ficheiros

O potencial de armazenamento apresentado pela $cloud$ tornou propícia a implementação e utilização de serviços de armazenamento de ficheiros sobre este modelo. A existência dos mesmos tem, como consequência, a procura por funcionalidades de partilha e consequente desenvolvimento de sistemas que respondem a esses problemas.

A noção de partilha de dados refere-se apenas à possibilidade de os tornar acessíveis a terceiros. No entanto, esta funcionalidade tem vindo a evoluir, permitindo sincronização de pastas, $backup$ automático e divisão de permissões. Sistemas como Dropbox ou Google Drive são exemplos de ofertas de mercado populares, que abordam partilhas de ficheiros com caraterísticas adicionais. O desenvolvimento de mais avançadas funcionalidades de partilha implica o considerar de requisitos de segurança adaptados a esses propósitos.

Proteger a integridade e confidencialidade dos dados na presença de entidades não autorizadas é o requisito mais básico, uma vez que é necessário na forma mais simplista da partilha. Salvaguardar o controlo de acessos contra adulteração também é um requisito essencial, sempre que o sistema recorra a um mecanismo que associe ficheiros a utilizadores com permissões de acesso. Detetar tentativas de personificação também pode ser

relevante, uma vez que não implica adulterar o controlo de acessos, mas a autenticação em si. Limitar os utilizadores às suas permissões representa outra exigência dos sistemas que permitam diferentes tipos de partilha, como $read\text{-}only$ e $read\text{-}write$. A definição destes requisitos é fulcral no desenvolvimento de sistemas seguros e vai consistir num dos objetivos iniciais deste trabalho.

1.3 Modelos de confiança

Uma prática habitual na utilização de sistemas digitais consiste na distribuição de diferentes poderes e permissões entre as entidades que fazem parte do mesmo. Como tal, é importante a definição da confiança depositada nessas diferentes entidades, de modo a atender aos requisitos de segurança exigidos ao sistema.

As abordagens práticas tendem a assumir que os provedores de $cloud$ são confiáveis. Isto permite que certas medidas de segurança possam ser realizadas do lado dos servidores, como a cifragem ou o controlo de acessos. Desta forma, o sistema atinge uma melhor performance e os servidores continuam a poder realizar operações sobre a informação, uma vez que têm acesso ao texto-limpo.

Porém, a necessidade de manusear informação confidencial ou sensível exige um maior cuidado com estas assunções, tornando este nível de confiança impraticável. Não assumindo confiança no provedor, as medidas implementadas vão ter de contemplar ameaças provenientes dos servidores que armazenam informação, implicando mais processamento do lado do cliente e reduzida performance do serviço.

Relaxar este modelo permite que as entidades confiáveis funcionem com menos obstáculos e, consequentemente, maximizem a sua contribuição para o sistema em geral. Neste trabalho vão ser avaliados os diferentes níveis de confiança que podem ser depositados num ambiente de partilha de ficheiros na $cloud$ e de que forma esses influenciam as medidas de segurança. A finalidade desta análise é a de encontrar um grau de confiança adequado à utilização para clientes com informação sensível, mantendo uma performance viável na aplicação prática.

1.4 Objetivos e contribuições

Atendendo aos problemas discutidos acima, o trabalho realizado nesta dissertação pode-se dividir em quatro objetivos principais. Os dois iniciais são focados na análise do estado da arte prático e teórico, enquanto que os restantes abordam as contribuições em si:

i. Análise de segurança dos sistemas de partilha. Esta fase vai consistir numa abordagem do estado da arte da partilha de ficheiros na $cloud$, de forma consistente com a metodologia de análise de segurança para sistemas digitais. Espera-se conseguir uma imagem das medidas de segurança impostas pelos serviços e das respetivas fraquezas.

ii. Estudo de soluções alternativas. Observando os problemas assinalados em (i), esta fase consiste na investigação de protocolos que apresentem respostas às insuficiências atuais. O objetivo consiste em listar e definir os diferentes avanços teóricos e experimentais, bem como contextualizar a sua utilização na partilha de ficheiros segura.

iii. Experimentação das abordagens teóricas, isto é, a recolha de dados práticos que permitam avaliar as alternativas referidas em (ii). Esta contribuição passa pela implementação de um $benchmark$ que deve executar testes com as diferentes soluções alternativas identificadas, bem como a análise dos resultados obtidos com o mesmo.

iv. Implementação de um sistema de partilha de ficheiros seguro, recorrendo às abordagens estudadas. Este sistema deve contemplar os requisitos funcionais e de segurança estudados em (i), utilizar abordagens criptográficas referidas em (ii) e, apoiado em dados obtidos em (iii), apresentar uma validação de segurança que permita comparabilidade com serviços atuais.

O objetivo central deste trabalho passa por abordar as alternativas aos sistemas existentes e implementar um sistema que faça uso dessas soluções numa aplicação prática. Desta forma, é realizada uma contribuição para os sistemas seguros de $cloud$ apoiada por dados experimentais, bem como implementada uma prova de conceito que pode ser utilizada como base para futuros desenvolvimentos.

1.5 Organização do documento

Esta dissertação está estruturada em sete capítulos seguidos de três de anexos, contendo informação adicional ao trabalho desenvolvido. Assim, nos capítulos 2, 3 e 4 são abordados

e estudados os conceitos relacionados com a dissertação. No capítulo 2, é analisada a segurança dos sistemas e da informação, enquadrando e servindo como base à abordagem realizada nos restantes capítulos. No capítulo 3, é explorado o estado da arte da partilha de ficheiros na $cloud$, atendendo às necessidades funcionais e a uma análise de segurança. O capítulo 4 é dedicado à avaliação dos obstáculos às soluções práticas atuais, apresentando alternativas teóricas de protocolos e de primitivas de segurança.

O capítulo 5 consiste na implementação e análise de resultados de um $benchmark$. O propósito do mesmo é o recolher de dados que possam servir como argumento para as primitivas exploradas no capítulo 4.

O capítulo 6 engloba a implementação e validação de um sistema de partilha de ficheiros seguro. A sua composição passa por: i. especificações de implementação, ii. componentes e operações, iii. análise de segurança, iv. ambiente e configuração e v. validação.

O capítulo 7 consiste na apresentação de conclusões relacionadas com o trabalho realizado, bem como propostas para desenvolvimento futuro.

Em anexo, encontra-se material mais específico do que o abordado ao longo da dissertação, de modo a manter o nível de detalhe sem pesar na leitura do documento em geral. Neste capítulo podem-se consultar construções de esquemas criptográficos, descrições extensivas de operações do sistema e diagramas de atividade.

Capítulo 2

Segurança da informação e de sistemas

2.1 Segurança de sistemas informáticos

Documentos de conteúdo confidencial, ou de tal forma crítico que exigem um cuidado acrescido no seu manuseamento, são habitualmente categorizados como "informação sensível". Este termo pode ser atribuído a quaisquer dados que devem ser protegidos contra divulgação, adulteração e ataques à sua acessibilidade. Sistemas seguros, por sua vez, procuram oferecer aos utilizadores um serviço reforçado com garantias de confiabilidade na sua utilização, de modo a aceder a estas necessidades.

No entanto, apesar das medidas de salvaguarda serem a origem deste tipo de sistemas, é importante ter em conta a usabilidade dos mesmos. Um sistema seguro ideal não só fornece certas garantias aos utilizadores, como também pode ser utilizado para os mesmos propósitos que outros sistemas do mesmo género, uma vez que as medidas de segurança que envolvam a perda de funcionalidades ou a redução excessiva de performance podem tornar um serviço inutilizável na prática. A aplicação de medidas de segurança deve contemplar estas noções, focando-se não numa perspetiva binária de "seguro" ou "inseguro", mas sim em conseguir as garantias de segurança necessárias para o sistema ser seguro o suficiente, mantendo uma qualidade de utilização adequada.

Segundo [30], é aconselhado seguir várias diretrizes na aproximação à segurança digital, devendo esta ser custo-eficiente, centrada no objetivo da organização, de periódica

verificação, dependente de fatores sociais, entre outras. Assim, para uma abordagem mais explícita e minuciosa, a segurança do sistema exige a definição de responsabilidades para os diferentes intervenientes, o estabelecimento de ameaças e riscos ao bom funcionamento do sistema e a delimitação da política de segurança da empresa perante esses critérios.

2.1.1 $Roles$, ameaças e políticas de segurança

Responsabilidades e $roles$

Todos os envolvidos na utilização de um certo sistema estão responsáveis pela sua segurança. No entanto, como a prática habitual das organizações envolve a divisão de cargos entre os seus funcionários, a definição de $roles$ vai permitir que o sistema possa atribuir as diferentes responsabilidades na sua operação. A título de exemplo:

- Os administradores devem estabelecer o programa e as políticas de segurança a seguir, delimitando as prioridades e os objetivos que procuram servir os interesses da organização.
- Os gestores de segurança devem ser encarregues de gerir interações entre os diferentes elementos que podem interferir na segurança de um sistema, bem como da gestão da segurança digital aplicada ao mesmo.
- Os utilizadores devem ser responsabilizados pelo detalhar das suas exigências a nível de segurança perante o sistema.

É aconselhável que esta divisão de responsabilidades seja feita recorrendo a grupos de intervenientes, aqui tendo sido abordados apenas 3, dependendo estes grupos do caso específico em que estão inseridos. Uma divisão granular de obrigações simplifica a utilização e torna o processo de avaliação mais fácil e detalhado.

Ameaças e riscos

Os sistemas informáticos são vulneráveis a várias ameaças, que podem causar diferentes tipos de danos. Estas ameaças englobam atividades de entidades externas ($hackers$) que divulgam informação privada, ações de utilizadores maliciosos do sistema que corrompem a informação, ataques de $denial\text{-}of\text{-}service$ (**DoS**) que levam à inacessibilidade dos dados, entre outros. Uma vista geral do tipo de ameaças presentes no sistema vai permitir que a

salvaguarda e as medidas de recuperação de recursos sejam mais adequadas aos acontecimentos.

Como nas responsabilidades, as ameaças também costumam ser agrupadas por tipo, agregando: erros e omissões, fraude e roubo, sabotagem por parte de empregados, perda de infraestruturas físicas, $hackers$ maliciosos, espionagem industrial, código malicioso ou ameaças à privacidade pessoal. Um conhecimento vasto das ameaças é crucial para a implementação de segurança custo-eficiente, uma vez que, quanto mais pormenorizadamente forem conhecidos os adversários ao funcionamento do sistema, mais específicas e adaptadas vão ser as respetivas contramedidas.

As atitudes perante as ameaças avaliadas nunca conseguem elimina-las por completo, apenas reduzir a probabilidade da sua ocorrência. O processo de avaliação de risco, redução de risco e manutenção das medidas para manter o risco em níveis aceitáveis chama-se gestão de risco.

A avaliação de risco consiste em:

1. Identificar bens e ameaças presentes no sistema, explicitando as componentes abordadas.

2. Avaliação de consequências, isto é, dos danos causados pelo sucesso das ameaças nos bens abordados.

3. Análise de salvaguarda, atribuindo o grau em que uma dada medida consegue, realmente, proteger os bens perante as ameaças.

4. Análise de vulnerabilidade, contrária ao ponto anterior, avaliando o quão vulnerável é o sistema, relativo à fraqueza das medidas aplicadas (ou até à ausência das mesmas).

5. Análise de possibilidade, contemplando o quão provável é que uma dada ameaça ocorra no sistema.

A boa inclusão deste processo na segurança de um sistema depende da aplicação inteligente de medidas de redução de risco, devendo estas não deteriorar a performance do sistema por demais, e da aceitação do risco residual, tornando-se explícito o que o sistema considera “seguro o suficiente” e o $racionale$ associado ao estabelecimento desse critério.

Políticas de segurança

Todas as implementações e escolhas de medidas resultam em políticas de segurança que devem ser relembradas para futuras avaliações e reavaliações. A escrita de documentos tão gerais como estes é muitas vezes auxiliada por *standards* organizacionais, diretrizes ou procedimentos.

As políticas de segurança podem ser divididas em três tipos:

- *Program policy* - Estas são políticas de segurança que consistem num estabelecimento das medidas gerais de segurança e da estrutura do sistema. Devem abordar os objetivos do programa (o que se pretende que o programa consiga realizar), o âmbito (quais recursos vão ser abordados pelo documento) e as responsabilidades (referidas acima).

- *Issue-specific policies* - Estas são políticas de segurança que abordam tópicos mais específicos, como decisões de planeamento ou metodologias particulares de gestão de risco. Devem ser constituídas por uma introdução ao discutido (o *issue*), a posição da organização perante esse assunto, a aplicabilidade e as responsabilidades associadas.

- *System-specific policies* - Estas são políticas de segurança de baixo nível, sendo específicas a um sistema e apontando a colmatar falhas de precisão nas descrições de segurança. O seu conteúdo pode ser dividido em duas partes, a definição de objetivos de segurança, descrevendo com precisão as caraterísticas de segurança e o grau com o qual se pretende reforçar as medidas associadas, e as regras operacionais, delimitando os *roles* que podem desempenhar determinadas tarefas e as regras que devem seguir ao faze-lo.

2.1.2 *Common criteria*

Em 2.1.1 foi referido que, para auxiliar na documentação de políticas de segurança, é aconselhada a utilização de *standards*. O objetivo destes é o de permitir que as organizações recorram a *guidelines* exaustivamente estudadas para desenharem os seus sistemas de segurança e de normalizar e facilitar a avaliação de confiabilidade, através de uma estrutura "*if ... then*".

Conceito

O *common criteria* (**CC**) [34] é um *standard* que permite a comparabilidade entre resultados de análises de segurança independentes, oferecendo um conjunto de requisitos de segurança e medidas de confiabilidade envolvidas no processo de avaliação. As categorias de proteção envolvem a confidencialidade, integridade e disponibilidade.

Este *standard* conjuga três perspetivas principais de utilização. **Avaliação**, recorrendo aos critérios associados ao *protection profile* (**PP**) para avaliar os requisitos de segurança de um *target of evaluation* (**TOE**). **Desenvolvimento**, recorrendo às *guidelines* para definição de *security targets* (**ST**) utilizados na implementação e aos critérios de responsabilização para apoiar a avaliação do **TOE**. **Utilização**, recorrendo aos **PP** para expressar as suas necessidades de segurança de uma forma inambígua, ou às avaliações de diferentes **TOE**s para fins comparativos.

O **CC** começa por definir as entidades a proteger, denominadas *assets*, que podem passar por conteúdos de ficheiros, disponibilidade de um serviço ou acesso a uma determinada função. Depois, é importante definir o ambiente no qual essas entidades vão ser colocadas, como uma rede interna, entre escritórios, ou a internet. Após estes pontos, identificam-se as entidades que podem causar problemas ao sistema (*threat agents*) as ameaças (*threats*) e as contramedidas (*countermeasures*) inerentes do sistema caraterizado, analisando-se o risco como o resultado de uma dada ameaça enfrentada através de certas contramedidas. Esta dinâmica é representada na figura 2.1.

Requisitos, *security targets* e *protection profiles*

Para a definição de requisitos de segurança, são apresentados dois tipos abordados:

- *Security functional requirements* (**SFR**), contemplando os objetivos de segurança para o **TOE**.
- *Security assurance requirements* (**SAR**), descrevendo como deve ser estabelecida confiança na forma como o **TOE** atinge os **SFR** apresentados.

Os **ST**s têm duas finalidades: apresentar o que vai ser avaliado, enumerando as propriedades do **TOE** e tornando inteligível o foco da avaliação e explicar o que foi avaliado, servindo como descrição do compromisso entre exigências e nível de garantia providenciado. A estrutura dos mesmos passa por uma introdução, definição do problema de segurança, objetivos de segurança, requisitos de segurança (**SFR** e **SAR**) e especificação do **TOE**.

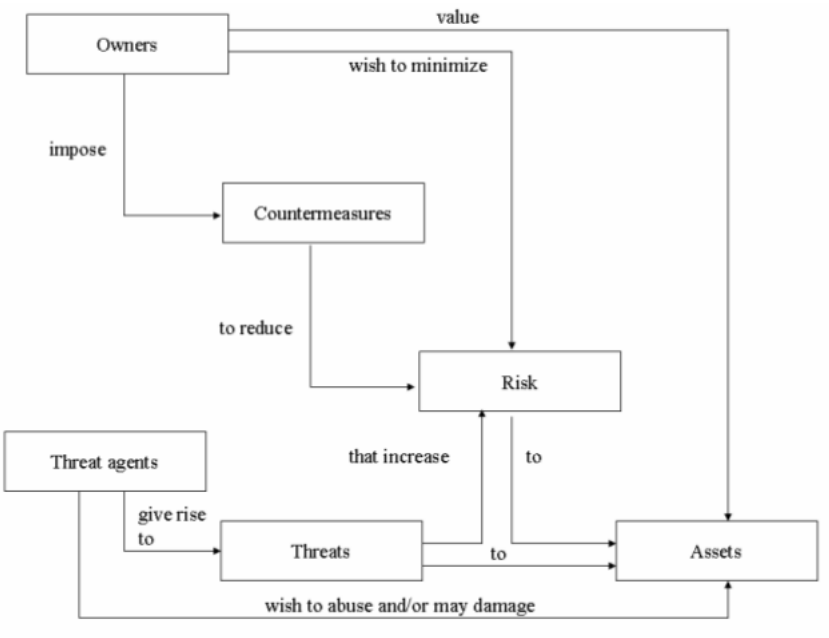


Figura 2.1: Avaliação de risco no ***CC*** *- Imagem de [30]*

Os **PP**s e os *packages* são dois construtores essenciais do **CC**. Um *package* pode ser funcional, contendo apenas **SFR**s, ou de garantias, contendo apenas **SAR**s, o objetivo dos mesmos é serem reutilizáveis, sendo importante a sua simplicidade e interrelacionabilidade. Um **PP**, por outro lado, descreve um tipo de **TOE**, abordando as exigências gerais do mesmo, podendo ser utilizado como *template* para diferentes **ST**s.

Avaliação

Avaliar um sistema para o **CC** significa avaliar o(s) **TOE**(s), o que implica recorrer aos construtores referidos previamente (fig. 2.2).

1. Inicialmente, é avaliado o **PP**, consoante os requisitos gerais contemplados. Após sucesso de avaliação, este pode ser reutilizado para ponto de partida de **ST**s, evitando gastar recursos na repetição deste passo inicial.

2. É avaliado o **ST**, consoante as caraterísticas do **TOE** e os objetivos de segurança, verificando se os **SFR**s são verificados com os **SAR**s descritos.

3. Por final, é analisado o **TOE**, consoante as descrições específicas do mesmo. Caso estas últimas duas avaliações se verificarem, é obtido um *conformance claim* que

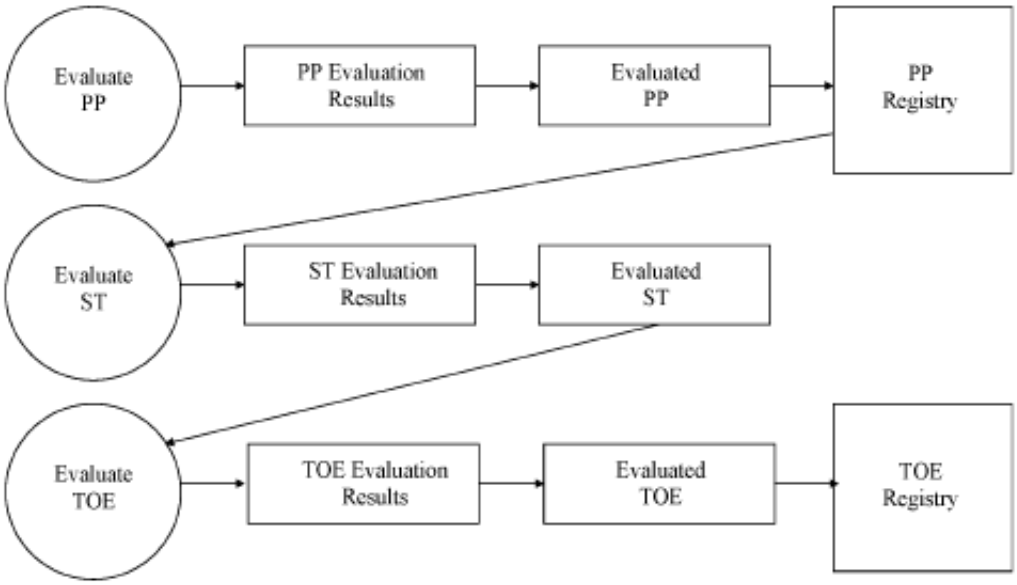


*Figura 2.2: Avaliação **PP-ST-TOE** em **CC** - Imagem de [30]*

descreve a versão de **CC** na qual foi avaliado do **PP**/**ST**, bem como alguns critérios adicionais, como especificações de **SFR**s/**SAR**s ou comentários relevantes.

Os resultados obtidos desta avaliação podem ser utilizados pelo dono da entidade ($asset$) na decisão de aceitar os riscos inerentes à sua utilização, ou no caso de querer reavaliar o caso perante outras variáveis, como outro ambiente ou contemplando diferentes vulnerabilidades.

2.2 Segurança da informação

Nesta secção explicitam-se as diferentes primitivas e noções criptográficas abordadas ao longo da dissertação. Estas primitivas serão descritas pela sua funcionalidade, pelo modelo de segurança associado e, quando adequado, por construções das mesmas.

2.2.1 Funções de $hash$ resistentes a colisões

Funções de $hash$ são funções que recebem $strings$ de tamanho arbitrário, comprimindo-as em $strings$ mais pequenas, habitualmente utilizadas em estruturas de informação para

realizar $lookups$ rápidos. No contexto de segurança, uma função de $hash$ é tão útil quanto mais difícil for encontrar uma colisão. Esta é a diferença fundamental entre uma função de $hash$ clássica e uma função de $hash$ criptográfica.

Uma função de $hash$ é um par de algoritmos (Gen, H) tal que:

- Gen é um algoritmo probabilístico que recebe um parâmetro de segurança 1^n e devolve uma chave k.
- Existe um polinómio l de forma que H é um algoritmo determinístico que recebe uma chave k e uma string $m \in \{0,1\}^*$ e devolve uma string $\{0,1\}^{l(n)}$.

As funções de $hash$ criptográficas devem ser $one\text{-}way$ e $collision\ resistant$. Uma função ser $one\text{-}way$ implica que é de fácil computação um $H(x)$ tendo um x, mas que é $hard$ o cálculo do seu inverso, isto é, obter o x tendo o $H(x)$. Uma função de $hash$ é $collision\ resistant$ quando é $hard$ o problema de encontrar um x e um x' de tal forma que $x \neq x'$ e $H(x) = H(x')$.

Habitualmente, são utilizadas funções de $hash$ que utilizam uma chave fixa e não contemplam a utilização de um Gen. Exemplos de funções desse género utilizadas na prática são o **MD5** e o **SHA-1**, ambas funções de compressão com tamanho fixo e a utilizarem chaves públicas, ao invés de um algoritmo de geração de chaves.

O **MD5** já não oferece garantias de segurança adequadas, e o **SHA-1** ainda é muito utilizado. Porém, existe uma tendência para a utilização de técnicas que recorram a chaves com maior tamanho, como o **SHA-256** ou **SHA-512**.

2.2.2 Cifras simétricas

Atualmente, um sistema de cifragem é considerado seguro se um adversário não conseguir computar nenhuma função aplicável ao texto-limpo no criptograma. Esta propriedade pode ser analisada em diferentes cenários, sendo importante a definição das respetivas ameaças. Para esta avaliação, deve ser definido o que é considerado quebrar a segurança de um sistema e qual é o assumido poder computacional dos adversários, podendo-se afirmar que um esquema criptográfico é seguro para uma designada tarefa se um adversário com um dado poder não conseguir quebrar o sistema. Atualmente, os diferentes cenários de ataque são:

- $Ciphertext\text{-}only\ attack$: um adversário observa um criptograma e tenta derivar a respetiva mensagem.
- $Known\text{-}plaintext\ attack$: um adversário conhece um par de texto-limpo/criptograma cifrados utilizando uma dada chave e tenta derivar a mensagem de um outro criptograma.
- $Chosen\text{-}plaintext\ attack$ (**CPA**): Um adversário que consegue obter a cifragem de mensagens à sua escolha tenta derivar um texto-limpo de um criptograma.
- $Chosen\text{-}ciphertext\ attack$ (**CCA**): Um adversário que consegue obter a decifragem de criptogramas à sua escolha tenta derivar um texto-limpo de um criptograma que ainda não decifrou diretamente.

O $design$ de esquemas de cifragem simétrica visa a indistinguibilidade entre criptogramas. Na construção de esquemas com tamanho variável de mensagens, são frequentemente utilizadas permutações pseudoaleatórias. Estes esquemas chamam-se cifras por blocos e recorrem a diferentes modos de operação.

$Cipher\ Block\ Chaining$ (**CBC**) é um modo de operação para este tipo de esquemas que recorre a um vetor de inicialização (**IV**), sendo que a cifragem parte de uma permutação que depende do bloco anterior (utilizando **IV** para o primeiro bloco). Como se trata de uma cifragem probabilística, o esquema satisfaz a noção $standard$ de segurança **IND-CPA**. Toda a cifragem tem de ser realizada sequencialmente (uma vez que depende do criptograma).

O $Advanced\ Encryption\ Standard$ (**AES**) é uma cifra por blocos criada para substituir o $Data\ Encryption\ Standard$ (**DES**) que utiliza uma rede de permutação-substituição no processo de cifragem. São realizadas várias rondas, dependendo do tamanho de chave adotado, em que cada uma faz uso de um $array$ bidimensional de estado e opera em quatro fases, resumidamente descritas de seguida:

1. $AddRoundKey$: Derivação de uma chave de 16 $bytes$ a partir da chave mestra.
2. $SubBytes$: Operação de substituição de $bytes$.
3. $ShiftRows$: Operação de $shift$ às linhas do $array$ de estado.
4. $MixColumns$: Operação de transformação linear invertível sobre as colunas do $array$.

O **AES** é considerado, neste momento, uma instanciação plausível de uma função/permutação pseudoaleatória.

2.2.3 Cifras de chave pública, certificados e PKIs

Num cenário de cifragem com chave pública, a entidade recetora gera um par de chaves: a chave pública (pk) e a chave privada (sk): (pk, sk). A chave pública é utilizada pelo emissor para cifragem e, por sua vez, o recetor utiliza a chave privada para decifrar o recebido. A construção destes esquemas passa pela utilização de um conjunto de três algoritmos: (Gen, Enc, Den) e opera como se segue:

- $Setup$: Recebendo um parâmetro de segurança 1^n, entrega $(pk, sk) \leftarrow Gen(1^n)$.
- $Encrypt$: Recebendo pk e uma mensagem m, entrega $c \leftarrow Enc_{pk}(m)$.
- $Decrypt$: Recebendo sk e um criptograma c, entrega $m \leftarrow Dec_{sk}(c)$.

Criptografia de chave pública pode também ser utilizada para a geração e verificação de assinaturas digitais. O propósito destas assinaturas é o de validar a autenticidade e integridade de informação, que pode ser verificada por qualquer entidade com acesso à respetiva chave pública. Assinaturas digitais desfrutam da propriedade de não-repúdio, significando que a entidade responsável por uma dada assinatura é, irrefutavelmente, o proprietário da chave privada associada.

Nesta aplicação de criptografia assimétrica, a entidade emissora gera um par de chaves: a chave pública (pk) e a chave privada (sk): (pk, sk). O emissor gera uma assinatura associada a uma dada mensagem, sendo posteriormente transferida para o recetor que, possuindo a chave pública respetiva, pode verificar a validade do recebido. Estes esquemas seguem uma construção baseada num tuplo de três algoritmos $(Gen, Sign, Vrfy)$:

- $Setup$: Recebendo 1^n, entrega $(pk, sk) \leftarrow Gen(1^n)$.
- $Sign$: Recebendo sk e uma mensagem m, entrega $\sigma \leftarrow Sign_{sk}(m)$.
- $Verify$: Recebendo pk, uma mensagem m e uma assinatura σ, entrega $b \leftarrow Vrfy_{pk}(m, \sigma)$. Este b toma o valor de 1 quando a assinatura é válida e 0 quando esta não se verificar.

Uma $Public\ Key\ Infrastructure$ (**PKI**) tem como objectivo a gestão segura e eficiente de chaves e certificados para permitir essa mesma utilização. Esta define-se pelo conjunto de $hardware$, $software$, pessoas, políticas e procedimentos necessários para criar, gerir, armazenar, distribuir e revogar certificados de chave pública.

2.2.4 *Identity-Based Encryption* e *Certificateless Cryptography*

Num cenário que assume a utilização de *Identity-Based Encryption* (**IBE**), a chave pública de uma identidade é gerada a partir de um identificador da mesma, recorrendo-se a uma *third party* de confiança para gerar chaves privadas. Esta entidade externa está responsável por distribuir as chaves privadas apenas às entidades que se consigam autenticar perante a mesma.

Através do seu modo de funcionamento, o **IBE** contorna a limitação de recorrer a canais autenticados e certificados para reforçar as chaves públicas com uma nova dificuldade de autenticação de chaves privadas. Simplificando a geração de chaves públicas e reduzindo as exigências computacionais que advêm da verificação de certificados [29], o **IBE** centraliza processamento na *third party*. Esta desvantagem assume duas naturezas: i. olhando o problema de uma perspetiva funcional, é exigida uma disponibilidade muito grande à *third party*, devendo esta estar maioritariamente acessível para realizar autenticações e gerar chaves privadas com uma performance adequada ao funcionamento do serviço, ii. tendo em conta que a *third party* é responsável por gerar e distribuir todas as chaves privadas, cria-se um *key escrow* na própria, podendo esta armazenar localmente uma cópia das chaves geradas e ganhando acesso a toda a informação que tenha sido cifrada utilizando **IBE**.

Técnicas de *threshold* [21, 28], utilizando múltiplas entidades de geração de chaves privadas, podem ser aplicadas para amenizar este problema, mas estas opções acabam por exigir um peso adicional de comunicação e infraestrutura. Adicionalmente, caso a entidade de confiança seja comprometida, as consequências são mais graves que na utilização de um **PKI** tradicional, reduzindo a aplicabilidade do **IBE** para grupos pequenos de aplicações com reduzidos requisitos de segurança.

O aspeto da **IBE** que evita a utilização de certificados é interessante numa perspetiva funcional, mas assumir confiança total numa entidade nem sempre é uma atitude que se pode tomar, quanto mais permitir um *key escrow* que garanta acesso a toda a documentação armazenada. De modo a contornar este problema, foi introduzido um novo paradigma criptográfico: *Certificateless public key cryptography* (**CL-PKC**) [3].

Um sistema de **CL-PKC** também recorre uma terceira entidade para geração de chaves, o *key generation center* (**KGC**) mas, contrariamente ao que acontece no **IBE**, esta entidade não tem acesso às chaves privadas dos intervenientes. Isto acontece porque o **KGC**

fornece uma chave privada parcial D_A, computada a partir do identificador ID_A e de uma chave mestra s. A entidade A agora utiliza a chave parcial D_A, alguma informação secreta Q_A e os parâmetros públicos do **KGC** para gerar a chave pública Pub_A. Esta chave pública é posteriormente partilhada com as restantes entidades do sistema que, através dos parâmetros públicos disponibilizados pelo **KGC**, podem verificar a sua validade.

Esta é uma técnica que não beneficia da caraterística simplista de geração de chaves públicas do **IBE**, mas em contrapartida permite eliminar o problema do $key\ escrow$, reduzindo o poder da entidade de geração de chaves.

Neste modelo, as entidades publicam as próprias chaves públicas sem recorrer a certificados ou outras medidas de segurança. No entanto, ao contrário do que aconteceria num sistema clássico com **PKI**, a substituição da chave pública por um adversário, sem quaisquer medidas adicionais, não traria grande benefício ao atacante. A título de exemplo, no caso em que um adversário substitua a chave pública de A por uma chave falsa Pub_A, este não conseguirá decifrar a informação cifrada com Pub_A sem a chave privada parcial D_A, que apenas se obtém quando autenticado como A perante o **KGC**. Caso, por outro lado, o **KGC** queira tentar decifrar a informação no sistema com os dados a si acessíveis, não se espera que tenha sucesso, uma vez que a decifragem de informação cifrada com Pub_A exige, para além da privada parcial D_A, um valor secreto Q_A, inacessível ao **KGC**.

Esta nova abordagem assume que o **KGC** não realiza nem colabora em ataques às identidades, onde se podia fazer passar por entidades, gerando a chave privada e disponibilizando a chave pública associada. As melhorias a este nível residem no afastar da confiança total depositada pelo **IBE** na $third\ party$, que permitia acesso total à informação cifrada com chaves geradas pela mesma. Neste sentido, o **CL-PKC** encontra-se num ponto intermédio entre o **PKC** com certificados e o **IBE**.

A implementação de $Certificateless\ cryptography$ de adotamos nesta dissertação é baseada em $pairings$, tal como a abordagem de **IBE** em [51], permitindo que qualquer infraestrutura preparada para funcionar com esquemas **IBE** possa ser utilizada para esquemas **CL-PKC**. Para estes casos, até as chaves privadas através do **IBE** podem ser válidas num esquema **CL-PKC**, perdendo o benefício da chave pública ser derivável a partir da identidade.

Tal como o **PKC**, também o **CL-PKC** pode ser utilizado para cifragem (**CL-PKE**) ou para assinaturas digitais (**CL-PKS**). Estas duas vertentes têm 5 etapas partilhadas e 2 específicas à aplicabilidade:

- Partilhadas:
    1. $Setup$: realizado pelo **KGC**, consiste na geração dos parâmetros públicos e da chave mestra do sistema.
    2. $Partial\text{-}Private\text{-}Key\text{-}Extract$: realizado pelo **KGC**, consiste na derivação da chave privada parcial a partir de um identificador e da chave mestra.
    3. $Set\text{-}Secret\text{-}Value$: realizado pelos utilizadores, consiste na computação de uma informação aleatória secreta.
    4. $Set\text{-}Private\text{-}Key$: realizada pelos utilizadores, consiste na combinação do obtido nos pontos (2) e (3) para gerar a chave privada a utilizar.
    5. $Set\text{-}Public\text{-}Key$: realizada pelos utilizadores, consiste na utilização do identificador e do valor em (3) para gerar a chave pública a utilizar.
- Específicas a **CL-PKE**:
    1. $Encrypt$: realizada pelas entidades emissoras, consiste na utilização da chave pública obtida e dos parâmetros públicos do **KGC** para validar a chave e encriptar uma mensagem a transmitir.
    2. $Decrypt$: realizada pelas entidades recetoras, consiste na utilização da chave privada para decifrar um criptograma recebido.
- Específicas a **CL-PKS**:
    1. $Sign$: realizada pelas entidades emissoras, consiste na utilização da chave privada para gerar uma assinatura digital para uma dada mensagem.
    2. $Verify$: realizada pelas entidades recetoras, consiste na utilização da chave pública e dos parâmetros públicos do **KGC** para validar a chave e verificar a assinatura associada a uma dada mensagem.

Vão ser contempladas duas construções de **CL-PKE** em [3], nomeadamente `BasicCL-PKE` e `FullCL-PKE`. Estas encontram-se em A.1 e A.2, respetivamente. A formalização detalhada dos mesmos pode-se encontrar no respetivo $paper$.

Em 2005, foram assinaladas fraquezas de segurança na construção de assinaturas digitais **CL-PKS**, proposta por Al-Riyami e Paterson [33]. Como tal, nesta dissertação vai ser contemplado o primeiro esquema de **CLS** proposto em [32]. A construção associada pode encontrar-se em A.3, sendo que a formalização do mesmo se encontra no artigo referido.

Capítulo 3

Sistemas de partilha de ficheiros na cloud

3.1 Conceito

Em 1990 foi proposto o projeto da $World\ Wide\ Web$ [9], utilizando um computador NeXT (fig. 3.1) como o primeiro $web\ server$ e para programar o primeiro $web\ browser$. Protocolos de partilha de ficheiros não eram ainda muito usados, mas a sua popularidade cresceu em proporção com o desenvolvimento do WWW (ou $W3$, como era chamado na altura).

Em 1999 foi criado, por Shawn Fanning, o Napster [10], um sistema de partilha de ficheiros com estrutura centralizada, que mantem os ficheiros nos respectivos $hosts$, utilizando ligações $peer\text{-}to\text{-}peer$ para transferências. Um ano depois foi criada a primeira rede de partilha de ficheiros descentralizada Gnutella, gerando enorme interesse nesta metodologia de partilha, e originando estudos relacionados com a viabilidade e escalabilidade deste tipo de sistemas [17, 39, 49].

2000 marca um novo início da partilha de ficheiros moderna, com a introdução de muitos protocolos de partilha de ficheiros, sobretudo pela contribuição de Ian Clarke [18], que propôs um $software$ chamado $Freenet$, permitindo a partilha de ficheiros anonimamente, a procura e a publicação de $freesites$, (acessíveis através desta rede dedicada), com objetivo de promover a liberdade de expressão e abolir a censura.

Figura 3.1: O computador NeXT

Em 2001 é lançado o BitTorrent por Bram Cohen, este protocolo de *upload* e *download* de ficheiros é muito usado ainda nos dias de hoje, e a sua estrutura de *tit-for-tat* garante incentivos aos utilizadores que mais contribuírem ao sistema (permitindo *upload*) de modo a não haver demasiada exploração egoísta do mecanismo de partilha.

Em 2005 foi lançado o Megaupload, para alocação e partilha de ficheiros *online* cujas acusações de pirataria e ataques aos direitos de autor em 2012 levaram ao congelamento de cerca de 42 milhões de dólares de ativos. Mega, o sucessor de Megaupload, foi lançado em 2013 da Nova Zelândia. Também em 2005 foi criado o YouTube, um *website* de partilha de vídeos, também este frequentemente atacado com queixas de quebras de direitos de autor.

Também em 2005 foi lançado o primeiro sistema de partilha e gestão de ficheiros na *cloud*, referido na secção 3.2: Box, seguido de outros sistemas como o Dropbox em 2008 e o Google Docs e SugarSync em 2009. Este *trend* de serviços de *cloud* para partilha de ficheiros com sincronização de pastas ainda se encontra em crescimento, apontando para maior espaço oferecido e melhor acessibilidade para o utilizador [54].

3.2 Funcionalidade

A nossa abordagem vai contemplar as funcionalidades comuns oferecidas pelos sistemas disponíveis. Como tal, segue-se uma breve descrição dos mesmos:

- **Dropbox**[1]: Um serviço grátis (até 2Gb) de armazenamento de ficheiros e sincronização de pastas na *cloud* que, com a sua forma de operação simples e intuitiva, conseguiu promover-se com frases como: "Trabalhe com a sua equipa como se todos estivessem usando um único computador" e introduzindo muitos utilizadores ao conceito da utilização da nuvem para armazenamento de informação.

- **Google Drive**[2]: O Google Docs, antes de passar a fazer parte do Google Drive, permitia aos utilizadores partilharem documentos entre si, ajustarem permissões a nível do ficheiro com "apenas escrita" ou "escrita/leitura" e realizarem *collaborate editing*, isto é, utilizadores que partilhassem permissões sobre um ficheiro podiam observar as alterações a serem feitas ao mesmo em tempo real. Com a adaptação deste *Software-as-a-Service* (**SaaS**), o google drive adicionou a possibilidade de partilha e sincronização de pastas a estas funcionalidade já existentes (até 15Gb).

- **Box**[3]: Com até 5Gb livres, é muito mais focado para um ambiente de partilha de ficheiros em equipas, com facilidades em atribuição de tarefas, versionamento de ficheiros e até integração Google Docs.

- **OwnCloud**[4]: Uma plataforma *opensource* que, entre múltiplas aplicações disponibilizáveis, permite o armazenamento e partilha de pastas na *cloud*. Para além das várias ferramentas que permitem a visualização/alteração de ficheiros através do interface de *web*, possui também um cliente de sincronização para *desktop* e oferece a capacidade de estabeler permissões de acesso a nível do ficheiro.

- **SugarSync**[5]: Uma aplicação que permite armazenamento e partilha de ficheiros na *cloud*, focado em providenciar acessibilidade, *backup* automático e partilha entre grupos de trabalho, com diferentes permissões atribuíveis.

- **Mega**[6]: O serviço sucessor ao *Megaupload* permite o armazenamento de até 50Gb grátis e a partilha de ficheiros na *cloud*, na qual é utilizado um sistema de *always-on privacy*[7].

[1] https://www.dropbox.com/home
[2] https://drive.google.com/
[3] https://www.box.com/
[4] https://owncloud.com/
[5] https://www.sugarsync.com/
[6] https://mega.co.nz/
[7] *Always-on privacy* é um termo utilizado para transmitir a ideia de que os ficheiros estão sempre seguros, sendo transferidos do cliente já cifrados

A lista de serviços semelhantes pode continuar, com nomes como **OpenDrive**[8], **Hightail**[9] ou **JustCloud**[10], mas as metodologias utilizadas e os serviços que oferecem não se afastam muito dos apresentados na secção acima.

A partilha de ficheiros, como conceito, é abrangente a qualquer género de partilha, ou até de ficheiro. No entanto, para poder ser compreendido melhor que tipos de partilha podem ser oferecidos aos utilizadores, é necessário analisar diferentes características que se podem identificar na partilha em si. Alguns utilizadores vêem a $cloud$ como um ponto comum para trabalho, procurando a sincronização de pastas entre os vários elementos da empresa ou do mesmo grupo de colegas. Outros utilizam a $cloud$ como um sistema de publicação, exigindo uma partilha com mais granularidade e com diferentes tipos de permissões atribuíveis. Outros ainda fazem da $cloud$ um sistema de $backup$, do qual podem querer esporadicamente partilhar ficheiros.

A nossa identificação do que é um storage system baseia-se nas funcionalidades comums à maioria dos sistemas. Tendo isto em conta, vão ser de seguida analisados os diferentes tipos de partilha na $cloud$, bem como caraterizadas as funcionalidades oferecidas por estes serviços. Inicialmente, especificam-se apenas as funções essenciais à partilha, encontrando-se na maioria dos sistemas deste género:

Armazenar ficheiro - Possibilita ao utilizador o armazenamento de ficheiros no serviço de $cloud$

Descarregar ficheiro - Possibilita ao utilizador recuperar um ficheiro previamente armazenado.

Partilhar ficheiro - Possibilita ao utilizador a partilha de um dado ficheiro armazenado na $cloud$ com uma outra entidade. A essa entidade, dependendo dos casos, pode ser exigida a criação de uma conta no designado serviço.

Revogar permissões - Possibilita ao utilizador a revogação de permissões previamente estabelecidas.

Tendo sido estabelecidas as funcionalidades básicas dos sistemas, apresentam-se funções avançadas, oferecidas pelos casos referidos em 3.2:

[8] https://www.opendrive.com/
[9] https://www.hightail.com/
[10] http://www.justcloud.com/

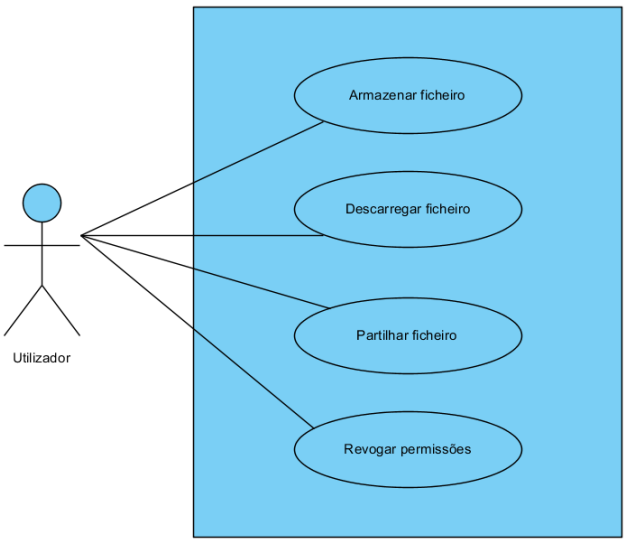


Figura 3.2: Funcionalidades básicas de partilha

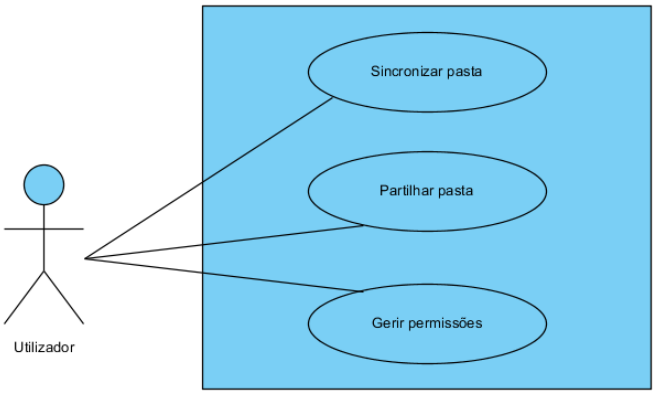


Figura 3.3: Funcionalidades avançadas de partilha

Sincronizar pasta - Possibilita ao utilizador assinalar uma pasta na sua máquina, de modo a que seja frequentemente sincronizada com o serviço de *cloud*.

Partilhar pasta - Possibilita ao utilizador a partilha de uma pasta que se encontre sincronizada com o sistema.

Gerir permissões - Possibilita ao utilizador atribuir diferentes tipos de permissões para os ficheiros a partilhar. Habitualmente, estas permissões consistem em *read-only* ou *read-write*.

Os diagramas de casos de uso 3.2 e 3.3 representam as funcionalidades abordadas acima. Naturalmente, existem pontos não referidos, uma vez que cada serviço pode providenciar

as funcionalidades de uma forma específica. Porém, a perceção destas funcionalidades gerais é um ponto importante para a compreensão do funcionamento dos sistemas atuais, bem como para uma correta análise de segurança.

3.3 Performance e deduplicação

Duplicação é o ato de criar uma réplica de uma dada informação. Deduplicação, por sua vez, é uma técnica de redução de espaço de armazenamento que utiliza a ideia de que, se existe uma cópia no sistema, talvez não haja necessidade de adicionar mais, utilizando-se a já presente. Nomes diferentes têm vindo a ser atribuídos a esta técnica: compressão inteligente, armazenamento de instância única, armazenamento de optimização de capacidade, entre outros. Trata-se de identificar a informação redundante, eliminar todas as cópias excepto uma e criar apontadores lógicos para essa informação, de modo a que todos os utilizadores consigam aceder a esse material.

Segundo Juan Orlandini [27], é um conceito que existe desde 1960, no entanto, o primeiro produto moderno a utilizar esta técnica foi o DD200, apenas lançado em 2004. Uma vez que se trata de reduzir a quantidade de duplicados num sistema, esta metodologia é tão eficiente quanta mais informação reduzir desta forma, ou seja, funciona melhor em espaços grandes de informação, preferencialmente com baixa entropia. Resultados práticos têm vindo a revelar uma melhoria em termos de espaço economizado, mas a escala dessa redução varia consoante os estudos: há quem diga que é possível armazenar 20 vezes mais informação [27], outros afirmam que metade do espaço utilizado é ocupado por duplicados [22] e há quem apresente resultados com 30% de redução do espaço utilizado [41].

A realização da deduplicação implica um mecanismo de processamento de informação que identifique redundâncias de dados. Esta identificação pode ser realizada utilizando a técnica de $hashing$ ou a de $delta\ encoding$, sendo que para o caso de estudo vai ser abordada apenas a primeira.

O sistema deduplicado vai possuir um registo com a associação de identificadores de $hash$, que se acreditam únicos [47], aos respetivos contadores de referências, sendo que o processamento de nova informação consiste no cálculo do seu valor de $hash$ e posterior pesquisa no registo. Os identificadores podem fazer referência a ficheiros inteiros ($whole\text{-}file$

$hashing$: WFH) ou a blocos, sendo estes de tamanho fixo ($fixed\ block\ hashing$: **FBH**) ou variável ($variable\ block\ hashing$: **VBH**).

Este tipo de sistemas podem operar de forma síncrona ou assíncrona:

- **Deduplicação síncrona** - Nestes casos, as operações de deduplicação ocorrem sempre que são realizadas funções que visam alterar a informação armazenada. Para adicionar dados, caso já exista uma entrada no registo com o valor conseguido, é aumentado o contador de referências e é criado um apontador lógico para a informação já existente. Para os casos de remoção de dados, o contador de referências é crucial para o sistema perceber quando é possível eliminar o conteúdo, assinalando quando já não existem mais associações ao mesmo.

- **Deuplicação assíncrona** - Nestes casos, as medidas de deduplicação ocorrem em intervalos de tempo programados, executando sobre toda a informação nos servidores (ou sobre a que ainda não foi deduplicada, dependendo dos casos).

Para os casos em que a deduplicação é realizada do lado do utilizador, é possível dividir as operações de armazenamento em 2 passos: No primeiro é enviado apenas o valor de $hash$ do ficheiro e no segundo é enviado o ficheiro completo. Com este método de funcionamento, o servidor recebe o $hash$, avalia o registo para verificar se já possui o ficheiro na base de dados e, se tal for o caso, responde adequadamente ao cliente, evitando a necessidade de execução do segundo passo.

As exigências de performance dos serviços de armazenamento fazem com que a adaptação a estas técnicas seja um processo sensível. Assim, para a implementação de sistemas deduplicados, é importante ter em conta os fatores envolvidos na mesma e perceber de que forma estes são afetados pelas diferentes vertentes da sua operação.

Os fatores envolvidos na deduplicação consistem em:

- $Overhead$ - Uma vez que a deduplicação envolve cálculos adicionais de identificadores, o armazenamento e possível transmissão dos mesmos é um incremento que deve ser tido em conta.

- Consumo de recursos - O processo de cálculo de identificadores e reconstrução dos ficheiros exige um investimento adicional de poder de processamento.

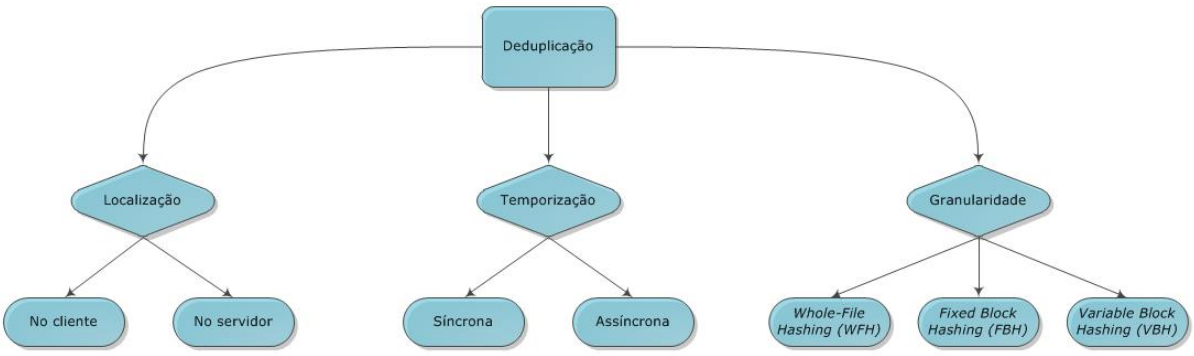


Figura 3.4: Escolhas na implementação da deduplicação

- *Fold factor* - Este fator é determinante para definir a eficiência da técnica de deduplicação, representando o nível de redução de informação armazenada.

As diferentes escolhas na implementação de sistemas de deduplicação envolvem a localização, temporização e granularidade das operações, como se pode observar na figura 3.4.

Em termos de localização, a deduplicação pode ser realizada no cliente ou no servidor. No lado do cliente torna-se possível o armazenamento em 2 passos, o que permite poupanças de largura de banda. No entanto, nos casos em que os servidores têm muito mais poder de processamento que os clientes, ou se o objetivo é conseguir clientes leves, faz sentido colocar o processo de deduplicação no lado do servidor.

Em termos de temporização, a escolha vai depender dos recursos envolvidos. Caso se trate de um sistema com fortes exigências de performance, faz sentido que a deduplicação seja realizada assincronamente, de modo a não pesar as operações do sistema com o incremento de funções a executar. No caso de ser importante a minimização de espaço armazenado, a aplicação de deduplicação síncrona vai certificar-se que o sistema se encontra sempre deduplicado, maximizando a eficácia da técnica. É importante notar que, para casos de quantidades elevadas de informação ou de sistemas que exigem elevada disponibilidade dos dados, a deduplicação assíncrona pode ser impraticável.

Em termos de granularidade, os critérios vão passar pelo *overhead* e pelo *fold factor*. Uma vez que dois ficheiros podem não ser estritamente iguais, mas conterem parcelas de conteúdo semelhantes (fig. 3.5), a divisão dos mesmos por blocos pode aumentar o *fold factor* da técnica. Quanto mais pequenos forem os blocos, mais provável é a existência

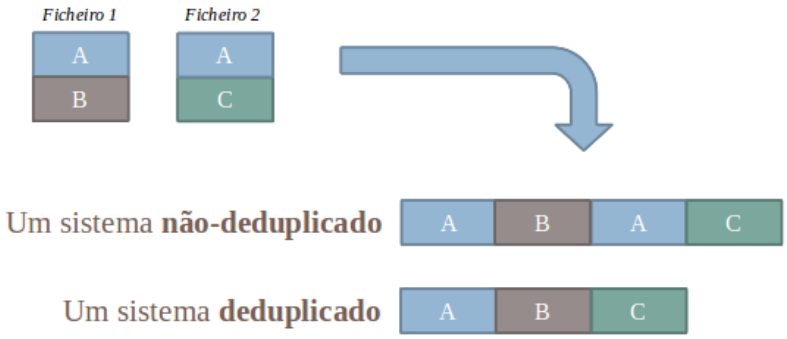


Figura 3.5: Exemplificação da deduplicação por blocos

desses casos, mas também maior vai ser o consumo de recursos e o *overhead* envolvido no cálculo dos identificadores. A utilização de blocos de tamanho variável permite que a deduplicação se adapte ao sistema, melhorando *fold factor*[11], mas os algoritmos utilizados para o cálculo do tamanho de blocos incorrem em mais custos de processamento.

3.4 Análise de segurança

De modo a ser possível desenhar métodos que salvaguardam o utilizador, é necessário que este trabalhe com os mesmos para atingir os objetivos, implicando assim um compromisso da parte dos utilizadores, de modo a assegurar o bom funcionamento dos mecanismos de segurança. Desta forma, um ponto importante é a clara definição das assunções de segurança, isto é, das atitudes que se espera que os utilizadores tenham, regra geral sendo medidas mínimas de defesa e mesmo de confiabilidade no sistema de *cloud* a utilizar:

- A máquina do utilizador não está ao alcance do adversário. Sistemas exigem esta garantia quando se focam em utilizar processos de cifragem e assinatura para que a informação do utilizador consiga o nível de segurança pretendido, quer durante a transferência, quer durante o armazenamento.

- Os ataques do servidor remoto à integridade dos ficheiros devem limitar-se à adulteração, não contemplando a sua destruição. Os mecanismos tipicamente implementados recorrem a assinaturas digitais para verificar alterações nos ficheiros, permitindo ao utilizador saber se este foi adulterado, mas não possibilitando a sua

[11] http://virtualbill.wordpress.com/2011/02/24/fixed-block-vs-variable-block-deduplication-a-quick-primer/

reconstrução. Uma resposta a este problema pode residir na utilização de sistemas tolerantes a falhas bizantinas [1, 15, 16], mas esse tipo de proteção não é contemplada nesta dissertação.

- As chaves privadas são protegidas pelo utilizador. Os métodos de cifragem centralizam-se nessas chaves, devendo assim o utilizador comprometer-se a não transmitir as chaves privadas do sistema a terceiros, a menos que o objetivo seja a partilha de todos os ficheiros por si acessíveis.

- Os dados são toda a informação a proteger. Neste tipo de sistemas, os metadados dos ficheiros, como nome de utilizador ou nome do ficheiro, podem-se encontrar sobre o formato de texto-limpo. Assume-se que assegurar a sua confidencialidade não é um ponto considerado necessário para os sistemas.

Como foi apresentado em 2.1.1, uma avaliação de segurança deve, inicialmente, definir os papeis desempenhados pelos intervenientes, bem como explicitar os agentes responsáveis pelas ameaças ao sistema.

Como $roles$, num sistema simples de $secure\ storage$, temos:

- Os utilizadores, que devem ser responsáveis pela especificação de requisitos de segurança, bem como pelo seguimento das regras de utilização do serviço. Adicionalmente, devem estar responsáveis pela proteção da própria máquina e das chaves privadas associadas ao funcionamento do programa.

- Os provedores, que devem estabelecer, delimitar e gerir as medidas de segurança a implementar. Devem responsabilizar-se pela prestação do serviço de armazenamento e partilha de dados e pela não-destruição/corrupção dos dados armazenados nos mesmos.

Como $threat\ agents$, vão ser contemplados:

- Utilizadores, que podem acidentalmente corromper a própria informação, perder chaves de acesso a documentos, tentar aceder a informação à qual não têm acesso ou atacar o próprio funcionamento do sistema.

- Provedores, que podem acidentalmente corromper a informação de utilizadores, perder registos ou associações de chaves, divulgar informação confidencial ou ocultar falhas do próprio sistema.

- Adversários externos, não pertencentes ao sistema. Estes participantes podem agir passivamente, avaliando os dados transferidos na rede e tentando inferir algum conhecimento relativo à informação armazenada, ou ativamente, através de quebras de disponibilidade, $queries$ aos dados ou personificação de outros utilizadores.

De seguida, é importante clarificar os requisitos de segurança associados ao tipo de serviço, através dos quais vamos poder realizar uma definição adequada dos critérios de segurança. Para serem especificados os requisitos de segurança dos sistemas de partilha de ficheiros, vão ser consideradas as diferentes garantias que devem ser oferecidas pelos serviços de $cloud\ storage$ [37]. Depois, consoante as funcionalidades especificadas em 3.2, serão avaliadas as necessidades centrais de segurança de serviços deste tipo.

Confidencialidade. Os clientes vão querer utilizar o sistema para armazenar e partilhar os seus ficheiros, no entanto, a informação que estes contêm pode ser sensível, levando a necessidades de salvaguardar esses dados. A confidencialidade trata-se de proteger a informação contra leituras inadvertidas, sendo assim importante que os clientes tenham garantias de que apenas os utilizadores com permissões para tal tenham capacidade de ler os seus ficheiros. Para os casos de menor nível de confiança no provedor, nem o próprio serviço de $cloud$ deve ter acesso à informação original dos mesmos.

Integridade. Uma vez que os utilizadores pretendem armazenar/partilhar a sua informação através do serviço, é importante garantir que os ficheiros que estes guardam do lado do servidor não sejam alterados sem conhecimento do cliente. Integridade garante a não-adulteração dos dados e, nestes casos, abrange os dados armazenados remotamente, exigindo uma proteção contra alterações inadvertidas. Esta defesa pode ser abordada tanto contra utilizadores sem permissões ou cujas permissões não abrangem essas capacidades, como contra os próprios superutilizadores que providenciam o serviço. Tipicamente são garantias que permitem que o utilizador saiba quando a sua informação foi alterada, mas nos casos mais robustos podem possibilitar até a recuperação da mesma.

Autenticação e autorização. Em sistemas que requerem garantias de confidencialidade e integridade, como é o caso, é importante que estes dois aspetos sejam também contemplados. O primeiro garante que a identidade das entidades com as quais se vai interagir é sempre verificada. Estas garantias devem ser contempladas tanto a nível do cliente, que deve ter certezas de que não está a transmitir informação para um servidor falso, como a nível do servidor, que deve implementar mecanismos que impeçam as tentativas de falsificação de

identidade. O segundo permite associar aos utilizadores às respetivas permissões, sendo especialmente importante a nível de partilha, de modo a que utilizadores não consigam ler ficheiros que não lhes competem (quebra de confidencialidade) ou alterar ficheiros cujas permissões não se extendam a tal (quebra de integridade).

Disponibilidade. Disponibilidade representa a parcela de tempo para a qual um sistema consegue disponibilizar o seu serviço[12]. A falta de disponibilidade, ou o $downtime$ do serviço, implica a incapacidade de acesso às funcionalidades de partilha, $backup$ de ficheiros e, para os utilizadores que fazem uso de diferentes máquinas para acesso à informação, pode significar a perda total de acesso à própria informação.

Independente controlo de acesso. O sistema não deve depender do servidor remoto para realizar controlo de acesso, devendo este providenciar as garantias desejadas, mesmo na presença de servidores comprometidos.

Revogação de permissões. O utilizador deve ser capaz de revogar permissões sobre os próprios ficheiros de uma forma simples e eficiente, e os utilizadores que perdem acesso a esses ficheiros devem ser afetados pelas mudanças o mais cedo possível, sem exigência de comunicação entre os intervenientes.

A avaliação do risco implica uma análise prévia das ameaças que o podem originar. As ameaças apresentadas de seguida provêm dos $threat\ agents$ enumerados e visam atacar um ou mais requisitos de segurança contemplados:

Quebra de confidencialidade é uma ameaça que pode surgir de qualquer um dos agentes. Trata-se da descoberta de informação confidencial na sua forma mais simples, consequência de escutas na rede por parte de adversários passivos ou da análise à informação armazenada por parte dos provedores do serviço.
Adicionalmente, abordando sistemas com deduplicação, é possível confirmar a presença de um dado ficheiro na $cloud$, avaliando a resposta do serviço quando se tenta armazenar um ficheiro igual (observando o mecanismo de deduplicação). Para os casos de sistemas com baixa entropia nos espaços de mensagens, esta é mais uma vertente que pode resultar em quebras de confidencialidade.

Alteração dos dados é o ataque à integridade de informação que aponta a substituir um F_1 por um F_2, sem que o dono do ficheiro se aperceba da alteração. Este tipo de

[12] Calculável pela divisão do $uptime$ pelo tempo total, isto é: $\frac{Up_t}{Up_t + Down_t}$

ataques pode tentar ser executado por qualquer elemento que possa agir ativamente com o sistema, i. e., todos exceto o adversário que opera passivamente.

DoS é um ataque à disponibilidade do serviço que aponta a tornar um sistema inacessível aos seus utilizadores. São ataques que podem ser realizadas por quaisquer entidades ativas, mas que à partida só serão úteis para adversários ativos, que não fazem uso do serviço.

Acesso inadvertido trata-se de uma entidade obter acesso a informação à qual não lhe compete. Pode passar por não-utilizadores quebrarem o controlo de acessos, utilizadores adulterarem as autorizações que a si estão associadas ou provedores tentarem subverter o sistema de autorização imposto pelos próprios.

Ataques por $rollback$ consistem na substituição de ficheiros centrais ao funcionamento do sistema por dados obsoletos, de modo a conseguir ganhar acesso a informação que não lhe compete, ou alterar o armazenado. Normalmente envolvem a substituição de ficheiros de metadados, recuperando permissões para o atacante.

Quebra de confiança consiste num dado utilizador, que se considerou inicialmente como confiável, começar a agir maliciosamente. Estes casos acontecem quando é atribuída confiança a fontes duvidosas, ou quando clientes legítimos são corrompidos por adversários. Também se pode aplicar ao provedor, mas não é tão adequado para os sistemas a avaliar.

$Takeover$ consiste na substituição de ficheiros chave de um sistema, com objetivo de ganhar posse de todos os dados de um determinado utilizador. Este ataque tira proveito de um único ponto de validação, envolvendo a alteração de todos os metadados associados à estrutura de dados do sistema.

Perda de chaves acontece quando os responsáveis pela gestão e utilização de chaves perdem acesso às mesmas. Esta ameaça depende das entidades responsáveis pela gestão das chaves.

Corrupção acidental é o tipo específico de ataque à integridade dos ficheiros que resulta de um acidente. Pode tratar-se de um acidente do utilizador, como a eliminação de dados importantes, ou do provedor, como uma falha no programa, ou um problema com as infraestruturas.

	PA	**AA**	Utilizadores	Provedores		Integridade	Confidencialidade	Disponibilidade
Quebra de confidencialidade	X	X	X	X			X	
Alteração dos dados		X	X	X		X		
DoS		X	X^*	X^*				X
Acesso inadvertido		X	X	X		X	X	
Ataques por $rollback$		X	X	X		X	X	
Quebra de confiança			X	X^*		X	X	
$Takeover$			X	X		X		X
Perda de chaves			X	X				X
Corrupção acidental			X	X		X		
Divulgação da informação				X			X	

Tabela 3.1: Ameaças vs requisitos e propriedades. X^* são ameaças menos relevantes.

Divulgação de informação passa pelos responsáveis pelo armazenamento de informação partilharem informação confidencial com terceiros. Muitas vezes passa por vender informação dos utilizadores a empresas que possam fazer uso desta no próprio negócio, como empresas que fazem concorrência com outras que são clientes do serviço em questão, por exemplo.

Na tabela 3.1 está representada a relação entre as ameaças descritas previamente com os $threat\ agents$ que as podem perpetuar e com as propriedades de segurança consideradas relevantes, nomeadamente confidencialidade, integridade e disponibilidade.

Depois de assinaladas as ameaças, vão ser abordados os mecanismos habitualmente utilizados pelos serviços de modo a atingirem certas garantias de segurança, nomeadamente confidencialidade, integridade e gestão de chaves. Faz sentido focar a avaliação de segurança nestes riscos em específico, uma vez que este trabalho visa enriquecer essa perspetiva de segurança.

As contramedidas habituais para responder a ataques à confidencialidade e integridade dos ficheiros passam pela transmissão da informação através de canais SSL e por sistemas de autenticação baseados em $login$ (C_1). Esta é a abordagem realizada por sistemas como Dropbox, Google Drive ou Box. Desta forma, a informação é armazenada em texto-limpo e o acesso/alteração da mesma é limitado às entidades com acesso aos dados. Esta

aproximação apenas é possível graças à assunção de que os clientes pretendem que os seus dados não se encontrem acessíveis ou alteráveis por outros utilizadores (quer durante a transmissão, quer no sistema), mas que estes confiam na legitimidade do serviço. Como alternativa a esta abordagem, com sistemas como o Mega, ou utilizando uma camada de segurança como o CloudFogger[13], é possível reforçar um sistema de $login$ com cifragem do lado do cliente, de forma a conseguir contrariar ameaças à confidencialidade da parte do provedor (C_2).

Para responder a ameaças acidentais, como a corrupção acidental ou a perda de chaves, são necessárias medidas específicas aos problemas. No caso da alteração dos dados, a utilização de protocolos bizantinos nos servidores e o frequente $backup$ dos ficheiros permite o acesso a informação antiga e a correção de possíveis lapsos. No caso de perda de chaves, as contramedidas habituais passam pela disponibilização de funcionalidades para recuperação de chaves. Para combater o **DoS**, podem ser utilizadas $firewalls$ ou mecanismos semelhantes. Estas são contramedidas menos relacionadas com o trabalho realizado e, como tal, são apresentadas de uma forma menos detalhada.

Ameaças mais específicas, como ataques por $rollback$ ou o $takeover$, não tendem a ser contempladas pelos serviços atuais. Ao assumirem algum nível de confiança nos serviços, rejeitam a possibilidade de alterações em ficheiros de metadados que permitam este tipo de ataques.

Para as alternativas que abordam a cifragem de informação armazenada, a partilha desses dados é habitualmente realizada através de cifras de chave pública (2.2.3) para partilhar chaves de cifras simétricas, por sua vez utilizadas na cifragem da informação (2.2.2). A cada utilizador é atribuído um par de chaves (sk, pk), que serão utilizadas para a transmissão de chaves de cifragem utilizadas nos ficheiros que se pretende partilhar. Como exemplo, assume-se que a Alice quer enviar um criptograma ao Bob (ambos clientes de um mesmo serviço de $cloud$), sendo esse criptograma F, o par de chaves do Bob (sk_B, pk_B), os algoritmos de cifragem/decifragem (Enc, Dec) e a chave de cifra simétrica do criptograma k, temos:

- O sistema entrega pk_B à Alice.
- A Alice executa $c \leftarrow Enc_{pk_B}(k)$.

[13] http://www.cloudfogger.com/

	PA	**AA**	Utilizadores	Provedores
Quebra de confidencialidade	C_1, C_2	C_1, C_2	C_1, C_2	C_2
Alteração dos dados		C_1, C_2	C_1, C_2	$-$
DoS		$-$	$-$	$-$
Acesso inadvertido		C_2	C_2	C_2
Ataques por $rollback$		C_1, C_2	C_1, C_2	$-$
Quebra de confiança			C_1, C_2	C_2
$Takeover$			C_1, C_2	$-$
Perda de chaves			$-$	$-$
Corrupção acidental			$-$	$-$
Divulgação da informação				C_2

Tabela 3.2: Abrangência das contramedidas apresentadas. "−" representam ameaças não contempladas pelas contramedidas mencionadas.

- A Alice partilha F e c.
- O Bob recebe F e c.
- O Bob executa $k \leftarrow Dec_{sk_B}(c)$.

Desta forma, o Bob consegue o criptograma F e a chave associada à sua decifragem k. Este tipo de partilha de chaves públicas envolve alguma confiança no serviço de $cloud$ intermediário (para que este não monte ataques de substituição de chave pública). Uma alternativa a este tipo de partilha de chaves é a transmissão por $e\text{-}mail$ da chave de cifragem, opcional no Mega, ou o recorrer a **PKI**s para validação de chaves públicas.

Na tabela 3.2 encontram-se representadas as ameaças protegidas pelas contramedidas abordadas. Como pode ser observado na mesma, as contramedidas de segurança aplicadas no cliente são mais abrangentes que as que implicam confiança na $cloud$. No entanto, a maioria dos serviços atuais opta pela segunda opção, consequência da dificuldade associada à conjugação dos requisitos de segurança em 3.4 com a boa performance das funcionalidades referidas em 3.2.

O mecanismo mais básico de segurança, que assume um provedor de $cloud$ confiável, assegura apenas a cifragem dos dados na sua transmissão, forçando tanto o cliente como o servidor a realizarem operações adicionais, degradando a performance do serviço.

Uma abordagem mais forte, que consiste em cifrar os dados do lado do cliente, enfrenta obstáculos de performance, de funcionalidade e de usabilidade. A nível de performance, o cliente é responsável por realizar as operações de cifragem e decifragem na própria máquina, não tirando proveito da capacidade de processamento do servidor remoto. A nível de funcionalidade, o servidor deixa de poder aplicar funções de aperfeiçoamento de serviço sobre os dados armazenados, como a deduplicação. A nível de usabilidade, é necessário que utilizador tenha acesso às chaves guardadas localmente para poder aceder aos ficheiros, o que significa que este terá de transferir as chaves para qualquer máquina na qual pretende fazer uso do sistema. Esta última questão seria contornável pela utilização de um outro sistema externo de confiança o que, por sua vez, resultaria num aumento de complexidade na arquitetura do serviço.

Para a partilha de ficheiros segura, os obstáculos na aplicação de cifras de chave pública dependem do grau de confiança depositado no serviço. Caso seja confiável que este não vai tentar executar ataques de substituição de chave pública, o servidor pode associar cada cliente registado à respetiva chave pública numa estrutura local, enviando-as aos utilizadores consoante sejam necessárias. Caso o serviço não seja confiável e os clientes devam partilhar chaves por $e\text{-}mail$, criam-se obstáculos de: i. usabilidade, sendo que ambas as partes envolvidas na partilha têm de enviar/receber as chaves por meios externos e ii. funcionalidade, pois no caso das cifras simétricas, a partilha é atómica, garantindo sempre acesso a leituras e escritas sobre o mesmo. Caso o serviço não seja confiável e os clientes recorram a **PKI**s, para além da quebra de usabilidade (os utilizadores continuam a recorrer a um meio externo), é necessário atender aos obstáculos de performance que advêm da necessidade adicional de processamento envolvido na validação/revogação das chaves públicas.

Os desafios associados ao estudo de soluções alternativas consistem em procurar $tradeoffs$ adequados entre garantias de segurança e funcionalidades providenciadas. Dessa forma, é importante responder a problemas como a granularidade de permissões ou o recorrer a servidores remotos não confiáveis com soluções que possuam um bom equilíbrio entre segurança e performance.

Capítulo 4

Estado da arte

4.1 Limitações das soluções existentes

A maioria sistemas comerciais referidos operam numa perspectiva de maximizar a performance e a eficiência, de modo a poderem oferecer o serviço a mais utilizadores e conseguirem maior retorno do seu investimento. Desta forma, os provedores preferem realizar uma partilha o mais generalista possível, levando à partilha de pastas ao invés de ficheiros individuais, minimizando o esforço da divisão de acessos e maximizando a sua eficiência. Isto não é necessariamente um ponto negativo, mas é importante ter em conta consoante o que o utilizador procura usufruir num sistema deste género, pois pode ser enganosa a declarada funcionalidade de "partilha de ficheiros".

Granularidade e precisão nas permissões

A abordagem realizada pelo dropbox restringe as permissões associáveis às pastas, enquanto que a granularidade da partilha de pastas no box tem limitações quando se tratam de subpastas. Insuficiências como estas são consequência da lógica referida anteriormente. Maior especificidade nas permissões de cada utilizador leva a um acréscimo de complexidade no controlo de acessos, o que origina num maior esforço para a gestão e aplicação desses mecanismos da parte do provedor.

Este é um obstáculo de performance, uma vez que as contramedidas existentes implicam um degradar da performance do serviço. Exemplos de soluções passam por listas de

controlo de acesso, que permitem atingir melhor especificação de permissões, ou cifragem com chave individual para cada ficheiro, que permite oferecer partilha ao nível do mesmo.

Servidores remotos não confiáveis e comunicação externa

A maioria das contramedidas referidas previamente partem da assunção que o provedor de $cloud$ é de confiança. Este é um ponto que tem vindo a receber forte atenção, uma vez que se trata de um luxo ao qual muitos utilizadores não se podem dar, quer pelo acesso a dados altamente confidenciais, quer pela manipulação de informação sensível demais para arriscar perdas de integridade.

Na seção 3.4 foi examinada a metodologia habitual para partilha de ficheiros segura: as cifras de chave pública, utilizando as chaves que os utilizadores mantinham na $cloud$ para transmitir o segredo utilizado na cifragem dos dados. O problema desta abordagem sobre esta nova perspectiva reside na possibilidade da $cloud$ substituir as chaves públicas armazenadas por chaves da própria $cloud$, conseguindo assim acesso aos dados armazenados pelo utilizador. Sem esta hipótese, cai sobre os utilizadores a responsabilidade da gestão, distribuição e validação das chaves públicas a utilizar, tarefa não-trivial e muito dificilmente abstraível.

Uma forma de resolver este problema é assumir que os utilizadores conseguem estabelecer uma ligação segura entre estes o que, resolvendo um problema, cria outros tantos.

Primeiro, é necessário que os utilizadores tenham um meio para comunicar entre eles que seja considerado seguro. Sincronamente, pode envolver uma ligação de TLS/SSL entre ambos ou a utilização de outros meios externos considerados pelos utilizadores como de confiança [48]. Assincronamente, pode implicar a troca de $e\text{-}mails$ ou a utilização de **PKI**s/**PGP** [1]. Este estabelecimento do meio externo origina obstáculos de arquitetura, na opção de recorrer aos sistemas externos, e obstáculos de performance, associados às técnicas que implicam processamento do lado do cliente [13].

Segundo, estes métodos nem sempre se apresentam como fáceis de utilizar, pelo menos sem conhecimento destas técnicas de segurança associadas. Isto leva a pior usabilidade e, implementado o seu uso de forma pouco intuitiva, pode alienar potenciais utilizadores do

[1] http://www.openpgp.org/

serviço.

Extração de conhecimento a partir de criptogramas

Como referido na seção 2.2, um sistema de cifragem é considerado seguro se um adversário não conseguir computar nenhuma função aplicável ao texto-limpo no criptograma. Isto implica que, nos casos gerais, é difícil (ou até impossível) extrair informação de criptogramas para os quais não se tem acesso à chave de decifragem associada.

Isto significa que, para a abordagem C_2 em 3.4, a cifragem de toda a informação implica que os servidores não vão ser capazes de aplicar quaisquer técnicas para melhoramento de performance. Esta atitude impossibilita a aplicação de técnicas de melhoramento de performance, como a deduplicação (3.3), e qualquer tipo de inferência sobre os dados armazenados.

4.2 Soluções propostas em sistemas experimentais

SiRiUS [35] e Plutus [36] são dois sistemas de $secure\ storage$ que têm como objetivo permitir partilha de ficheiros em servidores não-confiáveis. Ambos enfrentam as dificuldades referidas em 4.1 e 4.1.

A perspectiva da cifragem no cliente cruzada com partilha de ficheiros tem algumas características semelhantes à utilizada nos esquemas atuais. O método envolve associar a cada ficheiro uma chave de cifragem e uma chave de assinatura, sendo que os utilizadores com permissões de leitura vão ter acesso à primeira e os utilizadores com acesso de leitura/escrita vão ter acesso a ambas. Recorrendo a assinaturas da informação, os utilizadores com apenas a primeira chave não conseguem produzir alterações válidas, mesmo conseguindo cifrar/decifrar os dados. O acesso a estas chaves é atribuído através de cifras de chave pública, cifrando este par de chaves com as chaves públicas dos utilizadores-alvo (como descrito em 3.4).

Quanto ao obstáculo de arquitetura que resulta da gestão e distribuição de chaves públicas, os sistemas estudados evitam atacar o assunto. A atitude é a de assumir comunicação externa entre utilizadores e colocar a responsabilidade sobre os mesmos.

4.2.1 SiRiUS

O SiRiUS apresenta-se como uma camada de segurança para um sistema de ficheiros, de modo a poder jogar com a heterogeneidade dos diferentes sistemas e ser ortogonal às medidas (ou à ausência) de segurança de cada um. Neste caso, a facilidade no estabelecimento de um canal de comunicação seguro entre utilizadores é considerada irrealista, sendo que este evita ao máximo a comunicação externa entre intervenientes. Para tal, a partilha de ficheiros é feita recorrendo à cifragem de chave pública.

Cada utilizador necessita apenas de armazenar duas chaves, cujo par é gerado no início da utilização do sistema: **MEK** - $Master\ Encryption\ Key$ e **MSK** - $Master\ Signature\ Key$. Cada ficheiro criado vai ter a si associado uma chave simétrica **FEK** - $File\ Encryption\ Key$ e uma chave de assinatura **FSK** - $File\ Signature\ Key$. Assim, o armazenamento é composto por três tipos de ficheiros:

1. $d\text{-}file$ - O ficheiro que contém informação, nomeadamente a informação cifrada com a **FEK** e o seu hash cifrado com a **FSK**.

2. $md\text{-}file$ - O ficheiro de metadados, contendo uma assinatura do hash com **MSK**, o nome do ficheiro, um $timestamp$ da última alteração, a chave de verificação de **FSK** e uma lista de blocos cifrados com o **MEK** público de cada utilizador. Estes blocos podem conter o **FEK** ou o **FEK**+**FSK**, dependendo se é permitida leitura ou leitura/escrita.

3. $mdf\text{-}file$ - Um ficheiro de metadados localizado em cada diretoria que contem a raiz de uma árvore de $hash$ construída com os $md\text{-}files$ na mesma diretoria e todas as sub-diretorias.

Esta árvore de $hash$ serve como um mecanismo de atualização sobre os ficheiros de metadados, constantemente assegurando a integridade destes após um dado intervalo de tempo [26, 40]. Isto permite contrariar possíveis ataques por $rollback$ [2]. Para este mecanismo funcionar, a árvore deve ser constantemente verificada e a sua raiz atualizada por um $daemon$ do lado do cliente.

Para a revogação de acessos, o proprietário do ficheiro volta a cifrar o documento, renova o $md\text{-}file$ para contemplar a nova **FEK** e **FSK** e atualiza o $mdf\text{-}file$ para adaptar às alterações,

[2] $rollback$ é um tipo de ataque em que um utilizador substitui um ficheiro por uma versão mais antiga do mesmo ficheiro, com objetivo de tentar subjugar o sistema em questão

conseguindo assim uma revogação ativa. Os maiores problemas da abordagem SiRiUS são:

- Elevado processamento exigido ao cliente, forçando-o a cifragens, $hashing$ e múltiplas verificações de assinaturas.
- O $overhead$ do lado do sistema de ficheiros para armazenar $md\text{-}files$ e $mdf\text{-}files$, aumentando o espaço utilizado e a latência dos pedidos (devido às verificações).
- A partilha de ficheiros optimizada para grupos pequenos, não escalando bem para um aumento de utilizadores com permissões sobre um mesmo ficheiro. No entanto, a utilização do SiRiUS-NNL é uma alternativa adequada quando se pretendem grupos de partilha com tamanho próximo ao número de utilizadores do sistema.

4.2.2 Plutus

Plutus, ao contrário do SiRiUS, não só assume a existência de canais seguros entre utilizadores, como coloca a responsabilidade de gestão e distribuição de chaves totalmente do lado dos clientes. O objetivo desta medida é o de minimizar a responsabilidade do servidor remoto e atingir os melhores níveis de escalabilidade, uma vez que o incremento de processamento se encontra do lado do cliente.

Os ficheiros são cifrados por blocos com chaves individuais ($file\text{-}block\ keys$) e armazenados em $lockboxes$ cifradas com criptografia simétrica, resultando em $file\text{-}lockbox\ keys$ (cujo tamanho pode aumentar, mediante a granularidade de partilha que se quiser providenciar). Esta medida por si só é utilizada apenas no acesso ao(s) ficheiro(s), sendo que a mesma abordagem anterior de chaves para assinatura/verificação é utilizada para divisão de acessos em apenas-leitura e leitura-escrita.

Na revogação de acessos, é utilizada uma técnica de rotação de chaves para oferecer $lazy\ revogation$, significando que, mesmo que um utilizador perca permissões, este continua a ter acesso a versões antigas dos ficheiros. A ideia é a de utilizar a chave privada para exponenciar a chave atual e gerar chaves para o $lockbox$. Assim:

- Se um leitor possuir a chave na mesma versão do $lockbox$, vai conseguir concluir a operação.

- Se um leitor possuir uma chave desatualizada do $lockbox$, vai ter de pedir a chave atual ao $owner$ do ficheiro.
- Se um leitor possuir uma chave demasiado recente do $lockbox$, vai poder utilizar a chave pública para fazer $rollback$ e chegar à chave que precisa para concluir a operação.

Note-se que, mesmo que se esteja a tratar de $lazy\ revogation$, os metadados têm de ser imediatamente alterados, de modo a impedir escritas indesejadas, se tal for o caso. Para o Plutus, os seus maiores problemas são:

- Armazenamento elevado de chaves local, impedindo a utilização de um cliente com menos capacidade de armazenamento.
- Assunção de comunicação segura entre clientes exige canais seguros, que nem sempre são possíveis (considerado pelo SiRiUS como irrealista).
- $Lazy\ Revogation$ permite acesso a versões antigas dos ficheiros por parte de utilizadores revogados, o que não é conveniente.

4.3 Soluções criptográficas e *message-locked encryption*

Para a partilha de ficheiros segura num ambiente de $cloud$ são tipicamente utilizadas técnicas de cifragem de dados. A deduplicação, por outro lado, prospera da existência de duplicados e da capacidade de os identificar frequentemente para conseguir $hits$. Uma vez que dois ficheiros iguais cifrados com chaves diferentes geram documentos diferentes, a eficiência da deduplicação fica entregue ao raro acaso e espera-se que caia drasticamente.

Atacando a dificuldade abordada em 4.1, estes objetivos contraditórios têm de ser relaxados se é pretendido alcançar uma forma de fazer estas duas técnicas coexistir. Para conseguirmos deduplicação num sistema seguro temos de fazer sacrifícios na componente da segurança, e para conseguirmos adicionar segurança à deduplicação, vão ter de ser contemplados $tradeoffs$ na performance da mesma. Preferencialmente, o objetivo é conseguir atingir um nível de segurança mínimo exigido pelos utilizadores, providenciando uma eficácia de deduplicação que permita melhorias visíveis no espaço de armazenamento.

A abordagem deduplicação segura já é utilizada a nível prático em aplicações de armazenamento remoto [2, 5, 19, 50], muitas vezes denominada $Convergent\ Encryption$, mas com uma noção variante consoante o caso.

4.3.1 Funcionamento do *message-locked encryption*

Em 2013 foi formalizada a primitiva de $Message\text{-}Locked\ Encryption$ (**MLE**) [8]. Este artigo tem o objetivo de abordar os diferentes casos de deduplicação segura, generalizando o mecanismo envolvido nas técnicas usadas e clarificando as garantias de segurança oferecidas para este tipo de primitiva.

Esquemas de **MLE** são esquemas de cifra simétrica em que a chave em si é derivável da mensagem e são constituídos por 4 funções principais $\langle \mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{T} \rangle$ (Fig. 4.1):

- $\mathcal{K}$ é o algoritmo responsável pela derivação de chaves, isto é, recebe como $input$ a mensagem M e entrega como $output$ a chave K.
- $\mathcal{E}$ é o algoritmo responsável pela cifragem de dados. Para este, qualquer algoritmo de cifragem simétrica é aplicável.
- $\mathcal{D}$ é o algoritmo responsável pela decifragem de dados. O passo inverso ao anterior.
- $\mathcal{T}$ é o algoritmo responsável por gerar a tag, ou seja, o identificador utilizado para comparar dois criptogramas.

Utilizando estes quatro algoritmos, esquemas de **MLE** seguem uma construção como se segue:

- $Key\ generation$: Recebendo uma mensagem m, entrega $k \leftarrow \mathcal{K}(m)$.
- $Encrypt$: Recebendo k e uma mensagem m, entrega $c \leftarrow \mathcal{E}(k, m)$.
- $Tag\ generation$: Recebendo um criptograma c, entrega $t \leftarrow \mathcal{T}(c)$.
- $Decrypt$: Recebendo k e um criptograma c, entrega $m \leftarrow \mathcal{D}(k, c)$.

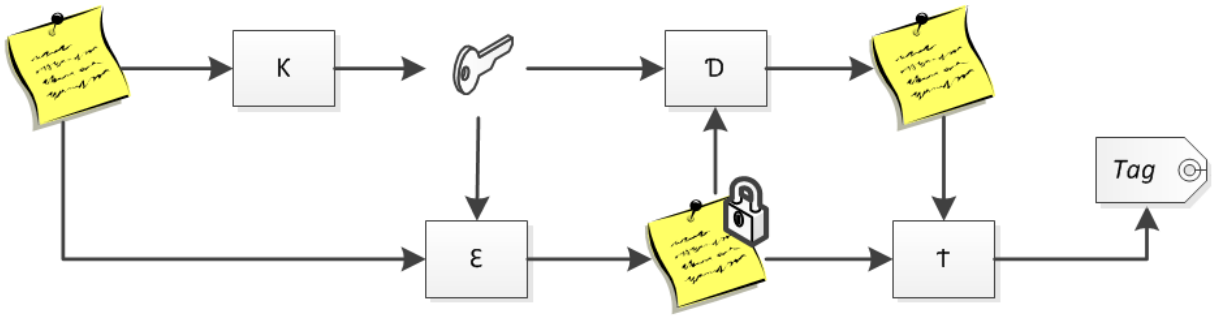


Figura 4.1: As 4 funções dos esquemas MLE

4.3.2 Revogação de permissões com *message-locked encryption*

A revogação *lazy* é, à partida, mais eficiente, uma vez que as alterações sobre o ficheiro ocorrem apenas durante a atualização, pois permite que o sistema otimize tendo isso em conta. No entanto, o preferível é realizar revogação ativa, o que implica cifrar novamente o ficheiro, com uma chave diferente, bloqueando imediatamente o acesso aos utilizadores indesejados [7].

O problema é que, na utilização de esquemas baseados em **MLE**, a chave deriva da mensagem, não sendo possível alterar a mesma sem alterar a informação. Essa natureza que liga a chave à mensagem e vice-versa implica que o melhor tipo de revogação possível de implementar neste sistema é a revogação *lazy*. Este é um exemplo do *tradeoff* que implica adaptar deduplicação a um sistema de partilha de ficheiros.

4.3.3 Segurança e contribuições práticas

Para ser possibilitada a deduplicação sobre texto cifrado, tem de ser possível recolher alguma informação do criptograma, nomeadamente, se dois criptogramas contêm a mesma informação. Nos esquemas propostos, a capacidade de verificação de igualdade consiste na caraterística das funções de *hash* utilizadas de produzirem *outputs* únicos para cada *input* [47], assumindo-se uma eficácia de deduplicação de 100%.

No entanto, com certezas da capacidade de comparar dois textos cifrados, as técnicas de **MLE** não atingem indistinguibilidade (2.2.2), uma vez que qualquer adversário que consiga

gerar o criptograma de um dado utilizador tem acesso à informação associada. Assim, a segurança associada à confidencialidade dos criptogramas depende da entropia presente no espaço de mensagens do sistema, sendo tão seguro quanto mais imprevisíveis forem as mensagens armazenadas no mesmo. Esta fraqueza leva a possíveis ataques de dicionário, por força bruta ou por tentativa/erro, consequências da segurança ser tão forte quanto a entropia das mensagens no sistema.

É importante também atender ao novo cuidado de proteger a tag. Uma vez que se assume não ser possível impedir a comparação de ficheiros cifrados, torna-se importante assegurar que dois ficheiros considerados iguais sejam, realmente, iguais. Esta propriedade é chamada de $tag\ consistency$ e tem como objetivo impedir ataques como o que se segue:

- Um adversário transmite um ficheiro maliciosamente gerado M_{Adv} com $tag\ T$.
- O Bob transmite um ficheiro legítimo M_B com $tag\ T$. O sistema reconhece as $tags$ como iguais e mantém apenas a primeira versão.
- O Bob tentaa fazer $upload$ do ficheiro e recebe o M_{Adv} criado pela entidade maliciosa.

Neste ataque, o Bob foi vítima de um ataque de $duplicate\text{-}faking$, já encontrado em sistemas que implementam este tipo de deduplicação [52].

$$\mathcal{T}(C) = \mathcal{T}(\mathcal{E}(\mathcal{K}(M), M)) \tag{4.1}$$

A propriedade de $tag\ consistency$ (**TC**) exige que seja "hard" [3] a criação de um par (M, C) congruente com 4.1, mas que verifique 4.2.

$$\mathcal{D}(\mathcal{K}(M), C) \neq m \tag{4.2}$$

Adicionalmente, existe a noção de $strong\ tag\ consistency$ (**STC**), que exige que seja "hard" a criação de um par (M, C) congruente com 4.1, mas que verifique 4.3. Note-se que **STC** é estritamente mais forte que **TC**.

$$\mathcal{D}(\mathcal{K}(M), C) = \perp \tag{4.3}$$

[3] Um problema $hard$ é um problema que se acredita não ter solução viável para um adversário cujo processamento está limitado a algoritmos polinomiais

O trabalho apresentado em [8] também inclui propostas de contribuições práticas. Nesse sentido, é abordado o $Convergent\ Encryption$ (**CE**), cuja utilização já se encontra presente em sistemas práticos, bem como dois adicionais: $Hash\ and\ Convergent\ Encryption$ (**HCE**) e $Randomized\ Convergent\ Encryption$ (**RCE**), para os quais serão atingidos diferentes $tradeoffs$ de eficiência e consistência de tag. Para a implementação destes esquemas, é assumida a existência dos algoritmos $\langle \mathcal{H}, Enc, Dec \rangle$, sendo $\mathcal{H}$ uma função de $hash$ criptográfica e (Enc, Dec) é o par de algoritmos para a cifra simétrica. As formalizações destes esquemas podem-se encontrar no artigo associado, seguindo-se, assim, uma resumida descrição dos mesmos.

O **CE** é uma generalização do esquema original proposto em [22], utilizando $\mathcal{H}$ para gerar uma chave simétrica K e recorrendo à mesma para cifrar a mensagem com Enc. A tag é calculada aplicando $\mathcal{H}$ a todo o criptograma. A construção deste esquema pode-se encontrar em A.4.

Em [8] são abordadas duas variações de **HCE**: **HCE1** e **HCE2**.

HCE1 é uma variante popular do **CE**, utilizada em vários sistemas [19, 20, 53]. Ao contrário do **CE**, neste esquema as $tags$ são computadas ao mesmo tempo que é realizada a cifragem, aplicando $\mathcal{H}$ à chave associada à mensagem. Este **HCE1** é vulnerável a ataques que quebram a **TC**, que consistem em escolher duas mensagens $M \neq M'$, computar $C \leftarrow Enc(\mathcal{H}(M), M')$ e $T \leftarrow \mathcal{H}(\mathcal{H}(M))$ e armazenar $(M, C \parallel T)$. Dessa forma, caso um adversário consiga antecipar uma mensagem que será armazenada por um utilizador, este pode indetetávelmente subistiuí-la por uma outra mensagem arbitrária à escolha do mesmo.

O **HCE2** é um novo esquema, que adapta o **HCE1** de modo a incluir um novo mecanismo que confira segurança **TC**. O processo de decifragem deste esquema volta a computar a tag utilizando a mensagem decifrada, confirmando assim a validade da tag associada. O **HCE** abordado neste trabalho refere-se a esta variação, e a respetiva construção pode ser consultada em A.5.

O **RCE** trata-se de uma variação do **HCE** que visa recorrer a aleatoriedade para maximizar a paralelização das operações envolvidas. Isto é conseguido através da escolha de uma chave aleatória de cifra simétrica L, seguida da execução simultânea de $Enc(L, M)$ e de $\mathcal{H}(K)$. Finalmente, L é cifrado com K utilizando $one\text{-}time\ pad$, e a tag é derivada a partir de K. A construção do **RCE** pode-se encontrar em A.6.

Capítulo 5

Viabilidade de cifras *message-locked encryption*

5.1 Características a avaliar com o programa

De modo a ser corretamente avaliado o impacto das técnicas de **MLE** num sistema deduplicado, é importante focar nas vantagens que esse tipo de sistema trás, e de que forma pode esta adição prejudicar as diferentes entidades que fazem parte do sistema. Para esta análise vamos assumir um sistema de deduplicação síncrona realizada do lado do cliente, incidindo sobre blocos de ficheiros e tendo em conta as diferentes características referidas em 3.3.

Do um lado temos o cliente. O benefício do mesmo com a deduplicação limita-se à possível redução de informação transmitida na rede (caso a comparação seja realizada em 3 passos) e/ou a uma indireta melhoria de serviço pela parte dos provedores do serviço. Assim, faz mais sentido perceber de que forma a aplicação de técnicas **MLE** pode sobrecarregar os clientes a nível de consumo de recursos, ou seja, tamanho das mensagens e tempo de execução de operações básicas.

De outro lado temos o servidor. As vantagens da deduplicação incidem maioritariamente nesta parte, reduzindo o espaço de armazenamento efetivamente utilizado a troco de uma operação de verificação realizada na inserção de ficheiros. Esta operação tende a ser

relativamente leve, implicando uma procura de um $output$ de função de $hash$ num registo de ficheiros. Assim, vamos procurar perceber se a aplicação destas técnicas resulta num degradar na eficiência da deduplicação.

Tendo isto em conta, o $benchmark$ foi dividido em duas abordagens, uma da perspetiva do servidor e outra da perspetiva do cliente:

Perspetiva do cliente - Performance: nomeadamente, tempo de execução e tamanho de mensagens.

Perspetiva do servidor - Eficiência da deduplicação: nomeadamente, quantidade de informação que é efetivamente deduplicada.

5.1.1 Técnicas abordadas

As operações que vão estar sobre teste pelo $benchmark$ devem produzir resultados passíveis de conclusões úteis ao problema estudado. Para tal, é tão importante a escolha de elementos a serem testados como a escolha de termos de comparação.

Este problema pode ser abordado por dois pontos de partida iniciais:

1. Por um provedor de serviço que faz uso da deduplicação. Esta permite que as medições possam ser usadas para explicitar a eficiência a abdicar nos seus servidores e compreender o quão grave é a sobrecarga das medidas de segurança para os clientes do seu serviço.

2. Por um provedor de serviço de $secure\ storage$. Neste ponto de vista, é interessante perceber de que forma pode ser poupado espaço de armazenamento com o relaxar do seu modelo de segurança e de que forma estes métodos afetariam o seu programa do cliente.

De modo a atender a estes pontos de vista, são identificados dois termos de comparação importantes: deduplicação simples e cifragem simples.

Na perspetiva do cliente, é importante avaliar a degradação de performance que advém da aplicação de técnicas **MLE**. Como tal, vai ser executado: deduplicação simples, cifragem

simples e um esquema de **MLE**. Estes testes permitirão mensurar o degradar de performance espectável quando se parte de um sistema que já aplique a deduplicação, bem como quando se parte de um sistema que já aplique métodos de cifragem convencionais.

A respeito da escolha do esquema de **MLE**, uma vez que os diferentes esquemas são caraterizados pelos diferentes níveis de paralelização de operações, teve-se em conta o reduzido tempo de execução das funções e o peso associado às *threads* de `java.lang.Thread` [38]. Como tal, o **HCE** revelou-se como o esquema mais rápido para estes casos e foi escolhido para comparação com as restantes operações.

Na perspetiva do servidor, é importante avaliar a eficiência da deduplicação nos diferentes casos. Para este ponto, vai ser mensurada a deduplicação por ficheiro completo e por blocos, atendendo ao número de vezes que é identificado um duplicado e ao espaço que é poupado desta forma. A nível de operação, e seguindo o raciocínio aplicado na perspetiva do cliente, vai ser contemplado: deduplicação simples, cifragem simples e um esquema de **MLE**.

Adicionalmente, o *benchmark* aborda dois esquemas adicionais, que tentam jogar com a natureza das primitivas de **MLE** e com o ambiente de teste, de modo a produzir resultados mais valiosos: **MLESalted** e **CipherCE**.

MLESalted é uma experiência com a eficiência da deduplicação na presença de aleatoriedade acrescentada nas mensagens. A ideia passa por aumentar a entropia do sistema através da inserção de bytes aleatórios que tornem as mensagens mais imprevisíveis, visando sacrificar alguma eficácia de deduplicação por um aumento na confidencialidade oferecida. Como as restantes operações, vai ser executada tanto a nível do bloco como a nível do ficheiro completo. A construção deste esquema pode ser consultada em A.7

Deduplicação por blocos tende a ser mais eficiente que a por ficheiros completos, porém, esta é também mais pesada, uma vez que cada bloco implica uma derivação de chave. **CipherCE** é uma abordagem cujo objectivo é perceber até que ponto se pode tirar partido de um cabeçalho comum dos ficheiros para derivar a chave. Caso ficheiros com blocos comuns tendam a partilhar o mesmo bloco inicial, a operação de derivação de chave precisa apenas de ser feita uma vez (para este cabeçalho), sendo identificáveis os blocos iguais entre ficheiros, mesmo não sendo estes totalmente iguais. A construção deste esquema encontra-se em A.8

Assim, as técnicas abordadas englobam:

- **Client-Side**:
  - $Deduplicação$ - Termo de comparação com a deduplicação simples.
  - $Normal\ Cipher$ - Para comparação com a cifragem normal.
  - **HCE** - Um esquema **MLE**, alvo de comparação.
- **Server-Side**:
  - $Deduplicação\ -\ WF$ - Deduplicação de ficheiro completo.
  - $Deduplicação\ -\ Bl$ - Deduplicação realizada a nível do bloco.
  - **HCE** $-\ WF$ - Um esquema **MLE** a executar com ficheiros completos.
  - **HCE** $-\ Bl$ - Um esquema **MLE** a executar a nível do bloco.
  - **CipherCE** - Um esquema de **MLE** que deriva a chave do primeiro bloco, utilizando-a para os restantes.
  - **HCESalted**$-\ WF$ - Um esquema de **MLE** com entropia injetada ($salt$) que funciona com o ficheiro completo.
  - **HCESalted**$-\ Bl$ - Um esquema de **MLE** com entropia injetada ($salt$) que funciona a nível do bloco.
  - $Std\ Cipher\ -\ WF$ - Uma cifra simétrica sem contemplar deduplicação, a funcionar com ficheiros completos.
  - $Std\ Cipher\ -\ Bl$ - Uma cifra simétrica sem contemplar deduplicação, a funcionar a nível do bloco.

5.2 Apresentação e análise dos resultados

5.2.1 Especificações do $benchmark$ e da máquina

Para testar a viabilidade dos esquemas baseados em **MLE** explicados em 4.3.3, foi implementado um programa específico à aplicação. O objetivo do mesmo passa pela medição de

tempos associados às técnicas de comparação e a técnicas de $baseline$ para interpretação de resultados.

Este $benchmark$ foi implementado em Java[1], consistindo em cerca de 1200 linhas de código, sendo por volta de 400 dedicadas à implementação das técnicas de **MLE** a serem utilizadas. Foi utilizada a versão de Java 1.6.0_27 com OpenJDK Runtime Environment (IcedTea6 1.12.6), **SHA-256** para funções de $hash$ (2.2.1), **AES**, para funções criptográficas simétricas, em modo **CBC** com $padding$ **PKCS5** (2.2.2). Para paralelização de processos foi utilizado a classe `java.lang.Thread` e, para aleatoriedade, a classe `java.security.SecureRandom` utilizando o construtor `public SecureRandom()` para inicialização do algoritmo.

Os diferentes testes abordados pelo $benchmark$ foram executados numa máquina com um processador Intel(R) Core(TM)2 Duo CPU T9300 @ 2.50GHz, com 4130Mb de memória RAM e sobre o sistema operativo Ubuntu 12.04.2 LTS.

5.2.2 Resultados da perspetiva do cliente

Para recolha de tempo de execução foram gerados, aleatoriamente, 40 ficheiros, com tamanho desde 250Kb a 10Mb (de 250Kb em 250Kb). Depois, foi repetida a operação a testar um n número de vezes para cada ficheiro gerado previamente, sendo medido o tempo de cada execução. Dividindo a soma destas operações pelo n escolhido obteve-se a média, sendo posteriormente construído um gráfico que ajuda a perceber como a operação reage a aumentos no tamanho dos ficheiros.

Deduplicação simples: o termo mais importante de comparação com técnicas de deduplicação é a deduplicação na sua forma mais simples. Para este teste será executado apenas uma operação de $hash$, tipicamente utilizada como termo de comparação entre dois ficheiros num sistema que contemple deduplicação. Como pode ser observado na figura 5.1, as operações de $hash$ são executadas inicialmente com tempo inferior a 4ms, crescendo linearmente com o aumentar do tamanho dos ficheiros. Neste caso, o tamanho de mensagem desta operação será apenas influenciado pela função de $hash$ (que, para este caso com **SHA-256**, será 256 bits). Sendo a mensagem $M = \{0,1\}^n$, o tamanho das mensagens será $n + 256$.

[1] http://www.oracle.com/technetwork/java/javase/downloads/index.html

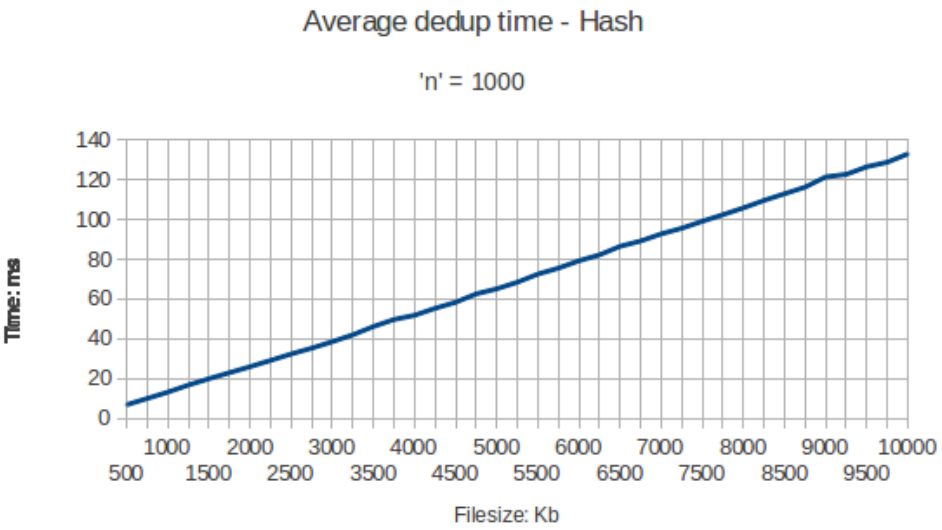


Figura 5.1: Tempos de execução do processo de deduplicação para n operações

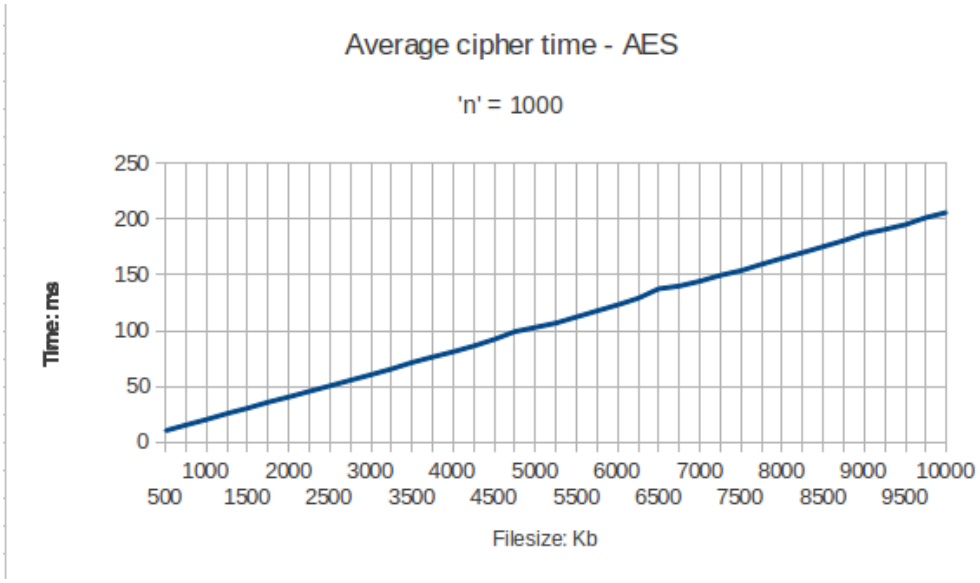


Figura 5.2: Tempos de execução do processo de cifragem para n operações

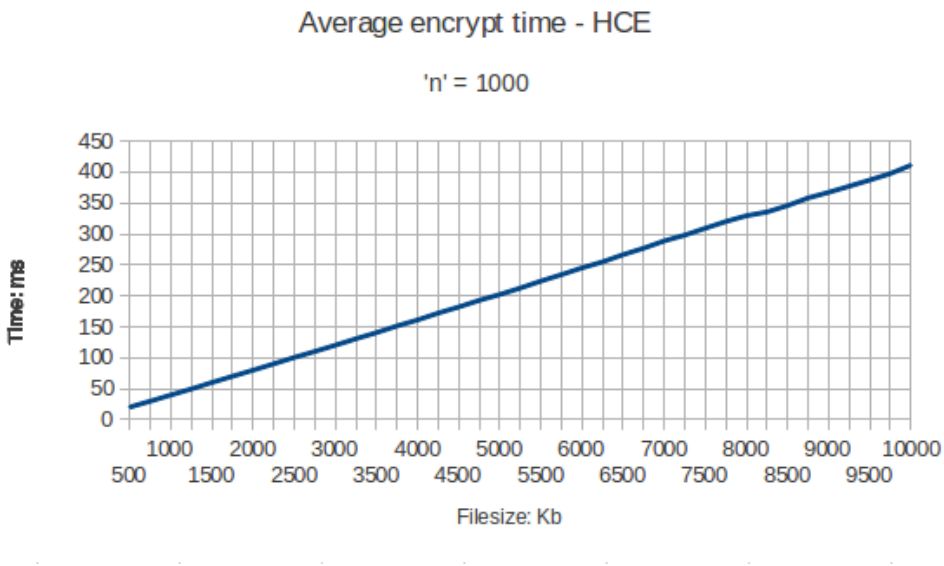


Figura 5.3: Tempos de execução do processo de cifragem HCE para n operações

Cifragem simples: neste teste será executada uma operação de cifragem **AES-CBC**, utilizando uma chave aleatória. O objectivo é obter um termo de comparação entre o custo de operação do esquema **MLE** com uma cifra $standard$ simétrica. Como pode ser observado na figura 5.2, as operações executadas demoram, inicialmente, cerca de 10ms, crescendo linearmente com o aumento do tamanho dos ficheiros. Para uma mensagem $M = \{0,1\}^n$, o tamanho de mensagens será n.

MLE-HCE: neste teste será executado o esquema de **MLE** denominado **HCE** (A.5). Como pode ser observado na figura 5.3, as operações iniciam num tempo de execução de 25ms e, tal como os anteriores, vão aumentando linearmente. Os valores obtidos são consistentes com o espectável, uma vez que o processo sequencial seria uma operação de $hash$ e uma de cifra simétrica, sendo o adicional provocado pela inicialização de $threads$ pela $Java\ Virtual\ Machine$ (**JVM**). Para esta operação, deve ser considerado o agregado da operação de $hash$ com a operação de cifragem. Sendo assim, temos que o tamanho das mensagens é $n + 256$.

Os tempos obtidos das operações de **HCE** são congruentes com o seu comportamento 4.3.3. De um ponto de vista funcional, pode-se aproximar o $tradeoff$ necessário à aplicabilidade deste esquema ao de adicionar a operação básica que falta a cada uma das abordagens. Isto significa que i. se quisermos partir de um sistema de deduplicação, devemos esperar uma degradação de performance ligeiramente superior a uma operação de

FileCount	BlockCount	Size	Dedup – WF	Dedup – BI	HCE – WF	HCE – BI	CipherCE	HCESalted – WF	HCESalted – BI	Std Cipher – WF	Std Cipher – BI
5443	5501	623Mb	1683 – 87Mb	1696 – 87Mb	1683 – 87Mb	1696 – 87Mb	1696 – 87Mb	144 – 3,0Mb	145 – 3,4Mb	0 – 0Kb	0 – 0Kb
6014	6156	1,1Gb	1972 – 91Mb	1980 – 91Mb	1972 – 91Mb	1980 – 91Mb	1980 – 91Mb	161 – 3,1Mb	151 – 2,9Mb	0 – 0Kb	0 – 0Kb
9720	9814	673Mb	2074 – 16,3Mb	2080 – 16,3Mb	2074 – 16,3Mb	2080 – 16,3Mb	2080 – 16,3Mb	99 – 0,2MB	71 – 0,2Mb	0 – 0Kb	0 – 0Kb
25202	25328	1,1Gb	5955 – 86,9Mb	6087 – 93,1Mb	5955 – 86,9Mb	6087 – 93,1Mb	6063 – 86,9Mb	737 – 0,2Mb	745 – 0,2Mb	0 – 0Kb	0 – 0Kb
27118	27263	1,4Gb	6386 – 100,6Mb	7121 – 106,8Mb	6386 – 100,6Mb	7121 – 106,8Mb	7002 – 100,6Mb	776 – 0,3Mb	501 – 0,1Mb	0 – 0Kb	0 – 0Kb
41574	41737	1,9Gb	11116 – 204Mb	12013 – 210Mb	11116 – 204Mb	12013 – 210Mb	12011 – 204Mb	810 – 0,4Mb	559 – 0,3Mb	0 – 0Kb	0 – 0Kb
91595	91843	2,3Gb	46447 – 390Mb	46581 – 394Mb	46447 – 390Mb	46581 – 394Mb	46580 – 390Mb	6069 – 31Mb	6096 – 32Mb	0 – 0Kb	0 – 0Kb
50763	51094	3,1Gb	14018 – 366Mb	15172 – 372Mb	14018 – 366Mb	15172 – 372Mb	15170 – 366Mb	934 – 1,7Mb	1521 – 2,3Mb	0 – 0Kb	0 – 0Kb

Tabela 5.1: Eficiência da deduplicação com blocos de 400Kb e 1 byte de salt

cifragem, e que ii. se quisermos adicionar a funcionalidade de deduplicação a um sistema que já realiza cifragem, esperamos um degradar de performance igualável a pouco mais que uma operação de $hash$.

A nível de tamanhos de mensagem, o aumento é quase negligenciável. No caso de um sistema que aplique cifragem, temos o aumento equivalente ao $output$ da função de $hash$ sendo este, pela própria natureza das funções, reduzido.

5.2.3 Resultados da perspetiva do servidor

Para uma avaliação de eficiência dos diferentes métodos a serem testados, foi escolhida uma amostra previamente utilizada num estudo de deduplicação [45, 46]. Como seriam executadas várias operações pesadas a nível de processamento, esta foi dividida em 8 sub-amostras. Estes resultados do $benchmark$ podem ser consultados na tabela 5.1.

As primeiras três colunas apresentam o número de ficheiros e o número de blocos verificados em cada run, seguidas do tamanho da amostra avaliada. As operações do $benchmark$ foram executadas para blocos de 400Kb.

Seguem-se colunas que apresentam os resultados da deduplicação a nível do ficheiro completo ($Whole\ File$: **WF**) e a nível do bloco ($Block$: **BI**). Ao contemplarem operações básicas de deduplicação, as duas primeiras colunas de resultados vão permitir estabelecer um valor base de eficácia, quando comparadas às restantes operações.

Imediatamente após a deduplicação encontram-se os resultados associados à aplicação da técnica **HCE**, de maneira a poder avaliar um dos esquemas propostos por Bellare em [8] em termos de $hits$ de deduplicação. Também neste ponto vai ser contemplada a avaliação por ficheiro completo e por blocos.

De seguida, encontram-se os resultados dos esquemas **CipherCE** e **MLESalted**. O primeiro apenas faz sentido ser avaliado com ficheiros completos, uma vez que tenta fazer uso do $header$ dos mesmos, enquanto que o segundo vai avaliar tanto ficheiros completos como por blocos. Para o mesmo, foi escolhido um tamanho de $salt$ de 1 byte.

Finalmente, como comparação final, é avaliada a qualidade da deduplicação quando executadas operações de cifragem que não estejam preparadas para deduplicação. Para este teste foi utilizado o **AES-CBC**.

Da referida tabela, podem tirar-se múltiplas conclusões:

1. A eficácia da deduplicação segura (esquemas de **MLE**) é igual à da deduplicação normal: 100%.
2. A deduplicação é inaplicável com técnicas de cifragem habituais (**AES-CBC**): 0%.
3. A nível de poupança de espaço, as técnicas de deduplicação por blocos conseguem atingir resultados superiores às que contemplam apenas ficheiros inteiros. De notar que esta é uma caraterística que pode melhorar com a diminuição do tamanho de blocos e que deve ser balanceada com o consequente aumento de processamento exigido.
4. A técnica de **MLESalted**, cujo objetivo passava por aumentar a entropia das mensagens, apresentou níveis extremamente reduzidos de eficácia na deduplicação, tendo esta sido reduzida para valores entre 8% e 0.2%. Daqui pode-se tirar que acréscimos na entropia das mensagens resultam em valores baixos de deduplicação.
5. A técnica de **CipherCE**, que recorre ao primeiro bloco para derivar a chave, não conseguiu níveis de eficiência superiores ao da deduplicação por ficheiro completo. Isto significa que, na execução deste $benchmark$, não foram encontrados dois ficheiros com blocos deduplicáveis, que tivessem os primeiros blocos iguais.

5.3 Conclusões do impacto e da utilidade das técnicas baseadas em MLE

Recorrendo aos resultados obtidos, podem-se agora tirar conclusões quanto à usabilidade das técnicas de **MLE** para finalidades de armazenamento seguro de ficheiros. Com finalidade de fazer inferências sobre os dados alcançados, vamos dividir os aspectos a discutir

por três pontos: duas das características discutidas na secção de deduplicação, nomeadamente performance e eficiência, e segurança, razão da existência deste tipo de esquemas.

A nível de performance, tal como foi observado em 5.2.2, os clientes têm o acréscimo das operações de identificação e de cifragem associadas à técnica a utilizar. Isto reflete-se como sendo um $tradeoff$ análogo à da transição para um sistema que aplique técnicas de segurança do lado do cliente, tanto a nível de tempo de execução como a nível de tamanho das mensagens. Em relação à eficiência, as técnicas de **MLE** atingiram valores semelhantes aos dos métodos de deduplicação habituais, podendo-se assumir que os servidores não esperam nenhum deteriorar na transição para este tipo de esquemas, a nível de espaço economizado.

A segurança, no entanto, apresenta-se como a maior falha dos esquemas de **MLE**, uma vez que os nossos testes revelam que pequenos acréscimos na entropia das mensagens, que poderiam dar alguma esperança de confidencialidade, implicam deduplicação praticamente zero.

Adicionalmente, a relação de segurança/eficiência é reduzida ainda mais se tivermos em conta a própria natureza das técnicas de deduplicação: a segurança está limitada à entropia presente no sistema, no entanto, a deduplicação prospera com a existência frequente de duplicados, o que se traduz em sistemas de baixa entropia. Isto significa que os esquemas de **MLE**, com uma segurança limitada pela já reduzida entropia espectável de um sistema alvo de deduplicação, são ainda mais vulneráveis a ataques de força bruta ajustados ao tipo de informação armazenada no sistema.

Capítulo 6

Sistema proposto

6.1 Especificação do sistema

Para este sistema, pretende-se que sejam disponibilizadas as contramedidas do tipo C_2, sendo aplicadas as técnicas de segurança do lado do cliente. Para tal, vai ser seguido o esquema de funcionamento e estrutura de dados apresentados em [35]. Tal como o referido, este sistema tem o objetivo de funcionar como uma camada de segurança, oferecendo garantias de segurança independentes do serviço no qual esta instalado.

Ao contrario da atitude tomada em [35, 36], este trabalho pretende enfrentar a problemática da partilha de chaves através da aplicação de **CL-PKC** (2.2.4). Deste modo, o sistema proposto não tem necessidade de delegar responsabilidades de gestão de chaves para os utilizadores, nem exige confiança total numa entidade de distribuição de chaves.

Numa perspetiva de expandir as funcionalidades oferecidas, o sistema deve disponibilizar a possibilidade de funcionar com esquemas de **MLE** (4.3). Desta forma, pode ser realizada uma adaptação ao próprio, caso as exigências de segurança sejam relaxáveis ao ponto do MLE ser aceitável para os utilizadores.

6.1.1 Critérios de design

Antes de apresentar a arquitetura e a estrutura do sistema, vão ser descritos alguns critérios que foram tidos em conta no desenvolvimento do mesmo. A explicação destes pontos

pode ser usada para, posteriormente, medir e justificar a validade das decisões tomadas na implementação, bem como avaliar os resultados conseguidos com o sistema em si.

- Sistema como camada. Este sistema deve funcionar como uma camada de segurança a ser aplicada sobre um serviço de partilha de ficheiros, não podendo assim exigir que o sistema sobre o qual está inserido providencie operações não espectáveis de serviços do género.

- Programa minimal. Sendo um sistema que corre na máquina pessoal de um utilizador de serviço de $cloud$, não se pode esperar que esta incorra em operações pesadas a nível de processamento ou que necessitem de grande capacidade de armazenamento local.

- Performance adequada. Para a aplicação deste sistema na prática, este tem de apresentar níveis de performance equiparáveis a serviços semelhantes. Caso contrário, é improvável que utilizadores mais exigentes a este nível mostrem interesse na sua utilização.

- Baixa largura de banda. Este sistema deve recorrer a uma largura de banda comparável à de serviços semelhantes, ou os utilizadores com fracas conexões não poderão usufruir do programa.

- Leve distribuição de chaves. Para o envio de chaves entre utilizadores, a comunicação extraordinária ao sistema deve ser minimizada, sendo utilizado um esquema que permita a partilha de chaves sem exigência de excessiva informação transmitida no processo.

6.1.2 Arquitetura do sistema

O sistema é constituído por dois componentes, o programa do utilizador e o $daemon$ de atualização - fig. 6.1.

O programa do utilizador é a parte do sistema que vai interagir com o próprio, sendo esse a "ponte" entre os dados que o cliente possui no disco e a pasta a ser armazenada na $cloud$. Este é o componente central e o único cujo funcionamento o utilizador deve compreender de modo a poder utilizar adequadamente o sistema. Este está encarregue de permitir ao

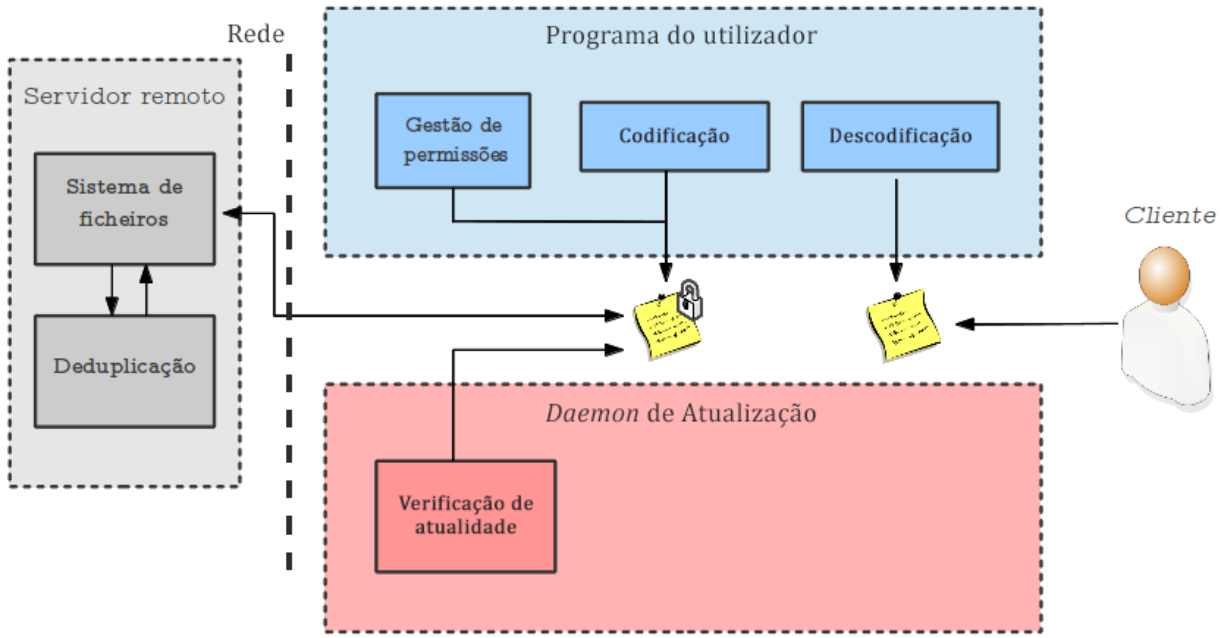


Figura 6.1: Arquitetura do sistema

utilizador aceder às funcionalidades do mesmo, executando codificações e descodificações, bem como todas as operações relacionadas com permissões sobre os ficheiros.

O *daemon* é um componente que pode passar despercebido ao cliente menos informado, executando em *background* e exigindo apenas capacidade de processamento periódica. Este é responsável pela integridade dos ficheiros de posse, o que faz com que, não sendo um componente sobre o qual o cliente interage ou sequer tem necessidade de saber muito sobre a sua função, represente uma parte importante nas metas de segurança a atingir.

6.1.3 Estrutura dos dados

Seguindo o tipo de estrutura proposta no sistema SiRiUS, cada utilizador vai possuir apenas dois pares de chaves, que tem de gerar no inicio da utilização do sistema: *User Encryption Key* (**UEK**), um par de chaves assimétricas utilizadas na cifragem/decifragem e *User Signature Key* (**USK**), um par de chaves assimétricas utilizadas na assinatura/verificação de ficheiros. Desta forma, cada utilizador só tem de armazenar a versão privada desses pares. Para além destas chaves, o sistema faz uso de quatro estruturas associadas ao armazenamento da informação. Uma delas é referente aos dados, apenas associando identificadores

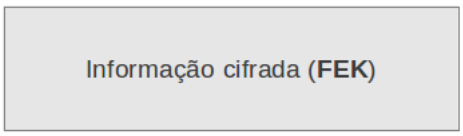


Figura 6.2: Ficheiro de dados: d-file

à informação respetiva. A segunda engloba metadados referentes a ficheiros individuais. A terceira é a de posse, sendo que contém apenas o nome do ficheiro, a lista de utilizadores com permissões de escrita sobre um determinado ficheiro e a assinatura do dono do mesmo. Finalmente, a quarta estrutura permite manter a atualidade dos ficheiros de posse, mantendo o $hash$ e o $timestamp$ assinados pelo utilizador.

- d-$file$, o ficheiro que contém informação. Uma vez que o sistema suporta esquemas baseados em **MLE**, cada d-$file$ vai estar associado a um identificador relativo à sua informação. - 6.2.
- md-$file$, contendo o identificador da informação para recolha, o utilizador que realizou a última alteração, o nome do ficheiro, uma lista de blocos contendo a chave do ficheiro, cifradas com as **UEK** públicas dos utilizadores com permissões para acesso ao ficheiro e a assinatura dos dados do ficheiro md-$file$ em questão com **USK** do utilizador que realizou a última alteração - 6.3.
- own-$file$, contendo o nome do ficheiro, uma lista de utilizadores com permissões de escrita (que, por consequência, podem assinar o md-$file$) e a assinatura do $owner$ do ficheiro, com a sua **USK** - 6.4.
- $ownf$-$file$, contendo o nome da diretoria, o $hash$ dos $hashes$ dos ficheiros own-$file$, concatenados de forma lexical, um timestamp e, caso se trate de um ficheiro de $ownf$-$root$, tudo isto assinado pela **USK** do utilizador - 6.5.

6.1.4 O $daemon$ e os ficheiros de atualidade

Ataques de $rollback$ consistem na substituição de um determinado ficheiro no sistema por uma versão mais antiga do mesmo, com objectivo de utilizar isso para contornar algum mecanismo de segurança no sistema. Como exemplo, a Alice inicialmente atribuiu permissões

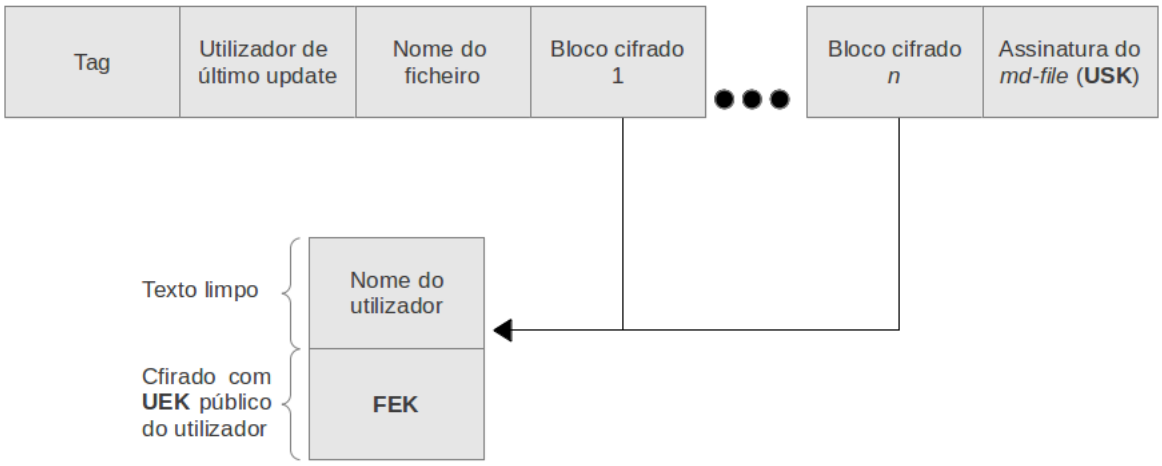


Figura 6.3: Ficheiro de dados: md-file

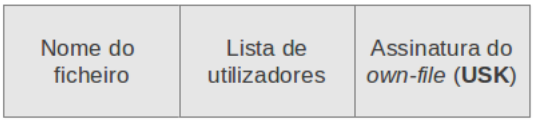


Figura 6.4: Ficheiro de dados: own-file

Figura 6.5: Ficheiro de dados: ownf-root

ao Bob para aceder a um dado ficheiro F, no entanto, a Alice recebeu notícias que indicam que o Bob pode ser malicioso e revoga as permissões do Bob. Neste caso, o Bob tenta substituir o ficheiro de permissões recente pelo ficheiro de permissões existente quanto ainda tinha acesso a F, cuja assinatura é válida, recuperando acesso ao ficheiro que pretende atacar. Atualidade trata-se de uma garantia de que os ficheiros foram verificados recentemente e permite contrariar este tipo de ataques, rejeitando qualquer ficheiro cujo $timestamp$ não se encontre dentro do intervalo de tempo considerado válido.

O sistema permite garantir a atualidade dos seus ficheiros de permissões recorrendo ao mecanismo utilizado pelo SiRiUS. A solução do mesmo passa por utilizar uma $hash\ tree$ [43] que engloba todos os $own\text{-}files$. Cada diretoria tem um ficheiro $.ownf$ que garante a atualidade da mesma diretoria, contendo o $hash$ de todos os ficheiros $.own$ e todos os $.ownf$ de sub-diretorias, sendo o ficheiro $.ownf$ da pasta principal chamado $ownf\text{-}root$. Para o processo existem três funções principais envolvidas: A geração inicial dos ficheiros $.ownf$ e $ownf\text{-}root$, a atualização periódica de $ownf\text{-}root$ e a verificação de validade da árvore. O $daemon$ é responsável por estas funções, enquanto que o programa do cliente vai realizar verificações e atualizações esporádicas, quando os ficheiros de posse tiverem de ser consultados ou alterados.

De seguida, vai ser descrito como o sistema realiza as operações indicadas acima:

- Gerar ficheiros $ownf$:

    1. Aplicar uma função de $hash$ a cada ficheiro de posse, verificando as respetivas assinaturas.

    2. Concatenar os $hashes$ de cada ficheiro de posse com os $.ownf$ de cada sub-diretoria em ordem lexicográfica.

    3. Aplicar uma função de $hash$ à concatenação.

    4. Caso se trate de um ficheiro $ownf\text{-}root$, marcar com o $timestamp$ atual, assinar o conteúdo e armazenar em $cache$.

    5. Armazenar o ficheiro.

- Verificar atualidade da árvore:

1. Recalcular o ficheiro $.ownf$ da diretoria, seguindo os 3 primeiros passos da operação anterior, e comparar ao armazenado no $.ownf$ a avaliar. Caso a comparação falhe, abortar.

2. Se a diretoria é $root$, verificar $timestamp$ e assinatura, caso não seja, aplicar esta operação recursivamente subindo na árvore.

- Atualizar o ficheiro $ownf\text{-}root$: Este processo é realizado por ambos os componentes. O $daemon$ tem obrigação apenas de o validar e atualizar o $timestamp$. Para isso, este faz uso da $cache$ local para comparar ao $ownf\text{-}root$ atual, refrescar o $timestamp$, assinar a nova informação e voltar a guardar em $cache$, tornando esta uma operação relativamente leve. Por outro lado, o programa tem de executar esta operação quando forem realizadas alterações nas permissões. Graças à estrutura da árvore, mudanças em ficheiros de posse alteram apenas a diretoria atual e todas as diretorias acima na árvore, diminuindo a quantidade de ficheiros a serem recalculados de forma semelhante à descrita nos passos 1-5 da operação de geração.

6.2 Componentes e operações

Este sistema é composto por dois componentes. O primeiro é o programa de codificação, que disponibiliza todas as operações executadas pelo cliente. Como tal, a inicialização do mesmo exige:

- `Username` - O nome do utilizador. Este nome vai ser utilizado para o programa saber que chave utilizar, quando for necessário realizar operações criptográficas, e para identificar o utilizador nas alterações (para assinaturas de ficheiros $.md$).

- `PathToStore` - A localização da diretoria onde armazenar ficheiros codificados.

- `PathToRestore` - A localização da diretoria onde armazenar os conteúdos descodificados da pasta definida em `PathToStore`.

- `PathToKeys` - Predefinido como "`/tmp/keys`", é a localização para o programa armazenar as chaves geradas e onde procurar chaves necessárias às operações.

- `TimeDif` - A diferença de tempo (em segundos) a partir da qual um ficheiro é considerado obsoleto. Este critério define o quão estrito vai ser o sistema contra ataques de $rollback$.

O segundo componente do sistema é o $daemon$ de atualização. Como o objetivo deste passa pela execução de funções em $background$, é importante a correta definição dos argumentos das mesmas:

- `Username` - Tal como no programa, utilizado para identificar o utilizador, de modo a saber que chaves utilizar.
- `Path` - O $daemon$ está dedicado à pasta codificada, portanto, este argumento deve ser o mesmo utilizado em `PathToStore` no programa do cliente.
- `PathToKeys` - Análoga à do componente anterior
- `TimeDif` - Valor igual ao escolhido no programa, neste caso representando o intervalo de tempo entre atualizações.

6.2.1 Diagramas e detalhar de funções

Para a abordagem detalhada das funcionalidades do sistema, é importante ter em conta os algoritmos referidos em 6.4, uma vez que serão utilizados para estes métodos. As descrições extensivas, por uma questão de facilidade de leitura, podem-se consultar no anexo B

Programa de codificação

- Codificar pasta - Esta função recebe, como $input$, a pasta que o utilizador deseja que seja codificada, executando as cifragens e decifragens necessárias.
- Descodificar pasta - Operação inversa à codificação.
- Atribuir permissões - Operação que permite o utilizador atribuir permissões, consoante o $input$ da mesma (ficheiro, utilizadores e opção).
- Revogar permissões - Operação inversa à de atribuição de permissões.

$Daemon$ de atualização

- Refrescar $hash\ tree$ - Única função do $daemon$. Esta é realizada periodicamente, sendo que a primeira iteração envolve a geração da árvore (ver funções auxiliares).

Funções auxiliares

- Verificar atualidade da árvore - Esta operação deve ser realizada sempre que se verifica uma assinatura, de modo a ser notificável a existência de possíveis ficheiros obsoletos.
- Gerar $hash\ tree$ - Esta operação é executada na primeira iteração do $daemon$ e por cada vez que o utilizador alterar permissões de escrita (o que envolve alterar ficheiros $.own$), consistindo na atualização da árvore.

Para a melhor compreensão das funções de gerar, verificar ou refrescar a árvore, no anexo C encontram-se diagramas de atividade que ilustram o seu funcionamento.

6.3 Análise de segurança

Esta seção visa abordar as ameaças de segurança listadas em 3.4 e apresentar contramedidas que atendam aos requisitos de segurança referidos em 3.4. Segue-se uma avaliação do risco resultante das soluções propostas, bem como uma validação dos requisitos contemplados.

De modo a atingir confidencialidade, o sistema recorre a cifras simétricas para a cifragem dos dados (C_1). Cada d-$file$ vai encontrar-se sempre em criptograma, dependendo do esquema utilizado para garantir confidencialidade da informação armazenada. É importante ter em conta que, uma vez que o sistema permite a utilização de **MLE**, o nível de segurança deve ter contemplar estes casos (C_1').

Para proteger a integridade dos dados, o sistema recorre a um sistema de assinaturas digitais (C_2). Estas vão ser utilizadas para ficheiros de metadados, como md-$file$, own-$file$ e $ownf$-$file$, e serão implementadas utilizando **CLS**.

	PA	**AA**	Utilizadores	Provedores
Quebra de confidencialidade	C_1, C_1'	C_1, C_1'	C_1, C_1'	C_1, C_1'
Alteração dos dados		C_2	C_2	C_2
DoS		C_5	C_5	C_5
Acesso inadvertido		$C_2 + C_3$	$C_2 + C_3$	$C_2 + C_3$
Ataques por $rollback$		C_4	C_4	C_4
Quebra de confiança			$C_2 + C_3$	$-$
$Takeover$			$-$	$-$
Perda de chaves			C_3^*	C_3^*
Corrupção acidental			$-$	$-$
Divulgação da informação				C_1, C_1'

Tabela 6.1: Abrangência das contramedidas apresentadas. "$-$" representam ameaças não contempladas no trabalho, "$*$" representam limitada cobertura à ameaça.

A partilha de chaves é realizada através de **CL-PKE**, utilizando uma metodologia semelhante à dos sistemas atuais (C_3). As chaves de cada ficheiro são armazenadas em conjunto com o mesmo e estão dependentes da integridade do próprio ficheiro de metadados.

A integridade dos ficheiros de metadados é conseguida através de uma $hash\ tree$ (C_4). Esta árvore é gerida por um $daemon$ de atualização, responsável por realizar constantes verificações aos ficheiros centrais do sistema. Este mecanismo é crucial para garantir o bom funcionamento do sistema de permissões.

A utilização do sistema como camada de segurança para codificação e descodificação de dados mantém uma cópia dos ficheiros localmente (C_5). Desta forma, o utilizador é salvaguardado contra possíveis $downtimes$ do serviço no qual está inserido.

Na tabela 6.1, encontram-se representadas as ameaças protegidas pelas contramedidas abordadas.

6.3.1 Avaliação do risco

Tendo sido enumerados os agentes e as respetivas ameaças, bem como descritas as contramedidas aplicadas para responder às mesmas, é importante atender ao risco associado.

Seguindo o esquema apresentado na fig. 2.1, vai ser detalhadamente analisado até que nível se espera que as contramedidas propostas resolvam os problemas que se espera enfrentar. Nesta análise, vão também ser tidas em conta as suposições de segurança (3.4), relativamente às exigências de cada ponto. Para auxilio visual, vai-se recorrer à tabela 6.1, descrevendo cada caso individualmente.

A quebra de confidencialidade deve ser analisada perante as duas vertentes. No caso de C_1, são utilizadas as cifras simétricas apresentadas em 2.2.2. Recorrendo a estes esquemas, os dados armazenados atingem a propriedade de indistinguibilidade e, caso seja utilizado um modo **CBC** ou semelhante, o esquema consegue ser **CPA**-seguro. Quando são utilizados esquemas de **MLE** (C_1'), os criptogramas são distinguíveis para sistemas que não possuam uma alta entropia associada às mensagens dos mesmos. Nestes casos, o nível de segurança é drasticamente reduzido, não se alcançando indistinguibilidade de criptogramas.

Caso os dados sejam alterados por adversários, a utilização de assinaturas digitais (C_2) permite que o utilizador se aperceba dessas mudanças. A alteração de dados pode também ser realizada indiretamente, contornando este mecanismo, mas esse é um tipo de ataque abordado posteriormente. Esta contramedida não contempla a recuperação de ficheiros corrompidos, sendo possível recorrer a sistemas tolerantes a falhas bizantinas caso sejam necessárias medidas adicionais de confiabilidade.

Ataques por **DoS** não são diretamente contrariados pelo sistema. No entanto, e uma vez que este funciona através de sincronização de pastas, o utilizador não perde completamente o acesso à sua informação. Este nível de segurança depende do quão frequentes forem realizadas as operações de codificação/descodificação e pode ser reforçado através da imposição de uma política de segurança associada a essa atitude.

Acesso inadvertido é impossibilitado pela assinatura dos metadados e das chaves para leitura dos dados. De forma a realizar alterações, é necessário ter acesso à chave privada de assinatura, e para conhecer a chave associada aos criptogramas armazenados, é necessário ter acesso à chave privada de decifragem. Assim, a contramedida C_2 responsabiliza-se por não ser possível realizar alterações maliciosas às permissões de escrita e a contramedida C_3 é aplicada para não ser possível subjugar a leitura à informação armazenada.

Os ataques por $rollback$ são contrariados, principalmente, pela utilização da árvore de $hash$ (C_4). Contudo, este tipo de ataques merece uma abordagem extensiva, de maneira a ser

integralmente compreendido e, como tal, os ficheiros passíveis de substituição vão ser abordados um a um: i. no caso de ser tentado substituir um ficheiro de dados por um obsoleto, o identificador associado ao mesmo vai ser incongruente com o presente nos ficheiros de metadados. ii. No caso de ser substituído o ficheiro de metadados por uma versão mais antiga, o identificador no próprio vai referir o ficheiro de dados anterior, maximizando o potencial deste ataque ao de restituir acesso de leitura a um ficheiro já revogado. Este ataque pode ser igualmente atingível através do $backup$ dos ficheiros acessíveis, considerando-se, assim, de menor relevância. iii. Caso o alvo seja um ficheiro de posse, permitindo acesso aos anteriormente referidos, C_4 vai garantir que apenas os $own\text{-}f$ mais recentes sejam considerados válidos, negando substituições de ficheiros com $timestamps$ mais antigos que o considerado aceitável. Neste último ponto, o intervalo também deve ser avaliado e adaptado consoante as necessidades específicas ao sistema.

No caso da perda de confiança num utilizador, é importante fazer uso dos mecanismos de revogação. Tendo em conta o tipo de revogação aplicada (4.3.2), se o utilizador comprometido tinha permissões de leitura do ficheiro, este vai conseguir manter acesso até a próxima alteração do mesmo (C_3). Por outro lado, se o utilizador comprometido tinha permissões de leitura e de escrita sobre o ficheiro, este perde imediatamente a possibilidade de realizar alterações (C_2).

O $takeover$ do sistema acontece quando um utilizador pretende alterar os dados alterando a totalidade da $hash\ tree$ para validação dos ficheiros de posse, essencialmente assumindo controlo do sistema de ficheiros. Este ataque é contrariável quando o utilizador original se aperceber da situação, tipicamente durante uma das atualizações periódicas. Apesar do sistema não proteger diretamente este ataque, este só funciona quando os utilizadores não tiverem noção do verdadeiro $owner$ dos ficheiros a serem acedidos.

Mesmo assumindo que o utilizador vai tentar proteger as chaves que possui, é sensato avaliar as consequências de uma possível falha. Para tal, vamos assumir que o Bob é o utilizador malicioso que obteve esta nova chave da Alice. Caso o Bob obtenha a chave de um dos ficheiros da Alice, este vai conseguir aceder aos dados do mesmo, sendo esta uma consequência semelhante à da necessidade de revogação de acesso a um utilizador com permissões de leitura (C_3). Caso o Bob obtenha a chave mestra privada de cifragem, este vai conseguir ter acesso a todos os ficheiros aos quais tinha acesso a Alice, atribuindo-se permissões de leitura sobre todos esses ficheiros, sendo necessária uma revogação em massa da informação partilhada com a Alice (C_3). Caso o Bob ganhe acesso à chave

mestra de assinatura da Alice, apesar de não conseguir ler a informação armazenada, passa a conseguir alterar esses ficheiros, atribuir permissões de escrita e revogar permissões sobre os mesmos. Este resulta em consequências semelhantes ao $takeover$ do sistema de ficheiros, sem a facilidade de identificação do adversário.

A corrupção de dados já foi referida previamente como um problema que o sistema consegue assinalar mas não contornar. Adicionalmente, é realizada uma suposição quanto à alteração dos dados significar corrupção e não destruição (3.4). Sobre essa assunção, o utilizador consegue sempre ter $feedback$ de possíveis alterações acidentais.

Com a mesma justificação que a quebra de confidencialidade, caso o provedor não consiga retirar informação dos criptogramas armazenados, não será possível que este divulgue a informação que possui para fins menos legítimos.

6.3.2 Validação dos requisitos

Na componente funcional, foi pretendido que os utilizadores conseguissem codificar e descodificar as suas pastas, recuperando sempre a informação armazenada. Este ponto é até ilustrado em 6.5, sendo que a pasta resultante da sequência de operações "codificar-descodificar" tem conteúdo semelhante à original. Desta forma, este requisito considera-se conseguido. Para além disto, definiu-se como importante que os utilizadores conseguissem partilhar os seus documentos e categorizar essas partilhas. `addPermitions()` e `removePermitions()` ,contempladas em 6.2.1, não só permitem que o utilizador estenda o acesso dos seus ficheiros a terceiros, como também esperam receber uma opção que define o tipo de permissão a receber: $read\text{-}only$ ou $read\text{-}write$. O programa pode ainda ser alterado em pontos como `PathToStore` e em `PathToKeys` para permitir a configurabilidade pretendida.

As funções de gerar, validar e refrescar $hash\ tree$, detalhadas em B, acedem às necessidades referidas na seção de requisitos operacionais, certificando-se que os ficheiros de posse não são alvos de ataques por $rollback$. De notar que estes métodos correm num $daemon$ em $background$ sem o utilizador necessitar de interagir diretamente com o mesmo, também indicado como requisito operacional.

A nível de requisitos de segurança, a validação vai ser realizada por partes, uma vez que depende das técnicas usadas. Como adição a este ponto, é aconselhado consultar a seção 6.3.

- Confidencialidade dos ficheiros de dados é assegurada do lado do cliente, significando que em nenhum ponto da transmissão/armazenamento dos dados estes se encontram decifrados. Dessa forma, a confidencialidade depende da segurança definida pelas técnicas de cifragem simétricas dos dados (4.3.3) e pelas técnicas de cifragem assimétrica das chaves (2.2.4).

- A integridade dos ficheiros é garantida através de assinaturas realizadas sobre os dados e, como tal, está dependente da robustez das técnicas adotadas (2.2.4).

- Uma vez que o serviço realiza medidas de segurança do lado do cliente, e que as partilhas de ficheiros são realizadas do lado do cliente, o controlo de acesso é independente do serviço remoto a utilizar.

- A revogação de permissões é permitida no sistema, como foi referido na componente funcional, e os seus efeitos ocorrem o mais rápido possível, atendendo à metodologia aplicada (4.3.2).

- Utilizando **CL-PKE**, a comunicação externa é limitada à transmissão das chaves públicas entre utilizadores e ao processo de geração de chaves (2.2.4)

- Como explicado em 6.1.3, cada utilizador apenas tem de armazenar duas chaves privadas, conseguindo-se assim reduzir o espaço de armazenamento exigido ao cliente.

Desta forma, foram abordados e atendidos todos os requisitos apresentados na seção anterior, atendendo às assunções relativas à segurança do sistema (3.4).

6.4 Ambiente e configuração

Para o desenvolvimento deste sistema, foram utilizadas múltiplas ferramentas e linguagens de programação, de modo a poder cruzar funcionalidades e disponibilizar os diferentes requisitos funcionais e de segurança. A abordagem do resultado final exige, portanto, uma especificação dos recursos utilizados e, se for o caso, a versão utilizada dos mesmos.

- **Especificações da máquina:**
  - $Intel(R)\ Core(TM)2\ Duo\ CPU\ T9300\ @\ 2.50GHz$

  - 4130Mb de memória RAM

- **Sistema operativo:**
  - *Ubuntu 12.04.2 LTS*.

- **Compiladores:**
  - *Gcc 4.6.3*
  - *Java 1.6.0_27* com *OpenJDK Runtime Environment (IcedTea6 1.12.6)*

- **Ferramentas de auxilio:**
  - *NetBeans 7.0.1*
  - *NetBeans C/C++ plugin 1.14.3.1*

Nas diferentes ferramentas, foram utilizados/implementados algoritmos criptográficos para as diferentes funcionalidades. De seguida, passamos a enumerar os mesmos:

- Utilizado **AES** de `javax.crypto.Cipher`, de acordo com o especificado por NIST em *FIPS 197*[1], a executar em modo **CBC**, como definido em FIPS *PUB 81*[1], recorrendo ao *padding* **PKCS5**, como descrito em [4].
- Utilizado **SHA-256** de `java.security.MessageDigest` e de `<openssl/sha.h>`, de acordo com o definido em FIPS *PUB 180-2*[1].
- Implementado `Basic CL-PKE` e `CLS`, como descrito em [3].
- Implementado **HCE** como descrito em [8].

6.4.1 Configuração e operação dos componentes

Uma vez que se trata de um protótipo de estudo, a sua utilização exige a instalação dos ambientes descritos em 6.4. Os componentes possuem uma pasta de *output* predefinida como `./Ready`, sendo o utilizador aconselhado a mover o programa e o *daemon* para uma

[1] http://csrc.nist.gov/publications/PubsFIPS.html

localização abrangida pelo serviço de $cloud$ que pretendem utilizar, ou alterar a pasta de $output$ para essa localização. Para além disso, a diretoria de chaves públicas está definida como `/tmp/keys`, sendo encorajado o utilizador à sua alteração.

Para a operação do sistema, é necessário que se atinjam as condições descritas no parágrafo anterior e que se inicialize o $daemon$ de atualização. Depois, todas as funcionalidades são acessíveis através do programa de codificação: `normFolder()` recebe como $input$ a pasta que se quer codificada, `denormFolder()` recebe como $input$ o local para onde o utilizador deseja que a pasta seja descodificada e `addPermitions()` + `removePermitions()` são as funções utilizadas para atribuir e revogar permissões, respetivamente, recebendo o ficheiro, uma lista de utilizadores e a opção (R para apenas leitura e RW para leitura e escrita).

6.5 Validação

Para a representação do funcionamento, foi escolhido um exemplo bastante simples, constituído apenas por 4 pastas (a principal e 3 sub-pastas) e 5 ficheiros, facilitando a visualização das diferenças entre codificação e descodificação. Executado foi: i. a operação de codificação, ii. a operação de descodificação e iii. um dos cálculos periódicos do $daemon$ responsável pela atualização dos ficheiros de posse:

- figura 6.6: A pasta escolhida pelo utilizador contém os ficheiros num formato de uso normal. Este é o alvo do nosso algoritmo de codificação.

- figura 6.7: A pasta codificada. Cada ficheiro resultou em 3 a si associados ($.d$, $.md$ e $.own$, como explicado previamente) e cada pasta possui um ficheiro de atualidade $.ownf$. O ficheiro de $root\text{-}ownf$ foi também criado no processo.

- figura 6.8: A pasta resultante. Trata-se da pasta descodificada, semelhante à da figura 6.6. Como registo, existe ainda um ficheiro "$Logfile.txt$", que contém informações relativas ao processo, como verificação de assinaturas, consistência de dados, permissões, etc. Este ficheiro não é necessário e a sua remoção não tem quaisquer repercussões negativas no sistema.

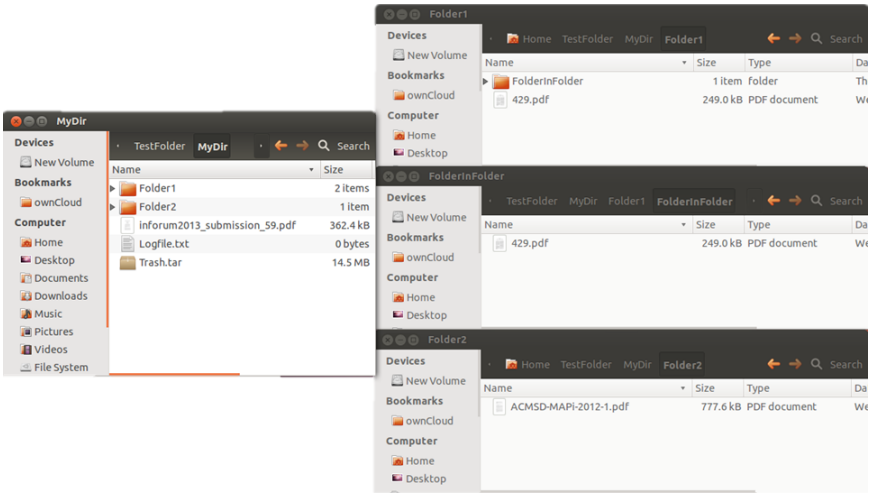


Figura 6.6: Pasta do utilizador

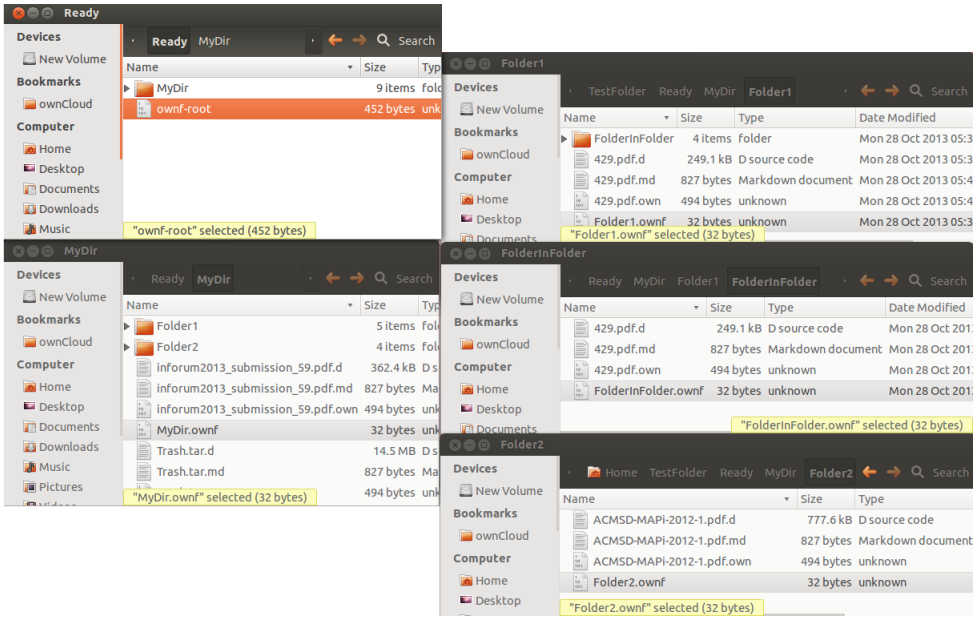


Figura 6.7: Pasta codificada

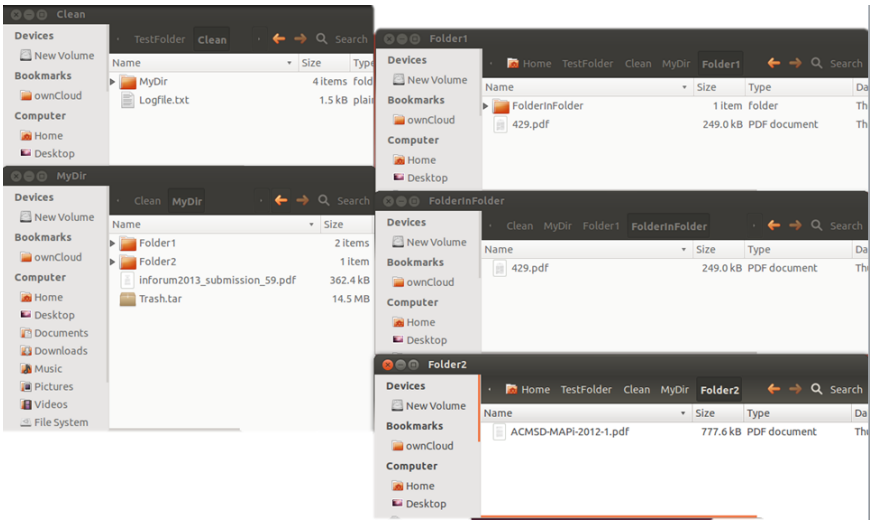


Figura 6.8: Pasta descodificada

De notar o processo de atualização, produzindo *hash trees* de tamanho equivalente à concatenação dos *hashes* individuais (explicado em 6.2.1). Como se pode observar na fig. 6.7, os ficheiros de atualidade possuem todos 32 bytes, o que é congruente com o *output* espectável da função de *hash* utilizada. O último passo envolve calcular o *root-ownf*, resultando num ficheiro maior (443 bytes), uma vez que este deve incluir o *timestamp* e deve ser assinado pelo utilizador.

Capítulo 7

Conclusões e trabalho futuro

7.1 Conclusões

Segundo a seção 1.4, os objetivos podem dividir-se em 4 pontos principais: análise de segurança dos serviços atuais, exploração de abordagens teóricas, experimentação de alternativas e implementação de um sistema. Como tal, a apresentação das conclusões vai seguir esta mesma estrutura.

A generalidade das soluções atuais encontra-se numa posição desvantajosa a nível de garantias de segurança oferecidas. A abordagem de confiança total no provedor é propícia a falhas do mesmo, não sendo um nível de confiança aceitável para entidades com informação sensível. A alternativa oposta, por outro lado, consiste em realizar a cifragem do lado do cliente, atribuindo ao sistema um nível de segurança adequado à utilização de servidores remotos não confiáveis. O problema desta solução trivial é que vai contra a ideia do modelo de $cloud$, colocando peso de processamento do lado dos clientes e inutilizando a capacidade dos servidores remotos. A falta de primitivas criptográficas adequadas a esta necessidade apresenta-se como o maior obstáculo, não existindo forma de realizar cálculo sobre informação cifrada, nem forma de partilhar chaves sem tornar o sistema demasiado obtuso.

As abordagens teóricas exploradas atacam dois problemas essenciais: i. partilhar informação sem confiança total no servidor remoto e ii. executar operações sobre criptogramas.

i. Foram explorados os sistemas SiRiUS e Plutus. O SiRiUS apresentou-se como o candidato mais adequado ao modelo de $cloud$ (4.2.1), demonstrando o funcionamento de uma camada que permite a aplicação de técnicas de segurança do lado do cliente com partilha de ficheiros e exigências reduzidas a nível de partilha de chaves. Porém, o processamento criptográfico exigido na sua execução e a ausência de resposta à problemática da partilha de chaves revelaram-se como as suas maiores desvantagens.

ii. Na execução de operações sobre criptogramas, foi considerada a utilização de esquemas baseados em **MLE**. Esta primitiva permite a comparação de dois criptogramas em relação ao seu conteúdo, de modo a poder ser realizada deduplicação. Para além das exigências de processamento adicionadas, esta tem a desvantagem de reduzir drasticamente o nível de segurança do sistema, consequência das inferências possíveis sobre os dados armazenados.

O $benchmark$ implementado para experimentação com técnicas de **MLE** permite que sejam atingidas conclusões mais específicas sobre a técnica em questão. Como apresentado em 5.2.2, os resultados dividem-se em perspetiva do cliente e perspetiva do servidor. Do lado do cliente, as técnicas de **MLE** não revelaram exigências inesperadas, conseguindo-se valores adequados em relação aos termos de comparação escolhidos. Do lado do servidor, o **MLE** apresentou eficácia elevada (semelhante à da deduplicação normal), mas a tentativa de gerar entropia nas mensagens para elevar o nível de segurança teve resultados insatisfatórios. Assim, pode concluir-se que o **MLE** é uma primitiva que, a nível de deduplicação, atinge excelentes níveis de eficácia mas que, a nível de segurança, exige um sério relaxamento do modelo de segurança. Adicionalmente, a ideia de atribuir segurança apoiada na entropia (criptogramas sem indistinguibilidade) é contraditória à natureza dos sistemas com mensagens previsíveis, propícios às técnicas de deduplicação.

O sistema de partilha segura de ficheiros implementado permite o desenvolvimento de diferentes pontos a explorar.

- A estruturação de dados e o funcionamento do SiRiUS pode ser adaptado à utilização de técnicas com **MLE**, quando acrescentada a noção de $ownership$ partilhada e movido o mecanismo de atualização para estes novos ficheiros. Esta nova abordagem faz uso do identificador necessário ao **MLE** para associar ficheiros de metadados com os respetivos dados.

- A utilização de **CL-PKC** nas diferentes funções de cifragem e assinatura é uma possível solução para o problema de distribuição de chaves. Sem necessidade do peso associado ao uso de certificados, estas técnicas não exigem a confiança total numa única entidade, como seria o caso da imediata alternativa, o **IBE**.

- A conjugação de técnicas de **MLE** com partilha de ficheiros (seguindo a metodologia em 3.4) implica a utilização de revogação $lazy$, resultado da ligação entre chave e mensagem associada (4.3.2).

7.2 Trabalho futuro

A nível do $benchmark$, pode ser realizada a implementação do mesmo numa linguagem com custo de paralelização mais leve, de modo a poder avaliar o **RCE** (4.3.3) perante as diferentes operações de base. Adicionalmente, a nível do servidor, podem ser adaptados os valores de tamanho de blocos, de modo a melhor mensurar o **CipherCE** a nível de eficácia de deduplicação. Uma bateria de testes com diferentes tamanhos de blocos, tanto a nível do servidor como do cliente, é também aconselhada, de modo a serem conseguidos resultados mais completos e, consequentemente, realizada uma análise mais meticulosa.

Tal como explicado em 1.1, os desafios nas soluções passam por equilibrar performance, segurança e funcionalidade. Assim, as possíveis contribuições ao sistema apresentado podem seguir um destes três caminhos:

- **Performance:** o sistema encontra-se implementado em $Java$, recorrendo a **JNI** para invocar as funções de **CL-PKC** implementadas em C. Assim, uma implementação da totalidade do sistema em C representaria uma melhoria à performance do próprio.

- **Segurança:** a implementação de funções baseadas em **CL-PKC** pode ser utilizada noutros sistemas que necessitem de distribuição de chaves semelhantes, como o SiRiUS. Por outro lado, a estrutura de dados, metadados e atualidade pode ser conjugada com outras abordagens de cifragem, da mesma forma que foram com o **MLE**, como o $Searchable\ Encryption$ [12] ou o $Order\text{-}preserving\ Encryption$ [11].

- **Funcionalidade:** a utilização de funções de codificação e descodificação não é a melhor em termos de usabilidade. A adaptação da camada de segurança com o

sincronizador de algum serviço de $cloud$ (como o Dropbox ou o OwnCloud) permitiria que os utilizadores pudessem recorrer aos sistemas que estão habituados, com as adicionais garantias de segurança que necessitam. De seguida, é aconselhada a definição de uma plataforma pública de relativa confiança, sendo esta escolhida para realizar a partilha das chaves de **CL-PKC**, abstraindo os utilizadores em relação a este processo.

Bibliografia

[1] Michael Abd-El-Malek, Gregory R Ganger, Garth R Goodson, Michael K Reiter, and Jay J Wylie. Fault-scalable byzantine fault-tolerant services. In *ACM SIGOPS Operating Systems Review*, volume 39, pages 59–74. ACM, 2005. 30

[2] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th symposium on Operating systems design and implementation*, OSDI '02, pages 1–14, New York, NY, USA, 2002. ACM. 45

[3] Sattam S. Al-riyami and Kenneth G. Paterson. Certificateless public key cryptography. In *ASIACRYPT'03*, pages 452–473. Springer-Verlag, 2003. 17, 19, 73

[4] RSA An. Pkcs# 5: Password-based encryption standard. 73

[5] Paul Anderson and Le Zhang. Fast and secure laptop backups with encrypted deduplication. In *Proceedings of the 24th international conference on Large installation system administration*, LISA'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association. 45

[6] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010. 3

[7] Michael Backes, Christian Cachin, and Alina Oprea. Lazy revocation in cryptographic file systems. In *Security in Storage Workshop, 2005. SISW'05. Third IEEE International*, pages 11–pp. IEEE, 2005. 46

[8] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication, 2012. 45, 48, 56, 73

[9] Tim Berners-Lee and R. Cailliau. WorldWideWeb: Proposal for a HyperText Project, November 1990. 21

[10] Kenneth P. Birman. *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. 21

[11] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: improved security analysis and alternative solutions. In *Advances in Cryptology–CRYPTO 2011*, pages 578–595. Springer, 2011. 79

[12] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Advances in Cryptology-Eurocrypt 2004*, pages 506–522. Springer, 2004. 79

[13] Dan Boneh, Xuhua Ding, Gene Tsudik, and Chi-Ming Wong. A method for fast revocation of public key certificates and security capabilities. In *USENIX Security Symposium*, pages 22–22, 2001. 40

[14] Scott Case. *Growth, litigation and innovation of peer-to-peer file-sharing networks*. PhD thesis, Massachusetts Institute of Technology, 2004. 1

[15] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999. 30

[16] Miguel Castro and Barbara Liskov. Proactive recovery in a byzantine-fault-tolerant system. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*, pages 19–19. USENIX Association, 2000. 30

[17] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 407–418, New York, NY, USA, 2003. ACM. 21

[18] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop*

on Designing privacy enhancing technologies: design issues in anonymity and unobservability, pages 46–66, New York, NY, USA, 2001. Springer-Verlag New York, Inc. 21

[19] Joe Cooley, Chris Taylor, and Alen Peacock. Abs: the apportioned backup system. *MITLaboratory for Computer Science*, 2004. 45, 48

[20] Landon P Cox, Christopher D Murray, and Brian D Noble. Pastiche: Making backup cheap and easy. *ACM SIGOPS Operating Systems Review*, 36(SI):285–298, 2002. 48

[21] Yvo G Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994. 17

[22] John R. Douceur, Atul Adya, William J. Bolosky, Dan Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, ICDCS '02, pages 617–, Washington, DC, USA, 2002. IEEE Computer Society. 26, 48

[23] Ernesto. File-sharing traffic predicted to double by 2015, Jun 2011. `http://torrentfreak.com/file-sharing-traffic-predicted-to-double-by-2015-110603/`. 1

[24] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 537–554, London, UK, UK, 1999. Springer-Verlag. 89

[25] Scott M. Fulton. Amazon ec2 now 42 supercomputer, ibm bluegenes in the dust, Nov 2011. `http://readwrite.com/2011/11/15/amazon-ec2-now-42-supercompute`. 3

[26] Blaise Gassend, G Edward Suh, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Caches and hash trees for efficient memory integrity verification. In *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on*, pages 295–306. IEEE, 2003. 42

[27] David Geer. Reducing the storage burden via data deduplication. *Computer*, 41(12):15–17, December 2008. 26

[28] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology—EUROCRYPT'99*, pages 295–310. Springer, 1999. 17

[29] Peter Gutman. Pki: It's not dead, just resting. *Computer*, 35(8):41–49, August 2002. 17

[30] Barbara Guttman and Edward A. Roback. Sp 800-12. an introduction to computer security: the nist handbook. Technical report, Gaithersburg, MD, United States, 1995. xiii, 7, 12, 13

[31] Brian Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, July 2008. 3

[32] Xinyi Huang, Yi Mu, Willy Susilo, Duncan S Wong, and Wei Wu. Certificateless signature revisited. In *Information Security and Privacy*, pages 308–322. Springer, 2007. 19

[33] Xinyi Huang, Willy Susilo, Yi Mu, and Futai Zhang. On the security of certificateless signature schemes from asiacrypt 2003. In *Cryptology and Network Security*, pages 13–25. Springer, 2005. 19

[34] Public Key Infrastructure and Token Protection Profile. Common criteria for information technology security evaluation. *National Security Agency*, 2002. 11

[35] Eu jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. Sirius: Securing remote untrusted storage. In *in Proc. Network and Distributed Systems Security (NDSS) Symposium 2003*, pages 131–145, 2003. 41, 59

[36] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable secure file sharing on untrusted storage. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, FAST '03, pages 29–42, Berkeley, CA, USA, 2003. USENIX Association. 41, 59

[37] Vishal Kher and Yongdae Kim. Securing distributed storage: challenges, techniques, and systems. In *Proceedings of the 2005 ACM workshop on Storage security and survivability*, StorageSS '05, pages 9–25, New York, NY, USA, 2005. ACM. 31

[38] Doug Lea. A java fork/join framework. In *Proceedings of the ACM 2000 conference on Java Grande*, pages 36–43. ACM, 2000. 51

[39] Qin Lv, Sylvia Ratnasamy, and Scott Shenker. Can heterogeneity make gnutella scalable? In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 94–103, London, UK, UK, 2002. Springer-Verlag. 21

[40] Umesh Maheshwari, Radek Vingralek, and William Shapiro. How to build a trusted database system on untrusted storage. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, pages 10–10, Berkeley, CA, USA, 2000. USENIX Association. 42

[41] Nagapramod Mandagere, Pin Zhou, Mark A Smith, and Sandeep Uttamchandani. Demystifying data deduplication. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, Companion '08, pages 12–17, New York, NY, USA, 2008. ACM. 26

[42] Matt Marshal. Dropbox has become "problem child" of cloud security, Aug 2012. `http://venturebeat.com/2012/08/01/dropbox-has-become-problem-child-of-cloud-secur` 2

[43] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 369–378, London, UK, UK, 1988. Springer-Verlag. 64

[44] Arif Mohamed. A history of cloud computing, Mar 2009. `http://www.computerweekly.com/feature/A-history-of-cloud-computing`. 3

[45] J. Paulo, P. Reis, José Pereira, and A. Sousa. Dedisbench: A benchmark for deduplicated storage systems. In Robert Meersman, Hervé Panetto, Tharam S. Dillon, Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou, Siani Pearson, Alois Ferscha, Sonia Bergamaschi, and Isabel F. Cruz, editors, *OTM Conferences (2)*, volume 7566 of *Lecture Notes in Computer Science*, pages 584–601. Springer, 2012. 56

[46] João Paulo. Efficient storage of data in cloud storage. Master's thesis, Universidade do Minho, 2009. 56

[47] Sean Quinlan and Sean Dorward. Awarded best paper! - venti: A new approach to archival data storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association. 26, 46

[48] Anirudh V. Ramachandran and Nick Feamster. Authenticated out-of-band communication over social links. In *Proceedings of the first workshop on Online social networks*, WOSN '08, pages 61–66, New York, NY, USA, 2008. ACM. 40

[49] M. Ripeanu. [15] peer-to-peer architecture case study: Gnutella network. In *Proceedings of the First International Conference on Peer-to-Peer Computing*, P2P '01, pages 99–, Washington, DC, USA, 2001. IEEE Computer Society. 21

[50] Phillip Rogaway, Mihir Bellare, and John Black. Ocb: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, August 2003. 45

[51] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc. 18

[52] Mark W Storer, Kevin Greenan, Darrell DE Long, and Ethan L Miller. Secure data deduplication. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 1–10. ACM, 2008. 47

[53] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 21–26. ACM, 2008. 48

[54] Jiyi Wu, Lingdi Ping, Xiaoping Ge, Ya Wang, and Jianqing Fu. Cloud storage as the infrastructure of cloud computing. In *Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on*, pages 380–383. IEEE, 2010. 22

Anexo A

Construções

A.1 *Basic CL-PKE*

O primeiro esquema é chamado de `Basic CL-PKE` e serve como base para um segundo esquema mais completo. Como foi referido previamente, a estrutura destes esquemas é bem definida e, como tal, faz sentido abordar o seu funcionamento através da mesma.

- *Setup*:

  1. Utilizar um *input* k para gerar um *output* $\langle \mathbb{G}_1, \mathbb{G}_2, e \rangle$, onde $\mathbb{G}_1$ e $\mathbb{G}_2$ são grupos da mesma ordem prima q e $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ é um *pairing*.
  2. Escolher um gerador arbitrário $P \in \mathbb{G}_1$.
  3. Escolher uma chave mestra s aleatória de $\mathbb{Z}_q^*$ e calcular $P_0 = sP$.
  4. Escolher duas funções de *hash* $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$ e $H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$.
  5. Entregar como *output* os parâmetros públicos.

  Os parâmetros públicos são: $\langle \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_0, H_1, H_2 \rangle$, a chave mestra é $s \in \mathbb{Z}_q^*$, o espaço de mensagens é $M = \{0,1\}^n$ e o espaço de textos-cifa é $C = \mathbb{G}_1 \times \{0,1\}^n$.

- *Partial-Private-Key-Extract*:

1. Computar $Q_A = H_1(ID_A) \in \mathbb{G}_1^*$.
2. Calcular a chave privada parcial $D_A = sQ_A \in \mathbb{G}_1^*$.
3. Entregar como $output$ a chave privada parcial D_A.

- $Set\text{-}Secret\text{-}Value$:
  1. Escolher um $x_A \in \mathbb{Z}_q^*$ aleatório.
  2. Entregar como $output$ o valor secreto x_A.
- $Set\text{-}Private\text{-}Key$:
  1. Computar $S_A = x_A D_A \in \mathbb{G}_1^*$.
  2. Entregar como $output$ a chave privada S_A.
- $Set\text{-}Public\text{-}Key$:
  1. Calcular $X_A = x_A P$.
  2. Calcular $Y_A = x_A P_0$.
  3. Entregar como $output$ a chave pública $P_A = \langle X_A, Y_A \rangle$.
- $Encrypt$:
  1. Verificar se $X_A, Y_A \in \mathbb{G}_1^*$ e se $e(X_A, P_0) = e(Y_A, P)$. Se não se verificar, abortar cifragem.
  2. Computar $Q_A = H_1(ID_A) \in \mathbb{G}_1^*$.
  3. Escolher um valor aleatório $r \in \mathbb{Z}_q^*$.
  4. Computar $C = \langle rP, M \oplus H_2(e(Q_A, Y_A)^r) \rangle$.
  5. Entregar como $output$ o texto-cifra C.
- $Decrypt$:
  1. Realizar o $parsing$ a C de modo a que $C = \langle U, V \rangle$.
  2. Computar $Res = V \oplus H_2(e(S_A, U))$.
  3. Entregar como $output$ o valor decifrado Res.

A.2 *Full CL-PKE*

O segundo esquema é chamado de `Full CL-PKE` e trata-se de uma adaptação do `Basic CL-PKE` de modo a tornar o sistema resistente contra **CCA**. Este nível de segurança é atingido utilizando a técnica de $padding$ de Fujisaki-Okamoto [24].

- **Setup** - Tal como `Basic CL-PKE`, com adição de duas novas funções $H_3 : \{0,1\}^n \times \{0,1\}^n \rightarrow \mathbb{Z}_q^*$ e $H_4 : \{0,1\}^n \rightarrow \{0,1\}^n$.
  Assim, os parâmetros públicos passam a ser $\langle \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_0, H_1, H_2, H_3, H_4 \rangle$, o espaço de mensagens é igual ao do `Basic` e o espaço de textos-cifra passa a ser $C = \mathbb{G}_1 \times \{0,1\}^{2n}$

- **Partial-Private-Key-Extract** - Idêntico a `Basic CL-PKE`.

- **Set-Secret-Value** - Idêntico a `Basic CL-PKE`.

- **Set-Private-Key** - Idêntico a `Basic CL-PKE`.

- **Set-Public-Key** - Idêntico a `Basic CL-PKE`.

- **Encrypt**:

  1. Verificar se $X_A, Y_A \in \mathbb{G}_1^*$ e se $e(X_A, P_0) = e(Y_A, P)$. Se não se verificar, abortar cifragem.
  2. Computar $Q_A = H_1(ID_A) \in \mathbb{G}_1^*$.
  3. Escolher aleatoriamente $\sigma \in \{0,1\}^n$.
  4. Calcular $r = H_3(\sigma, M)$.
  5. Computar $C = \langle rP, \sigma \oplus H_2(e(Q_A, Y_A)^r), M \oplus H_4(\sigma) \rangle$.
  6. Entregar como $output$ o texto-cifra C.

- **Decrypt**:

  1. Realizar o $parsing$ a C de modo a que $C = \langle U, V, W \rangle$.
  2. Computar $\sigma' = V \oplus H_2(e(S_A, U))$.

3. Computar $M' = W \oplus H_4(\sigma')$.
4. Calcular $r' = H_3(\sigma', M')$.
5. Testar se $U = r'P$. Se não se verificar, abortar decifragem.
6. Entregar como $output$ o valor decifrado M'.

A.3 CLS

Este esquema de assinatura é chamado de $Scheme\ I$ no artigo associado, e é utilizado para gerar assinaturas e realizar as respetivas verificações utilizando o tipo de chaves conseguidas com os esquemas $Certificateless$.

- **Setup** - Tal como `Basic CL-PKE`, mas apenas será utilizada uma função de $hash$ $H : \{0,1\}^* \times \mathbb{G}_2 \rightarrow \mathbb{Z}_q^*$.
  Assim, os parâmetros públicos passam a ser $\langle \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_0, H \rangle$, o espaço de assinaturas é $S = \mathbb{G}_1 \times \mathbb{Z}_q^*$
- **Partial-Private-Key-Extract** - Idêntico a `Basic CL-PKE`.
- **Set-Secret-Value** - Idêntico a `Basic CL-PKE`.
- **Set-Public-Key** - Idêntico a `Basic CL-PKE`.
- **Sign**:
  1. Calcular $h = H(M \parallel ID_A \parallel X_A)$.
  2. Computar $\sigma = D_A + x_A h$.
  3. Entregar como $output$ a assinatura σ.
- **Verify**:
  1. Verificar se $e(X_A, P_0) = e(Y_A, P)$. Se não seja o caso, abortar verificação.
  2. Computar $v1 = e(\sigma, P)$.
  3. Calcular $h = H(M \parallel ID_A \parallel X_A)$.
  4. Computar $v2 = e(Q_A, P_0)e(X_A, h)$.
  5. Se $v1 = v2$, entregar como $output\ TRUE$, caso contrário, entregar $FALSE$.

A.4 CE

- $Key\ generation$: Recebendo uma mensagem m, entrega $k \leftarrow \mathcal{H}(m)$.
- $Encrypt$: Recebendo k e uma mensagem m, entrega $c \leftarrow Enc(k, m)$.
- $Tag\ generation$: Recebendo um criptograma c, entrega $t \leftarrow \mathcal{T}(c)$.
- $Decrypt$: Recebendo k e um criptograma c, entrega $m \leftarrow Dec(k, c)$.

A.5 HCE

- $Key\ generation$: Recebendo uma mensagem m, entrega $k \leftarrow \mathcal{H}(m)$.
- $Encrypt$: Recebendo k e uma mensagem m, executa:

  1. $t \leftarrow \mathcal{H}(k)$.
  2. $c \leftarrow Enc(k, m)$.
  3. $\mathcal{E} = c \parallel t$.

  Terminando com o $output\ \mathcal{E}$.
- $Tag\ generation$: Recebendo um criptograma c, executa:

  1. $c \rightarrow c_1 \parallel t$.
  2. $\mathcal{T} = t$.

  Terminando com o $output\ \mathcal{T}$.
- $Decrypt$:Recebendo k e um criptograma c, executa:

  1. $c \rightarrow c_1 \parallel t$.
  2. $m \leftarrow Dec(k, c)$.
  3. $t' \leftarrow \mathcal{H}(\mathcal{H}(m))$.

  Caso $t == t'$, termina com o $output\ m$. Caso contrário termina com $\perp$.

A.6 RCE

- $Key\ generation$: Recebendo uma mensagem m, entrega $k \leftarrow \mathcal{H}(m)$.
- $Encrypt$: Recebendo k e uma mensagem m, executa:
  1. $l \leftarrow rand() \in \{0,1\}^{len}$.
  2. $t \leftarrow \mathcal{H}(k)$.
  3. $c_1 \leftarrow Enc(l, m)$.
  4. $c_2 \leftarrow l \oplus k$.
  5. $\mathcal{E} = c_1 \parallel c_2 \parallel t$.

  Terminando com o $output\ \mathcal{E}$.
- $Tag\ generation$: Recebendo um criptograma c, executa:
  1. $c \rightarrow c_1 \parallel c_2 \parallel t$.
  2. $\mathcal{T} = t$.

  Terminando com o $output\ \mathcal{T}$.
- $Decrypt$:Recebendo k e um criptograma c, executa:
  1. $c \rightarrow c_1 \parallel c_2 \parallel t$.
  2. $l \leftarrow c_2 \oplus k$.
  3. $m \leftarrow Dec(l, c_1)$.
  4. $t' \leftarrow \mathcal{H}(\mathcal{H}(m))$.

  Caso $t == t'$, termina com o $output\ m$. Caso contrário termina com $\perp$.

A.7 SaltedMLE

- $Key\ generation$: Recebendo uma mensagem m e um tamanho de salt $slen$, executa:

  1. $l \leftarrow rand() \in \{0,1\}^{slen}$.
  2. $m \leftarrow l \parallel m$.
  3. $k = \mathcal{H}(m)$.

  Terminando com $output\ k$.

- $Encrypt$: Idêntico a `HCE`.
- $Tag\ generation$: Idêntico a `HCE`.
- $Decrypt$ Idêntico a `HCE`.

A.8 CipherCE

- $Key\ generation$: Se é primeiro bloco, recebendo uma mensagem m, entrega $k \leftarrow \mathcal{H}(m)$.
- $Encrypt$: Idêntico a `CE`.
- $Tag\ generation$: Idêntico a `CE`.
- $Decrypt$: Idêntico a `CE`.

Anexo B

Descrição das operações do sistema

B.1 $Software$ de codificação

- Codificar pasta:
  1. Verificar existência de chaves em `PathToKeys`. Caso contrário, abortar.
  2. Localizar a pasta escolhida no $input$ em e analisar os seus conteúdos individualmente.
  3. Para cada pasta, criar um ficheiro $.ownf$ com o mesmo nome e executar recursivamente. Para os outros casos, criar os três ficheiros necessários:
     - Cifrar a informação com **HCE**, e assinar com a chave privada do utilizador.
     - Criar o ficheiro $.d$ com o obtido previamente.
     - Cifrar a chave do utilizador com a sua chave pública (`Full CL-PKE`).
     - Criar um ficheiro $.md$ e adicionar a chave à lista.
     - Acrescentar a tag ao ficheiro $.md$.
     - Acrescentar o `username`, assinar e guardar o ficheiro $.md$ (`CLS`).
     - Criar um ficheiro $.own$ e adicionar o utilizador à lista.

- Assinar e guardar o ficheiro $.own$.

- Descodificar pasta:

  1. Verificar existência de chaves em `PathToKeys`. Caso contrário, abortar.
  2. Localizar a pasta `PathToStore` e analisar os seus conteúdos individualmente.
  3. Para cada pasta executar recursivamente. Para os outros casos, verificar se se trata de um ficheiro $.md$ e:
     - Verificar a assinatura do ficheiro $.own$ com a **USK** do utilizador a que pertence.
     - Consultar `username` de última alteração e confirmar que este se encontra na lista de $.own$.
     - Verificar assinatura de $.md$.
     - Obter a chave de $.md$ relacionada com o utilizador.
     - Decifrar a chave com **UEK** privada.
     - Obter a informação em $.d$.
     - Decifrar informação com chave obtida e comparar com tag.
     - Guardar novo ficheiro com a informação conseguida.

- Atribuir permissões:

  1. Localizar a pasta `PathToStore`.
  2. Validar a $hash\ tree$ de atualidade.
  3. Validar a assinatura em $.md$ consoante o utilizador da última alteração.
  4. Extrair a chave associada ao próprio `username`, decifrando-a com a chave privada.
  5. Cifrar a chave com chaves públicas dos utilizadores recebidos como argumento e armazenar no ficheiro $.md$, adicionando o próprio `username` para a última alteração.

6. Assinar o ficheiro $.md$

7. Caso a operação seja RW, adicionar os utilizadores ao ficheiro $.own$, assinando-o e refrescando a $hash\ tree$.

- Revogar permissões - Operação inversa à anterior e, assumindo-se como natural a compreensão do seu funcionamento a partir da explicação anterior, foi omitida a descrição.

B.2 $Daemon$ de atualização

- Refrescar $hash\ tree$:

1. Localizar `PathToStore` e obter conteúdos.

2. Executar geração de $hash\ tree$.

3. Caso seja a primeira iteração, terminar.

4. Comparar ao valor em $cache$, caso não seja igual, abortar.

5. Inserir $timestamp$.

6. Assinar ficheiro.

7. Armazenar em cache.

8. Armazenar o ficheiro.

B.3 Funções auxiliares

- Gerar $hash\ tree$:

1. Localizar `PathToStore` e obter conteúdos.

2. Avaliar os conteúdos até terminarem. Caso seja um ficheiro, calcular o $hash$. Caso seja uma pasta, realizar recursivamente a operação.

3. Concatenar os valores obtidos.

4. Caso seja $root$, inserir $timestamp$, assinar e armazenar em cache.

5. Armazenar o valor.

- Verificar atualidade:

1. Executar geração de $hash\ tree$.

2. Comparar ao valor armazenado, caso não seja igual, abortar.

3. Caso não seja root, subir um nível na árvore da diretoria e executar recursivamente. Caso contrário, seguir em 4.

4. Validar $timestamp$ e verificar assinatura. Se as operações falharem, abortar.

Anexo C

Diagramas de atividade

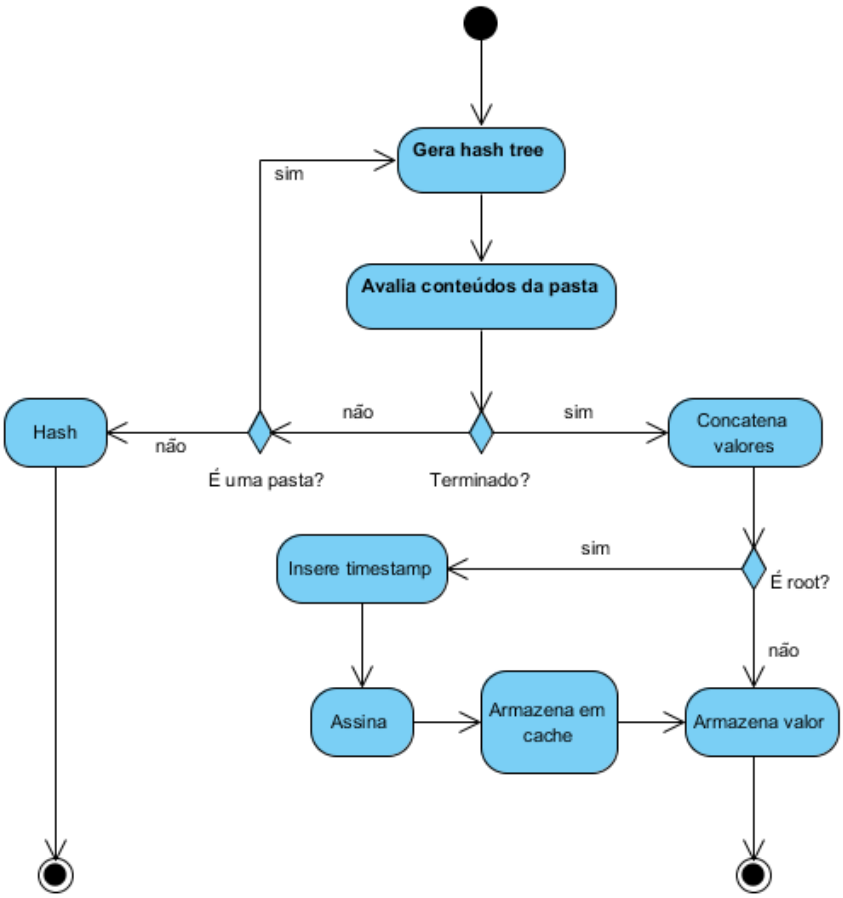


Figura C.1: Diagrama de atividades para a geração da hash tree

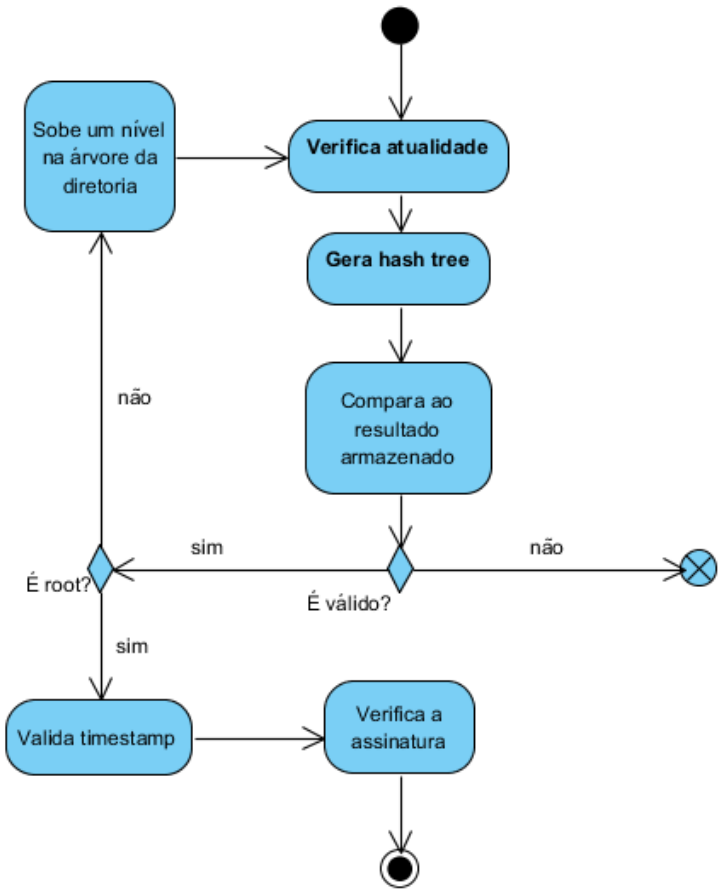


Figura C.2: Diagrama de atividades para a validação da hash tree

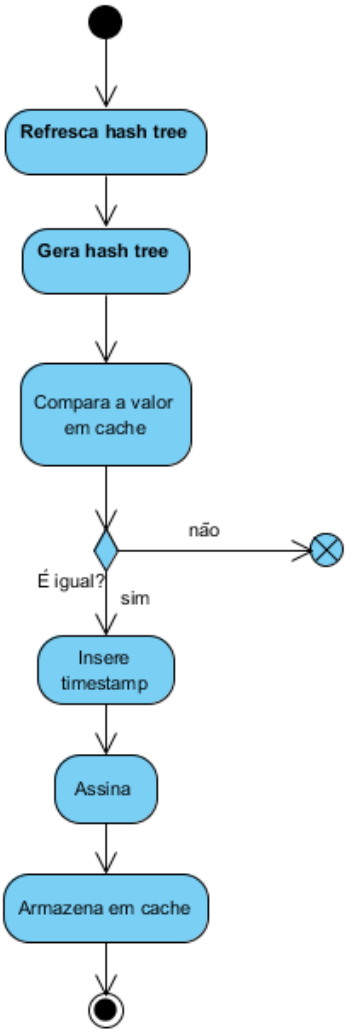


Figura C.3: Diagrama de atividades para o refrescar da hash tree