Introdução à Criptografia:
Algoritmos de Chaves Simétricas

Roteiros e tópicos para estudo por

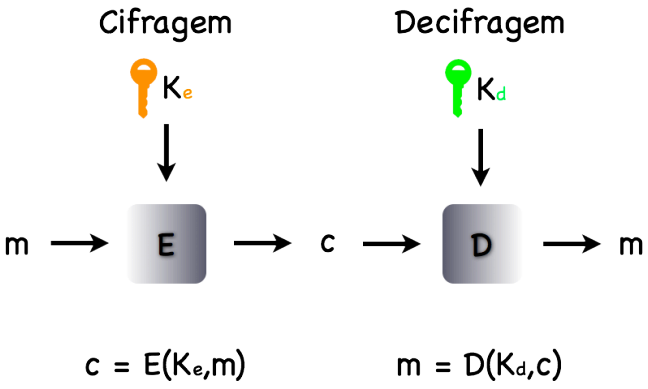
Vinícius da Silveira Serafim
professor@serafim.eti.br
http://professor.serafim.eti.br

Palavras-chave: algoritmo, chaves simétricas, chaves assimétricas

Simetria e Assimetria de Chaves

Até aqui estamos considerando que a chave **K** utilizada na cifragem deve ser a mesma chave utilizada na decifragem. Isso não é verdadeiro para todos os algoritmos criptográficos.

Voltemos ao nosso modelo de criptossistema na figura ao lado. A diferença marcante deste novo desenho com relação ao anterior é a diferenciação da chave utilizada na cifragem **K_e** da chave utilizada na decifragem **K_d**. Isso não significa que elas sejam necessariamente diferentes, é apenas uma distinção gráfica para tornar possível a explicação a seguir.



Algoritmos de Chaves Simétricas: $K_e = K_d$

Nesses algoritmos a chave de cifragem e a chave de decifragem é a mesma. Dizemos que há uma simetria de chaves entre os processos de cifragem e decifragem do algoritmo.

Atenção: é errado denominar esses algoritmos de *algoritmos simétricos*, o que é simétrico é a chave e não, necessariamente, o algoritmo.

Algoritmos de Chaves Assimétricas: $K_e \neq K_d$

Nesses algoritmos, uma chave é utilizada na cifragem e uma outra chave diferente é utilizada na decifragem. É claro que existe uma relação entre essas duas chaves e a estudaremos no próximo roteiro.

Há então, nesses algoritmos, uma assimetria de chaves entre os processos de cifragem e decifragem.

Atenção: é errado denominar esses algoritmos de *algoritmos assimétricos*, o que é assimétrico é a chave e não, necessariamente, o algoritmo.

Algoritmos de Chaves Simétricas: $K_e = K_d$

Esses algoritmos também são denominados de **algoritmos de chave secreta**. (PFLEEGER, 2006)

Uma grande vantagem desses algoritmos sobre os algoritmos de chaves assimétricas é a velocidade. Os algoritmos de chaves assimétricas são muito menos eficientes. (FERGUSON, 2003)

Essa vantagem é devida ao fato de operações simples como, por exemplo, XOR (ou-exclusivo), permutações e substituições serem a base dos algoritmos de chaves simétricas.

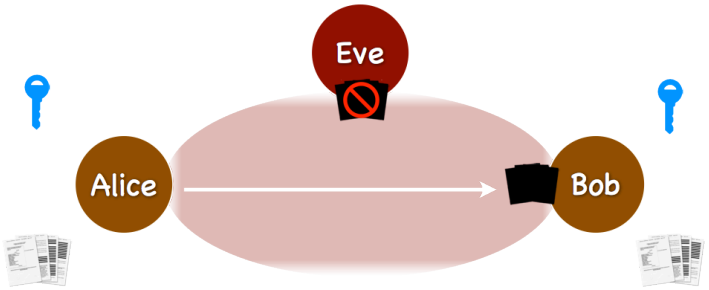
A seguir vamos estudar o que esses algoritmos nos oferecem, quais os seus problemas e algumas de suas aplicações.

O que oferecem?

Confidencialidade

Somente quem tem a chave **K** pode decifrar a mensagem **c** e ler a mensagem original **m**. (PFLEEGER, 2006)

Considere o cenário da figura a seguir em que uma parte, Alice, deseja enviar à outra, Bob, uma série de documentos através de um canal inseguro (ex.: e-mail).



O canal é considerado inseguro pois é sujeito à interceptação de terceiros, no caso Eve. Então Alice cifra os documentos **m** (texto em claro) com uma chave secreta **K** que tanto ela quanto Bob possuem.

$$Alice: c = E(K, m)$$

Alice envia para Bob o texto cifrado **c** que ela gerou. Essa ação vamos representar por:

$$Alice: c > Bob$$

Bob ao receber o texto cifrado o decifra usando a chave secreta que é de seu conhecimento e recupera os documentos **m**.

$$Bob: m = D(K,c)$$

Lembre-se de que Eve consegue capturar uma cópia do texto cifrado.

$$Eve < c$$

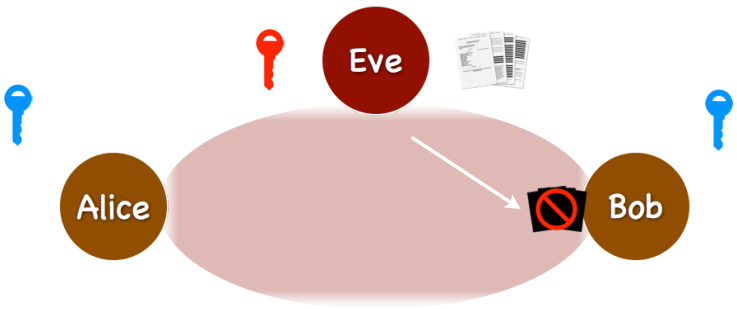
No entanto Eve não tem a chave secreta **K** utilizada por Alice e Bob e, portanto, não consegue realizar a decifragem do texto cifrado capturado.

$$Eve: ! m = D(K?, c)$$

A *confidencialidade* da mensagem é garantida desde que a chave secreta **K** seja mantida em segredo tanto por Alice quanto por Bob.

Autenticidade entre as partes

Somente quem possui a chave secreta **K** pode gerar um texto cifrado **c** válido. (PFLEEGER, 2006)



Quando Bob recebe uma mensagem de Alice, ele sabe que foi a Alice quem enviou a mensagem por duas razões:

- Somente ele e a Alice conhecem a chave secreta **K**; e
- Não foi ele, Bob, quem cifrou a mensagem.

As mesmas razões servem para Alice a respeito de uma mensagem recebida por ela.

Eve não conhece a chave e portanto não consegue enviar uma mensagem à Bob pretendendo que a mesma tenha sido gerada pela Alice e *vice-versa*. Se Eve cifrar uma mensagem com uma chave qualquer e enviar à Bob como tendo sido criada pela Alice, Bob, ao tentar decifrar com a chave correta, perceberá a tentativa de falsificação.

A *autenticidade* nesse esquema é frágil e só é garantida enquanto a chave secreta **K** for mantida em segredo pela Alice e por Bob. Veja também, logo adiante, o problema da irrefutabilidade.

Problemas

Irrefutabilidade não é garantida

Se a *autenticidade* e a *confidencialidade* podem ser garantidas enquanto a chave secreta estiver de posse apenas da Alice e do Bob, o mesmo não é verdadeiro pra a irrefutabilidade. (Se você não lembra o que é *isso* volte ao roteiro sobre os objetivos da segurança).

Imaginemos que Alice e Bob, após longo tempo de operações realizadas sob a proteção de algoritmos de chaves simétricas, se desentendam e entrem em uma disputa. Bob, por

exemplo, afirma que Alice enviou uma determinada mensagem e Alice nega a autoria da mesma.
Bob e Alice procuram então um terceiro, Tomas, que ambos confiam para tentar resolver a disputa.

Bob apresenta a mensagem cifrada à Tomas e afirma que, como não foi ele quem cifrou a mensagem com a chave secreta, somente conhecida por ele e pela Alice, certamente a mensagem foi gerada pela Alice.

Alice, ao contrário, nega a autoria da mensagem e afirma que Bob cifrou ele mesmo a mensagem e agora afirma ter sido ela quem a gerou.

Por mais confiança que Alice e Bob tenham em Tomas, a verificação de qualquer uma das afirmativas é impossível. Qualquer um, Alice ou Bob, pode ter gerado a referida mensagem. Tomas não tem solução para o caso e pode ponderar apenas sobre duas possibilidades:

- Um dos dois está mentindo;
- Ou alguém (ex.: Eve) teve acesso à chave secreta e os dois foram enganados.

Distribuição de chaves

Considerando novamente a situação anteriormente descrita na seção *confidencialidade*: Alice utiliza um algoritmo de chaves simétricas para garantir a confidencialidade dos documentos enviados à Bob pois o canal a ser utilizado para esse envio é inseguro.

Naquele momento simplesmente afirmei que tanto Alice quanto Bob conheciam a chave secreta. Mas essa afirmação ignora o seguinte problema: se Alice não tem um canal seguro para enviar os documentos à Bob, como ela envia a chave secreta que ela escolheu para ele?

Ou, em outras palavras, Alice precisa de um canal seguro para enviar a chave secreta à Bob, só então ela terá um canal seguro para os documentos. A solução, neste momento, é a Alice encontrar um outro canal (ex.: telefone) para passar a chave secreta ao Bob.

Se ela enviar a chave secreta ao Bob, pelo canal inseguro:

$$\text{Alice: } K > \text{Bob}$$

Eve, que tem a possibilidade de interceptar todas as comunicações entre Alice e Bob, também obtém a chave secreta.

$$\text{Eve} < K$$

E então Eve pode decifrar as mensagens trocadas entre Alice e Bob, comprometendo *confidencialidade*:

$$\text{Eve: } m = D(K,c)$$

E pode criar mensagens para Alice se passando pelo Bob e *vice-versa*, comprometendo a *autenticidade*:

$$Eve: c = E(K,m)$$

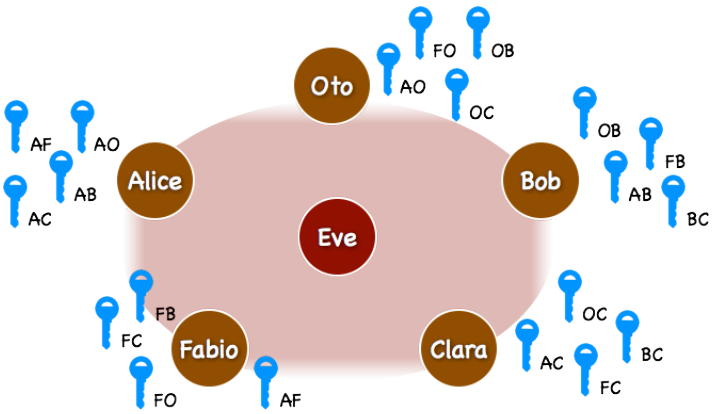
Gerenciamento das chaves

Uma vez que os algoritmos são públicos e que a segurança depende do segredo da chave **K** (lembra do princípio Kerckhoffs?), o gerenciamento das chaves de criptografia é algo crítico para qualquer criptossistema.

Vejamos o que diz Schneier sobre esse assunto: o gerenciamento das chaves é a parte mais difícil da criptografia. Projetar algoritmos criptográficos seguros não é fácil, mas para isso podemos contar com uma grande comunidade internacional de pesquisadores acadêmicos. Manter o segredo da chave secreta é muito difícil. (SCHNEIER, 1996)

Gerenciar chaves significa garantir que as mesmas sejam armazenadas e utilizadas de forma segura. A definição é simples, mas atingir esse resultado não é fácil.

No caso dos algoritmos de chaves simétricas o problema se mostra bem evidenciado quando são considerados outros participantes. Enquanto Alice se comunicava exclusivamente com Bob, bastava que ela armazenasse apenas uma chave secreta K_{ab} (ab significa Alice e Bob).



Agora veja o que acontece (na figura ao lado) quando outros três participantes passam a fazer parte do ambiente. Se todos quiserem conversar com todos, porém mantendo a *confidencialidade* e a *autenticidade* entre cada par, teremos 10 chaves secretas diferentes a serem gerenciadas, se contarmos as cópias de cada participante teremos 20 chaves a serem gerenciadas. Acrescente mais um participante e o número de chaves secretas diferentes sobe para 15, e um total de 30 chaves terão que ser gerenciadas.

Aplicações

Eis alguns exemplos de aplicações para os algoritmos de chaves simétricas:

- *Cifragem de arquivos*: proteção de alguns arquivos importantes em disco ou armazenados na núvem (ex.: em serviços como DropBox, Google Drive, SkyDrive) para futuro acesso. O programa WinZip, por exemplo, oferece cifragem do arquivo compactado com algoritmo de chaves simétricas.

- *Cifragem de backups*: proteção dos dados armazenados em mídias de backup (fitas, discos, etc.);

- *Cifragem de HDs e pendrives*: principalmente HDs de notebooks que podem ser facilmente alvo de furtos e roubos. Pendrives são ainda mais vulneráveis, pois além de serem furtados e roubados são frequentemente perdidos. O software TrueCrypt (gratuito e disponível em www.truecrypt.org) permite a cifragem de HDs e de pendrives; e

- Chaves de sessão: utilizadas para proteger dados trocados durante uma "conversação" entre duas partes. (mais detalhes quando estudarmos algoritmos de chaves assimétricas).

- *Armazenamento seguro de senhas*: o software KeePass (gratuito e disponível em keepass.info) é um aplicativo que armazena senhas e outras informações relevantes para autenticação em um pequeno banco de dados criptografado.

Como não usar

As formas erras de empregar algoritmos de chaves simétricas em um sistema são as mais variadas possíveis. Portanto, nesta seção, meu objetivo é trazer apenas alguns exemplos bastante comuns.

Armazenamento da chave secreta no código do sistema (*hardcoded*)

Não pense que a chave estará mais protegida no código do sistema, por este ser compilado, do que se fosse armazenada em um arquivo de configuração. Nesses casos, quem tem o código, tem a chave.

Tanto um quanto outro, se contiverem a chave, terão que ser adequadamente protegidos utilizando os recursos de controle de acesso fornecidos pelo sistema operacional.

No entanto, se estiver no código do sistema, uma eventual troca da chave exigiria uma atualização do sistema nos ambientes de produção nos quais esteja instalado. Mais grave ainda é a possibilidade de todas as empresas, usuárias daquele sistema, estarem usando a mesma chave.

Por outro lado, estando no arquivo de configuração, basta uma intervenção do usuário devidamente autorizado e a chave poderá ser alterada.

Armazenamento de senhas de usuários em um sistema

Alguns projetistas e desenvolvedores, visando proteger as senhas dos usuários dos seus sistemas, cifram essas senhas usando algoritmos de chaves simétricas. Nessa situação temos dois problemas graves:

- A chave secreta tem que estar disponível o tempo todo para o sistema para que este consiga autenticar os usuários. Voltamos aqui à questão do armazenamento da chave do item anterior; e

- O processo é reversível, ou seja, quem estiver de posse da chave secreta (ex.: um cracker ou mesmo um funcionário mal intencionado) pode decifrar a senha de todos os usuários. Mesmo que o sistema não seja de grande importância, não olvidemos o fato de que é muito comum os usuários utilizarem a mesma senha nos mais diversos sistemas e ambientes.

A forma correta de fazer isso é com *algoritmos criptográficos de hash*, os quais estudaremos mais adiante.

Algoritmos atualmente em uso

O algoritmo de chaves simétricas que pode ser considerado como o primeiro padrão a ser utilizado é o DES (Data Encryption Standard), criado pela IBM da década de 70 em resposta a uma solicitação do governo americano. O DES foi mundialmente adotado porém, devido ao seu pequeno tamanho de chave (apenas 56 bits), seu uso não é mais recomendado.

Como o DES já estava amplamente disponível nas mais variadas plataformas de hardware, visando dar uma sobrevida a essas implementações, foi criado o 3DES (triple DES). De fato o 3DES não é um novo algoritmo, é apenas o DES utilizado da seguinte maneira: a chave secreta é, na verdade, composta por três chaves distintas; primeiro é feita a cifragem de **m** com uma chave, depois o resultado é decifrado com outra chave e, o resultado dessa decifragem é finalmente cifrado com a terceira chave. A segurança fornecida por esse esquema é equivalente ao de uma chave de 112 bits. (PFLEEGER, 2006)

Apesar disso, não é recomendado o uso do DES e nem do 3DES em novos projetos. (FERGUSON, 2003) Tanto o DES, quanto o 3DES, ainda estão bastante presentes nas operações do nosso dia-a-dia, principalmente no que diz respeito à transações financeiras.

Em 2001 foi anunciado o AES (Advanced Encryption Standard), substituto do DES e do 3DES. Assim como o DES, o AES também foi criado por incentivo do governo americano. (PFLEEGER, 2006) O AES aceita chaves de 128 bits, 192 bits e 256 bits.

De par com o AES, estão o *Serpent* e o *Twofish*. Ambos finalistas da seleção de algoritmos, junto com o *Rijndael* (o vencedor e escolhido para ser o AES). Qualquer um dos três pode ser utilizado. Porém, lembre-se da escolha correta do *modo de operação*.

Todos os algoritmos citados nesta seção, são cifras de bloco.

Considerações sobre o tamanho da chave

Esse assunto é tema suficiente para umas duas aulas (ou mais). Porém aqui o espaço é curto e farei apenas algumas poucas considerações. Se você for trabalhar de alguma forma com criptografia, recomendo fortemente que estude as bibliografias citadas aqui e no meu site.

De uma forma bem clara e direta podemos dizer que: quanto maior a chave **K**, desde que o algoritmo respeite o princípio de Kerckhoffs, mais seguros estão os dados.

A medida que o número de bits da chave aumenta, o número de chaves possíveis aumenta exponencialmente. Ou seja, uma chave de 56 bits possibilita 2^{56} ou $7{,}2057594 \times 10^{16}$ chaves diferentes. Já um algoritmo com chaves de 128 bits possibilita 2^{128} ou $3{,}40282367 \times 10^{38}$. Se você está muito enferrujado na matemática, compare os expoentes destacados em vermelho, eles são o número de zeros que você deve adicionar ao número.

Alguns ataques, como os ataques de colisão, podem reduzir a efetiva proteção à metade dos bits da chave. (FERGUSON, 2003) Portanto, um algoritmo com uma chave de 128 bits, teoricamente, teria uma segurança efetiva de uma chave de 64 bits. Ferguson e Schneier, portanto, recomendam uso de chaves de 256 bits.

A desvantagem de usar chaves de 256 bits é que o processo de cifragem fica mais lento. Entretanto temos que considerar que temos processadores cada vais mais velozes e cada vez mais computadores possuem um processador criptográfico.

Senhas e Chaves

Por fim, cabe aqui algumas palavras para diferenciarmos senhas e chaves. Uma chave criptográfica tem um tamanho fixo em bits (ex: 128, 192,...) e ela é utilizada diretamente pelo algoritmo no processo de cifragem.

Há casos em que o usuário define diretamente a chave como, por exemplo, nas redes wireless com uso do protocolo WEP 64 bits (de fato 40bits, 5 caracteres) e 128 bits (de fato 104 bits, 13 caracteres).

Porém, o mais comum, é o usuário fornecer uma senha. Esta senha é então utilizada para geração de uma chave com o tamanho requerido pelo algoritmo.

Bibliografia

ANDERSON, R. J. Security Engineering: A Guide to Building Dependable Distributed Systems. 2a edição. Wiley, 14 de abril de 2008. 1080 pág.

FERGUSON, N. F.; SCHNEIER, B. Practical Criptography. 1a edição. Wiley, 17 de abril de 2003. 432 pág.

MENEZES, A.; OORSCHOT P. V.; VANSTONE, S. **Handbook of Applied Cryptography**. 1a edição. CRC Press, 16 de dezembro de 1996. 780 pág.

PFLEEGER, C. P. Security in Computing. 4a edição. Prentice Hall, 23 de outubro de 2006. 880 pág.

SCHNEIER, B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. 2a edição. Wiley, 18 de outubro de 1996. 758 pág.