universidade de aveiro
theoria poiesis praxis

A few years ago I heard a quotation, and I am going to modify it here: "If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology".

Bruce Schneier

SEGURANÇA

4

Atenção!

Todo o conteúdo deste documento pode conter alguns erros de sintaxe, científicos, entre outros... **Não estudes apenas a partir desta fonte**. Este documento apenas serve de apoio à leitura de outros livros, tendo nele contido todo o programa da disciplina de Segurança, tal como foi lecionada, no ano letivo de 2017/2018, na Universidade de Aveiro. Este documento foi realizado por Rui Lopes.

Ao longo de toda a formação de um estudante na área das ciências computacionais e outras suas relacionadas existe um largo conjunto de estudos que são perfeitamente transversais e independentes da sua aplicação. Um desses estudos, e decididamente o mais fulcral, é o da segurança, o qual tentamos abordar neste documento.

1. Introdução aos Conceitos de Segurança

O conceito de segurança é algo muito vago num só termo, mas muito mais significativo quando aplicado a uma determinada atividade. Quando pesquisamos num dicionário o significado de **segurança** por si só, encontramos significados como libertação de perigos (o que em inglês se descreve facilmente por *safety*), libertação de medos ou ansiedades, algo dado como confiado ou algo que assegura, isto é, que permite que alguém esteja prevenido sobre casos de espionagem, sabotagem, crime, ataque ou fuga [1]. Em termos de **segurança informática**, a sua descrição interseta-se com a genérica, mas especifica-se o seu significado dentro de três grandes categorias.

segurança

segurança informática

Áreas de atividade da segurança informática

Como referido, existem três grandes categorias sobre as quais se definem as **atividades** possíveis e aplicáveis de segurança informática.

atividades

A primeira de todas as categorias é a defesa contra **catástrofes físicas** que tenta encontrar um ou mais métodos para assegurar que um sistema computacional consegue sobreviver a danos físicos. Entre os vários danos físicos podemos apontar catástrofes materiais (como degradação dos equipamentos ou roubo), catástrofes políticas (como ataques terroristas) ou catástrofes naturais (como tremores-de-terra, incêndios, ...). A resolução de muitas destas graves consequências, conforme o que abordámos em Arquitetura de Computadores Avançada (a4s1), passa por aumentar um fator de **redundância** do material que compõe os sistemas em causa. Note-se o caso que foi referido nesta disciplina, em que para tratar do aumento do MTBF (*Mean Time Between Fails*) em termos de acesso a informação armazenada em discos, se criavam *arrays* de discos, a cuja estrutura se dá o nome de RAID (*Redundant Array of Independent Disks*). Outra forma de salvaguarda de eventos desta escala, mas não de contenção, é a criação de discos de recuperação, também conhecidos por **backups**.

catástrofes físicas

redundância

backups

Na criação de *software* também há, sempre, a possibilidade de serem lançados conjuntos de erros a qualquer momento, pelo que tais falhas se tornam algo previsíveis. Este fenómeno, inevitável, cria assim o conceito da segunda de três categorias da aplicação da segurança informática (**tolerância a falhas**). Por exemplo, quando há uma quebra de energia, um bloqueio de uma execução de aplicações ou meras falhas de rede, devemos considerar que tais eventos são previsíveis, pelo que poderão ser resolvidos com geradores de energia (UPS) ou fontes redundantes, redundância de equipamentos para resolução dos bloqueios, entre outros [2].

tolerância a falhas

Por exemplo, na disciplina de Base de Dados (a3s2) pudemos reparar que em sistemas mais críticos, para não colocar em risco uma transação ou a integridade de uma base de dados, são divididas operações por passos mais pequenos e, assim, de forma atómica, são realizadas e tomadas as diferentes ações, através de comandos como `commit` ou `rollback`.

A terceira e última categoria de aplicação de segurança informática está na prevenção contra **atividades não autorizadas**. Enquanto que os acontecimentos que registámos anteriormente são, por norma, fortuitos e fruto do acaso, estes são usualmente previsíveis, uma vez que são os que mostram más intenções por parte de terceiros, de agir de forma ilícita.

atividades não autorizadas

Existe um vasto conjunto de atividades consideradas ilícitas como o acesso a informação privilegiada (ou seja, não tornada pública), a alteração de informação (que de forma camuflada ou explícita altera ou elimina informação pertencente a terceiros, sem autorização), a utilização exagerada ou abusiva de recursos computacionais, o impedimento da prestação de serviços ou **DoS** (acrónimo para *Denial of Service* - recusa de serviço) ou o

DoS

puro vandalismo. Uma recusa de serviço significa que há terceiros que estarão a "inundar" uma rede de computadores, tornando assim mais difícil a tarefa de resposta, por parte de quem fornece o serviço [2].

Vulnerabilidades, ataques, riscos e defesas

Existem alguns conceitos que são importantes referir ao longo deste documento, uma vez que serão muito usados no estudo de segurança informática.

vulnerabilidade

Um primeiro conceito a abordar é o de **vulnerabilidade**. Uma vulnerabilidade é uma caraterística de um sistema que o torna sensível a certos ataques [2]. Basicamente, acaba por ser uma fraqueza num sistema computacional, presente num ou mais procedimentos, no planeamento (desenho) ou até mesmo na implementação, que poderá ser alvo de danos. Por exemplo, um sistema em particular poderá ser vulnerável a manipulação de dados de forma indevida por um utilizador, se não for capaz de verificar a identidade deste antes de lhe fornecer o acesso aos dados [3].

ameaças

As vulnerabilidades dão assim azo a que haja a possibilidade de ocorrerem **ameaças**. Uma ameaça é um conjunto de circunstâncias que têm o potencial de causar danos. Consideremos, por exemplo, a Figura 1.1 [3], onde uma parede aguenta uma grande quantidade de água. A água que se encontra à esquerda é uma ameaça para o homem à direita, uma vez que se aumentar, poderá passar para cima dele através da fenda, e se se manter como está, põe em causa a parede, que poderá ruir, pondo o homem em causa mais uma vez.

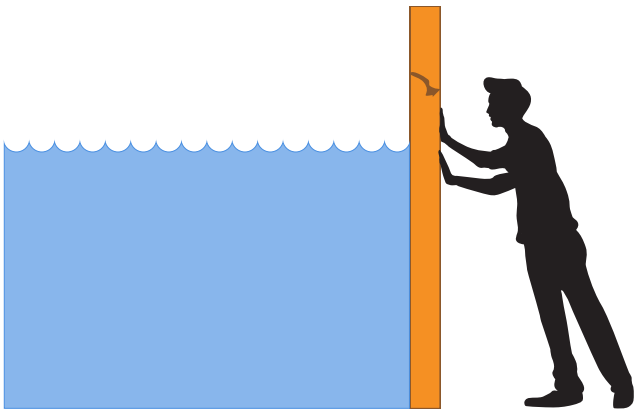

figura 1.1
vulnerabilidades e ameaças

ataques

As ameaças, quando executadas, provocam **ataques** ou tentativas destes. Os ataques são conjuntos de passos executados em função de uma vulnerabilidade que permitem fazer uma ação ilícita. Esta ação, por si terá **riscos** associados, se for executada, mas poderá também ter um conjunto de **contra-medidas** que protejam os sistemas computacionais atacados - mecanismos de **defesa**.

riscos

contra-medidas

defesa

Contrariamente ao que se possa pensar, é importante que se refira que nenhum produto pode ser considerado como cem porcento seguro. Isto significa que todos os produtos informáticos, seja em que fase de desenvolvimento for, poderão estar sempre sujeitos a novos ataques de segurança. Não é muito difícil perceber porquê: nos primórdios da criação de *software* computacional (as primeiras criações de programas) os componentes principais de tais módulos eram variáveis e outros componentes inseridos num corpo único de uma função — na verdade, no início, nem sequer havia a noção de funções, procedimentos ou métodos. A complexidade da implementação de detalhes de segurança numa só função, qualquer que fosse o seu tamanho, é algo grande, entre programação defensiva, gestão e manipulação de erros, ... Com o avançar dos tempos os programas começaram a atingir tamanhos muito maiores, sendo hoje em dia, conjuntos de componentes e módulos, que por si só, contêm inúmeros métodos, procedimentos e funções. Com isto, a complexidade da implementação de detalhes de segurança tornou-se algo muito difícil, humanamente impossível de se tomar como certas, garantias de segurança.

De forma a tentar aproximar um produto do ideal de cem porcento seguro, é importante assim explorar o maior número de mecanismos de defesa. Esta exploração de mecanismos pode acontecer entre duas formas distintas: de um modo a explorar o **perímetro**, de forma análoga a uma muralha, que evita o contacto direto entre o exterior e o interior de uma região; ou de um modo a explorar em **profundidade** uma determinada região onde o perigo se pode instalar. A exploração de defesas em perímetro, por norma, tenta separar o ambiente de perigo (exterior ao programa) do ambiente próprio do programa, que deverá estar limpo de qualquer intruso e consequentes danos. Teoricamente, esta aproximação não será considerada como a melhor, uma vez que não se há certezas de que o perigo já não se encontra no interior do programa. Já a exploração de defesas em profundidade torna o processo de proteção contra intrusos e danos mais forte, isto é, consegue proteger-se sobre ataques, mesmo que estes não ultrapassem um perímetro de segurança, uma vez que sejam iniciados internamente. Da mesma forma que o anterior, também permite a defesa de ataques provenientes do exterior e que cruzem o perímetro de segurança. Como podemos verificar, por esta razão, este procedimento torna-se muito mais eficaz, contudo muito mais complexo e difícil de gerir. [4]

perímetro

profundidade

Em caso de ataque, os procedimentos de defesa, muito possivelmente, já não serão importantes. Neste caso, os programas deverão ter escolhas entre **contra-medidas**, de forma a cortar o acesso contínuo a dados e sistemas sem permissão. Entre as várias contra-medidas possíveis, temos o desencorajamento (quer por lei, com procedimentos de coima, quer por barreiras de segurança que possam ser instaladas pós-ataque, isto é, mecanismos de *firewall*, autenticação ou *sandboxing* previamente instalados na rede, de forma a que o atacante depois de entrar não consiga ter o ambiente completamente aberto), os sistemas de deteção (como auditorias, análise forense de entrada forçada ou sistema de deteção à intrusão), a deceção (fenómeno forçado com **honeypots** — pequenas regiões de atração locais, com dados de uma rede computacional, de aparente interesse para o atacante, não o permitindo fazer nada mais, atrasando-o de ataques posteriores e mais severos, e deixando a oportunidade a um administrador de sistemas de estudar os mecanismos do ataque, de forma a que possa responder com mais sapiência [5]), a prevenção (como a atribuição do **princípio do menor privilégio**, isto é, a atribuição dos privilégios mínimos a cada utilizador que se considere como parte integrante de um projeto, para que ninguém tenha mais permissões que aquelas de que realmente necessita) e a recuperação (como a criação de cópias de segurança, a instalação de sistemas redundantes ou a recuperação de dados por vias forenses) [6].

contra-medidas

honeypots

princípio de menor privilégio

Deteção, listagem de vulnerabilidades e zero-day attacks

Como já foi referido, é humanamente impossível fazer a deteção ou a listagem de todas as vulnerabilidades de um sistema computacional, quer este seja distribuído ou não. Para melhorar esta limitação, existem *softwares* que já se encarregam de o fazer. Ainda assim muitas vulnerabilidades escapam aos olhos dos administradores de sistemas ou de quem cria os projetos.

Algumas das vulnerabilidades que não são encontradas têm a sua identificação ao longo do tempo de utilização do programa onde estão implementadas. Quando são detetadas tais vulnerabilidades e alguém responde a esta com um ataque, diz-se que se efetuou um **ataque de dia-zero** (em inglês *zero-day attack*)[1]. Estes ataques, em particular, dificultam muito a ação e possíveis contras-medidas por parte do administrador de sistemas, uma vez que não permitem que este analise com tempo o acontecimento que despoletou o ataque ou, até mesmo, de que forma é que o ataque foi executado.

ataque de dia-zero

Então como é que podemos previnir os nossos projetos de tais ataques? Pensemos a larga escala. Os sistemas operativos são projetos cuja escala é muito larga, isto é, por si só podem ser objetos de ataques furtivos, e que atingem um número muito grande de utilizadores, cujas informações dependem dos mecanismos de segurança que nos sistemas e-

[1] Também poderá ser uma ameaça de dia-zero (em inglês zero-day threat) se não se tratar de um ataque, mas antes de uma ameaça.

xistem implementados. Se compararmos os atuais (janeiro de 2018) sistemas operativos mais vendidos (Microsoft Windows™ e macOS) podemos verificar que o número de ataques é muito mais forte e incidente sobre os vários computadores com Windows™ instalado, ao invés de macOS. Isto acontece essencialmente porque a **diversidade** é muito grande em termos de implementação de *software* sobre os variados *hardwares* que existem no mercado. Contrariamente, nos locais onde podemos consultar e usar o macOS, a diversidade de *hardwares* é relativamente pequena, sendo exclusiva a um fabricante só — a Apple. Contudo, isto não significa que os equipamentos da Apple não sejam alvo de novos ataques — apenas significa que será considerado mais difícil atacar um equipamento da Apple, em comparação aos outros existentes no mercado.

diversidade

Note-se, no entanto, que o facto de haver o controlo de diversidade sobre um conjunto ou categoria de equipamentos não é ponto essencial e solução de problemas de segurança. Há um *trade-off*, em casos como o da Apple, em que os seus vários equipamentos executam o mesmo sistema operativo (o macOS), onde se um equipamento for vulnerável a uma ameaça ou ataque, então se pode assumir que todos os outros são. Por outras palavras, enquanto que a diversidade permite-nos melhorar a nossa proteção em termos de *zero-day attacks*, também nos pode tocar, caso façamos parte de uma comunidade muito grande de equipamentos com muitos elementos em comum, onde basta um ser atacado, para que todos os outros se tornem ameaçados com facilidade.

Uma segunda forma de tentarmos combater a possibilidade de sermos alvo de um ataque de *zero-day* é fazermos o nosso produto o mais robusto possível usando a ajuda de dicionários de vulnerabilidades já conhecidas em termos mundiais. Estes dicionários, vulgarmente denominados de **CVE** (acrónimo inglês para *Common Vulnerabilities and Exposures*), são listas públicas[2] onde se podem consultar as várias vulnerabilidades, de forma a podermos previnir os nossos códigos de tais perigos.

CVE

Como o próprio nome o indica, as CVE não só listam as vulnerabilidades, como também mostra alguns pontos de exposição de informação a terceiros, cujos privilégios não são básicos, isto é, confidenciais ou com outros níveis de importância, mas que não poderão ser acessíveis a qualquer pessoa (uma falha na aplicação do princípio do menor privilégio).

As CVE podem ajudar a salvaguardarmo-nos de *zero-day attacks*, mas não nos asseguram que estes não possam acontecer!

Do lado de quem encontra novas vulnerabilidades, também é possível (e até se aconselha), a preencher uma entrada no CVE. Para isso, há que ir à sua página principal e registar uma nova entrada, sobre a qual se receberá um número de identificação, fornecido por uma **CNA** (*Candidate Number Authority*) do CVE. De seguida, o pessoal oficial e regulador dos dicionários CVE vão a votações sobre se a entrada requirida é válida para entrar no dicionário ou não. Se for válido, então passamos a ter uma nova entrada no dicionário, como podemos ver na Figura 1.2, onde temos a entrada com o nome "*git-shell in git before 2.4.12, 2.5.x before 2.5.6, 2.6.x before 2.6.7, 2.7.x before 2.7.5, 2.8.x before 2.8.5, 2.9.x before 2.9.4, 2.10.x before 2.10.3, 2.11.x before 2.11.2, and 2.12.x before 2.12.3 might allow remote authenticated users to gain privileges via a repository name that starts with a - (dash) character*" (entrada com identificação CVE-2017-8386, para futura referência).

CNA

↬ **CVE-2017-8386**

Para saber mais detalhes sobre a própria entrada do CVE, podemos pedir para saber mais entrando na página descritiva da mesma, sob os serviços da **NVD** (acrónimo para *National Vulnerabilities Database*). Aqui, em termos de detalhes, conseguimos ter uma ideia mais correta sobre a avaliação que foi feita à vulnerabilidade em termos de local do possível ataque, da sua complexidade, dos seus privilégios requiridos, interação com o utilizador, âmbito, confidencialidade, integridade e disponibilidade. Na Figura 1.3 podemos ver de que forma é que a vulnerabilidade do `git` da Figura 1.2 é descrita na NVD.

NVD

Na Figura 1.3, note-se que também temos, no início, uma descrição da severidade do assunto. Neste caso, sendo bastante alta, numa escala de 0 a 10 tem também um impacto avaliado de 5.9 e uma pontuação para a sua exploração (devida ou indevida), baixa, de 2.8.

---

[2] Estas listas podem ser consultadas em http://cve.mitre.org/cve.

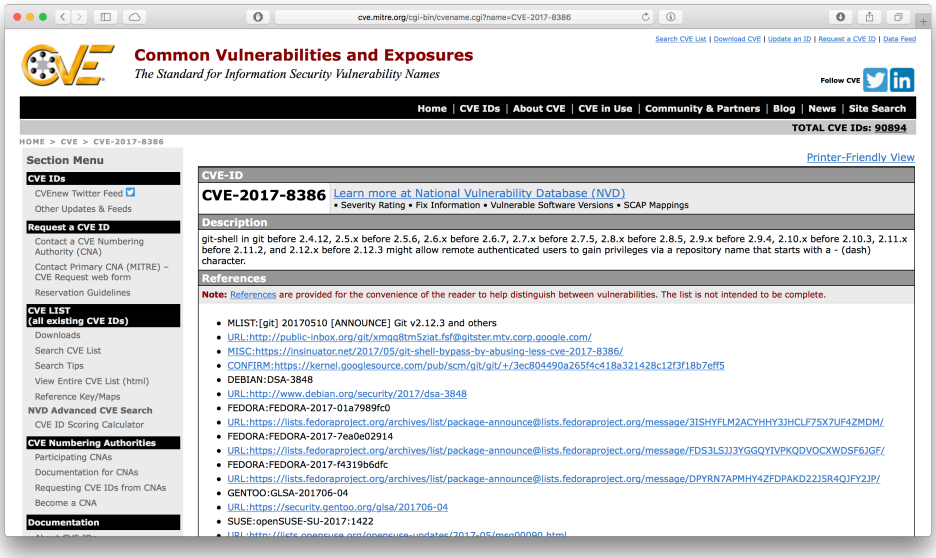

figura 1.2
CVE-2017-8386

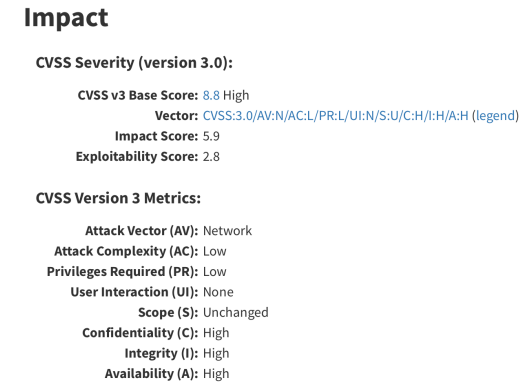

figura 1.3
classificação de vulnerabilidade

Mas a NVD não é a única base de dados de vulnerabilidades que existe — tal como o seu nome indica é uma base de dados de vulnerabilidades nacional, em termos dos Estados Unidos da América. Outras bases de dados são recorrentes por todo o mundo, mas há principalmente uma cujo centro de coordenação se encontra na Universidade de Carnegie Mellon, denominada de **CERT**. Esta organização, devota a assegurar boas práticas na gestão de sistemas e tecnologias em termos de resistência a ataques em redes, limitando ao máximo danos maiores e mantendo serviços críticos *on-line* [6], apareceu depois de um jovem, de seu nome Robert T. Morris, ter lançado para a Internet, em 1988, um "vírus" (em inglês *worm*) que provocou que o desempenho das máquinas "penetradas" fosse ficando cada vez mais degradada até uma paragem total. Estes sintomas eram provocados por uma replicação de um pequeno programa várias vezes, que ocupava a memória, mas que não colocava em risco nem os ficheiros de sistema, nem os seus dados [7]. Este vírus ficou conhecido por **Morris Worm**, em homenagem ao seu criador.

CERT

⊙ **Robert Tappan Morris**

Morris Worm

Em Portugal temos uma sucursal da CERT[3] sob o nome de Centro Nacional de Cibersegurança (em janeiro de 2018).

2. Criptografia

Uma das aplicações mais vulgares de segurança no meio informático está na comunicação entre mais do que uma parte, numa rede de computadores. Num cenário em que temos duas pessoas com um canal de comunicação estabelecido entre elas é importante que

3 Ver em https://www.cncs.gov.pt.

se consiga garantir que a partilha de mensagens que possa ocorrer neste canal seja feito apenas entre as duas partes, isto é, que um terceiro não possa chegar e receber e interagir com a mesma partilha. Para isto usamos a criptografia, como forma de garantir que as mensagens permanecem visíveis apenas para os que nela participarem.

Ocultar mensagens num canal de comunicação

A **criptografia** é uma técnica, ciência e arte que permite escrever mensagens entre duas pessoas num canal de comunicação, sem que terceiros possam interagir, lendo as mensagens ou até mesmo respondendo em concordância. Em termos básicos, o que a criptografia faz é modificar a forma da mensagem para que esta se torne totalmente inteligível para terceiros, tornando-o, por consequência, numa tarefa muito difícil para quem quer mesmo visualizar a informação. Na Figura 2.1 podemos ver um exemplo genérico da perceção de um terceiro numa comunicação estabelecida entre a Alice e o Bob[4], onde se aplica a criptografia, e a Eve quer auscultar a comunicação.

criptografia

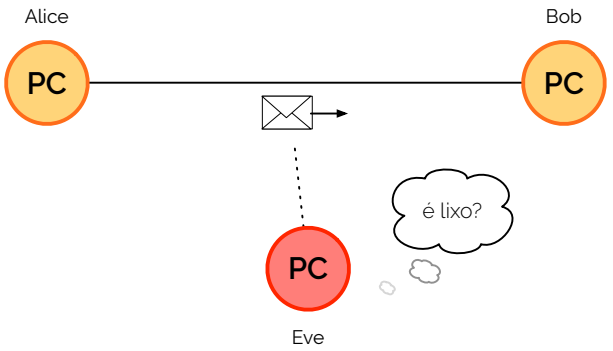

figura 2.1
Eve olha para criptografia

Uma alternativa à criptografia (uma aproximação distinta) é a **estenografia**, onde ao invés de tornar a mensagem inteligível no canal de comunicação, transforma-se a mensagem como peça integrante de um objeto. Neste caso, a Eve a olhar para o canal de comunicações muito possivelmente veria um objeto que lhe fosse estranho ao inesperado, e onde ela não soubesse que estivesse incluída a mensagem, como podemos ver na Figura 2.2.

estenografia

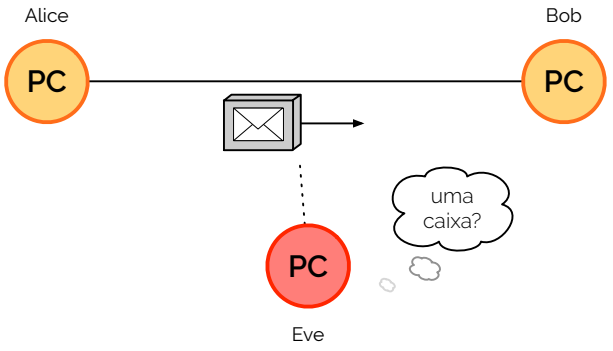

figura 2.2
Eve olha para estenografia

A criptografia e a estenografia são ambas analisadas e violadas por uma área denominada de **criptanálise**.

criptanálise

Noção de chave, cifra e sua evolução histórica

De forma a que um determinado objeto que se pretende ser transmitido entre duas ou mais partes por um canal de comunicação seja inteligível por terceiros, é importante que ambas as partes envolvidas na comunicação percebam como interpretar as mensagens. Ve-

[4] Ao longo deste documento, e por norma no contexto de segurança, chamamos às partes com boas intenções Alice e Bob, e a uma parte de má intenção Eve.

jamos como é que funciona em nossas casas. Consideremos assim que dentro de uma determinada casa vivem cinco pessoas. Todas estas pessoas sabem de todas as conversas que se fazem dentro de casa, mas alguém estranho ao pessoal não. O que é que tal pessoa necessita para se inserir dentro de casa? Ora, para tal necessita de uma **chave**, de forma a que possa entrar em casa e participar na conversa (ouvi-la e falar). Da mesma forma, em sistemas de redes de computadores, é importante que as duas ou mais partes envolvidas numa comunicação tenham uma chave para que possam abrir a informação apenas nos espaços onde se considera seguro fazê-lo, como na máquina de cada um, em oposição ao próprio canal de comunicação.

chave

De facto, a chave está na base de como é que a criptografia funciona, sendo que permite a interpretação de **cifras**[5], isto é, dos objetos inteligíveis que são comunicados ao longo de uma rede.

cifra

A utilização de criptografia e de estenografia, na verdade, embora já existisse, partiu fortemente da necessidade de ocultar mensagens nas Grandes Guerras, mais em particular na Segunda Guerra Mundial, onde ambas as tropas alemãs comunicavam remotamente em mensagens cifradas (com os britânicos a tentarem ler as mensagens, com a ajuda da criptanálise) e as tropas aliadas trocavam mensagens cifradas entre eles (com os alemães a tentarem lê-las de forma equivalente). Nesta história, os britânicos (aliados) foram os que mais sucesso tiveram na decifra de mensagens, enquanto que os alemães foram os que mais máquinas para a cifra de mensagens.

As primeiras cifras possuíam a sua segurança no secretismo do algoritmo. Usadas, por exemplo, pelos espartanos (portanto entre 900 a.C. e 192 a.C. [8]), estes baseavam a sua cifra num bastão com um dado diâmetro à volta do qual era enrolado um pergaminho com uma mensagem (que só poderia ser lida se o pergaminho fosse enrolado num outro bastão com o mesmo diâmetro) — a mensagem poderia inclusive ser transmitida num cinto das calças de um mensageiro, mas só poderia ser lida com um bastão específico (a chave era o diâmetro do bastão) [2], como podemos ver na Figura 2.3 [9].

figura 2.3
mensagem num bastão de diâmetro específico

Já no tempo romano, uma cifra hoje em dia perfeitamente passível de ser decifrada de cabeça, era a **cifra de César**, que usava uma mera adição de índices no alfabeto para constituir as mensagens cifradas. Por exemplo, sobre essa cifra, à palavra `HELLO` com chave 3 será atribuída a palavra `KHOOR` (cada letra foi substituída pela terceira letra seguinte no alfabeto).

cifra de César

Estas cifras simples, como já fora referido, foram substituídas posteriormente por máquinas, ou seja, por processos mecânicos que aumentaram drasticamente a complexidade destas. Conhecemos assim a famosa máquina Enigma (que veremos mais à frente aquando do estudo dos tipos de cifra), inventada em 1918 pelo alemão Arthur Scherbius, como podemos ver na Figura 2.4 [10], que usava um sistema de **rotores** mecânicos que encaminhavam sinais elétricos por caminhos dentro da máquina que dependiam da posição atual de cada rotor e da configuração inicial da máquina.

⊙ **Arthur Scherbius**

rotores

[5] Note-se bem no uso do termo cifra, ao invés de criptograma – é mais correto usar o termo cifra, uma vez que se trata de um erro de vocabulário português (de Portugal) referir a ação de encriptar e desencriptar, mas antes cifrar e decifrar.

Da mesma forma que o tempo foi ditando a história e os equipamentos de cifra foram tornando-se cada vez mais complexos, a criptanálise por si também aumentou e a sua investigação levou a exemplos como uma máquina programa desenvolvida por matemáticos no centro de criptanálise de Bletchley Park, no Reino Unido, onde trabalhou Alan Turing — as máquinas **Turing Bombe** e a **Colossus** (Figura 2.5 [11], [12]). Estas máquinas foram ambas feitas para criptanalisar, em tempo útil, as cifras criadas pela Enigma e pela Lorenz.

⊙ **Alan Turing**
Turing Bombe, Colossus

figura 2.4
máquina Enigma

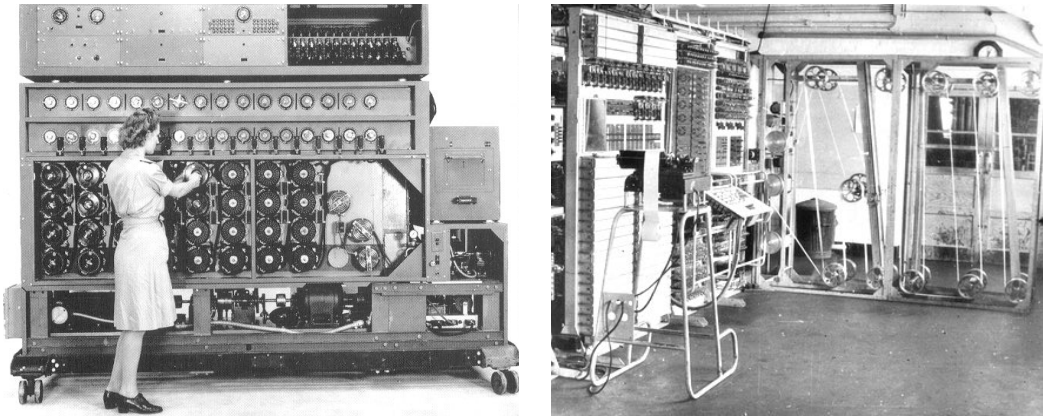

figura 2.5
Turing Bombe (à esquerda) e Colossus (à direita)

Os tempos seguintes à era da Turing Bombe e da Colossus permitiram uma grande evolução do mundo da computação, pelo que a complexidade dos sistemas aumentou, o que também potenciou o número de vulnerabilidades. Por esta mesma razão, os mecanismos de segurança que envolvem cifras também foram todos melhorados, uma vez que o seu contexto de aplicação também passou de procedimentos mecânicos, para fluxos contínuos de bits (ou blocos de bits).

Tipos de cifra

Como já pudemos verificar, nos últimos 4 000 anos, as civilizações têm usado vários tipos de cifra. Cada um destes tipos só nos poderá ser realmente útil se, uma vez a informação toda misturada, conseguimos voltar a um ponto inicial e ler a informação novamente, ou seja, se a conseguimos reconstruir.

No início do uso de cifras surgiram assim duas grandes soluções: as cifras de substituição e as cifras de transposição. Veremos cada uma delas com atenção.

cifra de transposição

Uma **cifra de transposição** é tal que um texto em aberto inicial terá os seus conteúdos com uma ordem trocada, segundo uma determinada regra secreta (uma chave). Por exemplo, se considerarmos um texto como ABCD EFGH IJKL MNOP, com uma mera permutação 4132 podemos obter o texto DACB HEGF LIKJ PMON, isto é, podemos trocar a ordem do primeiro, segundo, terceiro e quarto elementos de cada bloco de quatro carateres para as posições quarto, primeiro, terceiro e segundo, respetivamente.

cifra de substituição

Por outro lado, uma **cifra de substituição** é tal que ao invés de trocar a ordem dos carateres, substitui-os segundo uma determinada regra de aplicação de um novo carater. Dentro deste tipo de cifras, podemos ter três sub-tipos, entre os quais cifras monoalfabéticas, polialfabéticas e homofónicas.

monoalfabética

Uma cifra de substituição **monoalfabética** é uma cifra que usa apenas um alfabeto de substituição, isto é, sempre que um carater x é cifrado, terá sempre a mesma substituição y. Para fazer este tipo de cifras vários tipos de abordagem poderão ser tomadas. Um exemplo muito conhecido, e já referido anteriormente, é a **cifra de César**, onde a cada letra de um alfabeto original é somada uma quantidade constante que consideramos como chave, sendo que o resultante final é o resto da divisão da soma pelo cardinal do alfabeto original.

cifra de César

aditiva
multiplicativa

No entanto, uma cifra monoalfabética não tem de ser **aditiva**, podendo também ser **multiplicativa**, caso ao invés de somarmos uma quantidade constante (chave), a multiplicarmos pela quantidade constante, novamente dentro do mesmo módulo. Ainda mais, também poderá ser possível conjugar ambas cifras aditivas e multiplicativas para se obter uma **cifra afim**, que terá a seguinte forma: $y = x.k_1 + k_2 \bmod \#A$, onde x é a letra original, k_n é a chave n (sendo 1 para a multiplicativa e 2 para a aditiva) e $\#A$ é o cardinal do alfabeto (número de carateres do alfabeto A).

chave afim

Assim sendo, por exemplo, com uma mera cifra de César cuja chave é +3, podemos escrever a mensagem ABCD EFGH IJKL MNOP como DEFG HIJK LMNO PQRS.

Note-se, no entanto, nas fragilidades deste método de cifra. As nossas linguagens como o português ou o inglês possuem frequências relativas de palavras que não são iguais para todas. Por exemplo, a palavra "a" em português é muito usada, tal como a palavra "the" em inglês. Depois de efetuarmos uma substituição monoalfabética, a frequência das mesmas palavras, com formas diferentes, será precisamente a mesma, sendo que, com muito cuidado e alguma facilidade, é possível inferir a mensagem oculta mesmo sem conhecimento da chave.

cifras polialfabéticas

Para corrigir alguns dos problemas das cifras monoalfabéticas surgem depois as **cifras polialfabéticas**. Estas, já mais evoluídas, tentam usar um número finito de alfabetos para serem aplicados a cada letra de um alfabeto original, mantendo, ainda, a forma cíclica de uso nos alfabetos, ou seja, a aplicação de várias cifras monoalfabéticas.

⊙ **Giovan Battista Bellaso**

cifra de Vigenère

⊙ **Blaise de Vigenère**

A cifra polialfabética mais conhecida é a **cifra de Vigenère**, que consiste na aplicação de n cifras monoalfabéticas aditivas, sendo que a chave acaba por ser o valor atribuído a

cada letra de uma frase-chave com n letras (A=0, B=1, …, Z=26). De forma gráfica, a cifra apresenta-se como um quadrado de 26 elementos em cada lado, onde cada coluna representa o alfabeto de substituição usado por cada letra da frase-chave e cada linha a letra do texto original, como podemos ver na Figura 2.6.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

figura 2.6
cifra de Vigenère

Note-se, no entanto, que embora este método já seja melhor que as monoalfabéticas, ainda assim é possível inferir o texto oculto sem a presença de uma chave, uma vez que a frequência de letras individuais e de conjuntos caraterísticos de letras poderão formar um **período** T que, analisado por testes como o de **Kasiski** ou o **índice de coincidência**, poderão revelar o texto inicial [2].

⊙ **Friedrich Kasiski**
período, Kasiski, índice de coincidência

Por fim, as cifras **homofónicas** são tais que uma letra poderá ser cifrada numa ou mais letras finais. Neste caso, note-se que cada letra poderá ter um número de substituições diferentes das outras. No entanto, com este método ainda é possível conservar a frequência de letras e de conjuntos delas (palavras ou partículas destas), podendo, com criptanálises muito extensivas, deslindar o texto que outrora estaria oculto.

homofónicas

No início dos anos '40 Claude Shannon começou uma era de análise mais profunda da criptografia e das suas técnicas, de um modo teórico. Generalizou então, que uma cifra será considerada **perfeita** se, por um criptograma c, a probabilidade dele corresponder a um texto original m e ter sido gerado por uma chave k é igual à probabilidade da ocorrência do texto m. Por outras palavras, significa que uma cifra é perfeita quando um criptanalista capturar um criptograma c, tal que não consiga concluir qual o texto original correspondente, porque para cada texto candidato existe sempre uma chave que pode ter efetuado a transformação. Para que isto possa acontecer, então é necessário que o processo

⊙ **Claude Shannon**

perfeita

de escolha das suas chaves seja verdadeiramente aleatório para tornar todas as chaves equiprováveis [2].

Olhando para este cenário, podemos concluir que o cardinal do espaço das chaves tem de ser igual ou maior ao cardinal do espaço de textos em aberto. Embora isto seja algo muito exigente, para uma cifra — note-se que geralmente as chaves possuem tamanhos finitos, o que por si, faz com que o cardinal do conjunto das chaves se torne muito mais pequeno que o cardinal do espaço de mensagens —, existe uma cifra perfeita: a **cifra de Vernam** (mais vulgarmente denominada de *One-Time Pad* ou OTP), como podemos ver na Figura 2.7 [11].

cifra de Vernam
⊙ **Gilbert S. Vernam**

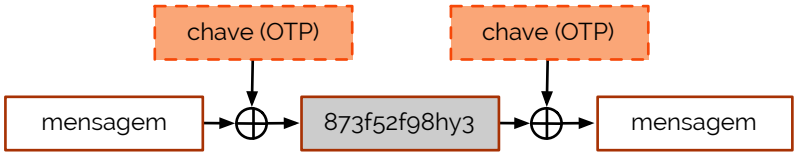

figura 2.7
cifra de Vernam

Esta cifra de Vernam, embora tenha muitas desvantagens como as chaves terem de ter um comprimento igual ou superior ao dos textos e para cada texto ter de ser usada uma chave diferente, já foi usada em condições extremas como no Projecto Venona [12], que foi uma colaboração secreta de larga duração entre as agências de inteligência secreta dos Estados Unidos e do Reino Unido, para a criptoanálise de mensagens da União Soviética, sobretudo durante a Segunda Guerra Mundial (entre 1943 e 1980).

Existem cinco conceitos que foram criados para avaliar sistemas de informação: **confidencialidade** é a capacidade de cifrar ou codificar uma mensagem a ser transmitida sobre uma rede insegura; **controlo de acesso** é a capacidade de controlar o nível de acesso que um indivíduo ou entidade possuem a uma rede ou sistema e quanta informação podem receber; **autenticação** é a capacidade de verificar a identidade de indivíduos ou entidades numa rede; **integridade** é a capacidade de garantir que uma mensagem ou dados não são alterados em trânsito, desde um remetente a um destinatário; e **não-repúdio** é a capacidade de previnir indivíduos ou entidades de recusarem que enviaram ou receberam um ficheiro, quando de facto o fizeram [13].

confidencialidade
controlo de acesso
autenticação
integridade
não-repúdio

Claude Shannon criou, no seu estudo teórico sobre as cifras, cinco critérios importantes para a sua descrição: a quantidade de secretismo oferecido (por exemplo pelo tamanho das chaves), a complexidade da chave selecionada (deteção de chaves fracas e geração de chaves), implementação de simplicidade, propagação de erros (canais de comunicação com ruído) e dimensão de textos cifrados (em comparação ao tamanho dos textos em aberto).

Existem, contudo, dois outros conceitos importantes que foram criados para caraterizar cifras. Num primeiro conceito, a **confusão** é considerada como uma relação complexa entre uma chave, um texto aberto e um texto cifrado. Por relação complexa pretende-se designar que é difícil descobrir partes do texto aberto desconhecidas, mesmo já possuindo conhecimento sobre outras (da mesma forma torna-se difícil a dedução da chave usada).

confusão

Num segundo conceito importante, a **difusão** pretende designar o caos a cujas estatísticas do texto aberto são sujeitas aquando da sua transformação para texto cifrado. Em termos mais práticos, cada bit de informação do texto original aberto deverá influenciar diversos bits do criptograma.

difusão

Depois da década de '50 o paradigma de aplicação da criptografia, num contexto mundial, começou a mudar para ambientes digitais. Como consequência deste feito surgiram novas categorias de cifras, entre as quais as **cifras contínuas**, que se basearam essencialmente na cifra de Vernam. Esta aproximação consistiu em substituir a chave infinita e aleatória (a designada *one-time pad*) por uma **chave contínua**, isto é, por um conjunto de dados produzidos por um gerador pseudoaleatório, de dimensão finita. Por fim, a chave produzida é misturada com o texto original de uma forma facilmente invencível,

cifras contínuas
chave contínua

nomeadamente somas lógicas exclusivas (XORs). Na Figura 2.8 podemos ver um esquema ilustrativo da operação destas cifras. [2]

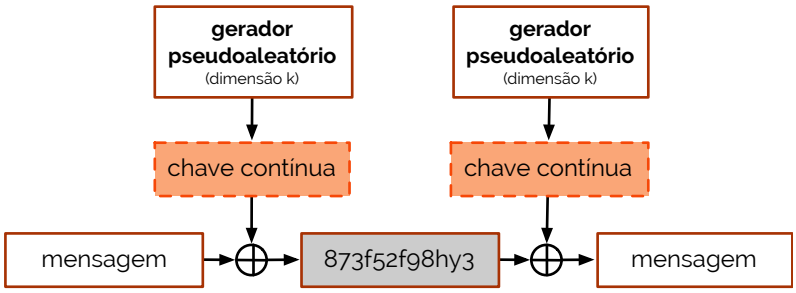

figura 2.8
cifras contínuas

Note-se que, no procedimento de cifras contínuas, como descrito acima, as chaves deverão ser o mais próximas possíveis de *one-time pads*, pelo que o período de uma chave contínua deverá ser o mais longo possível (melhor se for mais longo que o texto a cifrar) e a sequência de bits de uma chave contínua deverá aparentar ser verdadeiramente aleatória.

Cifras modernas

Com o surgimento das cifras contínuas, rapidamente as necessidades de evolução dos seus mecanismos aumentaram drasticamente, acompanhando o número de vulnerabilidades dos sistemas que se tornam cada vez mais complexos ao longo dos tempos.

Estas cifras poderão classificar-se dentro de duas categorias: as cifras em bloco (monoalfabéticas) e as cifras contínuas (polialfabéticas). Mas ainda podemos ter dois tipos para as caraterizar: as cifras simétricas e as cifras assimétricas.

cifra simétrica

Consideremos primeiro o caso das cifras simétricas. Designamos por **cifra simétrica** toda a cifra que tem uma chave partilhada por duas ou mais partes, sendo que se permite a confidencialidade nas mensagens partilhadas entre os vários intervenientes (quem tem chave). Este método, claramente, possui logo um problema de segurança aquando de uma má gestão de partilha de chaves — a partir do momento em que uma chave é mal partilhada, isto é, é partilhada com um indivíduo ou entidade que não deveria ter privilégios sobre a informação cuidada, a segurança implícita ao sistema é anulada ou perde o efeito.

rede de Feistel
⊙ **Horst Feistel**

As cifras simétricas costumam ser aplicadas em blocos de bits com tamanhos consideravelmente grandes como 64 bits, 128 bits, 256 bits, entre outros..., e através de métodos que garantem difusão e confusão como permutações, substituições, expansões, iterações, compressões ou aplicação de redes de Feistel. Uma **rede de Feistel**, como podemos ver na Figura 2.9 é tal que permitem que a tarefa de cifra e decifra sejam praticamente a mesma, pelo que diminuem a dificuldade de conseguir um algoritmo de cifra onde a sua função inversa seja difícil de encontrar [14].

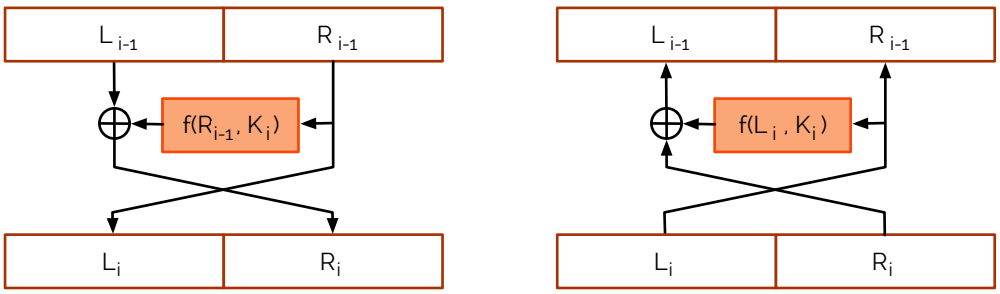

figura 2.9
redes de Feistel

DES

Um dos algoritmos mais conhecidos para a cifra de dados através de cifras simétricas é o **DES** (acrónimo inglês para *Data Encryption Standard*). Em 1972 um algo revolucionário foi pedido pela antiga NBS (sigla inglesa para *US National Bureau of Standards* — agora *National Institute of Standards and Technology*, NIST): pretendia-se que se criasse uma

norma para a criação de cifras nos Estados Unidos da América. A ideia-base concentrava-se então na busca de um só algoritmo criptográfico seguro que pudesse ser usado para um variado conjunto de aplicações. Até este ponto os vários governos que trabalhavam sobre cifras consideravam-nas tão importantes que achavam por bem mantê-las sob segredo absoluto, para evitar criptanálises. No entanto, a década de '70, principalmente nos Estados Unidos, exigiu que as cifras passassem a ser usadas também em outras várias aplicações comerciais, principalmente em bancos, de forma a que não houvesse qualquer tipo de colapso económico.

Neste concurso, vários candidatos participaram, mas entre todos a IBM destacou-se com um algoritmo baseado numa cifra denominada de *Lucifer*. Esta cifra *Lucifer* era uma família de cifras desenvolvidas por Horst Feistel nos fins da década de '60 e foi uma das primeiras ocorrências de cifras de blocos, operacionais, em ambientes digitais. O que tal cifra fazia era cifrar blocos de 64 bits usando chaves com tamanho 128 bits.

De forma a investigar a segurança das cifras submetidas, a NBS pediu colaboração da NSA (sigla inglesa para *National Security Agency*, que na altura ainda nem sequer era conhecida entre o povo americano) que, decerto, influenciou algumas mudanças feitas às próprias cifras. Uma das alterações que alegadamente a NSA fez foi mudar o tamanho da chave proposta de 128 bits para 56 bits, o que tornou a cifra muito mais vulnerável a ataques de força-bruta, como se viria a comprovar anos mais tarde [14].

A cifra DES é então uma cifra simétrica, isto é, uma cifra que usa a mesma chave tanto para a tarefa de cifra, como para a tarefa de decifra, sendo um processo meramente iterativo. No fundo, por cada bloco de texto em aberto a cifra seria feita em 16 iterações, todas elas efetuando precisamente a mesma operação, como podemos ver na Figura 2.10.

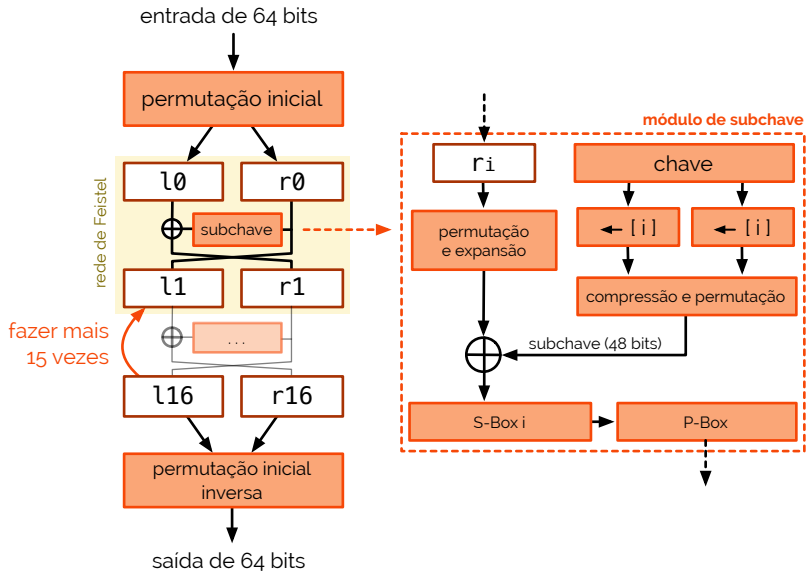

figura 2.10
algoritmo DES simplificado

Como podemos ver na Figura 2.10 a mesma operação que é executada nas 16 iterações é basicamente a aplicação de uma rede de Feistel com uma função que produz substituições através de *S-boxes*, permutações através de *P-boxes*, expansões e compressões. O seu funcionamento, basicamente, depois de uma permutação inicial e *bitwise* de um texto aberto x, este é dividido em duas metades L_0 e R_0. Estas duas metades de 32 bits cada são a entrada para a rede de Feistel, que consiste nas 16 rondas. Aqui, a metade direita R_i é re-entregue à função f, pelo que a saída desta é atribuída em módulo 2 (sob uma operação de XOR) à parte esquerda L_i de 32 bits. Finalmente, ambas partes direita e esquerda ficam trocadas. Em termos lógicos, no fim, ficamos com $L_i = R_{i-1}$ e $R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$, onde $i = 1, 2, \ldots, 16$. Depois da iteração 16, ambas as metades de 32 bits L_{16} e R_{16} são novamente trocadas e a permutação final é a última operação do DES.

Na rede de Feistel a função que denotamos como $f()$ é tal que, em cada ronda, derive uma nova chave (subchave) através da original de 56 bits — **key schedule**.

key schedule

Em termos de segurança, o DES pode usar todos os valores de 56 bits como chaves igualmente seguras, havendo apenas um conjunto de chaves (fracas ou semi-fracas) que não devem ser usadas porque pouco ou nada alteram o texto original[6] [2]. Notou-se, com os tempos, que a chave original de 56 bits era pequena, pelo que os ataques de força-bruta poderiam ser fáceis de efetuar com o desenvolvimento do potencial computacional no evoluir dos tempos. Como solução a este problema criou-se o conceito de cifra em cascata ou múltipla, isto é, num procedimento análogo a ter chaves de 112 ou 168 bits, passamos a ter duas ou três chaves e fazemos três vezes a cifra. Esta alteração ao DES é vulgarmente denominada de **3DES** (acrónimo inglês para *Triple-DES*).

3DES

Outros algoritmos também vulgares de cifras em bloco e simétricas são a AES, a IDEA, a Blowfish, RC5, entre outros...

O desenvolvimento de cifras simétricas de bloco também possui vários **modos** de aplicação. Entre vários existentes, o DES teve duas propostas de implementação iniciais, denominadas de ECB e CBC. O modo de **ECB** (sigla inglesa para *Electronic Code Book*) é bastante simples, pelo que para cada bloco de bits b aplica uma função de cifra para formar uma transformação direta do mesmo bloco em c e aplica uma função de decifra para formar através de c um bloco t. Este método pode ser visto na sua representação na Figura 2.11.

modos

ECB

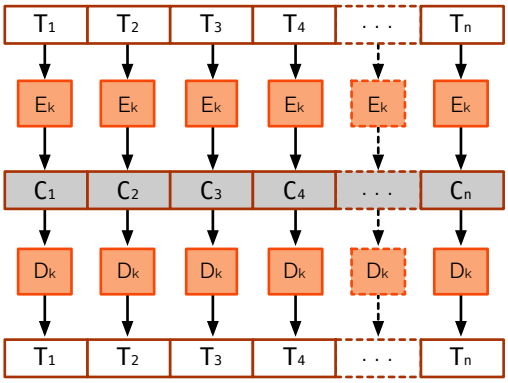

figura 2.11
Electronic Code Book (ECB)

Por outro lado, o DES também recebeu a proposta de implementação do modo CBC. O modo de **CBC** (sigla inglesa para *Cipher Block Chaining*), tal como o próprio nome o indica, tenta encadear o valor da transformação de cifra de um bloco com o seu anterior, sendo que no caso do primeiro bloco encadeia com um determinado **valor de inicialização** dado *a priori* (valor normalmente abreviado por IV, de *Initialization Value*). Já a decifrar o processo é o mesmo, mas no sentido inverso, isto é para decifrar um bloco é necessário a decifra do bloco anterior, sendo que no primeiro bloco é necessário o valor de inicialização. Podemos ver o seu funcionamento na representação do mesmo na Figura 2.12.

CBC

valor de inicialização

Então mas e o que é que acontece quando um bloco não é preenchido até ao fim? De forma algo análoga ao que temos visto em disciplinas de arquiteturas de computadores, em que os dados em memória, numa arquitetura RISC, precisam todos de estar alinhados a um determinado valor (porque os espaços de memória tinham todos o mesmo tamanho — estão organizados em blocos de bytes), aqui, em segurança, também será necessário **alinhar** o conteúdo dos blocos, para que este não fique preenchido até meio. Para isso usamos funções específicas de **padding**. Estas funções de aplicação de *padding* estão todas definidas no PKCS #7[7] que especifica que para um bloco de tamanho b, para obter x espaços de *padding* ter-se-á de calcular b - (m mod b), onde m é o tamanho da mensagem total. Para

alinhar

padding

6 Existem 4 chaves fracas, 12 semi-fracas, isto é, pares de chaves que produzem o mesmo resultado, e 48 potencialmente fracas, que só produzem 4 subchaves distintas, ao invés de 16.

7 Note-se que existe uma outra norma PKCS, a PKCS #5 que especifica todos os mesmos critérios que a PKCS #7, apenas com a nuance de que, obrigatoriamente, b tem o valor de 8.

o *padding*, depois podemos preencher os últimos x espaços com um determinado valor X[8]. O seu funcionamento está representado na Figura 2.13.

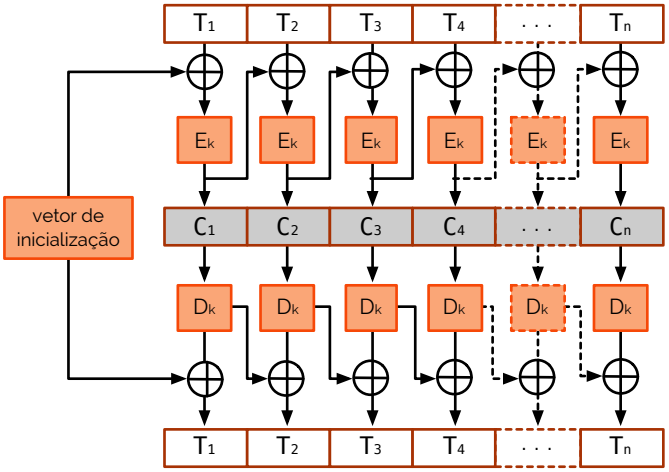

figura 2.12
Cipher Block Chaining (CBC)

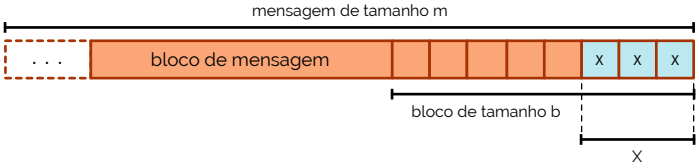

figura 2.13
padding PKCS #7

Olhando agora para cifras simétricas, mas de fluxo (ou contínuas), temos várias aproximações possíveis, que foram feitas ao longo dos tempos. Numa primeira aproximação podemos usar registos de deslocamento com *feedback* linear (em inglês *Linear Feedback Shift Register*, **LFSR**). Esta forma é assim baseada em circuitos de pequenas dimensões que, por si, contêm um número de células de memória, cada uma capaz de preservar um bit de informação, formando, num conjunto, e como vimos em Introdução aos Sistemas Digitais (a1s1), um **registo**. Estando o seu funcionamento descrito na Figura 2.14, em cada ciclo um conjunto predefinido de células são controladas e o seu valor é levado para uma função de *feedback*, também designado como polinómio de realimentação. Esta função serve para calcular o bit a introduzir no LFSR de cada vez que o seu estado interno é deslocado ($S_{n-1}(t+1)$) e, para o efeito, usa coeficientes C_i, com valores 0 ou 1, por cada registo interno S_i, e soma módulo 2 [15].

LFSR

registo

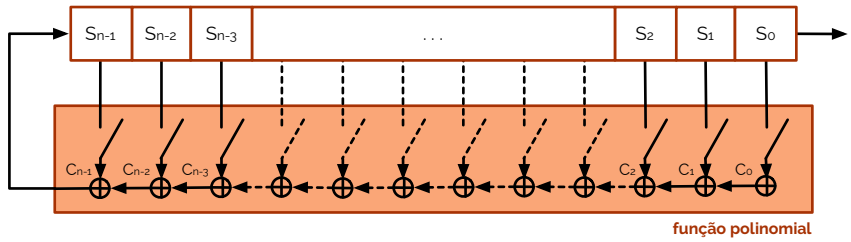

figura 2.14
diagrama de um LFSR

Note-se que um LFSR pode produzir várias sequências diferentes, no máximo 2^n -1, dependendo esse número da função de *feedback*. Cada sequência tem um período máximo limitado, que também não excede 2^n -1 bits. Para um dado polinómio de realimentação, se uma das sequências tiver o período máximo (sequência máxima) então todas as outras também o têm (nesse caso o LFSR apenas produz duas sequências, uma nula e outra de comprimento máximo). Designam-se por polinómios primitivos todos os que geram sequências máximas [2].

8 Estes valores de padding, ao longo dos processos de cifra e decifra mantém-se sempre iguais, provocando com que o último bloco íntegro "troque" informação com o bloco anterior.

A utilização de um LFSR como gerador de chaves contínuas pressupõe que alguns dos seus parâmetros operacionais sejam derivados de uma chave secreta, como o estado inicial do registo de deslocamento e o polinómio de realimentação. No entanto, fazer este último depender de uma chave é complicado, porque nem todos os polinómios são primitivos (o que poderia dar origem a sequências não máximas). O que se costuma fazer em tais casos é usar polinómios primitivos fixos, afetando-se o estado inicial do registo de deslocamento ao valor da chave secreta [2].

GSM

Esta aproximação de chaves simétricas contínuas é usada na tecnologia **GSM**. No entanto esta foi mantida em segredo durante algum tempo para que o sistema GSM se pudesse considerar como menos inseguro [16]. No sistema GSM os LFSRs foram implementados segundo um algoritmo denominado de **A5** (também conhecido como A5/1). Na verdade, neste algoritmo são usados três LFSR, cada um com diferente número de registos (19 bits, 22 bits e 23 bits) e funções de *feedback* que são polinómios primitivos. Uma descrição do circuito está representada na Figura 2.15.

A5

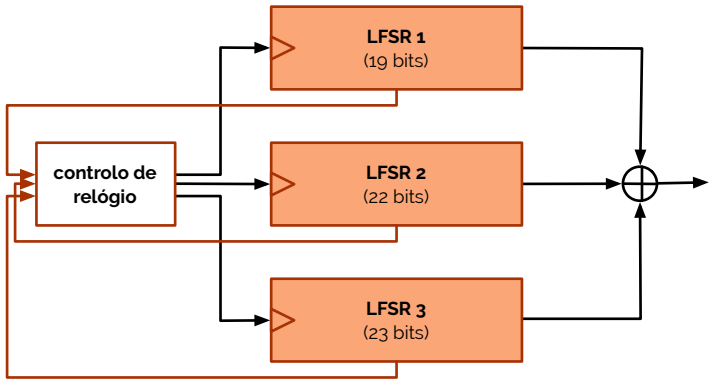

figura 2.15
uso de LFSR em A5/1

Da mesma forma que existem duas formas de execução do DES por cifras de bloco, também é possível a sua implementação com dois modos de aplicação com cifras contínuas: o modo OFB e o CFB.

OFB

No modo **OFB** (sigla inglesa para *Output Feedback Mode*), representado na Figura 2.16 [2], note-se que a chave contínua (*key stream*) não é gerada de forma *bitwise*, mas antes *blockwise* (bloco-a-bloco, ao invés de bit-a-bit). A saída da cifra, neste caso, dá-nos b bits de chave contínua, sendo que b é o tamanho do bloco de cifra usada, com o qual poder cifrar b bits de texto em aberto usando operações aritméticas de módulo 2.

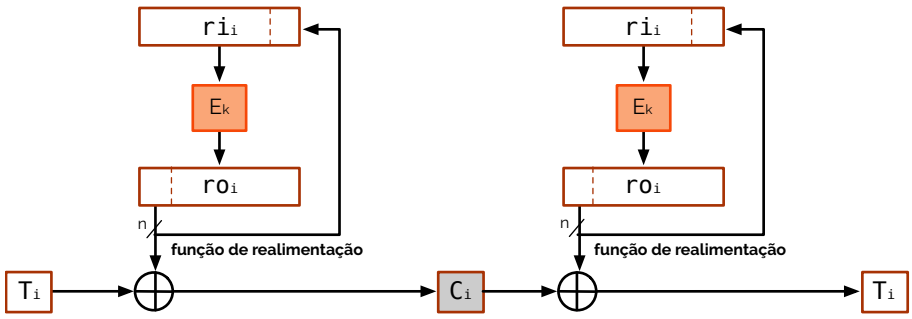

figura 2.16
Output Feedback (OFB)

A ideia por detrás do OFB até é bastante simples. Começando por cifrar um bloco com um valor de inicialização (IV), a saída do OFB dá-nos o primeiro conjunto de n bits de chave contínua. O próximo bloco de bits da chave é calculada depois através da realimentação da saída anterior novamente para o bloco da cifra e cifrando-o. Este processo, como podemos ver na Figura 2.16, é repetido várias vezes, até que se cubra o bloco por inteiro.

O modo OFB forma assim uma cifra contínua síncrona, sendo que o fluxo da chave não depende nem do texto em claro, nem cifrado. Como o modo OFB também forma uma cifra contínua, então pode-se dizer que as operações de cifra e decifra são precisamente as

mesmas, como podemos ver na Figura 2.16, uma vez que o recetor não usa o bloco de cifra na função $e^{-1}()$ para decifrar o texto cifrado — isto acontece uma vez que a cifra é executada por uma operação de módulo 2 (XOR) que, para a inverter, apenas temos de executar outra vez a mesma operação. Isto acontecer também em contraste com modos como o ECB e CBC, onde os dados são cifrados e decifrados através do bloco de cifra.

Como no OFB usamos um valor de inicialização, então tornamos este processo teoricamente não-determinístico, sendo que cifrar o mesmo texto em aberto duas vezes dá resultados diferentes (no caso do CBC o IV deverá se usado uma só vez) [14].

CFB

Por alternativa ao OFB, podemos também aplicar o modo **CFB** (*Cipher Feedback*), que volta ao uso de uma cifra de bloco como bloco construtor para uma cifra contínua. De forma algo semelhante ao OFB, o CFB, ao invés de realimentar o sistema com a saída do bloco de cifra, é o próprio texto cifrado que é objeto de realimentação. A ideia geral deste sistema é que para gerar o primeiro bloco de chave contínua s_1 se tenha de cifrar IV. Para todas os blocos de chave subsequentes s_2, s_3, ... podemos cifrar o texto cifrado anterior, como podemos ver na Figura 2.17.

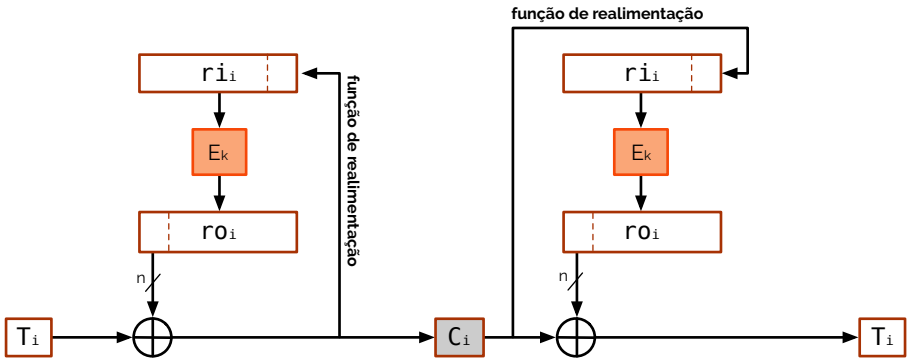

figura 2.17
Cipher Feedback (CFB)

Novamente, como o modo CFB também forma uma cifra contínua, então pode-se dizer que as operações de cifra e decifra são precisamente as mesmas. No entanto, e em contraste com o OFB, o modo CFB é um caso de cifra contínua assíncrona, uma vez que a saída da cifra é também uma função do texto cifrado.

Counter, CTR

Finalmente, um outro modo que também usa uma cifra de bloco como uma cifra contínua é o modo **Counter** (**CTR**). Voltando a um tratamento *blockwise*, neste caso a entrada para o bloco de cifra possui um contador que assume um valor diferente para cada vez que é calculado um novo bloco de chave contínua. Este processo está representado na Figura 2.18.

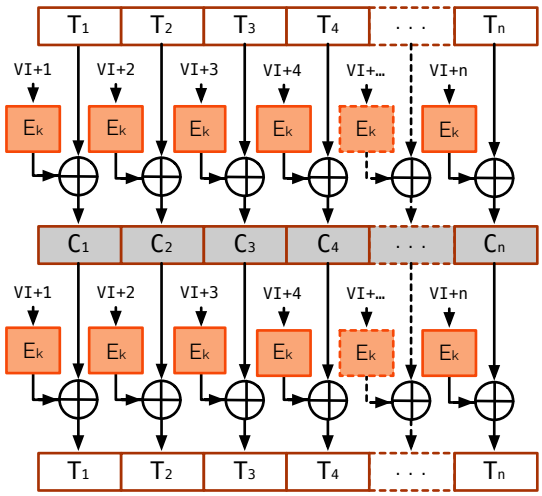

figura 2.18
Counter Mode (CTR)

Devemos ter cuidado com a forma como inicializamos a entrada do bloco de cifra, uma vez que devemos ter especial atenção a não dar a mesma entrada duas vezes. Caso tal aconteça, então se um atacante souber de um dos dois textos abertos que foram cifrados

com a mesma entrada, então ele poderá calcular o bloco de chave contínua e assim decifrar de imediato o outro texto cifrado. Para que se possa obter o caráter único, geralmente costuma-se fazer o seguinte: considerando uma cifra de bloco com uma entrada de 128 bits (como na cifra AES), escolhe-se um valor de inicialização (VI) — que se usa apenas uma vez — com um tamanho mais pequeno que o bloco (como 96 bits). Os 32 bits restantes são então usados por um contador com o valor CTR (inicializado a zero). Por cada bloco cifrado durante a sessão, o contador é incrementado, mas o IV fica igual. Neste exemplo que estamos a tentar descrever, o número de bloco que podemos cifrar sem escolher um novo IV é 2^{32}. Como cada bloco consiste em 8 bytes, então um máximo de $8 \times 2^{32} = 2^{35}$ bytes (aproximadamente 32 gigabytes) poderão ser cifrados antes que um novo IV seja gerado [14].

Na Figura 2.19 podemos agora comparar os vários modos de cifra que estudámos numa só tabela, onde pomos a teste as suas capacidades como a ocultação de padrões de entrada, confusão da entrada da cifra, uso da mesma chave para mensagens diferentes, dificuldade de interação com o algoritmo, propagação de erros e capacidade para recuperação de perdas [18].

	ECB	CBC	OFB	CFB	CTR
ocultação de padrões de entrada		aplica-se	aplica-se	aplica-se	aplica-se
confusão na entrada da cifra		aplica-se	só com contador	aplica-se	se o VI for segredo
mesma chave para mensagens diferentes	aplica-se	aplica-se	se os VI's forem diferentes	se os VI's forem diferentes	se os VI's forem diferentes
dificuldade de previsão de alterações	é difícil	é difícil, contudo possível	é fácil	é difícil, contudo possível	é fácil
pré-processamento			aplica-se		aplica-se
processamento em paralelo	aplica-se	aplica-se somente na decifra	só com pré-processamento	aplica-se somente na decifra	aplica-se
acesso aleatório uniforme					
propagação de erros do criptograma	no bloco atual	no bloco atual e no seguinte		apenas em alguns bits seguintes	
capacidade de recuperação de sincronismo	perdem-se blocos	perdem-se blocos		perdem-se múltiplos n bits	

figura 2.19
modos de cifra em suma

Um outro tipo de cifras muito usadas são as **cifras assimétricas**. Contrariamente às chaves simétricas, aqui usamos **pares de chaves**, entre as quais uma é considerada como uma **chave pública**, isto é, uma chave que designa, de forma pública, um indivíduo ou entidade e que serve principalmente para, de certa forma, garantir que se é a pessoa correta que está ou pretende aceder a uma determinada informação cifrada, e uma **chave privada**, que é pessoal e intransmissível, e permite a identificação inequívoca numa comunicação privada do indivíduo ou entidade que a possui.

cifras assimétricas
pares de chaves
chave pública
chave privada

Este método de chaves assimétricas permite a confidencialidade, uma vez que não há troca de segredos entre as várias partes integrantes de uma comunicação segura e autenticação, uma vez que o conteúdo transmitido é íntegro. No entanto, em comparação com as cifras simétricas, estas têm o forte prejuízo de serem bastante "pesadas" no seu processamento tanto em termos de tempo como em espaço de memória. Apesar disso, há que considerar que uma grande vantagem do seu uso é que como n pares necessitam de chaves comunicantes, então temos n pares de chaves finais. Ainda assim é necessário ter cuidado com aspetos de manutenção e gestão de chaves como a distribuição de chaves públicas e a vida de uso dos pares de chaves.

Então e em termos de processamento das operações com as chaves, a diferença é assim tão grande das chaves simétricas? Sim. Na verdade ambas as chaves do par constituintes

têm papeis completamente diferentes no processo. Comecemos com uma mensagem que pretendemos cifrar: se a Alice pretender cifrar a mensagem, então só tem de pegar na sua chave pública e cifrar a mensagem com esta. No fim deste processo ficamos com um texto cifrado que a Alice pretende enviar ao Bob. A Alice envia o texto ao Bob, mas agora como é que o Bob lê a mensagem? Ora, para abrir a mensagem apenas terá de usar a correspondente chave privada, dele, para o fazer. Note-se que, neste processo, só o par de chaves do destinatário é que está envolvido, pelo que para enviar qualquer ficheiro ou mensagem com confidencialidade para o Bob, então apenas é necessário ter conhecimento da chave pública dele (K_{Bob})[9]. Este processo poderá ser visto na Figura 2.20.

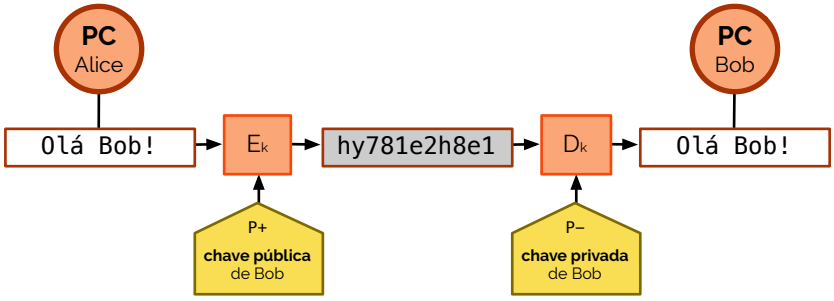

figura 2.20
cifras assimétricas

Se invertermos o esquema e a Alice decidir então cifrar uma mensagem com a sua chave privada, como podemos ver na Figura 2.21, o Bob recebe uma mensagem que não possui qualquer tipo de confidencialidade, uma vez que quem tiver a chave pública (e por ser pública qualquer um a pode ter) poderá abrir o conteúdo.

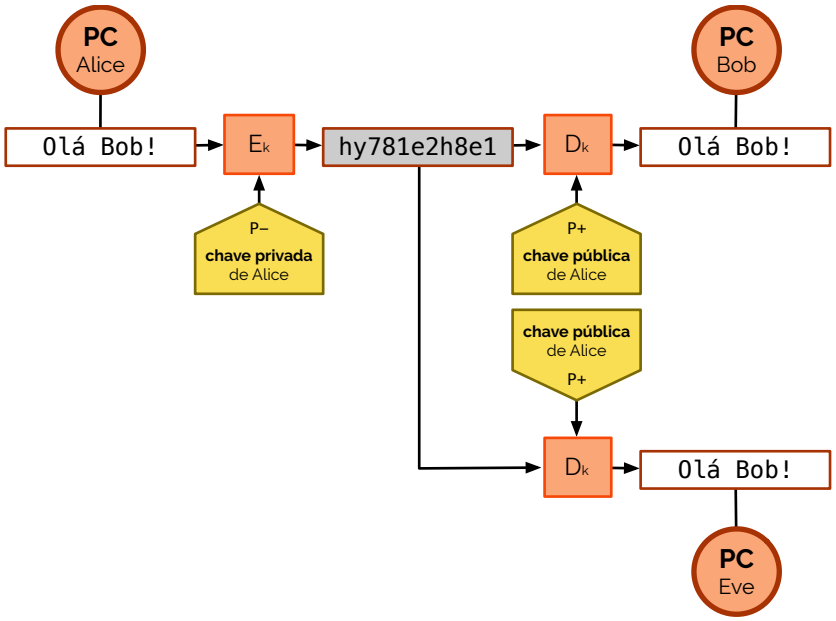

figura 2.21
cifras assimétricas (falta de confidencialidade)

As cifras assimétricas, dada a sua complexidade de processamento, são feitas em blocos para simplificação das suas manipulações possíveis. Existem algumas aproximações que foram criadas ao longo dos anos para a criação de métodos de pares de chaves, como logaritmos discretos de números muito grandes, fatorização de números muito grandes inteiros ou problemas do tipo *knapsack* (como abordámos em Algoritmos e Complexidade (a2s2)). Alguns dos algoritmos que daqui surgiram foram o RSA, o ElGamal ou as curvas elíticas (ECC).

[9] Repare-se que não existe autenticação da parte de quem envia um conteúdo. Na verdade Bob não tem como saber quem é que produziu os textos cifrados, isto dado que, se a sua chave pública for distribuída de forma pública mesmo, então toda a gente lhe poderá enviar mensagens cifradas.

Antes do desenvolvimento de tais algoritmos, surgiram primeiro outras técnicas para pares de chaves assimétricas, como o acordo entre chaves **Diffie-Hellman**. O acordo de troca de chaves Diffie-Hellman (também conhecido como DHKE), proposto por ambos Whitfield Diffie e Martin Hellman em 1976, foi de facto o primeiro esquema assimétrico publicado na literatura para o público. Este procedimento fornece uma solução prática para a distribuição de chaves, permitindo que duas partes derivem um segredo comum através de uma comunicação num canal inseguro. Esta é, na verdade, uma aplicação de logaritmos discretos de números muito grandes, que ainda está implementada em algumas soluções tecnológicas dos dias que correm (embora com algumas nuances), como o `ssh`, o `tls` ou o IPSec.

Diffie-Hellman
- Whitfield Diffie
- Martin Hellman

A ideia por detrás do DHKE está na aplicação de uma função unívoca, com um p sendo um número primo muito grande, onde a exponenciação é comutativa, isto é, $k = = (\alpha^x)^y = (\alpha^y)^x \bmod p$, onde k é o segredo comum que pode ser usado como chave de sessão entre as duas partes.

Vejamos então como é que se processa o algoritmo de Diffie-Hellman. Neste protocolo temos então duas partes, a Alice e o Bob, que pretendem estabelecer uma chave partilhada e secreta entre eles. Neste passo, eles (ou um terceiro que os ajude), terão que seguir os seguintes passos:

1. Escolher um número primo muito grande, p;
2. Escolher um inteiro $\alpha \in \{2, 3, \ldots, p-2\}$;
3. Publicar tanto p como α.

Estes dois valores publicados p e α são vulgarmente denominados de parâmetros de domínio e se ambos Alice e Bob tiverem conhecimento destes, então já poderão estabelecer uma troca de chaves de forma segura até conseguirem um k. Para isso a Alice deverá escolher um valor a, qualquer, desde dentro do conjunto $\{2, \ldots, p-2\}$, da mesma forma que o Bob também o deverá fazer para um valor b, dentro do mesmo conjunto. De seguida, cada um deverá calcular A (ou B, no caso de Bob) como $\alpha^a \bmod p$ (ou $\alpha^b \bmod p$), como chave e enviando-a para o Bob (para a Alice). Recebendo a chave de Alice (de Bob), então podemos chegar a um consenso em que $k_{AB} = B^a \bmod p$ (do lado de Bob $k_{AB} = A^b \bmod p$). Para provarmos que realmente isto funciona, a Alice poderá calcular B^a como sendo $(\alpha^a)^b$, isto é, $\alpha^{ab} \bmod p$, enquanto que o Bob calcula A^b como sendo $(\alpha^b)^a$, sendo também $\alpha^{ab} \bmod p$. Este processo encontra-se descrito na Figura 2.22 e a chave poderá ser agora partilhada para estabelecer uma ligação segura entre a Alice e o Bob, usando k_{AB} como chave para um algoritmo simétrico como o AES ou o 3DES [2].

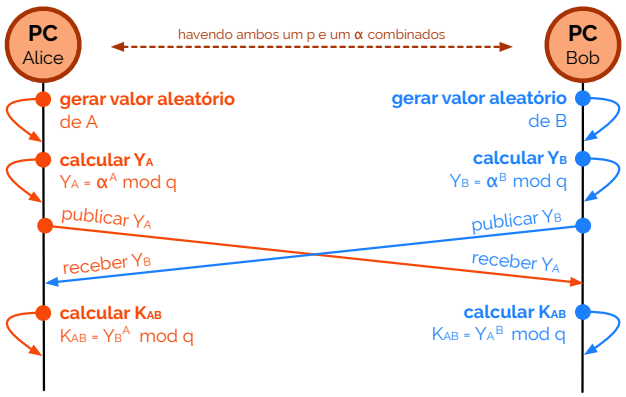

figura 2.22
algoritmo de Diffie-Hellman

Note-se que facilmente poderá colocar-se um terceiro interveniente neste processo, efetuando um ataque denominado de **Man-in-the-Middle** (vulgarmente abreviado de MITM), em que basicamente alguém (o Mallory) coloca-se na posição do Bob em relação à Alice (e na posição de Alice em relação ao Bob), trocando mensagens que para a Alice (para o Bob) seriam perfeitamente normais [18].

Man-in-the-Middle

Depois de Diffie e Hellman terem introduzido uma cifra por chave pública em 1976 como já estudámos, um vasto conjunto de novas oportunidades de evolução dos algoritmos de criptografia foram criadas. Como consequência, criptólogos começaram a procurar novas formas de aplicação das chaves públicas. Em 1977, uma equipa formada por Ronald Rivest, Adi Shamir e Leonard Adleman, propôs um esquema novo que se tornaria no algoritmo criptográfico e assimétrico mais usado de sempre até à data (novembro de 2017) — o **RSA**.

- Ronald Rivest
- Adi Shamir
- Leonard Adlemand

RSA

O algoritmo de RSA (também por vezes referida como o algoritmo de Rivest-Shamir-Adleman), patenteado nos Estados Unidos da América, é vulgarmente usado para a cifra de pequenos conjuntos de dados (especialmente para o transporte de chaves) e para assinaturas digitais (assunto a abordar mais à frente). Note-se, não obstante, que a cifra RSA não foi criada para substituir cifras simétricas, uma vez que, por ser assimétrica, é muito mais lenta que, por exemplo, uma cifra AES. Isto acontece porque há alguns cálculos efetuados no algoritmo RSA que precisam de algum esforço computacional.

O grande problema que o RSA tenta resolver é a função unívoca para um problema de fatorização de inteiros. Quando multiplicamos dois números primos, mesmo que num computador, o cálculo é bastante simples (até o conseguimos fazer mesmo com um lápis e papel), mas quando se trata de fatorizar o produto final então a tarefa toma dimensões completamente diferentes. Depois este algoritmo ainda se complementa com o cálculo de logaritmos discretos.

A cifra e decifra em RSA é feita através de um espaço inteiro e onde as operações de módulo tomam o papel principal. A primeira coisa que se tem de fazer é escolher dois números muito grandes primos p e q, sendo que devemos calcular um n para que seja o produto de p e q. De seguida, temos que escolher um co-primo[10] com o produto de $p-1$ com $q-1$, sendo que um d terá de ser calculado tal que $e \times d = 1 \bmod (p-1) \times (q-1)$. Tendo estes valores, a chave pública será $K = (e, n)$ e a chave privada será $K^{-1} = (d, n)$, pelo que cifrar um texto x com a chave pública teria a forma de $x^e \bmod n$. Já obter um texto aberto de um cifrado c com a chave privada teria a forma de $c^d \bmod n$. Pela ordem inversa, com as chaves restantes, cifrar um texto x com a chave privada terá a forma de $x^d \bmod n$ e decifrar um texto cifrado c com a chave pública teria a forma de $c^e \bmod n$ [2, 14, 19].

O RSA, contudo, na sua forma mais original, possui algumas vulnerabilidades, uma delas porque, por tentativa-erro, é possível ir descobrindo as partes do algoritmo que supostamente seriam secretas (enumerem-se e e d, mas mais em particular d). Para isso uma boa forma de resolver o problema seria trazendo tópicos de aleatoriedade para o algoritmo, de forma a torná-lo não-determinístico, ou seja, de modo a que n cifras de um mesmo valor, com a mesma chave, possam fornecer n resultados diferentes (com uma probabilidade de colisões teoricamente nula). Um esquema que é muito usado é o **OAEP** (sigla inglesa para *Optimal Asymmetric Encryption Padding*) e tem a forma da Figura 2.23.

OAEP

Este algoritmo, quando é usado com o RSA toma o nome de RSA-OAEP e basicamente o que faz é adicionar um *padding* à mensagem original que juntamente com funções de *hash* (que iremos ver de seguida) permite que a tarefa de tentativa-erro na procura de obter valores secretos se torne cada vez mais difícil.

Funções de hash

Em Métodos Probabilísticos para Engenharia Informática (a2s1) vimos que uma **função de hash** era uma função usada para mapear dados de um tamanho arbitrário para um determinado tipo de dados (geralmente inteiro) com um tamanho fixo.

função de hash

Pelo princípio da gaiola de pombos, também já sabemos que se o cardinal do conjunto de valores iniciais possíveis (antes da execução de uma função de *hash*) for maior que o cardinal do conjunto dos valores pós-*hash*, então haverá pelo menos dois valores iniciais que passaram a estar representados por um só mesmo valor pós-*hash*.

---

10 Um número co-primo de um segundo se e só se o único número positivo e inteiro que divide ambos é 1. É habitual escolher-se um valor para e como 3, 17 ou 65537 [16].

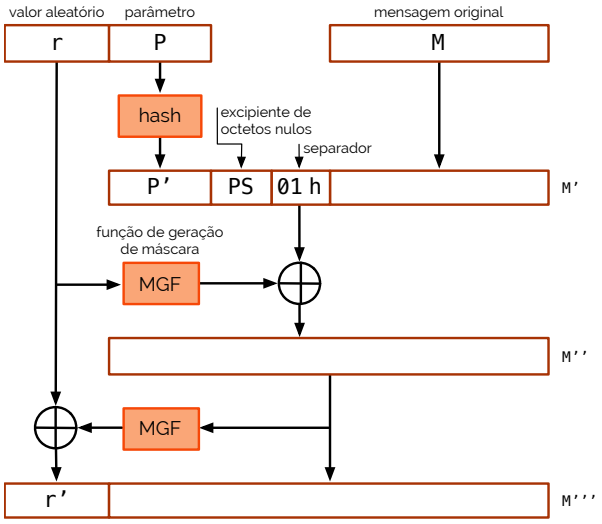

figura 2.23
OEAP

Estas funções, para serem admitidas como **digest functions** deverão passar três critérios essenciais: dado um valor de *hash* h deverá ser difícil encontrar qualquer mensagem m tal que $h = hash(m)$, pelo que a função deverá ser unívoca e resistente a pré-imagens; dado uma entrada m, deverá ser difícil encontrar uma entrada m_2 tal que $hash(m) = = hash(m_2)$ — isto descreve o efeito avalanche, onde a partir da alteração de um só carater de uma palavra, por exemplo, terá de resultar num $hash(m)$ completamente diferente; deverá ser difícil descobrir duas mensagens m_1 e m_2 distintas tais que $hash(m_1) = hash(m_2)$, uma vez que se trata de uma **colisão**.

digest functions

colisão

Alguns dos algoritmos mais conhecidos para **síntese** (funções de *hash* ou *digest*), em criptografia, são o **MD5** (que já não é mais considerado seguro, uma vez que descobriram várias colisões possíveis), o **SHA-1** (sigla para *Secure Hash Algorithm*, que com 160 bits também já lhe descobriram colisões em 2017) e a **SHA-2** (versão 2 com 256 ou 512 bits).

síntese

MD5

SHA-1

SHA-2

Códigos de autenticação de mensagens (MAC)

Uma forma de adicionar informação sobre um "autor" de um conteúdo, numa mensagem que deve ser transmitida sobre um canal seguro é feita através da adição de uma partícula contígua à mensagem denominada de **MAC** (acrónimo inglês para *Message Authentication Codes*). Este campo, serve assim de autenticação de uma mensagem perante um segundo interveniente numa comunicação.

MAC

Em termos básicos, o que acontece é que através de uma função de *hash*, é-lhe entregue uma chave com que a função de *hash* deverá trabalhar para produzir um valor de tamanho limitado para ser representado na mensagem a ser enviada, como podemos ver na Figura 2.24.

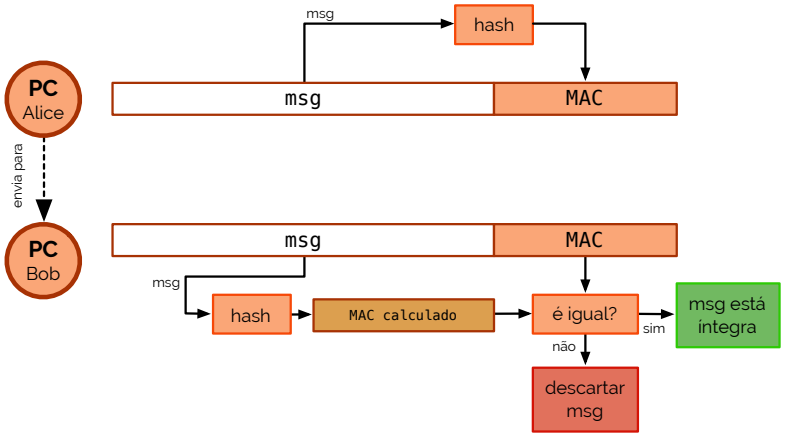

figura 2.24
MAC

Uma implementação possível deste esquema está no **HMAC**, onde se acaba por adicionar uma chave aos próprios dados *hashed* (sintetizados) e onde esta decide qual o tamanho do campo MAC final, dependendo da função H que for usada.

HMAC

Mais, a aplicação do campo MAC final, no envio da mensagem poderá ser feita numa de três formas recorrentes: poderá ser feita segundo um esquema **MAC-then-encrypt**, onde o MAC é calculado através do texto aberto e depois cifrado; poderá ser feita segundo um esquema de **encrypt-and-MAC**, onde o MAC é apenas calculado do texto aberto; ou simplesmente através de um esquema de **encrypt-then-MAC**, onde o MAC é calculado diretamente do criptograma.

MAC-then-encrypt

encrypt-and-MAC

encrypt-then-MAC

Assinaturas digitais

Como referimos, o MAC é uma forma de fornecer autenticação perante uma mensagem que é enviada num canal de comunicação, mas de forma seguro, mantendo a identidade inicial. Mas em que casos da nossa rotina é que precisamos de garantir que somos mesmo nós quem redigiu ou acatou uma mensagem ou decisão?

Consideremos que temos um contrato à nossa frente. Para que tal contrato seja aceite e devidamente celebrado entre nós e mais uma entidade é necessário que construamos um novo documento a comprovar que lemos o contrato e que tenha alguma informação escrita por nós a aceitar o mesmo. Para que tal novo documento seja aceite é necessário que nele, de alguma forma, exista uma marca única nossa — uma **assinatura**.

assinatura

Em termos digitais aplica-se a mesma lógica — uma forma clara que comprovar que fomos nós que criámos um determinado conteúdo é colocando uma marca de assinatura sobre ele. Note-se que tal assinatura, contrariamente à que fazemos à mão com uma caneta, não é gráfica, mas antes uma **assinatura digital**.

assinatura digital

Em comparação a técnicas como a MAC, neste caso estamos a manter informação legível sobre a origem de um documento mesmo que este seja entregue a um terceiro, fora do nosso canal de comunicação, informação que não passa no caso da aplicação de MAC [2].

Mais do que uma assinatura convencional consegue fazer, uma assinatura digital também server para verificar se alguma partícula de um ficheiro mudou depois da assinatura ter sido criada. Quando entregamos um contrato assinado à mão por nós, não ficamos com garantia de que os dados não são alterados depois de termos assinado. No entanto isso não acontece com as assinatura digitais, uma vez que a assinatura passa a estar inválida uma vez que o conteúdo mudou sem nova revisão de quem assinou. Ainda com esta funcionalidade, esta técnica é mais eficiente do que a anteriormente enunciada.

Devido ao elevado custo computacional das cifras assimétricas, não interessa cifrar a totalidade do documento a assinar, mas antes um valor de pequena dimensão que represente o mesmo, ou seja, uma síntese do mesmo. Usando a síntese obtém-se uma maior eficiência na geração e validação de assinaturas digitais e obtêm-se assinaturas mais reduzidas. Assim, a geração (2.1) e validação (2.2) de assinaturas atribuíveis à entidade x são feitas da seguinte forma [2]:

$$A_x = \text{contexto de assinatura}, E_{K_x^{-1}}(\text{hash}(\text{contexto de assinatura}, \text{documento})) \quad (2.1)$$

$$D_{K_x}(A_x) \equiv \text{hash}(\text{contexto de assinatura}, \text{documento}) \quad (2.2)$$

Na Figura 2.25 podemos ver um diagrama onde especificamos como é que as assinaturas são representadas num ficheiro assinado.

Quando acontece um erro de verificação da assinatura, isto significa que os mecanismos de controlo de integridade de algoritmos como o de RSA tiveram de intervir para avisar que o conteúdo de um determinado ficheiro já não se encontra da forma como alguém o deixara e verificara. Quem fala de RSA para as assinaturas digitais, também pode

referir o **DSA** (sigla inglesa para *Digital Signature Algorithm*), sendo uma variação do algoritmo de assinatura ElGamal. **DSA**

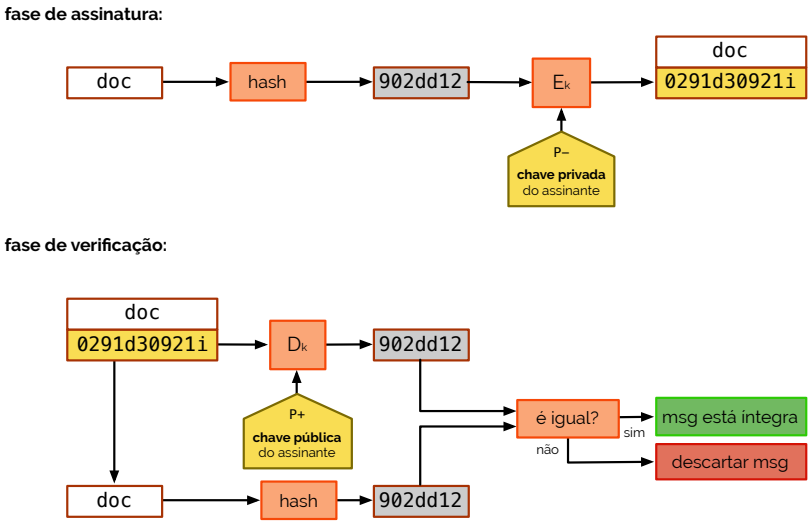

figura 2.25
assinaturas digitais

Consideremos agora um sistema de votação eletrónica. É importante, num cenário deste tipo, que sejam feitas assinaturas digitais sobre os votos realizados, uma vez que estas podem garantir que o seu valor mantém-se igual durante o tempo e não foi corrompido. No entanto, o voto é secreto por lei (contrassenso ironizado na Figura 2.26 [20]). Então como é que podemos fazer uma assinatura de forma "anónima"?

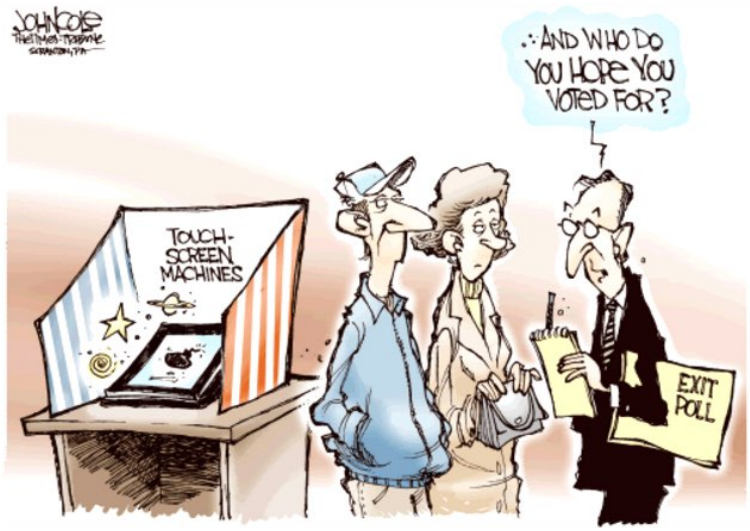

figura 2.26
assinatura às cegas

A ideia de uma assinatura em votos eletrónicos não é difícil de fazer, uma vez que se podem fazer **assinaturas às cegas**, isto é, permitindo que se validem os boletins de voto pertencentes a votantes autorizados, sem conseguir fazer qualquer ligação entre a assinatura gerada, o voto e o votante [2]. É também análogo a uma assinatura feita sobre um envelope fechado onde se colocou o documento a assinar debaixo de uma folha de papel químico, com o qual, após a remoção do envelope, se recupera o documento convenientemente assinado e que o assinante não viu. **assinatura às cegas**

Estas assinaturas às cegas têm o seu requerente, então, primeiro a esconder o texto a assinar (de modo a transformá-lo em algo totalmente aleatório e não associável ao texto inicial), sendo que o assinante assinará o texto escondido e devolve a assinatura (que será posteriormente alterada pelo requerente, para que não haja efeitos secundários da política normal e comum de assinaturas digitais). No final o requerente obtém uma assinatura digital normal do texto, essa que todos poderão validar, ainda que o assinante não consiga

ligar a assinatura feita sobre o texto escondido à assinatura do texto que todos conseguem validar e ver que é sua.

3. Gestão de Chaves Assimétricas

Quando introduzimos as chaves assimétricas tivemos oportunidade de referir que no meio de muitas vantagens de tal tipo de chaves perante as simétricas, uma das grandes desvantagens é a sua gestão, isto é, o seu mau uso poderá cancelar por completo qualquer uma das vantagens deste sistema de segurança. Por esta mesma razão é importante saber como gerir estas chaves ao longo do seu tempo de uso e saber porque é que é importante nunca esquecer de renovar as chaves.

Problemas de má gestão de chaves

Com chaves assimétricas a nossa unidade de criptografia aplicável são os **pares de chaves**, isto é, pares de chaves constituídos por uma chave privada (explicitamente exclusiva de um indivíduo ou entidade) e por uma chave pública (chave entregue ao público que dela necessita). O que é que pode acontecer quando uma destas chaves cai na mão errada?

pares de chaves

Consideremos então o primeiro cenário — decerto o mais simples de todos — em que uma chave privada de um par de chaves cai nas mãos erradas: que mal poderá acontecer? Ora, se um terceiro receber a nossa chave relembremo-nos que essa pessoa poderá assumir a nossa identidade num contacto com quem possui a nossa chave pública. Isto acontece porque uma das caraterísticas da chave privada é precisamente a autenticação num canal de comunicação.

Num segundo cenário, consideremos que alguém indevido fica com a nossa chave pública. Que mal poderá acontecer, já que a chave, por si, é pública? Estando a chave pública nas mãos erradas, então qualquer conteúdo que nós pretendamos que seja confidencial deixará de o ser, uma vez que de todas as pessoas com chave pública que poderiam desvendar o conteúdo cifrado, agora uma delas vai conseguir abri-lo e saber quem somos.

Mas vejamos ainda um último cenário, mais simples, em ligação com o último tópico abordado — as assinaturas digitais. Se nos ficarem com a chave privada, então nós iremos ter necessidade de repudiar as assinaturas realizadas a partir desse momento, pelo que será o único caso em que não seremos nós a concretizar uma assinatura digital, mesmo que apareça com o nosso nome.

Como podemos ver, existem vários casos em que a segurança terá de ter uma presença não só passiva, mas como ativa. Para isso precisamos de saber como manter as nossas chaves privadas privadas (perdoe-se o pleonasmo), como distribuir as chaves públicas e que **tempo de vida** lhes havemos de dar, isto porque passado algum tempo de uso chegamos a um ponto em que já não lhes havemos de saber onde e na posse de quem é que elas ainda existem, enquanto válidas.

tempo de vida

Distribuição de chaves públicas e certificados

Numa comunicação entre duas entidades ou indivíduos, se a Alice pretender comunicar com o Bob de uma forma segura, então a Alice precisará de ter a chave pública do Bob, para que este consiga decifrar a conversa com a sua chave privada. Assim sendo, isto significa que o Bob partilhou a sua chave pública com a Alice, uma vez que ambos precisaram de estabelecer um contacto (neste caso numa só direção).

Consideremos agora uma situação em que recebemos uma chamada de uma pessoa desconhecida (mas identificada por um número), mas em que um amigo nosso A, ao nosso lado, nos diz "atende, que fui eu que dei o teu contacto a um amigo B meu". Que raciocínio acontece neste caso, e que nos poderá fazer atender a chamada? Ora, como nós somos amigos de A e confiamos nele, e A é amigo de B e confia no seu contacto telefónico, então nós devemos poder confiar no contacto telefónico de B. Formou-se assim uma **cadeia de confiança**.

cadeia de confiança

A distribuição de chaves públicas também pode funcionar de forma análoga. Vejamos a Figura 3.1.

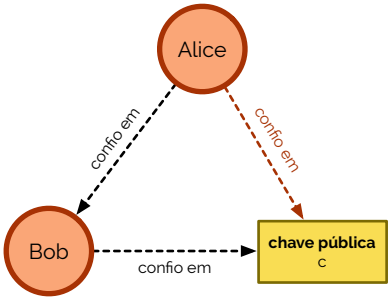

figura 3.1
cadeia de confiança

Na Figura 3.1 podemos reparar que se a Alice confia no Bob e se o Bob confiar numa chave pública *c*, então a Alice também pode confiar na chave *c*. Isto é uma cadeia de confiança que foi estabelecida, com a forma de um grafo.

Com este raciocínio sobre confianças podemos chegar a um novo conceito: o conceito de certificado. Um **certificado** é uma declaração assinada por um assinante (em inglês *issuer*) sobre um determinado sujeito (em inglês *subject*) e a sua chave pública [21]. O assinante aqui poderá ser uma entidade denominada de *Certification Authority* (**CA**) que acaba por ser quem tem a responsabilidade de ligar uma chave pública com uma entidade (indivíduo, servidor ou serviço), verificar se os documentos são públicos (se estes não contêm qualquer informação que seja privada) e se são criptograficamentem seguros, isto é, se estão digitalmente assinados pelo *issuer*, não podendo ser alterados. É da responsabilidade de uma CA, também, distribuir chaves públicas através de cadeias de confiança, disponibilizando as várias chaves e deixando para quem precisa dos certificados, a capacidade de os validar. Se a chave pública do assinante (CA) é confiada e a sua assinatura estiver correta, então quem recebe o certificado poderá confiar na chave pública (também certificada).

certificado

CA

Os certificados em si, para que todos estes processos de confiança possam existir e funcionar em uníssono, estão devidamente normalizados em diversos protocolos de certificação, como é o caso do **X.509v3** (que especifica obrigatoriamente campos como a versão, sujeito, chave pública, datas de lançamento e de validade, *issuer*, assinatura, entre outros e algumas extensões possíveis de implementar[11]) e da norma de apresentação binária **ASN** (em particular a ASN.1, sigla para *Abstract Syntax Notation*, que é a base de representação, por exemplo, de sistemas como o SNMP para a gestão de manutenção de uma rede de computadores, como vimos em Arquitetura de Redes (a3s2)) e das definições do PKCS #7 (norma sobre forma das mensagens criptográficas) e do PKCS #12 (norma sobre protocolo e sua sintaxe de trocas de informação pessoais).

x.509v3

ASN

Mas quem são estas entidades de certificação (CA) que atribuem chaves públicas a nomes? Quando pretendemos criar um certificado devemos pensar em quem é que deverá autorizar o lançamento destes. Por exemplo, é fácil uma empresa tomar uma posição de certificação dentro do conjunto dos seus empregados: uma empresa não poderá mudar o nome de um seu funcionário, mas sabendo a forma como é chamado dentro das suas instalações pode ser critério para que a empresa possa certificar que determinado empregado tem um nome *A*, quando na verdade o nome dele é *B* [17].

As entidades de certificação (CA), para além do lançamento de certificados, também têm autoridade suficiente (e responsabilidade) de revogar certificados, distribuí-los e lançar e distribuir as respetivas chaves privadas. Com isto, devemos ficar intrigados sobre como é que uma entidade CA poderá revogar um certificado. Ora, para o **revogar**, da mesma forma que existe um controlo sobre a distribuição de certificados, também tem de haver uma forma de gerir várias **listas de revogação** de certificados. Surgem assim os **CRL** (sigla inglesa para *Certificate Revocation Lists*) que são mensagens denominadas **base** e **delta** onde

revogar

listas de revogação, CRL

base, delta

11 As extensões estão definidas pelo PKCS #6.

se especificam listas de certificados a revogar e atualizações sobre a revogação de certificados, respetivamente.

Estes certificados cujas informações são atualizadas por vias de CRLs são revogados de forma prematura, isto é, antes de os próprios expirarem. A sua distribuição é tarefa do protocolo de **OCSP** (sigla para *Online Certificate Status Protocol*) que atualiza regularmente os vários certificados existentes numa máquina ligada à rede (pelo menos uma vez por dia). Caso haja um pedido de revogação, uma lista de detalhes sobre tal acontecimento será debitada igualmente no mesmo CRL, definido pelo RFC 3280. Na verdade, isto também acontece porque cada CA mantém o seu CRL e permite-lhe o acesso público, pelo que os vários CAs trocam CRLs para facilitar a sua propagação [22].

OCSP

↘ **RFC 3280**

Na Figura 3.2 podemos ver um caso de aplicação de CRL onde inicialmente (no instante de tempo t_1) é lançado um ficheiro CRL (número n) com uma lista de certificados a revogar. Em t_3 é lançada uma atualização num novo CRL denominado de ficheiro-delta (com número $n + 1$) onde é revogado o certificado com número de série x, permanecendo tal informação no CRL. A mesma coisa acontece em t_5, onde um novo certificado y é revogado e tal informação é acrescentada ao CRL anterior, atualizado agora com o número $n + 2$ (num novo ficheiro-delta).

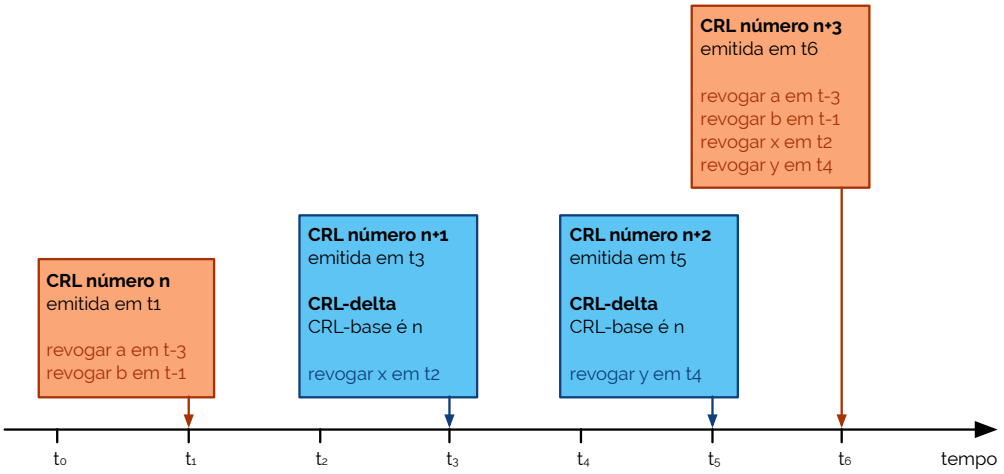

figura 3.2
OCSP

Gestão de entidades de certificação e hierarquias de certificação

O uso de pares de chaves e de certificados é então garantido por uma *interface* denominada de **PKI** (sigla inglesa para *Public Key Infrastructure*) que é um conjunto de políticas e procedimentos necessários para criar, gerir, distribuir, usar, guardar e revogar certificados digitais, tal como gerir as cifras com chaves públicas. O seu propósito é, então, facilitar a transferência de informação de uma forma mais segura, em atividades ruidosas como *internet banking*, mensagens de correio eletrónico confidenciais ou até lojas *online* (*e-commerce*). É também o PKI que se encarrega de verificar e inserir elementos em cadeias de confiança (nas hierarquias) e ativar regularmente os serviços *online* OCSP.

PKI

As hierarquias de certificação que hoje se formam têm base em dois modelos: o PEM e o PGP. O modelo **PEM** (acrónimo de *Privacy Enhanced Mail*) baseia-se num ideal de colocar uma hierarquia mundial (em forma de monopólio) com uma só raiz (a IPRA, acrónimo *Internet Policy Registration Authority*) e vários filhos PCA (*Policy Creation Authorities*), cada um com vários CAs (possivelmente pertencentes a várias organizações e companhias) [22, 23]. Note-se que este modelo nunca foi verdadeiramente implementado, tendo sido antes adaptado para caraterizar os ficheiros que preservam as chaves nas nossas máquinas, com uma forma humanamente legível — os ficheiros com extensão `*.pem`. A razão pela qual nunca foi implementado da forma como foi desenhado está no facto de ficarmos com uma floresta de hierarquias no fim, cada uma com uma CA raiz, em modo de **oligarquia**.

PEM

oligarquia

Num modelo mais realista surgiu então o **PGP** (sigla para *Pretty Good Privacy*) onde são estabelecidas redes de confiança. Em tais redes não existe nenhum nó central com autoridade sobre as confianças, pelo que cada indivíduo é um certificador potencial, certificando, se pretender, uma chave pública (lançar um certificado) e publicando-a. Para isto estão definidos dois tipos de confiança: a confiança nas pessoas conhecidas e a confiança no comportamento dos certificados (consideração de que se deve saber o que fazer quando se lança um novo certificado).

PGP

Smart Cards

Uma forma alternativa de portar uma possibilidade de assinar digitalmente um documento está no uso de **smart cards**. Um *smart card* é um cartão que possui um *chip* com um sistema de ficheiros e protocolos únicos que sejam capazes de fazer operações de assinatura digital e/ou de autenticação.

smart cards

Necessitando de uma interface própria de comunicação com o exterior (com uma aplicação cliente), um *smart card* é tal que troque um conjunto de mensagens próprias, com um dispositivo, denominadas de **APDU**. Um APDU (sigla inglesa para *Application Protocol Data Unit*) é uma unidade de comunicação entre um leitor de *smart card* (uma *interface* de comunicação própria) e um *smart card*, descrita no ISO 7816-4.

APDU

↘ ISO 7816-4

O datagrama de um comando APDU está representado na Figura 3.3 (as dimensões indicadas estão em bytes).

figura 3.3
comando APDU

Olhando para o comando APDU representado na Figura 3.3 podemos verificar que primeiro temos um campo de 1 byte denominado `CLA` onde se designa a classe da instrução, seguido da descrição da mesma codificada igualmente em 1 byte no campo `INS`, de instrução. De seguida temos dois campos, ambos de 2 bytes, denominados de `P1` e de `P2` que servem para transportar dois parâmetros para as instruções a serem executadas. Continuando podemos verificar o campo `LC` que serve para indicar o tamanho dos dados opcionais do comando APDU, terminando aqui a porção do datagrama respetiva ao cabeçalho do mesmo. O corpo deste comando APDU está no desígnio dos dados opcionais e na descrição do tamanho dos dados esperados em resposta, no campo `LE`, sendo que, se este campo estiver a zeros, significa que não se espera por qualquer tipo de resposta.

O APDU em reposta a um comando, responde com um ADPU Response, cujo datagrama é visível na Figura 3.4.

figura 3.4
resposta APDU

No caso de uma resposta APDU, esta vem com dados opcionais seguidos de dois campos de estado, entre os quais o `SW1` e o `SW2`, ambos de 2 bytes, sendo que, por exemplo, se `SW1` possuir o valor `0x9000` então significa que a operação foi bem sucedida.

Na Figura 3.5 podemos agora ver como é que funciona a interação entre o *smart card* e o dispositivo que usa o leitor de *smart cards* para lhe aceder às propriedades e funcionalidades.

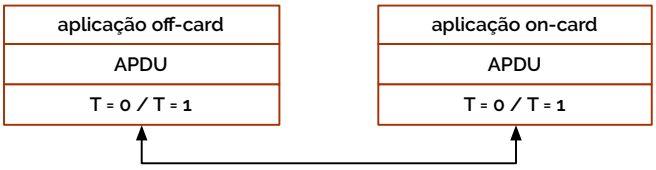

figura 3.5
stack protocolar de comunicação (smart-card)

Como podemos ver na Figura 3.5 temos, para além da *interface* de controlo de entrada e saída, e das APDUs, um primeiro nível que identifica o protocolo de transmissão (designado pela norma ISO 7816-3). Esta identificação é dada pelo valor de T como sendo 0 ou 1 (critério de escolha) [24].

ISO 7816-3

Os objetos, da forma como são guardados nos *smart cards*, precisam de estar sob uma estrutura que seja capaz de os representar devidamente, uma vez que dois objetos não terão de representar uma mesma entidade. Existem, por esse motivo, duas técnicas que lhes são aplicadas para que estes possam coexistir num cartão: o TLV e o ASN.1 BER. O **TLV** (sigla para *Type-Length-Value*) é um esquema de codificação usado para informação considerada opcional (sob a forma de elementos) dentro de um protocolo. Neste esquema de codificação existem assim três campos: um **tipo**, que é um código binário (geralmente alfanumérico) que indica o tipo do campo TLV; um **tamanho**, do valor representado no campo; e um **valor**, que é o objeto que pretendemos preservar. Algumas das regras de codificação para o TLV são especificadas pelo **ASN** (sigla para *Abstract Syntax Notation*).

TLV

tipo

tamanho

valor

ASN

Os *smart-cards*, como também já foi referido anteriormente, também possuem um sistema de ficheiros próprio, formado por um conjunto de ficheiros identificados por um número ou nome. No entanto, a forma como estes ficheiros existem neste sistema de ficheiros não é igual à forma com que trabalhamos, por exemplo, em sistemas de ficheiros casuais para sistemas UNIX ou UNIX-*like*. No caso de *smart-cards* temos um sistema de ficheiros muito próprio que é formado por três tipos de ficheiros: o **master file** (MF) que é o ficheiro-raiz de todo o sistema (que possui um ID `0x3F00`), os **elementary files** (EF) que são ficheiros de dados comuns com tamanho fixo e definido aquando da sua criação e os **dedicated files** (DF) que, de forma semelhante a diretórios, são contentores de ficheiros do tipo EF ou DF.

master file

elementary files

dedicated files

Este sistema de ficheiros também pode possuir algum controlo no seu acesso, sendo totalmente protegido (aqui os APDUs com pedidos de acesso a ficheiros deverão vir com MACs já calculados com uma chave partilhada entre o cartão e uma aplicação *off-card*) ou então com acesso autenticado numa aplicação exterior (aqui os APDUs com pedidos de acesso a ficheiros apenas serão permitidos de serem executados se o cartão já verificou a existência de uma chave partilhada com uma aplicação *off-card*).

Os *smart-cards*, portanto, permitem a execução de um largo conjunto de tarefas como a aplicação de cifras simétricas e assimétricas, a geração de chaves, a gestão de chaves (importação e exportação), a assinatura de documentos (ambas geração e verificação), a criação de *hashes* por vias de funções de síntese e a gestão de certificados de chave pública (mais uma vez ambas geração e verificação).

Para a integração entre as funcionalidades disponíveis para uso nos cartões e as aplicações de alto-nível que as pretendem usar existem ferramentas que podem facilitar os vários mecanismos através de interfaces próprias para as várias aplicações pretendidas: a estas ferramentas damos o nome de **middleware**, sendo mais precicsamente bibliotecas que fazem a ponte entre os cartões e as aplicações que pretendem usar as funcionalidades destes. Algumas das aproximações mais comuns são o PKCS #11 e o PKCS #15, para a definição do *Cryptographic Token Interface Standard* (uma norma usualmente denominada de **Cryptoki**) ou o CAPI CSP (definido para os sistemas operativos da família Microsoft Windows, sendo o *CryptoAPI Cryptographic Service Provider*).

middleware

Cryptoki

4. Protocolos de Autenticação

Até agora temos vindo a abordar detalhes de mensagens que nos chegam de determinados utilizadores ou diferentes entidades, mas nunca nos interrogámos sobre a validade de tais partes. Mas este cenário não se fica apenas pela identidade de um remetente, uma vez que, para entrar em alguns sistemas que não podem ser acedidos por terceiras partes desconhecidas, há igualmente a necessidade de saber, inequivocamente, identificar uma pessoa. Neste processo estamos a dar nome a um processo denominado de **autenticação**, que procura uma prova que uma entidade possui atributos, tal como o afirma ter.

autenticação

Tipos de autenticação e requisitos

Autenticar uma entidade ou ação pode ser uma atividade difícil, mas geralmente existem vários níveis a que esta ação poderá ser tomada.

Num primeiro nível de aplicação podemos considerar autenticar uma entidade através de alguma informação ou dado que temos da mesma. Este é o exemplo mais conhecido de todos, porque é o mais simples e rápido de pedir numa interação entre duas partes. Para isto as duas partes terão de ter algum dado com que possam confirmar as suas identidades, como um segredo ou palavra-passe. Por exemplo, a operação do Dia D (dia do desembarque na Normandia), da Segunda Guerra Mundial, tinha um conjunto de códigos transmitidos pela rádio britânica que serviam para que a resistência francesa pudesse saber, antecipadamente, quando é que a operação código Overlord iria começar oficialmente, para que pudessem iniciar alguns processos de sabotagem, principalmente das linhas férreas francesas neste departamento. Assim, havia um acordo em que, se a rádio de Londres emitisse a **palavra-passe** "*Les sanglots longs // des violons // de l'automne*" — primeiros versos do poema *Chanson d'Automne* de Paul Verlaine, então a resistência francesa autenticaria a mensagem como pertencente ao seu projeto e esperava pelo começo do Dia D dentro de duas semanas. Da mesma forma, se os seguintes versos fossem lidos "*Blessent mon cœur // D'une langueur // monotone*", então o Dia D começará dentro de 48 horas.

palavra-passe

Também podemos fazer uma autenticação com algo que uma entidade possui, como um objeto ou um *token* que só este poderá ter. Este exemplo é geralmente habitual em formulários HTML, o que permite que um preenchimento não seja misturado com um segundo mal-intencionado, uma vez que cada preenchimento é acompanhado de um *token* (um identificador único), que permite a identificação de uma entrada dada por um utilizador.

Por fim, podemos ter uma autenticação com algo que faz parte da própria definição da entidade em causa. Sendo um cenário cada vez mais habitual (como já podemos ver em telemóveis, computadores e outros dispositivos), aqui podemos usar a **biometria** para autenticar uma pessoa. A biometria é a métrica de objetos biológicos como as nossas mãos (através das impressões digitais), a nossa locomoção (através da análise da forma como caminhamos), os nossos olhos (identificação por retina), a nossa cara (identificação facial), a nossa voz, entre outros...

biometria

Embora distintos uns de outros, os vários tipos de autenticação poderão ser utilizados em simultâneo, no que se chama de **autenticação multi-fator**. Note-se também que a autenticação, como já foi implicitamente indicado através de exemplos vários, não terá de ter como seu objeto uma pessoa, mas antes uma qualquer entidade, como uma ação, um servidor ou outra máquina, uma empresa, uma rede, ...

autenticação multi-fator

No entanto, não devemos confundir autenticação com **autorização**. Em Arquitetura de Redes (a3s2) estudámos uma abordagem de aceitação de novas máquinas numa rede, sob um conjunto de regras, denominada de arquiteturas AAA, onde pudemos verificar que a autenticação vem reforçar o domínio da autorização, sendo que a última permite (ou não) um contacto inicial com a primeira entidade que puder avaliar a autenticidade da segunda.

autorização

Para que uma entidade possa cumprir os requisitos de conseguir autenticar ou ser autenticada perante uma segunda entidade é importante que seja confiável (tenha uma noção se é capaz de fornecer dados fáceis para autenticação ou de os pedir), seja sigilosa (não passe os seus dados — ou de quem quer ser autenticado — de autenticação a mais ninguém), seja robusta (não permita ataques de outros, seja de que forma for), seja simples (havendo simplicidade na forma como um processo de autenticação é feito, então torna-se mais difícil criar caminhos alternativos para a mesma ação) e seja capaz de trabalhar com possíveis vulnerabilidades provocadas por pessoas (como más palavras-passe, entre outros mecanismos, uma vez que as pessoas têm a tendência de tentar facilitar todos os processos, muitas vezes sem que as consequências sejam bem medidas).

Processos de autenticação com interação direta

Os processos de autenticação poderão variar de uma tarefa muito simples para tarefas com uma complexidade algo elevada, mas que garante ao máximo a não-entrada de terceiros num determinado sistema crítico. Ao longo deste gradiente de complexidade de processos podemos caraterizar algumas das suas formas dentro de dois grandes grupos: autenticações cuja interação é feita de forma direta (através do fornecimento de credenciais e espera da sua verificação e validação) e autenticações cuja interação é feita através de um desafio lançado e espera pela respetiva resposta.

Tal como já foi referido anteriormente, o método mais vulgar de autenticação aplicado em vários tipos de aplicação é, fora de dúvidas, a aplicação de uma palavra-passe. Este mecanismo passa pela comparação de uma palavra-passe fornecida por quem se pretende autenticar com um dado já previamente armazenado. Este processo é bastante comum uma vez que é simples, mas possui um vasto conjunto de problemas: primeiro este processo permite que se criem listas com as palavras mais comuns (Figura 4.1) [25], sendo que, por serem criadas por humanos, estas são facilmente alvo de ataques de dicionário por serem conjuntos de palavras comuns ou facilmente padronizadas; o facto de enviarmos uma palavra-passe para a segunda entidade que confirma a nossa identidade não é bom, uma vez que uma terceira entidade poderá estar a auscultar o canal de comunicação, obtendo a nossa palavra-passe; a palavra-passe que está guardada na entidade que autentica deverá estar representada sobre a forma de uma síntese (produto de uma função de *hash*), mas não temos garantia de que terá essa forma, podendo ser alvo de ataque posterior.

índice	palavra-passe
1	123456
2	Password
3	12345678
4	qwerty
5	12345
6	123456789
7	letmein
8	1234567
9	football
10	iloveyou

figura 4.1
10 palavras-passe mais comuns (2017)

A outra forma de proceder a uma autenticação direta pode acontecer usando a biometria. Durante muito tempo estes mecanismos eram frequentemente usados por corpos militares ou vistos apenas na indústria da sétima arte. Estávamos em 1989 quando no filme *Regresso ao Futuro II* Robert Zemeckis retrata uma situação em que Jennifer, atordoada depois de Doc Brown a ter adormecido para a acalmar do choque de estar no futuro, é levada para sua casa pela polícia, onde abre a porta através da sua impressão digital lida por um dispositivo à entrada e onde as luzes se acendem depois do sinal de Jennifer "*lights on?*". Era ainda muito cedo para que estas tecnologias chegassem ao público-geral, mas o sonho do realizador tornou-se realidade anos depois. E com este exemplo mostramos não só as vantagens de ter autenticação por dados biométricos, mas também uma desvantagem — Jennifer não se autenticou em sua casa por vontade própria, mas antes pela vontade da polícia, mas com os dados biométricos da dona da casa.

Embora seja uma tarefa muito árdua em grande parte dos casos, violar uma autenticação biométrica é possível, principalmente porque nem todos os mecanismos relativos a biometria estão apurados o suficiente para corrigir falsos-positivos. Mais, depois das credenciais de uma pessoa serem roubadas uma vez, nunca mais se poderão criar novas, como impressões digitais.

E este é um dos grandes problemas das chaves que são usadas nos métodos diretos de autenticação — grande parte das palavras-passe que as pessoas usam nas suas contas de correio eletrónico ou redes sociais são as mesmas para sempre, o que nunca deve ser feito, sendo que, sem sabermos, as nossas credenciais podem ter sido roubadas e ficamos com as nossas contas, igualmente para sempre, abertas aos terceiros que possuem as nossas informações. Por esta mesma razão, ao longo da história foram criadas formas diferentes de abordar este problema, um dos quais foram as **one-time passwords**, um mecanismo que é suportado por um largo número de aplicações e que permite que a palavra-passe seja única por utilização, por dia, por hora ou até mesmo por minuto. Um exemplo mais vulgar desta aplicação é o *home-banking* onde, para operações mais delicadas como pagamentos de serviços ou transferências bancárias, no fim das mesmas, para se confirmar uma transação, é necessário que se preencham três espaços com base numa matriz que o cliente deverá ter consigo na forma de papel ou outro tipo de formato.

one-time passwords

A grande vantagem das *one-time passwords* está assim na sua aleatoriedade de acordo com a regularidade temporal que no dispositivo portador está configurada, permitindo assim que uma chave se torne inútil depois de roubada por terceiros. No entanto, de nada servirá se uma terceira entidade nos ficar com o dispositivo que nos fornece a chave, pelo que com a sua posse as nossas credenciais estão em risco. Contudo, há sempre a possibilidade de pedirmos um novo dispositivo gerador.

Um outro exemplo de aplicação de *one-time passwords* pode ser visto com o dispositivo RSA SecurID que é uma peça variável de acordo com o negócio onde se aplica, mas que possui um pequeno ecrã e, com a introdução de um PIN sempre igual e por pessoa portadora (ou sem PIN), é mostrado de 30 em 30 segundos uma palavra-passe para que seja introduzida num determinado portal juntamente com um identificador do portador (utilizador) único. Neste caso em concreto, estamos a falar de um dispositivo que possui uma chave de 64 bits guardada em si, tendo também um relógio sincronizado no fabrico com uma fonte igual à de quem autentica. Este dispositivo depois teria de ser renovado quando perdesse energia (a bateria interna terminasse) ou quando fosse colocado em risco.

Processos de autenticação por desafio-resposta

Na Segunda Guerra Mundial, mais em particular com o Dia D, as tropas dos aliados possuíam um esquema para validar as várias mensagens que eram ditadas entre os vários soldados, uma vez que não se conheciam todos uns aos outros. Para validar, então sabiam que em resposta a uma palavra "*flash*" deveriam responder "*thunder*", quase como se de um desafio se tratasse. Na verdade até era mesmo isso — um **desafio** — e o que se esperava era uma **resposta** correta. Não podendo manter-se o mesmo desafio durante muito tempo, para não perder o significado, as tropas dos aliados tinham já um conjunto de desafios-resposta já combinados, alterados de 3 em 3 dias, por exemplo, "*thirsty*" como desafio e "*victory*" como resposta (em Dia D + 1 dia até Dia D + 3 dias, inclusive). Este é o primeiro momento recordado em que foi feita autenticação das mensagens por vias de um processo de **desafio-resposta** (em inglês *challenge-response*).

desafio

resposta

desafio-resposta

Em termos básicos, o que acontece aqui é que quem autentica fornece um desafio, este, que deverá ser transformado com as credenciais de quem quer ser autenticado numa resposta que envia de volta para o autenticador, que verifica o resultado. Repare-se que neste processo nunca há uma troca de credenciais ao longo de um canal de comunicação entre as duas entidades envolvidas no processo de autenticação, sendo uma vantagem deste sistema. Contudo, há que registar como desvantagens o facto de precisarmos de um *hardware* ou *software* especiais para responder ao desafio e o facto de que respondendo várias vezes a desafios podem-se revelar algumas das nossas credenciais (como chaves ou palavras-passe).

Um exemplo de aplicação do processo de autenticação por desafio-resposta está presente no uso de *smart-cards* como o Cartão de Cidadão português ou dispositivos semelhantes ao longo de todo o mundo, como o *Documento Nacional de Identidad* em Espanha. Aqui as nossas credenciais são mesmo o cartão (onde está preservada a nossa chave priva-

da) e um código PIN de 4 dígitos que serve para verificar se estamos qualificados a utilizar os dados presentes no dispositivo. Para que este processo possa correr bem, a entidade que autenticará precisará de saber previamente a chave pública correspondente (a nossa chave pública) ou outro identificador pessoal, que poderá ser relativo a uma chave pública acessível através de um certificado verificável.

Havendo tais credenciais a autenticação por vias de um desafio-resposta com um Cartão de Cidadão poderá ser executado através de um protocolo com base numa assinatura ou com base numa cifra. Para que o processo possa ser executado com base numa cifra é necessário que a decifra por chave privada seja disponibilizada a quem autentica, no entanto, no primeiro processo — com base em assinaturas — o autenticador gera um desafio aleatório (ou um valor que nunca fora antes usado) e envia-o para o cliente que pretende autenticação que, desbloqueando a sua capacidade de cifra através de um código PIN cifra o conteúdo do desafio com a sua chave privada, enviando o texto cifrado de volta para o autenticador esperando que este, decifrando com a chave pública de quem pretende ser autenticado, obtenha uma resposta correta para o desafio. Este processo pode ser acompanhado na Figura 4.2.

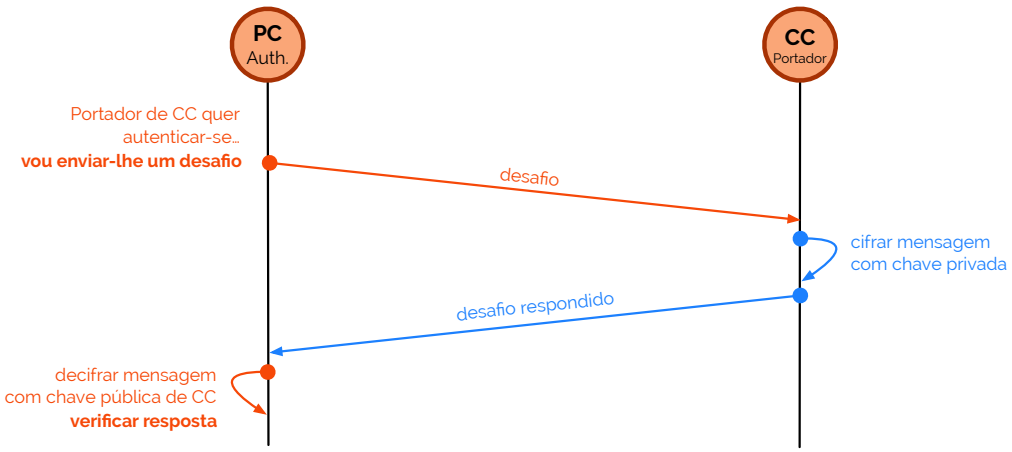

figura 4.2
desafio-resposta

Uma outra forma de aplicar um processo de desafio-resposta a uma tarefa de autenticação é tendo uma palavra-passe já memorizada. Aqui a entidade que autentica terá de ter conhecimento de todas as palavras-passe registadas ou de um elemento gerador das mesmas (que seja capaz de desginar as várias transformações que são feitas de forma a conseguir, inequivocamente, obter uma palavra-passe por cada iteração) — esta última abordagem é a preferida, sendo que sistemas já a implementam inclusive, juntamente com uma variação denominada de *salt* (elemento que se adiciona a um conjunto de dados para fornecer aleatoriedade).

De facto, este tipo de procedimento de desafio-resposta com palavras-passe memorizadas já e usado num largo número de sistemas através de protocolos que, alguns deles, já abordámos em disciplinas como Arquitetura de Redes (a3s2), como o PAP, o CHAP, o MS-CHAP versão 1 e 2 ou o S/Key. A forma como estes algoritmos trabalham, por alto, é a seguinte: a entidade que está encarregue de autenticar uma segunda lança um desafio aleatório e envia-o para quem pretende ser autenticado; a pessoa que recebe o desafio terá agora que calcular uma transformação do desafio juntamente com a sua palavra-passe (usando uma síntese conjunta do desafio e da palavra-passe (4.1) ou uma cifra do desafio com a palavra-passe (4.2)); terminando o processo, a primeira entidade — a que autentica — deverá fazer precisamente o mesmo que na fase anterior pela pessoa, verificando de seguida se o procedimento correspondeu ao mesmo resultado ou não, havendo autenticação positiva em caso igualmente positivo.

$$\text{resposta} = \text{síntese}(\text{desafio}, \text{palavra-passe}) \tag{4.1}$$

$$\text{resposta} = E_{\text{palavra-passe}}(\text{desafio}) \tag{4.2}$$

Comecemos então por analisar protocolos como o PAP e o CHAP, ambos usados em redes PPP (*Point-to-Point Protocol*) para efetuar uma **autenticação unidirecional**, isto é, onde a entidade que autentica não é verificada. Desenvolvida em 1992, esta tecnologia era basicamente aplicada a antigas ligações *dial-up* e mais tarde aplicadas em VPNs do tipo PPTP, como já estudámos em Arquiteturas de Redes (a3s2).

autenticação unidirecional

Embora tenham sido ambos protocolos implementados para o seu uso em PPP, os dois protocolos são distintos um do outro, sendo que o **PAP** (acrónimo para PPP Authentication Protocol) funciona com a verificação de uma simples palavra-passe ou identificador que, em texto em claro, é enviado do utilizador para o autenticador (Figura 4.3).

PAP

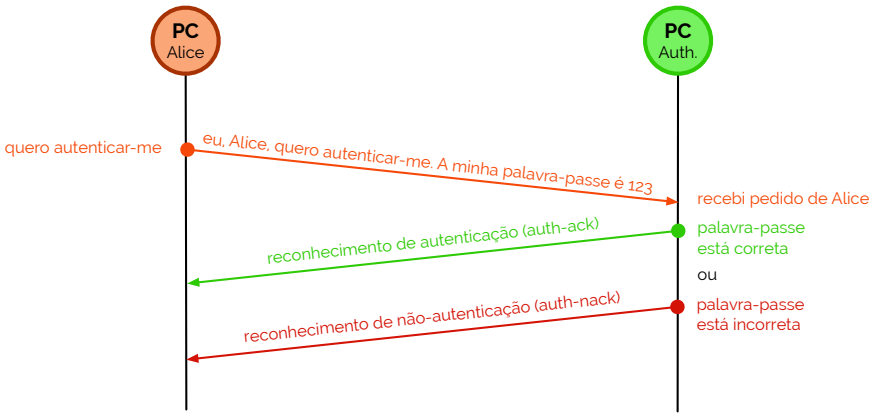

figura 4.3
PAP

Já o **CHAP** (acrónimo para Challenge-response Authentication Protocol) usa uma implementação do processo de desafio-resposta para efetuar um procedimento de autenticação unidirecional. Aqui, como podemos verificar na representação da Figura 4.4, a entidade autenticadora envia um desafio juntamente com um identificador do processo de autenticação para o utilizador que pretende autenticar-se. Este utilizador, recebendo esta mensagem, apenas terá de produzir uma síntese MD5 do identificador, da sua palavra-passe e do desafio e enviá-lo juntamente com o identificador do processo de autenticação e da sua identificação. Depois de enviada esta informação, a entidade autenticadora terá de verificar o resultado do desafio e enviar, por conseguinte, uma confirmação ou negação de autenticação para quem se sujeitou ao processo. Note-se que neste processo o autenticador poderá pedir uma reautenticação sempre que quiser e possui uma grande vulnerabilidade por obrigar o servidor de autenticação a manter senhas em claro (e não as suas sínteses), o que seria mais seguro caso o seu registo de utilizadores seja comprometido.

CHAP

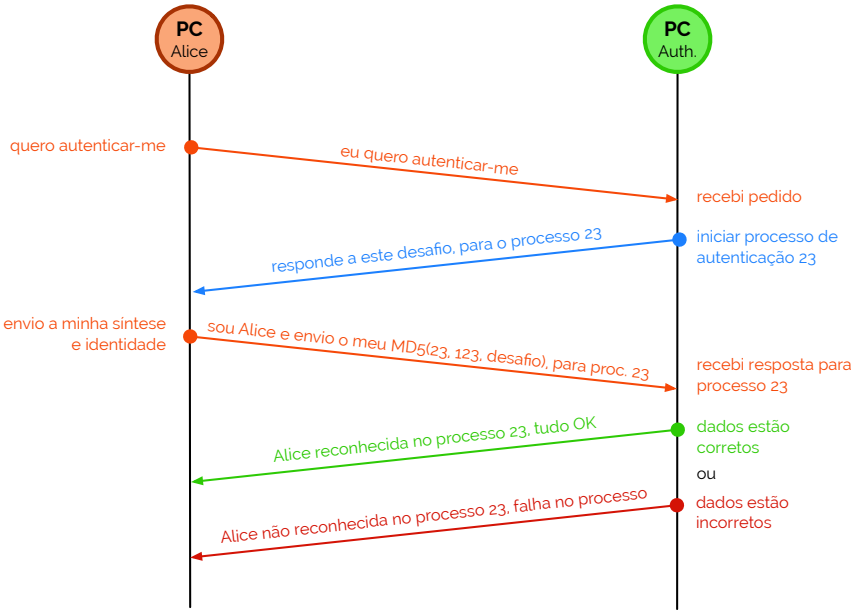

figura 4.4
CHAP

Uma melhoria do CHAP foi introduzida pela Microsoft com a sua versão **MS-CHAP**. No entanto, a primeira versão deste mesmo protocolo possuía vulnerabilidades diretamente relacionadas com compatibilidades do sistema operativo Microsoft Windows NT com o antigo LAN Manager, onde ambos os sistemas possuíam diferentes funções de síntese. Por haver duas funções de síntese, isto significa que as várias senhas tinham duas sínteses diferentes, sendo que a síntese proveniente da função do LAN Manager, por si, já era bastante sensível a ataques de dicionário, reduzindo o universo de tentativas para acertar na senha correta [2].

MS-CHAP

Numa segunda versão do MS-CHAP, denominada vulgarmente por **MS-CHAPv2**, as melhorias já foram bastantes, uma vez que deixámos de ter autenticação unidirecional, para agora ambas as entidades se verificarem uma perante a outra, usando uma senha partilhada e desafios (e consequentes respostas) trocadas de forma bilateral.

MS-CHAPv2

Um outro protocolo criado foi o **S/Key**, designado em ambos RFC 2289 e RFC 1998. Aqui, as únicas credenciais necessárias são uma palavra-passe, sendo que a entidade autenticadora necessita de ter conhecimento sobre a última *one-time password* (OTP) usada, qual foi o seu índice (dando importância à ordem entre os vários OTPs consecutivos) e um valor *seed* por cada OTP de uma pessoa.

S/Key
↖ RFC 2289
↖ RFC 1998

Em termos de procedimento, o que acontece é que a entidade autenticadora define um *seed* aleatório, sendo que a pessoa deverá gerar um OTP inicial como o resultado da enésima síntese MD4 aplicada à palavra-passe em causa, juntamente com a *seed*, como podemos ver em (4.3).

$$\text{OTP}_n = \text{MD4}^n(\text{seed}, \text{palavra-passe}) \tag{4.3}$$

Note-se que, embora em (4.3) estejamos a indicar o resultado da enésima aplicação de síntese da função MD4, algumas versões do S/Key podem usar funções como a SHA-1 ou a MD5.

Dado isto, o autenticador deverá então preservar o valor de *seed*, de n e de OTP_n como sendo as várias credenciais de autenticação, para depois testá-las da forma representada pela Figura 4.5 [26].

figura 4.5
S/Key

O facto da entidade autenticadora enviar ambos índice de OTP e *seed* da pessoa serve como desafio para o processo, sendo que a pessoa deverá gerar (*índices de OTP* − 1) e selecionar o último valor como resultado final: $resultado = OTP_{índice-1}$. De seguida o autenticador calcula o resultado sob forma de uma síntese e compara o resultado com o valor guardado de $\text{OTP}_{índice}$, sendo que se ambos coincidirem a autenticação é tomada como bem sucedida, guardando o índice recentemente usado e o seu valor de OTP.

Já em Arquitetura de Redes Avançadas (a4s1) estudámos o sistema **GSM**, de redes móveis. Este sistema possui, na sua arquitetura, módulos próprios para o registo de utilizadores junto de um MSC (*Mobile Switching Center*), através de uma unidade denominada de HLR (*Home Location Register*), que é uma base de dados para guardar dados permanentes e semi-permanentes sobre uma determinada área de utilizadores em MSs (*Mobile Stations*). No mesmo local também existe uma segunda entidade, denominada abreviadamente por AuC (sigla para *Authentication Center*) que é uma base de dados com as chaves de autenticação dos sobrescritores e respetivos algoritmos para calcular os parâmetros a serem levados para o HLR.

GSM

Como já podemos prever, uma vez que estamos a falar de chaves partilhadas, o GSM usa um processo de autenticação do tipo de desafio-resposta, mas utilizando uma chave partilhada ao invés de uma palavra-passe. Este processo acaba por ser mais robusto conrta ataques de dicionário, embora requerendo capacidade extra para preservar as chaves.

Embora inicialmente o esquema de arquitetura deste sistema não tenha sido divulgado ao público, o GSM possui todo um esquema de autenticação que incorpora unidades de relevo desde a estação móvel (MS) até às bases de dados estacionárias HLR e AuC. Entre o HLR e o MS uma chave partilhada é preservada, estando a mesma (K_i) preservada no cartão SIM no cliente, acessível depois do desbloqueio através da introdução de um código PIN. Também no cartão SIM estão presentes dois dos algoritmos necessários para este processo, entre os quais o A3 (algoritmo para autenticação) e o A8 (algoritmo para a geração de uma chave de sessão). Mas em todo o sistema ainda há mais um algoritmo necessário, denominado de A5 (algoritmo para cifrar as comunicações).

O procedimento, então, ocorre da seguinte forma: a estação móvel regista-se na estação-base (BS – *Base Station*) que, ligada ao circuito comutador GSM (MSC), faz com que este envie um pedido dos triplos com as informações do utilizador em causa à unidade de base de dados HLR e AuC; feito isto a HLR responde com os triplos pedidos que são enviados para a estação móvel, onde são realizados os algoritmos A3 e A8, cuja resposta SRES é enviado para o MSC, por vias da BS, para testar a validade da autenticação. As funções que são usadas pelos algoritmos A3/A8, uma vez que estaõ preservados no cartão SIM, podem ser à escolha do operador do cartão em causa, embora haja recomendações do consórcio GSM.

Voltando agora novamente para as redes locais, também temos o problema de autenticação nos nossos computadores ou em redes em que pretendemos comunicar com outras máquinas que não a nossa. Existem duas formas de autenticação em terminais computacionais, uma delas sendo feita através de nomes ou endereços de rede (soluções não muito fiáveis, uma vez que não há provas criptográficas de que estamos a referir as entidades que dizem ser), outra sendo feita através de chaves criptográficas, quer estejamos a falar de chaves simétricas (segredos partilhados entre máquinas) ou pares de chaves assimétricas por máquina (ou certificados de chave pública com qualquer parceiro de comunicação).

Para resolver esta questão de comunicações seguras sobre canais de comunicação estabelecidos em TCP/IP surgiu o **TLS** (sigla para *Transport Layer Security*), definido no RFC 2246 que, criado sobre o SSLv3 (sigla de *Secure Sockets Layer*) gere sessões seguras de comunicação entre aplicações sobre ligações TCP/IP. Os mecanismos de segurança que estão intrínsecos a este tipo de ligação garantem a integridade e a confidencialidade na comunicação através da distribuição de chaves e a autenticação de *endpoints* de comunicação, como servidores ou terminais comuns, ambos com pares de chaves assimétricas e certificados de chave pública, como veremos de seguida.

TLS

‹ **RFC 2246**

Na Figura 4.6 podemos verificar como é que se processa uma ligação por TLS, entre a Alice e um servidor SSL. Para começar a Alice deverá enviar uma mensagem a requerer uam ligação segura para com o servidor, enviando assim um **client hello**, onde detalha as suas capacidades de ligação como a versão de SSL e as cifras que suporta. O servidor terá de aceitar temporariamente a ligação para a troca inicial de informações, sendo que a primeira mensagem é um **server hello** que, analogamente à do cliente, transporta informações sobre a versão do SSL escolhida e o conjunto de cifras (note-se que aqui o servidor não propõe uma versão e um conjunto de cifras ao cliente, mas antes toma a decisão de acordo com as informações que já recebera do cliente). Sendo três passos opcionais, o servidor de seguida poderá enviar o seu certificado de chave pública, a sua chave pública (se não tiver ceritificado) e um pedido de ceritificado de chave pública do cliente (autorizado por uma entidade reguladora — CA — específica). No final deste processo o servidor envia uma mensagem de **server hello done** para concluir o primeiro processo de troca de mensagens.

client hello

server hello

server hello done

Do lado do cliente este só terá de responder às questões impostas pelo servidor se ainda pretender seguir a criação de uma ligação segura com ele. Assim sendo primeiramente deverá enviar (caso se aplique) o seu certificado de chave pública verificado pelo CA especificado pelo servidor. De seguida deverá enviar os seus dados cifrados com a chave pública do servidor e, caso tenha enviado o seu certificado, mais dados cifrados com a sua chave privada para que o servidor possa ter a certeza que o certificado é válido. Para terminar esta fase de troca de mensagens o cliente envia ainda um **change cipher spec** para

change cipher spec

indicar que a próxima mensagem a ser enviada será cifrada e servirá para terminar a fase inicial de mensagens trocadas.

Enviadas estas mensagens o servidor deverá igualmente enviar um *change cipher spec* passando a trocar mensagens até que a sessão entre ambos seja terminada, de forma cifrada.

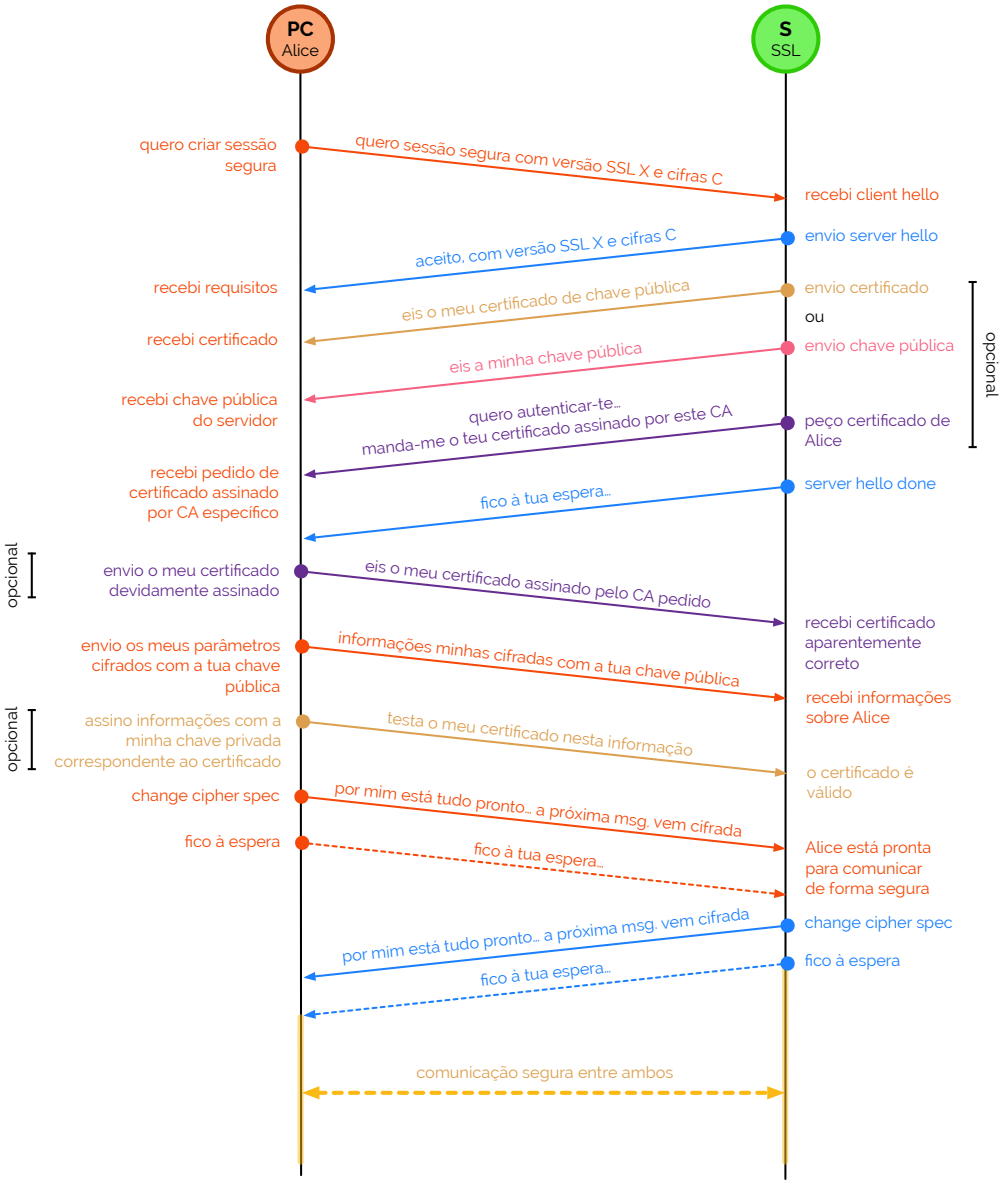

figura 4.6
ligação por TLS

O protocolo TLS embora cumpra os requisitos estabelecer uma comunicação entre duas máquinas (sejam qual forem as suas funções entre servidores e terminais) nem sempre cumpre as necessidades dos utilizadores, uma vez que foi feito para ser usado em aplicações que usam HTTP, entre outras. No entanto, se quisermos abrir uma sessão remota numa máquina através de uma consola como é que o podemos fazer?

Uma das aplicações mais conhecidas para abrir uma sessão numa máquina de forma remota é através da aplicação **SSH**, que é uma alternativa ao Telnet, uma aplicação que permitia, através de um protocolo feito para a mesma, aceder a uma máquina sem qualquer tipo de segurança, por defeito. A aplicação SSH, no entanto, não existe somente para criar uma sessão útil noutra máquina remota por vias de uma consola, mas também serve para criar ligações úteis para outras aplicações de transferência de dados de forma segura e criação de túneis seguros em sistemas UNIX.

SSH

Para que o SSH possa funcionar tem de haver uma troca de chaves assimétricas no início (não há certificados de chave pública). Depois desta tarefa ser concretizada, o uti-

lizador poderá efetuar a sua autenticação de duas formas: através da entrega de um nome de utilizador e de uma palavra-passe (este é o formato por defeito) ou através da entrega de um nome e do uso da chave privada, sendo que a nossa chave pública já deverá estar no servidor para teste.

Meta-protocolos de autenticação

Existem outros protocolos de autenticação que são colocados em prática dependendo da sua aplicação. Na disciplina de Arquitetura de Redes (a3s2) tivemos a oportunidade de estudar um conjunto de possíveis implementações de segurança em redes denominadas de **arquiteturas AAA**, que não eram nada mais do que arquiteturas de redes com separação de responsabilidades entre autorizações, autenticações e fatores contabilísticos. Neste tipo de redes, o que acontece é que, à entrada de um novo nó, há um pedido de acesso à rede que apenas é disponibilizado depois de tal nó confrontar um conjunto de informação perante um dispositivo encarregue pela autenticação. Aqui costumam-se usar aplicações como a RADIUS ou a 802.1X para se tratar dos processos de autenticação.

arquiteturas AAA

Contudo não é só nas arquiteturas AAA que podemos ter protocolos de autenticação diferentes. Por exemplo, também no âmbito da disciplina de Arquiteturas de Redes (a3s2) estudámos a criação de túneis seguros através do **IPSec**, que utiliza protocolos de gestão de redes como o **ISAKMP** (acrónimo para *Internet Security Association and Key Management Protocol*), ou a aplicação de critérios de segurança em redes do tipo 802.11 (WiFi), como o **EAP** (*Extensible Authentication Protocol*).

IPSec
ISAKMP
EAP

Este tipo de protocolos, no fundo, são formas de autenticação que, por si, encapsulam outros protocolos mais específicos de autenticação dentro. A este tipo de protocolos damos o nome de **meta-protocolos de autenticação**.

meta-protocolos de autenticação

Um último exemplo destes metaprotocolos de autenticação está na aplicação de **single sign-on** (SSO) como podemos ver em algumas instituições em que as unidades de autenticação se encontram centralizadas sobre a forma de um conjunto de serviços federados. Aqui, todas as identidades dos clientes, depois de uma autenticação, ficam a par de todos os serviços federados, sendo que os vários atributos dados a cada um destes poderá variar. Neste contexto o autenticador é tido como um **identity provider**, mais vulgarmente identificado pela sua abreviatura **IdP**.

single sign-on
identity provider
IdP

Pluggable Authentication Modules (PAM)

Os sistemas operativos tais como os conhecemos permitem que os seus serviços usem *middlewares* para autenticar os seus utilizadores. Para isso, os serviços não necessitam de lidar com os processos de autenticação de forma direta, mas antes através de uma camada de abstração que esconde a forma como a autenticação é realizada. Nos sistemas UNIX e UNIX-*like* tal é realizado através de uma infraestrutura denominada de **PAM** (acrónimo de *Pluggable Authentication Module*)[12].

PAM

Portanto, esta infraestrutura permite que se crie uma barreira entre a necessidade de uma aplicação realizar a autenticação de um utilizador e o modo como essa autenticação é efetivamente realizada. Para que isto possa acontecer foi criada uma biblioteca que cria um nível de abstração que usa um conjunto de ficheiros de configuração para orquestrar o processo de autenticação usando módulos com funcionalidades específicas, cuja arquitetura podemos ver na Figura 4.7. [2] Note-se que estes ficheiros de configuração poderão ser alterados a qualquer momento pelo administrador dos serviços.

Esta arquitetura para a biblioteca PAM possui quatro tópicos que são relevantes para a compreensão da importância do seu desenvolvimento e utilização em futuras implementações de serviços. O primeiro tópico a abordar será a **aquisição de credenciais** (ou serviços de autenticação), onde se enfatiza que o PAM lida com as várias atividades de um uti-

aquisição de credenciais

[12] Em sistemas operativos da família Microsoft Windows, o mesmo é realizado através da infraestrutura GINA (acrónimo de Graphical Identification and Authentication).

lizador perante um serviço. De seguida, a biblioteca PAM também aplica mecanismos de **gestão de contas**, sendo que permite o início de uma sessão depois da sua autenticação bem-sucedida (tendo em conta uma determinada hora, um número de utilizadores a usarem um sistema num determinado instante de tempo, entre outros...). Terceiro, a **gestão de sessões** lida com a gestão das sessões que foram iniciadas após uma operação de autenticação se ter dado como bem-sucedida e após uma autorização de uso do sistema, como a criação de uma sessão seguida do registo da hora de criação da mesma e o início de processos associados com a mesma, entre outros... Por último, a **gestão de palavras-passe**, permitindo assim a atualização de credenciais de acesso usadas pelos utilizadores para se autenticarem. [27]

gestão de contas

gestão de sessões

gestão de palavras-passe

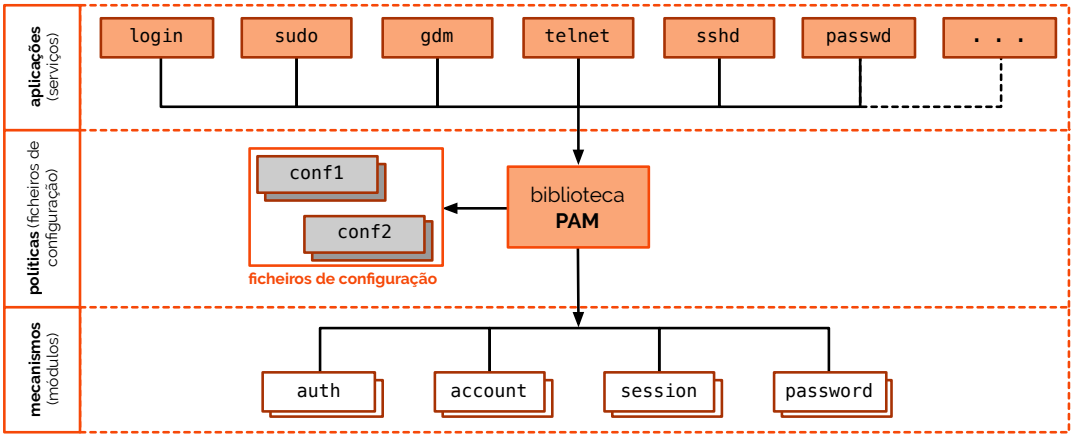

figura 4.7
arquitetura PAM

Como podemos ver pela Figura 4.7 existe um conjunto de ficheiros de configuração, geralmente um por aplicação-cliente PAM localizados no diretório `/etc/pam.d`, como, por exemplo, os ficheiros `/etc/pam.d/ftp` ou `/etc/pam.d/ssh`, podendo também ter fiicheiros partilhados como em `/etc/pam.d/common-auth`. Nestes ficheiros deverão estar, então, regras indicativas de que ações deverão ser aplicadas consoante a aplicação, como que bibliotecas dinâmicas é que se devem carregar (definindo assim que mecanismos usar), que parâmetros entregar a cada biblioteca e definições sobre de que modos é que podemos dizer que uma determinada ação foi completada. Dada esta arquitetura, cada módulo deverá usar um conjunto particular de recursos, como informações em ficheiros locais (como o `/etc/passwd` para saber informações acerca de utilizadores, ou `/etc/groups` para retirar informações sobre grupos, ...), ou informações em locais remotos, através de sistemas distribuídos como o NIS, LDAP ou Kerberos, entre outros...

Vejamos assim um exemplo de ficheiro de configuração para uma aplicação, no nosso caso, para o `sudo`, como é usada no sistema operativo macOS (Código 4.1).

código 4.1

```
$ cat /etc/pam.d/sudo
# sudo: auth account password session
auth       sufficient     pam_smartcard.so
auth       required       pam_opendirectory.so
account    required       pam_permit.so
password   required       pam_deny.so
session    required       pam_permit.so
```

Olhando para o Código 4.1 podemos primeiro verificar que a primeira coluna corresponde aos vários módulos indicados na Figura 4.7 e que enumerámos no início da explicação acerca do PAM. Podemos então saber que as primeiras duas linhas correspondem à aquisição de credenciais, a terceira linha corresponde a uma ação de gestão de contas, a quarta linha a uma gestão de palavras-passe e a quinta a uma gestão de sessões.

Mas, na segunda coluna podemos ver dois de quatro resultados possíveis: `required` e `sufficient` — o que é que significam? Ora, o sucesso das tarefas impostas dentro dos ficheiros indicados na terceira coluna necessita de ser controlado, uma vez que para atingir um determinado objetivo, por vezes, não é necessário que se façam todas as ações. Para

isso existem quatro indicadores de controlo de sucesso que permitem criar ligeiras modificações no fluxo de leitura destes ficheiros de configuração PAM. O primeiro modificador é o **requisite**, que significa que se o módulo falhar, então o resultado será retornado de imediato. Por outro lado, haver um modificador **optional** permite que o resultado seja ignorado, com a exceção de quando é o único módulo descrito na ação. Mas ainda temos o **required**, que, se o módulo falhar, o resultado será tomado, mas os seguintes módulos são chamados na mesma. Por fim, o modificador de controlo de sucesso **sufficient** permite que, se o módulo for executado com sucesso, então os módulos *required* anteriores serão tomados como com sucesso, mas caso falhe, o seu resultado será ignorado.

requisite

optional

required

sufficient

Voltando ao nosso exemplo, então primeiramente, para que o `sudo` possa ocorrer num sistema operativo macOS este irá executar o módulo `pam_smartcard.so`, seguido do `pam_opendirectory.so`. Este módulo `pam_opendirectory.so` permite que o sistema liste todos os seus utilizadores para depois poder questionar sobre que conta é que se pretende abrir, com o módulo seguinte, o `pam_permit.so`, permitindo a sua abertura ou, em caso de má inserção de palavra-passe, cortar o acesso com `pam_deny.so`. Feita a gestão de palavra-passe e de contas, então estando na possibilidade de abrir a sessão, o PAM tem uma ação especificada para o fazer, com o módulo `pam_permit.so`.

Em macOS a biblioteca PAM tem uma forma bastante simples, mas noutros sistemas como o Linux, os ficheiros já existentes possuem uma complexidade algo maior, com inclusões no próprio ficheiro de configuração e inclusive ficheiros de configuração partilhados.

Como pudemos verificar, PAM cria um nível de abstração que permite uma simplicidade na manipulação de mecanismos de segurança através do empilhamento destes e minimizando a perceção dos utilizadores em relação aos vários detalhes de implementação. Isto também permite que os próprios desenvolvedores de código possam prestar mais atenção e possam implementar múltiplos fatores de autenticação com diferentes chaves/segredos numa arquitetura puramente modular, com o carregamento de bibliotecas dinâmicas e com a manipulação de várias ações para além da própria autenticação como é o caso da gestão de palavras-passe, contas e sessões.

5. Módulos de Controlo de Acesso

Ao longo deste documento temos vindo a descrever métodos para assegurar que um determinado sistema não é facilmente alvo de ataques de segurança. Começámos por analisar como aplicar detalhes de segurança a um canal de comunicação e depois fomos verificar como autenticar as várias entidades participantes nessas comunicações. Agora, num grau ainda mais abstrato em relação ao detalhe do sistema em que pretendemos aplicar tais critérios, como é que controlamos quem é que lhe acede?

Surge assim a definição de **controlo de acesso**, como um conjunto de políticas e mecanismos que mediam o acesso de um determinado sujeito a um objeto. Por sujeito geralmente pretendemos indicar uma pessoa, no nosso caso habitualmente representada por programas em execução em seu nome. Já por objeto pretendemos referir locais onde determinadas ações poderão ser desempenhadas como ficheiros, tabelas, programas, objetos de memória, dispositivos físicos, *strings*, campos de dados, ligações de rede, ... Basicamente, podemos considerar um **sujeito** como algo que exibe atividade (como um processo, um computador ou uma rede) e um **objeto** como algo que é alvo de uma ação (como dados armazenados, tempo de CPU, memória, processos, computadores ou redes), sendo que ambos são entidades digitais.

controlo de acesso

sujeito

objeto

Princípio de menor privilégio

Em 1975, J. H. Saltzer e M. D. Schroeder explicitaram algumas regras essenciais para a proteção de informação em sistemas computacionais no *paper The protection of information in computer systems*, pelo IEEE. Dentro destas regras, um dos princípios redigidos reitera algo que já foi referido neste documento: o **princípio de menor privilégio**. Este princípio dita que se deve dar às pessoas e a cada entidade o menor número de privilégios

⊙ Jerome Howard Saltzer
⊙ Michael Schroeder

princípio de menor privilégio

necessários para que estes possam desempenhar as suas funções num sistema computacional. [28]

Pelo contrário, quantos mais privilégios forem atribuídos a uma entidade para desempenhar as suas tarefas, então mais a vulnerabilidades o sistema estará sujeito, isto porque poderá ser mais facilmente alvo de danos provocados por acidente ou erros de execução.

Tipos de controlo de acesso

O controlo de acesso a uma aplicação ou sistema poderá ser detalhado através de uma matriz de controlo de acesso, uma tabela onde se especificam todos os direitos de acesso dos sujeitos em relação aos objetos.

Tendo em conta esta tabela, que possui como colunas os vários objetos e, como linhas os vários sujeitos, podemos construir dois novos conceitos para o desígnio de níveis de controlo de acesso. Olhando para cada linha podemos criar mecanismos com base em **capacidades**, isto é, para cada objeto especificamos o nível de capacidade de um determinado sujeito. No entanto, se olharmos para cada coluna podemos visualizar outro tipo de atribuição de privilégios, através de **listas de acesso** (do inglês *access-lists*), que são lista de direitos de acesso para sujeitos específicos, podendo ser positivos ou negativos, mas sempre guardados junto dos objetos.

capacidades

listas de acesso

Dado isto, podemos ter dois tipos de controlo de acesso: um controlo obrigatório ou um controlo discricionário. O **controlo obrigatório** (do inglês *Mandatory Access Control*, vulgarmente denominado por **MAC**) é uma política estaticamente implementada por um **monitor** que não pode ser adaptada por um sujeito, contrariamente ao que se passa num **controlo discricionário** (do inglês *Discretionary Access Control*, vulgarmente abreviado por **DAC**), onde alguns sujeitos podem atualizar os seus direitos permitindo ou negando o acesso a outros para um determinado objeto (este procedimento é atribuído geralmente a donos de objetos e administradores de sistemas).

controlo obrigatório

monitor

controlo discricionário

DAC

Ainda não tendo distinguido vários tipos de utilizadores, vamos querer que alguns (como administradores de sistemas) possam ter privilégios significativos, enquanto que outros (como utilizadores regulares ou convidados) tenham privilégios menores. Contudo, em termos reais, implementado numa empresa, por exemplo, isto poderá tornar-se complicado quando um utilizador regular passa para um cargo de administração ou vice-versa, havendo a necessidade de troca de privilégios para estes utilizadores em concreto. Uma forma de contornar a dificuldade de implementação desta solução é aplicando-a aos papeis que estes utilizadores ocupam num determinado sistema. A este tipo de controlo de acesso damos o nome de **RBAC** (do inglês *Role-based Access Control*), sendo que o seu conceito não incorpora as decisões do tipo MAC ou DAC, mas antes decisões com base no papel de cada utilizador em todo o grupo de utilizadores de uma aplicação.

RBAC

A aplicação de conceitos como o de RBAC para controlar o acesso de utilizadores a uma determinada aplicação também é mais importante em cenários onde a complexidade de operações é elevada (não onde as operações são básicas como as de leitura, escrita ou execução), isto sendo que as operações poderão envolver vários objetos de nível mais baixo.

A **atribuição de papeis** do RBAC deverá ser cuidada e por essa mesma razão tem as suas regras definidas: todas as atividades de sujeitos num determinado sistema são conduzidas através do conceito de **transações**, cada uma destas aplicada a papeis específicos, sendo que todos os sujeitos ativos no sistema necessitam de ter um papel ativo. Note-se que um sujeito poderá executar uma transação se e só se foi selecionado para ou lhe foi atribuído um papel o qual poderá usar a transação. Estas transações também passam por um processo de autorização onde um sujeito pode executar uma transação se e só se esta é autorizada através das associações dos papeis deste e não há demais restrições que poderão ser aplicadas em vários sujeitos, papeis e permissões.

atribuição de papeis

transações

O próprio papel também possui uma autorização específica, onde um papel ativo de um sujeito deverá ser autorizado para o sujeito. Isto acontece também porque um papel é

uma coleção de permissões, que são garantidas aos sujeitos que, num determinado instante de tempo, reproduzem caraterísticas de um papel (sendo que um sujeito apenas poderá executar um papel de cada vez).

Os utilizadores, por si, por estarem organizados em grupos, também poderão receber permissões conjuntas, sendo que poderão pertencer a vários grupos em simultâneo.

contexto

A aplicação de restrições em termos de controlos de acesso também poderá ser aplicada tendo em conta o **contexto**. Neste caso o direito de acesso deverá depender de um contexto histórico, ou seja, dependendo do historial de um determinado utilizador poderá ser garantido, ou não, o seu acesso a um sistema — consideremos tal aplicação em cenários como o de uma *firewall* configurada de forma *stateful*. Este método, denominado de **CBAC** (sigla para *Context-based Access Control*) é uma aplicação de uma política chamada de política da **Muralha Chinesa** (uma política em que há uma barreira de informação erigida para previnir trocas ou comunicações que possam levar a conflitos de interesse por parte de vários utilizadores dentro de um mesmo grupo [29]).

CBAC

Muralha Chinesa

atributos

ABAC

Uma outra medida de controlo de acesso é aplicável com base em **atributos**, conforme o tipo **ABAC** (acrónimo de *Attribute-based Access Control*). Aqui, o controlo de acesso é feito de acordo com decisões com base em atributos relacionados com entidades de relevo. Um exemplo de aplicação deste conceito é a **arquitetura XACML** (sigla para *Extensible Access Control Markup Language*) que define um conjunto de quatro elementos, entre os quais: um **PAP** (*Policy Administrator Point*) como um gestor de políticas; um **PDP** (*Policy Decision Point*) como um membro autorizado a efetuar decisões com base em atributos dados ou relativos a critérios de avaliação de uma tarefa ou ação; um **PEP** (*Policy Enforcement Point*) como um membro que interceta pedidos de acesso a um recurso e confronta com as decisões dos PDPs; e um **PIP** (*Policy Information Point*) que fornece informação externa para um PDP.

arquitetura XACML

PAP, PDP

PEP

PIP

Neste caso, um sujeito que pretenda acesso a um objeto envia um pedido que será intercetado por um PEP. O PEP, de seguida, envia um pedido de autorização ao PDP, que avaliará o mesmo através das suas políticas armazenadas, alcançando uma decisão, retornando-a para o PEP.

Separação de deveres e fluxos de informação

separação de deveres

Um dos requisitos mais fundamentais de segurança contra fraudes e prevenção de erros é a **separação de deveres**, isto é, haver uma distribuição justa de tarefas e de privilégios associados para um determinado processo entre vários sujeitos envolvidos numa atividade.

Muitas vezes aplicado com a aplicação de controlo de acesso por RBAC, a separação de deveres perfaz também um controlo de danos, havendo assim uma segregação de deveres que reduz a possibilidade de criação de estragos num sistema, através das ações de uma pessoa.

níveis de segurança

Note-se, não obstante, que as autorizações deverão ser aplicadas a fluxos de informação, sendo que não se pretendem fluxos perigosos para um determinado sistema de informação (ou computacional). Para isto deverão ser criados **níveis de segurança** que permitam uma avaliação rápida e consisa sobre a classe de segurança (*clearance level*) de um determinado sujeito. Isto permite que sejam designadas classes de segurança para uma origem e para um destino de um fluxo de informação, classificando assim o mesmo através desse atributo.

O sujeitos ou papeis, então, agem em diferentes níveis de segurança, sendo que estes não se intersetam entre eles, tendo relações hierárquicas ou de malha (*lattice*). Dado isto, sujeitos (ou papeis) que se encontrem avaliados com a mesma classe de segurança têm o acesso garantido aos sistemas, mas se tiverem diferentes níveis, então o seu acesso é considerado como controlado, sendo que podem ser autorizados ou negados, numa base quase-militar em que se avalia a "necessidade de acesso". Estes níveis, dependendo da organização ou do estado onde se aplicam, podem possuir valores crescentes em segurança como: *unclassified*, *restricted*, *confidential*, *secret* ou *top secret*.

Modelo de Bell-La Padula

Uma forma de aplicar políticas de controlo de acesso a fluxos de informação está no uso do modelo de **Bell-La Padula**, criado em 1973. Este modelo permite que se atribuam critérios de confidencialidade e acesso a dados que possuem uma classe de segurança elevada, permitindo a divulgação da mesma apenas a quem de direito.

⊙ David Eliott Bell
Bell-La Padula
⊙ Leonard LaPadula

Para percebermos bem este modelo precisamos de ter noção do conceito de **etiquetas**, num fluxo de informação entre sujeitos e objetos. As etiquetas são definições de níveis de segurança designadas por *unclassified*, *restricted*, *confidential*, *secret* ou *top secret*. Estes níveis de segurança podem, então, ser aplicadas tanto a pessoas (os sujeitos), dando-lhes o nome de **classes de segurança** (ou em inglês níveis de *clearance*), e a documentos (os objetos), dando-lhes o nome de **classificação**. A notação de "pessoas" e de "documentos" provém de uma razão histórica, dado que este modelo foi introduzido pelo MITRE, um membro do MIT (Massachussets Institute of Technology) que nos anos '70 foi requisitado pela armada dos Estados Unidos da América para dar resposta ao problema "como é que as forças armadas podem assegurar os seus dados de fugas de informação?".

etiquetas

classes de segurança
classificação

Como resposta, Bell e La Padula esquematizaram um cenário em que uma pessoa que se encontra com uma determinada classe de segurança pode ler um documento avaliado à mesma classe ou mais baixa que à sua. Por exemplo, se a classe de uma pessoa *A* é *confidential*, então poderá ler documentos *unclassified*, *restricted* e *confidential*. No entanto, não faz sentido nenhum esta mesma pessoa ler documentos com classificação mais alta, pela simples razão de que não tem necessidade nenhuma de possuir essa informação, porque não está encarregue de tais responsabilidades. Estas decisões dependem essencialmente da posição que uma pessoa ocupa na escala de classes de segurança e no quão confiável ela é dentro do grupo de utilizadores. Assim sendo, um sujeito apenas poderá ler objetos ao mesmo nível ou inferior ao seu — por esta razão se diz que este modelo é **no read up**.

no read up

Em termos de escrita o cenário já muda de figura. Se formos uma pessoa *A* com nível de *clearance top secret*, então devemos conseguir escrever documentos com a mesma classe *top secret*. No entanto, será suposto podermos escrever acima do nosso nível ou abaixo? Vejamos, se escrevermos abaixo estamos a ser uma pessoa *top secret* a escrever documentos a um nível muito mais baixo, que poderá levar a fugas de informação involuntárias — é uma situação insegura. Por essa mesma razão, pelo modelo de Bell-La Padula, um sujeito poderá escrever objetos ao mesmo nível ou superior ao seu — por esta razão se diz que este modelo é **no write down**.

no write down

Na Figura 5.1 temos uma possível representação deste conceito.

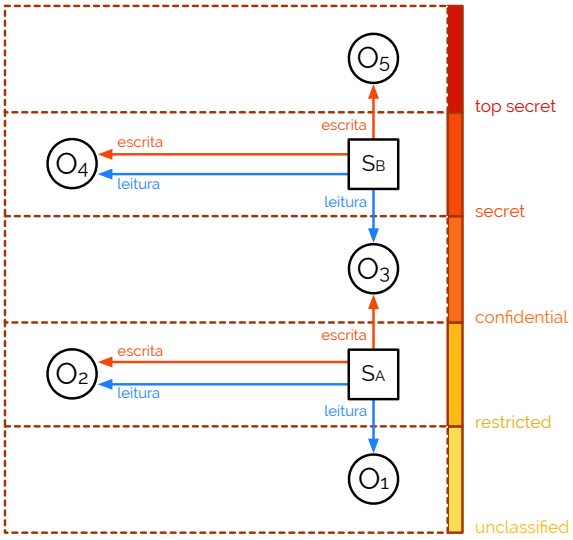

figura 5.1
modelo de Bell-La Padula

Modelo de integridade de Biba

Contrariamente à preocupação do modelo de Bell-La Padula, praticamente ao mesmo tempo, um investigador pelo nome de K. Biba criou um modelo para classificar os vários níveis de acesso a documentos militares por parte de pessoas através de uma outra variável crítica, ao invés da classe de segurança de cada sujeito — a **integridade**.

integridade

Saber distinguir dois documentos pela sua integridade é saber avaliá-los consoante a sua qualidade de apresentação e de escrita, e pelo seu grau de precisão no detalhe e conceito. Para aplicar esta modalidade surge então o **modelo de Biba**, um modelo que faz o controlo de acesso na leitura e na escrita, de pessoas, através da integridade de que cada um dispõe nos seus documentos.

modelo de Biba
⊙ **Kenneth Biba**

Consideremos assim que somos uma pessoa A, classificado como muito íntegro nos documentos que escrevemos. Fará sentido lermos algo menos íntegro? Ora, ao lermos algo com um grau de integridade mais baixo podemos estar a incorrer no perigo de perdermos a nossa integridade enquanto possessores de uma boa integridade de leitura. Contudo, se escrevermos, com o nosso grau de integridade, documentos menos classificados que os do nosso nível, estamos a acrescentar valor a estes, pelo que estamos perante uma ação que é realizável. Assim sendo, podemos dizer que, na leitura, segundo o modelo de Biba, é-nos permitido ler documentos com o mesmo grau de integridade ou superior e, na escrita, é-nos permitido escrever documentos com o mesmo grau de integridade ou inferior. Resumidamente, o modelo de Biba é *no read down*, *no write up*. Uma possível representação desta esquemática poderá ser vista na Figura 5.2.

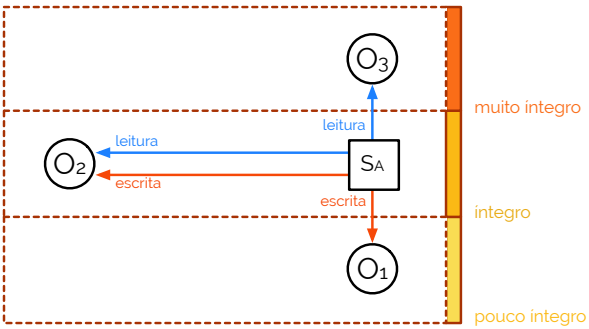

figura 5.2
modelo de Biba

Modelo de integridade de Clark-Wilson

Desenhado em 1987 e contrariamente ao modelo de Bell-La Padula e ao modelo de Biba, o modelo de **Clark-Wilson** foi criado para aplicações comerciais (e não com fins militares).

⊙ **David Clark**
Clark-Wilson
⊙ **David Wilson**

De forma algo semelhante ao modelo de Biba, este modelo reforça a integridade. No entanto, ao invés de desenhar todo o processo através de uma máquina de estados (como é o caso do modelo anterior), este define cada item de dados e restringe os vários programas que os poderão modificar. Para isso, este modelo usa etiquetas para garantir o acesso aos vários objetos, onde programas que não possuam a classificação necessária para a modificação dos objetos não o conseguirão fazer.

Vejamos primeiro alguns dos conceitos necessários para que possamos perceber com mais cuidado do que se pretende alcançar com este modelo de Clark-Wilson. Primeiro temos os itens de dados restringidos (**CDI**, sigla para *Constrained Data Item*), que são os dados que são, de facto, protegidos pelo modelo. Mas uma vez que nem todos os objetos podem ser protegidos, existem também itens de dados não restringidos (**UDI**, sigla para *Unconstrained Data Item*), como são o caso as fontes de *input* e de *output*, uma vez que são variáveis e o seu conteúdo não é possível de ser discriminado em tempo útil para uma determinada aplicação. Depois, para efetuar as verificações dos dados, temos os vários procedimentos de verificação de integridade (ou **IVP**, sigla para *Integrity Verification Proce-*

CDI

UDI

IVP

dure), que olham somente para os vários CDIs. Já os dados que não são restringidos são levados para transformação através dos procedimentos de transformação (mais vulgarmente conhecidos por **TP**, sigla para *Transformation Procedure*), onde se tentam que se passem de UDIs para CDIs, que são aceites somente se a conversão for bem sucedida dentro de determinados parâmetros. Mais precisamente, uma TP que tome uma UDI como parâmetro só poderá desempenhar transações válidas para todos os valores possíveis do UDI. Depois cabe ao TP aceitar o resultado (converter para CDI) ou rejeitar o UDI. Estes são os únicos procedimentos que podem tocar nos CDIs, mantendo a integridade dos dados.

TP

6. Segurança em Sistemas Operativos

O sistema operativo, como já abordámos em Sistemas de Operação (a3s1), é constituído por várias camadas de abstração, cada uma com um nível de responsabilidades completamente distinto das outras. Nessa mesma disciplina pudemos verificar que a primeira camada que se sobrepõe ao próprio *hardware*, fazendo interface com o mesmo, é o chamado **kernel** do sistema de operação — uma zona do sistema operativo que permite a virtualização do *hardware*, o reforço de políticas de proteção e a aplicação de mecanismos de proteção contra erros involuntários do utilizador e atividades consideradas ilegais no contexto do sistema.

kernel

Em cima do *kernel* do sistema operativo são então executadas as várias aplicações que são ditas correrem em **modo de utilizador**, isto é, num modo normal de execução do CPU. Este modo existe, uma vez que existe um conjunto de instruções que são delicadas e reservadas à execução em **modo de supervisão**, ou seja, ao modo em que as instruções são diretamente executadas sobre o *kernel* de um sistema operativo. Na Figura 6.1 podemos verificar uma pequena representação de como é que é estruturado um sistema operativo.

modo de utilizador

modo de supervisão

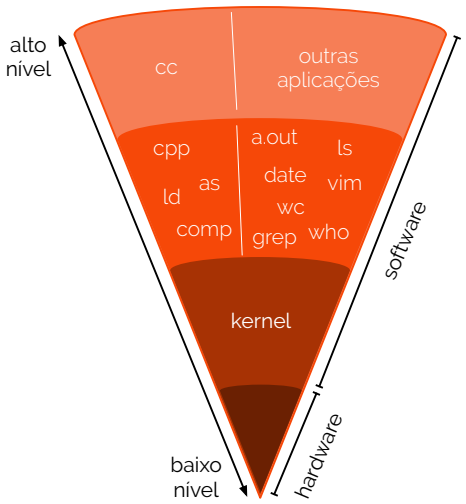

figura 6.1
organização de um sistema operativo

Definição de níveis de privilégio

Teoricamente, os processadores apenas têm dois modos de execução, que fornecem dois níveis de privilégio aos seus utilizadores por vias de um sistema operativo. No entanto, na prática, os processadores não têm apenas dois níveis.

Para que os vários processadores possam ser compatíveis com vários sistemas operativos, alguns com necessidades mais exigentes em termos de gestão de níveis de privilégios, estes precisam de ter mais do que dois níveis definidos. Por essa mesma razão os processadores hoje em dia estão organizados de forma a apresentarem, por norma, 4 níveis de privilégios que se organizam sob a forma de **aneis concêntricos**. Estes aneis são uma forma que os processadores têm de previnir que código não-privilegiado seja executado com códi-

aneis concêntricos

gos de operação privilegiados (exemplos disso são operações sobre o *input* e *output* ou manipulações sobre o *Translation Lookaside Buffer* — TLB). Nestes aneis o centro, dotado do identificador 0 é correspondente ao modo de supervisão (ou de *kernel*), enquanto que o identificador 3 é correspondente ao modo de utilizador. As várias transições entre níveis dos aneis requerem a utilização de **chamadas ao sistema**, ou também vulgarmente denominadas de **syscalls**.

chamadas ao sistema
syscalls

Quando estamos num ambiente em que estamos a usar máquinas virtuais o cenário é ligeiramente diferente, uma vez que neste caso o processador necessitará de improvisar um nível abaixo do anel 0 para destacar as funções de um **hypervisor** (sistema de gestão de virtualização) — isto se estivermos a falar de uma virtualização total, ficando assim o sistema dotado da capacidade de virtualizar *hardware* para muitos *kernels* de anel 0. Em alternativa a isto, com uma virtualização feita com base em *software* (solução mais comum) há uma execução direta de código do sistema *guest* em modo de utilizador, o qual é objeto de uma tradução de código privilegiado (os *kernels* dos sistemas operativos *guest* permanecem inalterados, mas não são executados diretamente na máquina anfitriã). Ao contrário, num sistema puramente virtualizado, não há a necessidade de tradução de binários o que permite que o sistema operativo convidado (*guest*) poderá executar muito mais depressa. [30]

hypervisor

Modelo computacional e controlo de acesso

Um sistema computacional como o que temos vindo a ilustrar ao longo dos vários apontamentos é um sistema tal que possui um conjunto de entidades (consideremos como objetos) geridos por um *kernel* de um sistema operativo. Tal conjunto de entidades é constituído por identificadores de utilizadores, processos, memória virtual, ficheiros e sistemas de ficheiros, canais de comunicação e dispositivos físicos como dispositivos de armazenamento (discos magnéticos, óticos, *tapes*, ...), interfaces de rede, interfaces de interação humano-computador (rato, teclado, ecrãs, ...) ou interfaces série/paralelo para I/O (USB, portas série, portas paralelo, infravermelhos, *bluetooth*).

Consideremos, a partir deste ponto, um sistema UNIX ou UNIX-*like* como uma distribuição Linux ou o sistema operativo macOS, para simplificarmos as nossas provas de conceito.

Quando falamos de utilizadores de um sistema UNIX vulgarmente atribuímos um nome para identificar um dono de uma conta. No entanto, para o sistema em si não é esta a correspondência que é feita, sendo que para o *kernel* do sistema um utilizador é sempre identificado de forma inequívoca através de um identificador numérico, que é estabelecido aquando de uma operação de *login* e é denominado de **UID** (sigla para *User ID*). Depois deste número ser estabelecido, todas as operações que são realizadas no computador são registadas como fazendo parte das ações do UID em causa, permitindo assim que o *kernel* verifique o que é que é permitido, ou não, executar em processos para este utilizador. No caso dos sistemas UNIX, o UID com identificador 0, geralmente, é o identificador do utilizador raiz (com permissões sobre todo o sistema — omnipotente) — para além do facto do utilizador `root` ser um nome como outro qualquer (pode ser alterado que o sistema não se queixará), mais à frente veremos uma exceção que poderá ocorrer e que contrapõe o identificador 0 ser sempre o utilizador raiz.

UID

Contudo, num sistema UNIX os utilizadores não se encontram apenas definidos como unidades, mas também como pertencentes a conjuntos denominados de **grupos**. No caso de um grupo, este embora seja identificado em modo de utilizador por um nome, também possui um identificador inteiro que é interpretado e com que é manipulado pelo *kernel* do sistema operativo, denominado de **GID** (sigla para *Group ID*). A ideia da criação de grupos parte também muito pelo facto de haver conjuntos de utilizadores que poderão ter várias permissões em comum, estas, que, no ponto de vista de um utilizador, são um extra visto que este será visto como um dotado de permissões como utilizador e de outras permissões como membro de determinados grupos. Contudo, se um utilizador pertencer a mais que um grupo, então dentro dos vários onde participa deverá existir um grupo con-

grupos

GID

siderado como primário (tipicamente usado para marcar a proteção de ficheiros) e os outros como secundários.

Outra questão que se levanta com a forma como os sistemas computacionais estão desenhados, e que deverá ser uma consideração nossa a tomar, são os **processos**. Um processo, tal como vimos na disciplina de Sistemas de Operação (a3s1) define o contexto de existência de uma atividade em execução no sistema. Este contexto de existência permite que o sistema consiga tomar decisões em termos de segurança (partilha de dados do processo em causa, escalonamentos, ...), mas também permite identificar as razões pelas quais o processo existe ou existiu, através da identificação do utilizador que o criou (através do UID) ou do grupo (GID) e listar os vários recursos que está a usar, como ficheiros abertos (ficheiros regulares ou comunicações por via de canais apropriados), áreas de memória virtual reservadas ou tempo de CPU usado.

processos

Dadas estas informações e a forma como o sistema operativo funciona, podemos então concluir que o *kernel* possui caraterísticas de um **monitor** de controlo de acesso, na medida em que controla todas as interações com o *hardware* e todas as interações entre entidades do modelo computacional descrito. Em termos dos vários sujeitos que possam requerer acesso a determinados serviços para efetuar as suas tarefas, estes, uma vez que necessitam de comunicar através da API de *syscalls*, passam pelo controlo marcado pelo *kernel* que os monitoriza — o mesmo acontece em caso de mensagens provenientes de outros terminais.

monitor

Olhando então para o *kernel* de um sistema operativo como um monitor de controlo de acesso, podemos verificar que estes possuem, por si, políticas de controlo de acesso obrigatórias, que já fazem parte do modelo lógico computacional, não podendo ser sobrepostas por novas políticas criadas por administradores do sistema (a não ser que mudem por completo o comportamento do *kernel* operado na máquina). Exemplos destas práticas são visíveis quando o *kernel* executa tarefas em modo de supervisão e o utilizador executa as suas tarefas em modo de utilizador, quando áreas de memória virtual se encontram separadas, quando há sinalização entre processos, entre outros...

Basicamente o sistema operativo tal como estamos habituados a utilizá-lo, está carregado de listas de acesso (para o controlo de acessos) que definem cada objeto, dizendo o que cada sujeito poderá fazer com estes. Dentro do conjunto destas listas de acesso, note-se que nem todas são obrigatórias, sendo que algumas são meramente discricionárias (podendo ser adaptadas ao sujeito que é alvo das mesmas). A execução destas listas de acesso é feita apenas quando o sistema sente que há necessidade de as testar, geralmente coincidindo com momentos em que uma determinada atividade, pela parte de um sujeito, quer manipular o objeto relacionado com a lista — mais uma vez, mostra-se a capacidade de monitorização por parte do *kernel* do sistema operativo.

Exemplos das listas de controlo aplicadas nos sistemas UNIX poderão partir, por exemplo, da proteção aplicada a cada ficheiro presente no sistema. Consideremos assim o Código 6.1 onde mostramos as informações relativas a um ficheiro depois de listarmos um diretório.

⊙ **Bruce Schneier**

código 6.1

```
$ ls -l
total 8
-rw-r--r--  1 apontamentos  wheel   121B Jan 15 09:41 bruceschneierfact.txt

$ cat bruceschneierfact.txt
Bruce Schneier knows a deterministic algorithm to generate non-pseudo random numbers
without need of an entrophy source.
```

Quando listamos, com detalhe, os vários ficheiros pertencentes a um diretório num sistema UNIX (ou simplesmente executando o comando `stat <ficheiro>`) podemos reparar que um conjunto de 10 carateres logo no início que geralmente apresenta conjuntos de carateres `'R'`, `'W'` ou `'X'`. O carater `'R'` serve para identificar se o sujeito possui direito de leitura ou de listagem. O carater `'W'` serve para identificar se o sujeito possui direito de escrita ou de criar e remover ficheiros ou diretórios. O carater `'X'` serve para identificar se o sujeito possui direitos de execução ou poderes de usar o ficheiro como diretório de trabalho do processo atual (em inglês *process' current working directory*). A apresentação

destes carateres é feito em conjuntos de três a partir do segundo carater da sequência, sendo que os conjuntos significam os sujeitos de utilizador (dono), através do UID, de grupo, através do GID, e de outros. No caso em causa, do Código 6.1, nós, como utilizadores temos direito a leitura, daí a concatenação do ficheiro no terminal, executado pelo comando `cat` ter sido feita com sucesso. No entanto, se removermos estes direitos, como podemos verificar no Código 6.2, deixamos de ter permissões de leitura do mesmo.

código 6.2

```
$ ls -l
total 8
----------  1 apontamentos  wheel   121B Jan 15 09:41 bruceschneierfact.txt

$ cat bruceschneierfact.txt
cat: bruceschneierfact.txt: Permission denied
```

Elevação de privilégios em sistemas UNIX

Em UNIX os priviégios de um processo estão então definidos através do seu UID e GID, que são herdados pelo processo-pai. Depois de uma operação de *login*, o processo inicial do utilizador que possui a sessão obtém o seu UID do utilizador e GID extraídos de ambos ficheiros `/etc/passwd` e `/etc/group`. Assim, os comandos executados a partir deste momento e da primeira aplicação com que a sessão foi iniciada (como um terminal `bash`) utilizarão o mesmo UID e GID. [31]

Consideremos agora um cenário em que um utilizador que já possui uma conta criada necessita de alterar a sua palavra-passe. Uma tarefa deste tipo deverá estar ao alcance de qualquer utilizador, quando este o pretender. Acontece que o ficheiro que possui uma transformação das palavras-passe dos utilizadores está armazenado num local em que utilizadores regulares (não-administradores) não podem modificar dados — uma vez que podem violar a integridade do ficheiro ou simplesmente roubar as credenciais. Mas os utilizadores devem ser capazes de alterar as suas palavras-passe… Como fazer?

set-UID

Para isso os sistemas UNIX criaram o conceito de mecanismo **set-UID**. O mecanismo set-UID permite que se altere o UID de um processo que corre um programa. Para que isto possa acontecer, a proteção dos ficheiros UNIX, para além das permissões que referimos no fim da secção anterior, antes, possui três *flags* que são indicativas da realização de uma operação de set-UID, uma operação de **set-GID** (semelhante à de set-UID, mas para grupos) e um *sticky bit* que existe por razões históricas, usado como uma pista para aumentar a permanência dos conteúdos do ficheiro na memória principal. Quando estas *flags* estão ativas em ficheiros executáveis, é permitido que todos os utilizadores executem tais executáveis com a identidade UID do utilizador dono ou GID do grupo. O seu objetivo é permitir que os utilizadores executem comandos específicos com possíveis privilégios elevados, de forma a que possam executar tarefas específicas que requeiram privilégios especiais.

set-GID

Exemplos de aplicação deste mecanismo são visíveis quando se pretende mudar a palavra-passe de *login*, montar/desmontar um sistema de ficheiros, partilhar uma sessão com outra identidade, entre outros…

Para verificarmos se um determinado executável usa o mecanismo set-UID (ou set-GID), podemos verificar as suas permissões através do comando `stat` ou listando o conteúdo, com detalhe, do diretório onde está presente. Vejamos assim as permissões de alguns dos ficheiros presentes no diretório `/usr/bin` (Código 6.3).

código 6.3

```
$ ls -l /usr/bin
total 183948
-rwxr-xr-x  1 root root      134968 Dec  8  2016 apt-get*
-rwxr-xr-x  1 root root      154328 Nov 28 18:50 curl*
-rwxr-xr-x  1 root root      125960 Oct 21  2013 diff*
-rwxr-xr-x  1 root root      775888 May  7  2016 gcc-4.8*
-rwxr-xr-x  1 root root     1577288 Oct  4 18:18 git*
-rwxr-xr-x  1 root root      170088 Jun  8  2013 make*
-rwxr-xr-x  1 root root       31328 Mar 10  2016 nice*
-rwsr-xr-x  1 root root       47032 May 16  2017 passwd*
-rwsr-xr-x  1 root root      155008 May 29  2017 sudo*
-rwxr-sr-x  1 root tty        19024 Nov 23  2016 wall*
```

Como podemos ver no Código 6.3 temos um conjunto de executáveis inseridos no diretório `/usr/bin`, sendo que três deles possuem elevações de privilégio por vias de set-UID e set-GID. Mas vejamos primeiro o caso do executável `git`. O `git` possui como permissões `-rwxr-xr-x`, isto é, é um ficheiro regular com permissões de leitura, escrita e execução para o UID atual, e permissões de leitura e execução para o grupo atual e outros. A interpretação deverá ser fácil, mas o mesmo cenário não será igual no caso do executável `passwd` ou `sudo`. Quando vemos as suas permissões podemos reparar que existe um `'s'` onde deveria estar um `'x'`, para os direitos de execução do utilizador — isto é o que identifica um executável que usa o mecanismo de set-UID. Já para o caso do set-GID temos o exemplo do executável `wall` que no lugar do `'x'` para os direitos de execução do grupo, tem um `'s'`, identificando o caso de aplicação do mecanismo set-GID.

Então mas o que é que acontece no sistema operativo, de especial, para que este mecanismo funcione? Ora, cada processo possui um UID real e um UID efetivo. O **UID efetivo** é o identificador que é efetivamente utilizado pelo sistema operativo para estabelecer os privilégios, por exemplo, de acesso a ficheiros. A *flag* de set-UID muda precisamente este valor, mesmo que este não seja o **UID real**.

UID efetivo

UID real

Por outro lado, o mecanismo de set-GID faz exatamente as mesmas ações, só que para um GID efetivo e um GID real. Note-se, não obstante, que as aplicações poderão reverter o valor dos seus UID/GID efetivos para o seu valor original antes das operações de set-UID ou de set-GID ocorrerem, cujos valores são preservados nos valores do processo de UID e GID reais.

Consideremos assim, sendo o utilizador número 3 (por exemplo), que estamos numa consola `bash` e executamos o comando `git pull` do Código 6.3. Quando isto acontece o processo responsável pela execução da consola `bash` fica suspenso enquanto um `fork` é feito de si próprio num processo-filho para a execução do ficheiro executável `git`, com argumento `pull`. Em todo este processo, e uma vez que o executável `git` tem permissões regulares de execução, o UID efetivo na execução deste executável é o mesmo que antes — 3 — como podemos ver pela Figura 6.2.

figura 6.2
execução de git (sem set-UID)

Consideremos agora, sendo novamente o utilizador com UID 3, que pretendemos mudar a nossa palavra-passe. Para isso temos de executar o comando `passwd`, executável que, vendo no Código 6.3, possui o mecanismo de set-UID ativo. Quando executamos tal comando, o que acontece é que será feito um `fork` para a criação de um processo-filho e depois é executado um `setuid()` com argumento 0, sendo que 0 é o UID da conta raiz, com permissões para executar as tarefas exigidas pelo `passwd`. Quando o comando termina o `setuid()` é usado para restaurar o valor original do UID, recuperando-o através do valor de UID real, como podemos ver na Figura 6.3.

Confinamento de execuções em UNIX (limitação de privilégios)

A informação que está preservada em sistemas de ficheiros está igualmente protegida, geralmente, de acessos inesperados e não-pretendidos através do uso de atributos de pertença (um determinado utilizador é dono de um ficheiro) ou de identidade e permissões

correspondentes. Contudo, estas proteções muitas vezes poderão ser circundadas por atacantes que são capazes de atingir altos privilégios dentro do sistema (como suportar uma identificação de utilizador raiz, num sistema UNIX) ou personificar outros utilizadores/grupos que possuem permissões relativamente a ficheiros-alvo. [31]

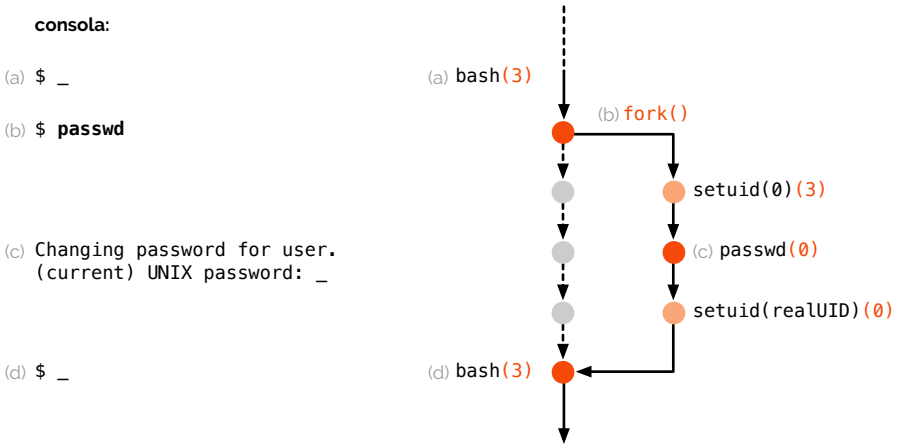

figura 6.3
execução de passwd (com set-UID)

Uma solução para resolver este tipo de problemas está na redução da visibilidade do horizonte por parte de um utilizador (representado por vias de um processo em execução), perante o sistema de ficheiros. De Sistemas de Operação (a3s1) sabemos que não há nenhum caminho acima de uma raiz de diretórios, pelo que só temos de conseguir aplicar este raciocínio a este caso em concreto. Para isso existe a ferramenta `chroot` (acrónimo de `change root`) que permite alterar o *working directory* do processo em execução, de forma a que este veja o seu próprio diretório de execução como a raiz do sistema de operação onde está inserido (Figura 6.4).

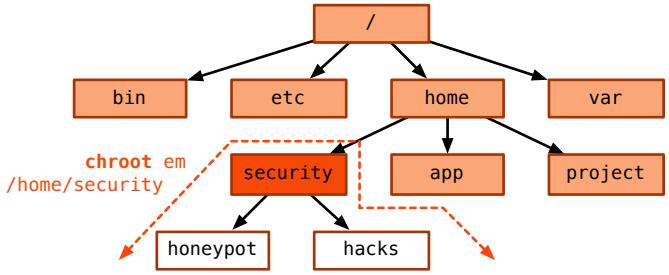

figura 6.4
efeito de chroot

Contudo, há um problema: o que é que acontece se possuirmos ficheiros no diretório atual que ligam, de alguma forma, a diretórios externos (anteriores) a esta raiz? Ora, quando se executa o comando `chroot` o sistema tratará de fechar algumas das portas que possamos ter deixado abertas, como descritores de ficheiros abertos ainda no sistema, mas caso o nosso diretório tenha *links* para locais remotos e anteriores à nossa nova raiz, então tais ficheiros deverão ser retirados, havendo o perigo do **confinamento** não ser íntegro a sua razão de existência.

confinamento

7. Armazenamento Seguro de Dados

Quando falamos em proteção dos dados geralmente a ideia que nos vem à cabeça foca-se muito por tentar criar mecanismos nos nossos sistemas operativos que nos permitam ter algum grau de certeza sobre a integridade das informações que manipulamos e que estão preservadas dentro das nossas unidades de armazenamento local, sejam elas um disco magnético, um SSD, um disco ótico ou outro tipo de dispositivo. Contudo de que nos servem tais mecanismos, se os ataques poderão ser conduzidos diretamente às nossas unidades físicas? Ou então, mesmo sendo alvo de ataque por vias de um sistema, de que

nos servirá tal camada de segurança se quem o fizer contornar os nossos mecanismos (por exemplo através de um acesso não-ético de um utilizador com várias permissões)?

E o problema nem para por aqui, uma vez que há um conjunto de sistemas que ainda não estamos muito habituados a referir, que são os sistemas distribuídos. Num sistema distribuído como é que podemos aplicar tais critérios de segurança? Com tantos grupos como é que podemos distinguir um grupo de administração possivelmente falso se não conhecermos o real? Mais, como é que podemos ter a certeza sobre aspetos como a autenticação de utilizadores, uma vez que o temos de fazer remotamente?

São muitas questões levantadas que só terão resposta se aplicarmos algum detalhe de segurança diretamente aos dispositivos ou à forma como despejamos os dados nestes.

Cifragem de ficheiros

Uma forma de tentar resolver o problema do armazenamento seguro de dados está na **cifragem dos dados** presentes nos dispositivos físicos. Esta, sendo muito possivelmente a forma mais fácil de o fazer na nossa perspetiva de quem trabalha ao nível lógico (nível de computadores e não ao nível físico como eletrónica), permite que os dados circulem de forma segura dentro de redes consideradas como comprometidas (no caso de sistemas distribuídos) ou que sejam guardados em discos que poderão ser alvo de ataques de integridade.

cifragem dos dados

Embora seja um método de fácil implementação, possui algumas desvantagens, entre as quais a partilha de ficheiros, que necessitará que quem receba os ficheiros possua a chave para os decifrar, a interferência com procedimentos normais de cópias de segurança da informação pela administração de um sistema, ou a obtenção de dados, que exigirá que o utilizador que faz a cifragem dos dados fique para sempre com uma chave de cifra/decifra.

Isto leva-nos a uma questão interessante: a que nível é que devemos implementar cifragem de dados? Ora, a cifragem de dados pode ser implementada ao nível da aplicação, ao nível do sistema de ficheiros ou a nível dos *device-drivers*, como podemos ver na Figura 7.1.

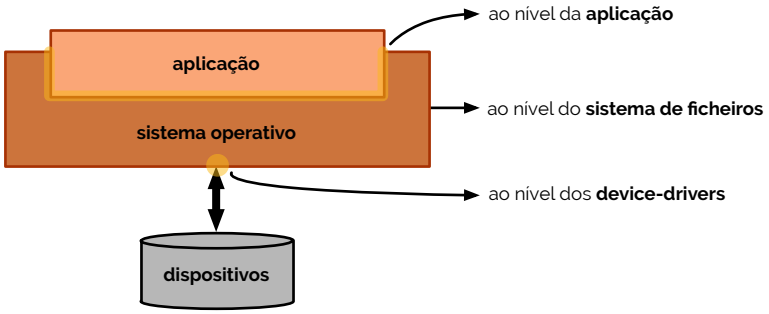

figura 7.1
cifragem de ficheiros (possíveis abordagens)

Seja de que forma for aplicada, a cifra e decifra terá de se mostrar completamente transparente para as várias aplicações que possam trabalhar sobre os dados concretos, tal como para os ficheiros de *cache* do sistema operativo. Contudo, analisemos, com algum detalhe, cada uma das possíveis implementações.

Implementar cifragem de ficheiros ao nível aplicacional significa que esta tarefa é realizada por aplicações autónomas, com pouca ou nenhuma integração com outras aplicações. Normalmente, dado que estas implementações são mais comuns quando pretendemos cifrar conjuntos pequenos de ficheiros, nota-se quando é que os ficheiros se encontram seguros ou não, geralmente indicado por um nome diferente, com uma extensão específica do mecanismo de cifra que lhe foi aplicado ou da aplicação que o usou. Estes procedimentos são também bastante flexíveis, uma vez que os dados poderão ser transformados com diferentes algoritmos — basta usar mecanismos diferentes, garantidos por aplicações igualmente diferentes. Contudo, usar múltiplas aplicações para o fazer dificutará o processo de recuperação de informação, sendo que mesmo para cifrar ficheiros com uma única

aplicação há a dificuldade de partilha das chaves para a decifra de conteúdos aquando de uma partilha de ficheiros.

Alguns dos exemplos de implementação de cifragem de ficheiros ao nível de aplicação são, por exemplo, o PGP (sigla de *Pretty Good Privacy*) ou o AxCrypt, entre outros...

Por outro lado, ao invés de implementarmos a cifragem de ficheiros ao mais alto nível disponível, podemos tentar implementar ao mais baixo nível possível — ao nível dos *device-drivers*. Neste caso, uma vez que estamos por debaixo de todas as camadas de sistema operativo (das quais as aplicações fazem parte), esta cifra/decifra terá de passar puramente transparente para as aplicações e sistema operativo. Contudo, este método de utilização poderá não ser o melhor caso se queira implementar sobre uma unidade de armazenamento que é partilhada por mais do que um utilizador, dado que só existirá uma chave — daí que este método seja aconselhado a unidades que são utilizadas por apenas uma pessoa.

Alguns exemplos de implementação de cifragem de ficheiros ao nível dos *device-drivers* podem ser vistos com o PGPdisk ou o LUKS (acrónimo de *Linux Unified Key Setup*). Também existem alguns dispositivos como os *Secure Digital Cards*, mais conhecidos como cartões SD, que já incluem algumas caraterísticas de segurança, mais precisamente no que toca a previnir os seus conteúdos de possíveis remoções ou modificações, previnindo o acesso ou protegendo o conteúdo através da gestão de direitos digitais de *copyright*.

Finalmente, temos a implementação de cifragem por via do sistema de ficheiros. Aqui os dados são transformados no caminho entre os dispositivos de armazenamento e a memória de aplicações. Esta transformação acontece porque os dados carregados do dispositivo de armazenamento necessitam de ser decifrados de forma a serem úteis para serem usados pelas várias aplicações presentes e sustentadas pelo sistema operativo. Veja-se assim a Figura 7.2.

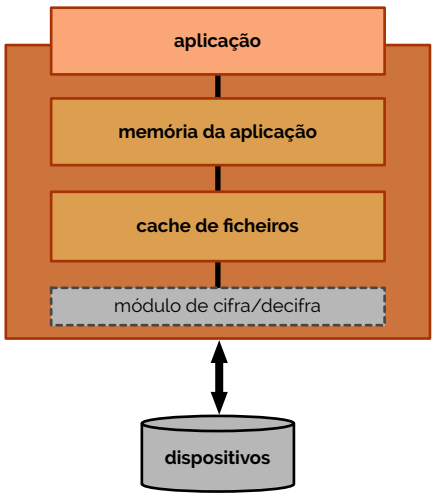

figura 7.2
cifragem de ficheiros (pelo sistema de ficheiros)

Para simplificar as várias transações realizadas com os vários dados que são carregados do disco, dado o esforço das operações de cifra e decifra, é importante que haja um módulo que sirva de intermediário, de forma a reduzir a carga de operações para a obtenção de um mesmo dado várias vezes. Este módulo, denominado de **cache de ficheiros**, posiciona-se entre a memória de aplicação e o *device-driver* de um dispositivo de armazenamento, como podemos verificar na Figura 7.2.

cache de ficheiros

Contudo, este tipo de modelos transporta consigo vários problemas. Não sendo todos os problemas necessariamente desvantagem destas aplicações, estes sistemas de ficheiros possuem uma integridade que deverá ser preservada, sendo que alguns dos atributos não poderão ser ocultos, de forma a manter a operação do sistema sobre o ficheiro, regular (exemplo disto será uma tarefa de administração como uma cópia de segurança, onde os atributos são importantes para saber detalhes acerca dos tamanhos dos ficheiros, por exemplo). Isto não significa que todos os atributos tenham de ser ocultos, no entanto, casos

como o nome dos ficheiros deverão, depois de cifrados, manter as normas de nomenclatura do sistema de ficheiros em causa e os conteúdos do ficheiro cifrado não deverão ter tamanhos diferentes do original. Mais, diretórios como os `"."` e `".."` não poderão ser modificados no processo de cifra/decifra, uma vez que antes ou depois do mesmo são ficheiros bem conhecidos e necessários na definição do sistema de ficheiros.

Entre os problemas que podem ser desvantagens dependendo da sua utilização temos que o acesso aleatório uniforme é uma tarefa difícil, uma vez que a decifra total para leitura é necessária para que o último byte de um ficheiro seja conseguido e a cifragem de um ficheiro também necessita de ser cumprida totalmente para que seja atualizado o primeiro byte. Mais, em termos de confidencialidade recomenda-se que não se use a mesma chave para ficheiros diferentes (tarefa árdua), uma vez que o contrário formará padrões que poderão revelar ficheiros de conteúdo e forma semelhantes. A uma escala mais pequena, o mesmo se aplica dentro de um ficheiro, uma vez que será possível perceber padrões dentro de um mesmo ficheiro de texto, por exemplo. No caso em que se pretenda usar uma mesma chave para ficheiros, não sendo um bom exemplo, o melhor será não usar uma chave contínua, uma vez que tornará possíveis ataques do tipo *known-plaintext*, que poderão revelar conteúdos de outros ficheiros.

8. Segurança em Base de Dados

base de dados

Em Bases de Dados (a3s2) estudámos que uma **base de dados** é uma coleção de dados e conjuntos de regras que organizam os dados especificando relações entre eles. Embora estes dados estejam guardados num ficheiro, para o utilizador não é problema, uma vez que este interage com os dados através de um Sistema de Gestão de Base de Dados (SGBD).

A ideia lógica que reside por detrás de uma base de dados é, então, que temos um único conjunto de dados, guardados e mantidos num só local, no qual várias pessoas têm acesso, de acordo com as necessidades destas. No entanto, uma implementação mais real poderá necessitar de outros locais de armazenamento ou níveis de acesso, sendo que o verdadeiro interesse de uma base de dados é que o utilizador nunca se tenha que preocupar com a componente física da mesma. Por conseguinte, há um conjunto de vantagens que são dadas aos sistemas de base de dados, em comparação a um mero sistema de ficheiros.

Primeiro, um sistema de base de dados possui acesso partilhado, o que significa que vários utilizadores poderão usar um só meio comum de partilha de dados, centralizado. Mais, podemos ter esse acesso controlado, de forma a que só utilizadores autorizados possuam permissões de visualizar ou modificar os dados presentes. Para isto, os utilizadores também não necessitam de ter uma cópia de parte dos valores a modificar para efetuar a mudança, uma vez que estes tipos de sistema nunca precisam de grandes níveis de redundância. Isto também permite que os próprios dados possuam facilmente consistência, de forma a que cada mudança efetuada tenha impacto sobre todos os restantes utilizadores, e integridade, de forma a que os vários valores armazenados possam estar protegidos contra acidentes ou mudanças maliciosas (ou indesejáveis).

Mas para que uma base de dados possa ser distribuída e o seu uso possa ser feito, existe um conjunto de requisitos básicos que deverão ser respeitados. O primeiro ponto que é necessário é que haja garantias de integridade da base de dados, em termos físicos, isto é, os dados da base de dados deverão estar imunes a problemas físicos, como falhas de energia, sendo que alguém poderá reconstruí-la se foi destruída depois de uma catástrofe. Da mesma forma, em termos lógicos, a estrutura da base de dados também deverá ser mantida intacta, sendo que um campo de um dado não poderá sofrer consequências por outro ter sido modificado. Mais, o valor de cada dado deverá ser o mais correto e preciso, sempre.

Fora questões de integridade, as bases de dados também deverão possuir capacidade de auditabilidade (capacidade de seguir quem ou quando é que um determinado elemento foi acedido — ou modificado), controlo de acesso (um utilizador deverá ter permissões a aceder determinados dados e diferentes utilizadores deverão ter diferentes modos de acesso,

como de escrita e de leitura), autenticação (todos os utilizadores deverão estar identificados, tanto para auditoria, como para o controlo de acesso) e disponibilidade (os utilizadores poderão aceder a uma base de dados em geral e a todos os dados para os quais têm acesso).

Transações

Um dos grandes problemas de um gestor de base de dados é quando o sistema que gere tem uma falha grave em plena modificação de dados. Se um determinado item de dados a ser modificado tiver grandes dimensões ou se as alterações consistirem num largo conjunto de dados a serem atualizados, caso algo corra mal a meio só parte dos dados ficam escritos, mas depois não conseguimos saber ao certo em que estado é que a base de dados ficou. Uma solução para este problema surgiu com o desenho das **transações** [32].

transações

As transações, também denominadas de mecanismo por duas fases, possui uma técnica de atualização que, durante a primeira fase — fase denominada de **intenção** — o Sistema de Gestão de Base de Dados carrega os recursos de que necessita para efetuar a atualização. Poderá considerar o carregamento de dados, criar registos *dummy*, abrir ficheiros, bloquear utilizadores e calcular resultados finais — poderá fazer tudo para preparar uma determinada operação a ser levada à base de dados, mas não chega a fazer qualquer tipo de alteração na mesma. Isto permite que caso o sistema falhe em pleno curso destas alterações, a base de dados não fique num estado incoerente e tudo possa ser recuperado, repetindo todos os passos depois do sistema voltar ao normal.

intenção

A última fase do processo de transações é denominada da fase de **commit** e permite que sejam postas em prática todas as alterações previstas na fase anterior, uma prática que não possibilita o retorno. Ou seja, depois do *commit* não há volta atrás — as alterações serão feitas.

commit

Este processo de transações é importante para manter alguma integridade de informação dentro da base de dados, tarefa que é feita tomando um conjunto de operações como uma só, em termos lógicos — através da **atomicidade**. Contudo, ainda que se faça o *commit* das operações todas de uma só vez, alguns pontos mais críticos neste processo são alvo de **shadow values**, ou seja, pequenas duplicações de registos ou campos de dados que são calculados e guardados localmente durante a fase de intenção. [3]

atomicidade

shadow values

Redundância, consistência e concorrência

Muitos SGBDs mantêm informação adicional para detetar inconsistências internas nos dados. Esta informação adicional varia entre vários bits a duplicados ou campos *shadow*, dependendo da importância dos dados.

Uma forma de redundância é a deteção de erros, como vimos em Fundamentos de Redes (a3s1), através de bits de paridade ou códigos de Hamming. Estes códigos podem ser aplicados a campos singulares, registos, ou a uma base de dados por inteiro. Por cada vez que um item é colocado na base de dados, os códigos de verificação apropriados são calculados e guardados. Por outro lado, cada vez que um registo é carregado da base de dados é calculado o código e comparado com o código que está guardado — se for igual então está tudo bem, caso contrário significa que o SGBD teve um erro na base de dados. [3]

⊙ **Richard Hamming**

Para além dos processos de correção de erros, um Sistema de Gestão de Base de Dados deverá manter um *log* dos vários acessos dos utilizadores, mais em particular de todas as suas modificações. Em caso de falha, a base de dados deverá ser recarregada com base numa cópia de segurança e em alterações posteriores gravadas em relatórios de auditoria.

Uma vez que os sistemas de base de dados são, geralmente, sistemas onde mais do que um utilizador trabalha, é importante que o acesso de dois ou mais utilizadores não provoque más consequências entre estes. Nestes casos o SGBD deverá usar técnicas como a de **simple locking** para permitir que para a mesma fonte de dados, somente um utilizador possa escrever, enquanto que muitos outros possam estar a ler. Contudo, continuamos a ter um problema quando um utilizador faz uma consulta à base de dados e um segundo

simple locking

atualiza o que foi consultado (neste cenário o primeiro utilizador não tomou conhecimento das alterações). Para resolver isto estas operações de **query-update** deverão ser vistas como sendo atómicas (como uma transação), sendo que deverão aplicar-se critérios de sincronização a estas, sendo que duas transações concorrentes não deverão poder escrever (e por vezes ler) no mesmo registo/campo.

query-update

Monitores de SGBDs e dados sensíveis

Pelos motivos apontados na secção anterior, os Sistemas de Gestão de Base de Dados deverão possuir unidades próprias e responsáveis pela integridade estrutural das suas bases de dados.

As unidades responsáveis por tal controlo deverão verificar os valores inseridos nas várias atualizações feitas pelos utilizadores, de forma a poder garantir consistência dos mesmos com as especificações dos campos de dados, registos ou restrições próprias da base de dados.

Sendo tais unidades denominadas por **monitores**, existem vários tipos, entre eles: monitores de comparação de intervalo de representação (unidade que verifica de um determinado valor pertence a um determinado intervalo ou gama de representação de um tipo de dado onde se está a tentar inserir); monitores de restrições de estado (unidade que descreve o estado da base de dados num determinado instante de tempo — por exemplo, para verificar a *flag* de *commit* de uma transação —, impondo regras de restrição para garantir integridade dos dados); e monitores de restrições de transição (unidade que descreve que condições são necessárias cumprir antes de alterar um ou mais dados da base de dados).

monitores

Ainda com os monitores, há vários dados que são considerados **sensíveis**, pelo que ainda precisam de proteção redobrada. Estes dados sensíveis dependem de base de dados em base de dados (porque dependem do negócio onde se aplicam), mas são tais que podem colocar uma empresa em risco caso haja uma fuga de informação — por exemplo, registos clínicos de pacientes de uma clínica. Esta sensibilidade, contudo, não é exclusiva ao facto de poderem ser vistos por quem não de direito, mas também à possibilidade de manipulações por terceiros não autorizados.

sensíveis

A sensibilidade de um dado, de facto, não é matéria clara e estrita, com que se possa caraterizar um conjunto de dados igual de base de dados em base de dados. Existem vários fatores que influenciam e tornam os dados sensíveis, vejamos: os dados poderão ser inerentemente sensíveis, se o valor por si for tão revelador que se torna sensível; os dados poderão ser provenientes de uma fonte sensível, neste caso, podendo colocar em risco a identidade da fonte dos mesmos; os dados poderão ser declarados sensíveis; os dados poderão ser provenientes de um registo que é considerado sensível; ou poderão ser sensíveis dada uma divulgação sua anterior, ou seja, por si, os dados não são sensíveis, mas juntamente com outros dados, o todo poderá ser sensível. [33]

Por si, a **divulgação** também é uma tarefa importante de ser designada em termos de bases de dados. Isto acontece porque há, por vezes, a necessidade de publicar parte dos dados para clientes, dependendo este da aplicação em causa. Contudo, o facto de parte de uma base de dados ser publicada indica que há limites que deverão ser estabelecidos. Assim sendo, existem alguns tipos de divulgação de dados sensíveis, entre os quais: dados exatos, sendo o valor exato de um determinado dado — a divulgação mais séria e cuidada de todas; determinação de limites, onde os dados sensíveis publicados deverão estar contidos entre um limite minorante e um majorante, ambos servindo para ocultar o restante trabalho; resultados negativos, sendo que obtendo um resultado negativo para uma consulta à base de dados com um dado sensível, um utilizador poderá concluir que um determinado valor possui um conjunto particular de resultados — considere-se o exemplo em que de uma lista de votantes efetivos, podemos concluir quem não votou; resultado de existência, quando a existência de um campo sensível num registo poderá ser, por si, informação sensível, dado que poderá revelar uma obtenção de dados oculta ou atividade de processamento sobre os mesmos; os valor provável, que se obtém pelo cruzamento de resultados

divulgação

de várias consultas, por onde podemos inferir a probabilidade de um determinado valor de um elemento.

Inferências e k-anonimato

Um dos vários ataques que se poderão conduzir numa base de dados, em termos semânticos, denomina-se de **ataques de inferência**. Por inferência, como abordámos em Engenharia de Dados e Conhecimento (a4s1), pretende-se referir uma forma de derivar novos dados a partir de outros dados. No nosso caso, aplicado ao nosso contexto de sensibilidade de dados, as inferências pretendem traduzir a criação de novos dados, considerados depois sensíveis, que foram criados através de dados que não são considerados sensíveis.

ataques de inferência

Este tipo de ataque poderá ser conduzido de uma de três formas diferentes. Num ataque de inferência direto, as várias consultas que são feitas terão uma mistura de regras de seleção de dados que usam tanto campos de registos que são sensíveis, como não-sensíveis. Aqui, o SGBD poderá ser despistado através das regras de seleção para campos não-sensíveis, que não possuem qualquer motivo para seleção de registos em particular.

Um outro tipo de ataque é o ataque de inferência indireto, que faz uma inferência de valores particulares obtidos por uma análise estatística de vários registos.

Por fim, existem os ataques de inferência por *tracker*. Aqui, a base de dados poderá esconder dados, quando um número muito reduzido de registos perfaz uma grande proporção dos dados revelados. Através deste tipo de ataque, é possível confundir o SGBD através de diferentes consultas (*queries*) que revelam os dados, combinando os vários resultados, conseguindo, o atacante, obter todas as informações sensíveis. [33]

Através da própria inferência podemos conseguir, então, obter novas informações sobre o nosso conjunto de dados, mesmo que a especificidade dos mesmos não for alta. Consideremos assim o exemplo da Figura 8.1, onde temos uma listagem dos vencimentos mensais de um conjunto de pessoas, obtido por uma equipa de investigação fictícia.

	nome	idade	localidade	vencimento
1	Alice	15 anos	Aveiro	1 200 €
2	Bruno	21 anos	Porto	1 100 €
3	Carla	30 anos	Coimbra	700 €
4	David	51 anos	Porto	1 500 €
5	Eduarda	24 anos	Porto	2 000 €
6	Filipe	43 anos	Aveiro	1 100 €

figura 8.1

Os dados que estão presentes na Figura 8.1 são considerados sensíveis, uma vez que a sua fonte é sensível — a sua fonte, aquando da sua entrevista, mostrou que não pretendia que a sua identidade fosse publicada. Agora, consideremos que esta equipa de investigação terá de publicar os seus resultados da investigação e precisa de publicar parte das informações da tabela da Figura 8.1 — como é que a equipa o poderá fazer?

A primeira tarefa a fazer poderá ser cortar o nome de cada pessoa inquirida, mostrando, no fim, uma tabela com a forma da Figura 8.2.

	nome	idade	localidade	vencimento
1	—	15 anos	Aveiro	1 200 €
2	—	21 anos	Porto	1 100 €
3	—	30 anos	Coimbra	700 €
4	—	51 anos	Porto	1 500 €
5	—	24 anos	Porto	2 000 €
6	—	43 anos	Aveiro	1 100 €

figura 8.2

Olhando para a tabela da Figura 8.2 podemos reparar que ainda conseguimos discriminar as várias pessoas envolvidas. Um dos dados que nos permite referir isso é o campo "idade", o que o torna sensível. Assim sendo, podemos aplicar-lhe o tipo de divulgação por limite, estabelecendo um limite entre vários níveis possíveis de idade, como podemos ver na Figura 8.3.

figura 8.3

	nome	idade	localidade	vencimento
1	—	entre 10 e 20 anos	Aveiro	1 200 €
2	—	entre 20 e 30 anos	Porto	1 100 €
3	—	entre 20 e 30 anos	Coimbra	700 €
4	—	entre 50 e 60 anos	Porto	1 500 €
5	—	entre 20 e 30 anos	Porto	2 000 €
6	—	entre 40 e 50 anos	Aveiro	1 100 €

Com a tabela da Figura 8.3 já podemos ter uma boa divulgação de dados sensíveis para o público, sendo que agora é difícil identificar quem é quem — mas não é tarefa impossível! Por exemplo, se conhecessemos a Eduarda iríamos ficar na dúvida se tem um vencimento de 1.100€ ou 2.000€.

A este tipo de redução de informação por anonimato damos o nome de **k-anonimato**, dado que tornámos os k valores de cada campo, anónimos.

k-anonimato

Segurança multi-nível

Da mesma forma que para o controlo de acesso a um sistema, os vários dados presentes numa base de dados também necessitam de estar enquadrados em classes de segurança como *unclassified*, *restricted*, *confidential*, *secret* ou *top secret*. Analogamente, as próprias consultas que são executadas também precisam de possuir um nível de segurança em relação à entidade que recebe o seu resultado. Assim sendo, há que ter mecanismos para previnir consultas de serem executadas (ou seja, do seu resultado ser visto) por entidades que não de direito por terem diferentes classificações de segurança.

Contudo, é possível que um determinado registo seja **poli-instanciado**, ou seja, que um registo com uma chave particular tenha um duplicado num nível de segurança distinto, possivelmente com um valor diferente também. Isto reduz o nível de precisão da informação presente na base de dados, sendo que a correção da informação depende diretamente da entidade que executa (ou recebe o resultado das) consultas.

poli-instanciado

Para resolver este problema podemos então criar **separações** dos vários dados presentes, por níveis de segurança já instaurados. Nessa linha de raciocínio, podemos aplicar várias **partições** à nossa base de dados, onde, dependendo do nível de segurança, consultas com um nível X seriam feitas exclusivamente à parte da base de dados de nível X. Esta solução é bastante fácil de implementar, contudo, cria redundância na informação, o que poderá causar problemas no acesso a registos com diferentes níveis de segurança.

separações

partições

Uma outra solução possível é a **cifragem** dos dados através de uma chave de segurança. Neste caso uma entidade de nível X precisa da chave k que decifra a informação de nível X. Aqui já não acontece redundância, dado que apenas teríamos controlo de acesso instaurado, sobre o mesmo conjunto de dados. Contudo, esta implementação cria algumas dificuldades especialmente pelo facto de haver a necessidade de ser feita uma decifra por cada consulta que é feita. Mais, em termos técnicos de criptografia, seria bom que campos iguais não gerassem o mesmo criptograma, uma vez que estaríamos sujeitos a ataques do tipo *known-plaintext* ou resultantes de análises estatísticas (problema que se resolve tendo chaves diferentes por registo ou diferentes vetores de inicialização por registo). Note-se, também, que neste cenário nenhum valor cifrado deverá ser atualizado depois de fornecer outro valor cifrado.

cifragem

Por fim, uma última solução será aplicar um **bloqueio de integridade**. Aqui a ideia é que cada item de dados está formada por três partes: um item de dados, uma etiqueta de sensibilidade (descrição se for inesquecível — não poder ser alterado —, único — não poder ser copiado para outro item de dados — ou escondido — não poder ser observado) e um *checksum*. Podendo ainda implementar esta solução num SGBD normal com uns *stored procedures* confiáveis para aplicar esta lógica, esta solução apenas peca por necessitar de espaço extra para preservar a informação acerca das etiquetas de sensibilidade e dos *checksums*.

bloqueio de integridade

Políticas de proteção de dados a nível internacional

Os vários níveis de proteção de dados preservados numa base de dados diferem de zona para zona do globo. A quantidade e a qualidade dos níveis de segurança implementados nos Estados Unidos é diferente dos níveis estabelecidos em Portugal. Isto acontece porque ligado à especificação dos níveis deve existir um conjunto de leis que estabelecem devidas consequências em caso de violação das regras impostas.

Em Portugal, a entidade responsável pela proteção de dados é a Comissão Nacional de Proteção de Dados (CNPD), que é quem necessita de autorizar todo o processamento de dados que envolve dados pessoais obtidos a partir de indivíduos.

Ataques a bases de dados por injeção de comandos em serviços

Nos vários serviços em que se tem uma base de dados como um componente vital da arquitetura de um sistema e uma outra máquina que disponibiliza uma interface de contacto com um cliente que faça uma consulta sobre a base de dados, é importante que se tenha segurança redobrada nas várias variações que se poderão permitir em relação ao *input* preenchido para consulta. Isto aconselha-se, uma vez que, em caso contrário, o sistema em causa poderá ser alvo de um **ataque por injeção**.

ataque por injeção

As tendências não são acaso, dado que já há bastante tempo que o ataque por injeção de comandos está no primeiro lugar das estatísticas dos ataques de segurança [34] da OWASP (acrónimo de *Open Web Application Security Project*). [35] Na descrição deste ataque temos que "as injeções (...) podem ocorrer quando dados não-confiáveis [o atacante] são enviados para um interpretador como parte de um comando ou consulta [a vítima] (...)" [34].

Grande parte dos SGBDs que processam bases de dados relacionais usam uma linguagem de descrição das mesmas denominada de **SQL**. Esta linguagem é, geralmente, usada noutras aplicações para aceder e efetuar consultas a uma base de dados que seja capaz de servir uma determinada aplicação ou serviço, muitas vezes deixando espaço para efetuar ataques de injeção, aqui denominados de **SQL injection** (ou SQLi). Aqui, um *browser*, por exemplo, poderá enviar *input* malicioso para um servidor.

SQL

SQL injection

Um exemplo bastante famoso de ataque por SQL injection ocorreu em junho de 2005, quando uma empresa de processamento de pagamentos por cartão de crédito foi assaltada através da sua rede, quando 263.000 cartões de crédito foram roubados da base de dados da companhia. Esta base de dados, por ter os cartões guardados sem qualquer tipo de cifra, deixou cerca de 43 milhões de cartões expostos [36].

Consideremos assim o Código 8.1, onde temos parte de um código para uma página de *login* em ASP.NET.

```
set ok = execute("SELECT * FROM Users
                  WHERE user=' " & form("user") & " '
                  AND   pass=' " & form("pass") & " '");
```

código 8.1

Olhando para o Código 8.1 o que é que poderá correr mal? Ora, para uma injeção de código SQL basta, no lugar o `user` ou da `pass` no formulário, inserir algum texto que provoque uma reação SQL. Mas para que seja interpretado, depois, como um comando SQL é necessário primeiro que se fechasse os apóstrofes delimitadores da *string* do SQL.

Consideremos assim que colocamos o texto `‘ or 1=1 --` no campo do utilizador no formulário. Se olharmos para a inserção desta mesma *string* no Código 8.1 podemos ver algo como o representado no Código 8.2.

código 8.2

```
set ok = execute(“SELECT * FROM Users
                 WHERE user=‘ ’ or 1=1 -- ‘
                 AND   pass=‘ ” & form(“pass”) & “ ‘”);
```

O que acontece no Código 8.2 é que a introdução do apóstrofe fechou a *string* em SQL e o resto foi interpretado como uma instrução, sendo que os dois traços inutilizaram tudo o que veio no resto da linha (indicação de comentário).

Quem faz um ataque como o do Código 8.2, também poderá fazer algo muito pior, como por exemplo apagar todas as tabelas (conteúdo e estrutura), com uma *string* `‘; DROP TABLE * –` ou executar um comando na máquina da base de dados com `‘; exec cmdshell ‘<comando>’/ ADD -- .`

Duas formas de resolver problemas de SQL *injection* são a limpeza do *input* dado pelo utilizador (técnicas de sanitização) — podemos bloquear apóstrofes como *input* — ou a utilização de procedimentos SQL para avaliar as entradas e unificar a forma como se recebem parâmetros provenientes de aplicações externas à base de dados [37].

9. Segurança na Máquina Virtual Java

Quando desenvolvemos uma aplicação, seja em que linguagem for, muitas vezes temos a necessidade de criar módulos que fazem uma tarefa específica. No entanto, se seguirmos uma organização modular em termos de programação podemos reutilizar módulos já criados, ao invés de reinventar a roda.

Contudo, com o uso de módulos realizados por outros desenvolvedores de *software*, como é que podemos estar seguros de que a implementação é confiável? Em Java, podemos usar classes implementadas por outros, mas será que as devemos usar, não sabendo se a fonte é confiável?

Sandboxing

O problema de trabalharmos com implementações feitas por outras pessoas é que estamos a tomar um risco muito grande de executar porções de código que não são propriamente desejáveis num determinado contexto, podendo suscitar, inclusive, consequências danosas para o nosso *software* ou até mesmo equipamento.

Muitas vezes, para evitar conflitos maiores e em caso de suspeição, é possível executar uma determinada implementação ou construção dentro de um ambiente considerado menos propício a más consequências — plataforma que dá pelo nome de **sandbox**.

sandbox

O conceito de *sandbox* é análogo a uma caixa de areia usada pelas crianças para construírem um castelo de areia, por exemplo. Se queremos que ela brinque com areia em casa não vamos querer que a criança sinta que pode usufruir da totalidade do espaço em casa para brincar, deixando um rasto de areia por toda a casa. Para corrigir isto criamos um espaço para que todo o tipo de brincadeiras possa ocorrer (onde ela possa fazer construções na areia, com e sem sucesso) dentro de determinados limites — uma caixa de areia, de onde esta não poderá sair.

A *sandbox* na área da computação é então uma barreira estabelecida entre o nosso ambiente computacional comum e um ambiente de teste (ou de produção, mas que não permite que más consequências abordem todos os recursos de uma máquina — apenas os que foram partilhados com a *sandbox*).

Modelo de segurança Java

No caso de uma aplicação realizada sobre a linguagem Java, o uso de classes de outros desenvolvedores de *software* é uma tarefa não muito crítica, uma vez que o próprio Java é

executado dentro de um ambiente *sandboxed*. Isto acontece porque todas as fases de manipulação do contexto Java (compilação e execução) são tarefas da responsabilidade da **máquina virtual** Java (JVM, de *Java Virtual Machine*), conceito o qual já incorpora o ambiente de *sandbox*. É, no fundo, uma consequência da JVM, ser uma *sandbox* segura para execução de programas Java.

máquina virtual

O facto de o Java ser executado sobre uma máquina virtual que garanta um ambiente *sandboxed* permite que políticas de segurança seja facilmente configuráveis para qualquer implementação feita e testada, com fáceis extensões das estruturas de controlo de acesso, de uso permitido em todas as classes Java.

O *Java Run-Time Environment* (vulgarmente abreviado de JRE) é assim um ambiente que possui um conjunto de regalias para este tipo de trabalhos, uma vez que é capaz de carregar novas classes (desde que invocadas no código), verificar a correção das classes carregadas (verificando integridade e consistência), compilar os *bytecodes* (somente para os métodos invocados, mantendo o *bytecode* original para reforçar possíveis validações que são executadas em *run-time*. Uma das vantagens do JRE é também a gestão de memória, uma vez que possui um *garbage collector* que limpa zonas de memória alocadas para objetos que já não estarão mais em uso. Mais, o código das classes é verificado na sua execução, havendo validações de integridade ao longo do seu período de *run-time* (verificando a existência de referências nulas (`null`), verificando o tipo de dados, as conversões entre tipos de dados com perda de precisão (*dynamic downcasting*), os limites dos *arrays*, ... Note-se que a indicação da visibilidade dos métodos e objetos também faz parte das verificações de integridade (mais em particular de segurança) em termos de controlo de acesso em *run-time*. [38]

Permissões em Java

Em Java, sempre que uma classe é carregada é testada o seu nível de permissões. Para isso usa-se a classe `ProtectionDomain` para verificar o estado das permissões de um código e mapear o código de uma classe ao domínio de proteção Java (JRE).

Uma **permissão** é assim algo que é permitido ou negado. Em Java, estas, são então subclasses da interface `java.security.Permission`. Existem pelo menos três possíveis implementações da interface `Permission`, são elas: o `BasicPermission` que possui um nome hierárquico e uma ação arbitrária (e booleana), entre `RuntimePermission`, `AWTPermission`, `ManagementPermission`, `NetPermission`, `PropertyPermission`, ...; o `FilePermission`, que indica um caminho para um ficheiro e a sua ação, entre leitura, escrita, execução ou remoção; e o `SocketPermission`, que permite indicar um porto, um terminal e uma ação, entre aceitação, conexão, escutar ou resolver.

permissão

Políticas de segurança e SecurityManager

Cada ambiente JRE mantém uma política de segurança configurada, determinando assim um conjunto de autorizações de permissão ou negação a recursos. Isto acontece através da definição de uma subclasse de `java.security.Policy`.

Note-se que existe sempre uma poítica instalada, indicada em `Policy.getPolicy()`, especificada com um ou mais ficheiros de configuração (geralmente posicionados em `$JAVA_PATH/lib/security/default.policy`. Ainda assim nada nos impede de sobrepor uma nova implementação de políticas de segurança, se bem que para isso é necessário haver uma permissão própria para executar `Policy.setPolicy(Policy)`.

Isto tudo funciona porque o Java possui um gestor de segurança intrínseco ao JRE. Pelo menos por cada JVM existe um `SecurityManager` que implementa uma política de segurança para uma determinada aplicação (especificando o que é permitido e o que não é). Isto, basicamente, ajuda a verificar se uma ação é permitida ser executada antes de a pedir, no contexto da *thread* executada.

A definição do `SecurityManager` é feita sob o pacote `java.lang` para a execução do gestor por omissão do Java. Contudo, este poderá ser redefinido, mas, mais uma vez, essa tarefa

requer uma permissão específica — permissão de execução de `setSecurityManager`. Esta permissão extra previne que classes maliciosas sejam executadas sobre um gestor de segurança pré-instalado.

O conteúdo desta classe `SecurityManager` é um conjunto de métodos de verificação de funções e privilégios, isto é, um conjunto de funções que permite a verificação de autorizações para ações específicas como as de leitura, escrita, execução, entre outras. Este módulo também usa a classe abstrata `AccessController` para fornecer as capacidades de controlo de acesso.

A classe abstrata `AccessController` é usada para decidir quando é que um acesso a um recurso crítico do sistema deverá ser permitido ou negado, de acordo com o conjunto de políticas em ação no momento da execução. Se o acesso for permitido para uma determinada classe em execução, então o código será marcado como privilegiado, o que afetará as subsequentes decisões de acesso. Neste passo, é obtido um *snapshot* do contexto atual de invocação, para que decisões de um módulo de controlo de um contexto diferente possam ser feitas acordando ao contexto guardado.

Carregamento de classes dinâmicas

Uma parte crítica da máquina virtual Java é o componente que carrega as classes Java, denominado de **class loader primordial**. Este componente é definido nas especificações da JVM e possui uma função muito importante de previnir o *spoofing* de nomes de bibliotecas Java sob o pacote `java.*`.

class loader primordial

Existem outros *class loaders*, que tanto podem ser definidos por utilizadores ou aplicações — podem ajudar uma aplicação a encontrar conteúdos de uma classe e descarregá-los, sendo que os *bytecodes* das classes são depois instalados pelo *class loader* da máquina virtual. Cada um destes *class loaders* definem espaços de nomes distintos por ambiente, onde cada classe é identificada com o *loader* que a carregou e classes num espaço de nomes não podem interagir com classes definidas noutros espaços de nomes. Este mecanismo, por outro lado, também acaba por permitir que diferentes versões de um mesmo nome de classe possam existir, o que tipicamente fica associado a código de diferentes origens.

As políticas de segurança para o carregamento de classes previnem, então, que não hajam carregamentos de pacotes `java.*` para além dos repositórios locais canónicos, de forma a evitar que haja uma sobreposição e reposição das classes Java básicas. As classes de diferentes servidores, por si, também não interagem entre si, uma vez que os seus domínios são diferentes, o que viola a regra de não-interferência entre códigos de diferentes fontes.

Então, para que uma classe seja dinamicamente carregada a máquina virtual Java faz os seguintes passos:

1. Localizar o binário da classe pedida (um ficheiro de extensão `*.class`);
2. Executar a tarefa de *parse* e de tradução para estruturas internas de dados para emulação das mesmas;
3. Reforçar as convenções de nomenclatura, verificando domínios, pacotes, classes, métodos e atributos, mas também níveis de acessibilidade (visibilidade) como público, privado ou pacote;
4. Verificar a correção do binário da classe, testando a integridade do ficheiro, da classe Java, do *bytecode* e depois do *run-time*;
5. Desempenhar as traduções necessárias de código e de metadados, permitindo que os métodos sejam passíveis de ser executados;
6. Inicializar a memória e passar o controlo para o motor de emulação.

Como podemos verificar pela enumeração acima, um dos passos consiste na verificação da correção do binário da classe, que testa primeiro a integridade do ficheiro `*.class` (verifica primeiro se o seu *magic number* é `0xCAFEBABE` — *magic number* das classes Java —, se os formatos usados estão corretos, se os componentes estão declarados e se os tamanhos

estão certos), depois testa a integridade da classe (verificando se esta possui superclasses e não é final, depois verificando se não é um *override* de um método final de uma superclasse, finalizando com a verificação da legalidade e legibilidade dos nomes e assinaturas dos vários métodos e atributos), a integridade do *bytecode* (através de uma análise ao fluxo de dados, à pilha de execução e aos vários tipos estáticos para argumentos de métodos e operandos *bytecode*), terminando com uma verificação de integridade em *run-time*, onde há testes às invocações de métodos.

E assim terminam os apontamentos da disciplina de Segurança. Existem muitos outros tópicos de relevo para a aplicação de segurança em redes informáticas, sendo que, dada a transversalidade do conteúdo, o seguimento deste estudo poderá ser feito em qualquer outra área de estudo, mas principalmente na área de redes computacionais.

65 SEGURANÇA

As referências abaixo correspondem às várias citações (quer diretas, indiretas ou de citação) presentes ao longo destes apontamentos. Tais referências encontram-se dispostas segundo a norma IEEE (as páginas Web estão dispostas de forma análoga à de referências para livros segundo a mesma norma).

[1] (September 14). *Security*. Available: https://www.merriam-webster.com/dictionary/security

[2] A. Zúquete, *Segurança em Redes Informáticas*, 4ª Aumentada ed. Lisboa: FCA - Editora de Informática, 2014

[3] C. P. Pfleeger, S. L. Pfleeger, and J. Margulies, *Security in computing*, Fifth edition. ed. Upper Saddle River, NJ: Prentice Hall, 2015, pp. xxxiii, 910 pages.

[4] A. Zúquete, "Introduction," 2017.

[5] D. Akkaya and F. Thalgott, "Honeypots in Network Security," Bachelor, School of Computer Science, Physics and Mathematics, Linnæus University, Småland, Sweden, 2010.

[6] A. Zúquete, J. P. Barraca "Vulnerabilities," 2017.

[7] T. Eisenberg, D. Gries, J. Hartmanis, D. Holcomb, M. S. Lynn, and T. Santoro, "The Cornell Commission: On Morris and the Worm," *Communications of the ACM,* vol. 32, pp. 706-709, June 1989.

[8] (October 11). *Sparta*. Available: https://en.wikipedia.org/wiki/Sparta

[9] (October 11). Figure of a Scytale. Available https://upload.wikimedia.org/wikipedia/commons/5/51/Skytale.png

[10] (October 11). Figure of the Enigma Machine. Available https://cdn.globalauctionplatform.com/7187abcf-14de-4d26-9a48-a48e012a3bd3/1f194fa9-87f4-45cd-b6db-a48e012de5c7/original.jpg

[11] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 1 ed. Boca Raton: Chapman & Hall/CRC, 2008.

[12] (October 31). Venona Project on Wikipedia. Available https://en.wikipedia.org/wiki/Venona_project.

[13] J. E. Canavan, Fundamentals of Network Security, 1 ed. Artech House, 2001.

[14] C. Paar and J. Pelzl, Understanding Cryptography: A Textbook for Students and Practioners. Berlin: Springer, 2010.

[15] N. Smart, Cryptography: An Introduction, 3 ed. Bristol, 2011.

[16] N. Smart, *Cryptography Made Simple*, 3 ed. Bristol: Springer, 2016.

[17] N. Ferguson, B. Schneier, and T. Kohno, Cryptography Engineering: Design Principles and Practical Applications, 1 ed. Indianapolis: Wiley Publishing, Inc., 2010.

[18] A. Zúquete, J. P. Barraca "Cryptography," 2017.

[19] R. Rivest, "Lecture 13: Public Key Encryption II," in *Course 6.857*, MIT, Ed., ed. Cambridge, Boston, 2017.

[20] J. Mittledorf and J. Cole. (2016, November 1st, 2017). *Statistical Evidence for Election Theft* [Article and Picture (Cartoon)]. Available: https://www.opednews.com/articles/Statistical-Evidence-for-E-by-Josh-Mitteldorf-Election-Integrity_Election-Integrity-161222-879.html

[21] R. Rivest, "Lecture 21: Quines, Anti-Virus Undecidability, Trusting Trust, SPKI & SDSI," in *Course 6.857*, MIT, Ed., ed. Cambridge, Boston, 2016.

[22] A. Zúquete, J. P. Barraca "Asymmetric Key Management," 2017.

[23] Michael A. Gurski (1995, November 3rd, 2017). *Privacy-Enhanced Mail (PEM)* [Web Article]. Available: https://www.csee.umbc.edu/~woodcock/cmsc482/proj1/pem.html

[24] W. Rankl and W. Effing, *Smart Card Handbook*, 4 ed. West Sussex: Wiley, 2010.

[25] K. Korosec (2017, January 13th, 2018). The 25 Most Common Passwords of 2017 Include 'Star Wars' [Web Article]. Available: http://fortune.com/2017/12/19/the-25-most-used-hackable-passwords-2017-star-wars-freedom/

[26] A. Zúquete "Authentication Protocols," 2017.

[27] A. G. Morgan (1997, January 15th, 2018). Linux Journal: Pluggable Authentication Modules for Linux [Web Article]. Available: http://www.linuxjournal.com/article/2120

[28] R. A. Smith, “A Comtemporary Look at Saltzer and Schroeder’s 1975 Design Principles”, *Lost Treasures*, pp.20-25, November 2012.

[29] Wikipedia (January 15th, 2018). Wikipedia: Chinese Wall [Web Article]. Available: https://en.wikipedia.org/wiki/Chinese_wall

[30] A. Zúquete “Security in Operating Systems," 2017.

[31] A. Zúquete “Practical Exercise: Execution of actions with different priviliges or in a confined environment,” 2017.

[32] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*, Third edition, New York: Springer-Verlag, 2011, pp. xx, 846 pages.

[33] A. Zúquete “Database Security," 2017.

[34] OWASP (2018). Top 10-2017 Top 10 - OWASP [Web Article]. Available: https://www.owasp.org/index.php/Top_10-2017_Top_10

[35] OWASP (2018). OWASP [Web Article]. Available: https://www.owasp.org/index.php/Main_Page

[36] B. Schneier (2018, June 23, 2005). CardSystems Exposes 40 Million Identities [Web Article]. Available: https://www.schneier.com/blog/archives/2005/06/cardsystems_exp.html

[37] J. Mitchell “Web Application Security,”, CS 155 — Spring 2017 — Stanford University, 2017.

[38] A. Zúquete “Java Virtual Machine Security," 2017.

Apontamentos de Segurança
1ª edição - janeiro de 2018

Autor: Rui Lopes
Agradecimentos: professor André Zúquete e professor João Barraca

Todas as ilustrações gráficas são obra de Rui Lopes e as imagens são provenientes das fontes bibliográficas divulgadas.

apontamentos