UNIVERSIDADE FEDERAL FLUMINENSE

INSTITUTO DE COMPUTAÇÃO

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

**Eduardo Fabiano de Souza Henriques**

**Ítalo Portinho**

Estudos de Algoritmos de Criptografia numa Aplicação Cliente-Servidor

NITERÓI

2011

UNIVERSIDADE FEDERAL FLUMINENSE

INSTITUTO DE COMPUTAÇÃO

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

**Eduardo Fabiano de Souza Henriques**

**Ítalo Portinho**

Estudos de Algoritmos de Criptografia numa Aplicação Cliente-Servidor

Trabalho de conclusão final submetido ao curso de Graduação de Ciência da Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientadora:

Profa. Anna Dolejsi Santos, D.Sc.

NITERÓI

2011

Estudos de Algoritmos de Criptografia numa Aplicação Cliente-Servidor

Eduardo Fabiano de Souza Henriques

Ítalo Portinho

Trabalho de conclusão final submetido ao curso de Graduação de Ciência da Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Aprovada por:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Profa. Anna Dolejsi Santos, D.Sc. / IC – UFF

(Orientadora)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. / IC – UFF

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. / IC – UFF

Niterói, de Março de 2011.

“Aprender é a única coisa de que a mente nunca

se cansa, nunca tem medo e nunca se arrepende.”

*Leonardo da Vinci*

Agradecimentos

À nossa professora orientadora Anna Doloseji que sempre nos motivou a obter o melhor e pela confiança depositada neste trabalho.

Aos professores da Universidade Federal Fluminense que me ensinaram e ajudaram na formação do meu caráter. A todos vocês, o meu muito obrigado!

Ao meu grande amigo Nei, pelas inúmeras ajudas com a linguagem Java. Ajuda fundamental que proporcionou a evolução da aplicação apresentada neste trabalho.

Especialmente aos meus pais pela paciência e por acreditarem na minha capacidade incondicionalmente. Vocês me criaram, me deram amor, carinho e atenção. À minha namorada Gabriella, que sempre me apoiou e compreendeu os finais de semana que passei em casa, cercado por livros e no computador.

Resumo

Este projeto tem por objetivo realizar um estudo dos algoritmos de criptografia de chave simétrica: DES, 3DES, AES e Blowfish, numa aplicação cliente-servidor de vídeo de fluxo contínuo. Estes algoritmos serão implementados na camada de aplicação utilizando-se o framework JAVA JCE, garantindo no servidor, a criptografia dos pacotes de dados que contém o vídeo. O cliente, opera de maneira semelhante realizando a decriptografia destes pacotes. A aplicação é executada sob os protocolos RTP, RSTP e UDP.

**Palavras-chave**: Criptografia de Chave Simétrica, DES, 3DES, AES, Blowfish, Protocolos de Rede, RTP, RSTP, UDP.

Abstract

This project aims to conduct a study of algorithms for symmetric key encryption: DES, 3DES, AES and Blowfish, in a client-server application for streaming video. These algorithms will be applied at the application layer using the Java JCE framework, ensuring the server, encryption of data packets containing the video. The client operates similarly performing the decryption of packets. The application is executed on the protocols RTP, RTSP and UDP.

**Keywords**: Symmetric Key Cryptography, DES, 3DES, AES, Blowfish, Network Protocols, RTP, RTSP, UDP.

Abreviações

3DES - Triple Data Encryption Standard

AES - Advanced Encryption Standard

ASCII - American Standard Code for Information Interchange

CRC - Cyclic redundancy check

DES - Data Encryption Standard

DNS - Domain Name System

IBM – International Business Machine

IP – Internet Protocol

JCE - Java Cryptography Extension

JPEG - Joint Photographic Experts Group

RTP - Real-Time Transfer Protocol

RTSP - Real-Time Streaming Protocol

TFTP - Trivial File Transfer Protocol

UDP - User Datagram Protocol

URL - Uniform Resource Locator

Sumário

[1. Introdução 1](#_Toc289152598)

[2. Criptografia 3](#_Toc289152599)

[2.1. Histórico 3](#_Toc289152602)

[2.2. A evolução 4](#_Toc289152603)

[2.3. Cifras de substituição 5](#_Toc289152604)

[2.4. Cifras de transposição 5](#_Toc289152605)

[3. Criptografia de Chave Simétrica 7](#_Toc289152606)

[3.1. Modos de operação 7](#_Toc289152608)

[3.1.1. Modo ECB 7](#_Toc289152614)

[3.1.2. Modo CBC 8](#_Toc289152615)

[3.1.3. Modo CFB 8](#_Toc289152616)

[3.1.4. Modo OFB 10](#_Toc289152617)

[4.2. Algorimos simétricos 11](#_Toc289152619)

[3.2.1. XOR 11](#_Toc289152621)

[3.2.2. DES 12](#_Toc289152622)

[4. Criptografia de Chave Pública 20](#_Toc289152624)

[5.1. Protocolo Diffie-Hellman 20](#_Toc289152626)

[5.2. Protocolo Diffie-Hellman modificado 22](#_Toc289152627)

[5.3. RSA 22](#_Toc289152628)

[5. Aplicações multimídia 25](#_Toc289152629)

[6.1. Arquitetura da Internet 25](#_Toc289152631)

[6.2. Protocolo UDP 26](#_Toc289152632)

[6.3. Protocolo RTP 27](#_Toc289152633)

[6.4. Protocolo RTSP 29](#_Toc289152634)

[6. Aplicação cliente-servidor 33](#_Toc289152635)

[7.1. Cliente 33](#_Toc289152637)

[7.2. Servidor 35](#_Toc289152638)

[7.3. RTPpacket 35](#_Toc289152639)

[7.4. VideoStream 36](#_Toc289152640)

[7. Resultados 37](#_Toc289152641)

[8.1. O ambiente 37](#_Toc289152643)

[8. Trabalhos futuros 39](#_Toc289152644)

[9.1. Criptografia Quântica 39](#_Toc289152646)

[9.2. Autenticação do cliente num servidor KDC 39](#_Toc289152647)

[9. Conclusão 40](#_Toc289152648)

Lista de Figuras

Figura 1 - O modelo de criptografia, utilizando chaves simétricas [TANENBAUM, 2003]. 4

Figura 2 - Encadeamento e blocos de cifras. (a) codificação. (b) decodificação [TANENBAUM, 2003]. 8

Figura 3 - Modo CFB. (a) codificação. (b) decodificação [TANENBAUM, 2003]. 9

Figura 4 - Modo OFB. (a) codificação. (b) decodificação [TANENBAUM, 2003]. 10

Figura 5 - A camada de protocolos da Internet [KUROSE, 2006]. 23

Figura 6 - Estrutura do protocolo UDP [RFC, 768] 25

Figura 7 - Posição do RTP na pilha de protocolos [KUROSE, 2006]. 25

Figura 8 - Campos de cabeçalho do pacote RTP [RFC, 1889]. 26

Figura 9 - Integração entre cliente-servidor usando RTSP [KUROSE, 2006] 28

Figura 10 - Diagrama de estados para um cliente usando o RTSP [KUROSE, 2006]. 30

Figura 11 – Esquema do cabeçalho RTP usado na aplicação Servidor 34

Lista de Tabelas

[Tabela 1 - Aplicação de cifras de transposição em uma mensagem 3](#_Toc288536575)

[Tabela 2 – Permutação a ser aplicada na chave *K*. 3](#_Toc288536576)

[Tabela 3 - As sub-chaves são obtidas com deslocamentos a esquerda a aplicados ao bloco anterior de acordo com a tabela. 3](#_Toc288536577)

[Tabela 4 – A permutação PC-2 é aplicada aos blocos que foram deslocados à esquerda para obter-se as sub-chaves. 3](#_Toc288536578)

[Tabela 5 – Permutação que é aplicada ao bloco de 64 *bits* *M* para se obter *M’*. 3](#_Toc288536579)

[Tabela 6 – Regra usada para expandir um bloco de 32 *bits* para 48 *bits*. 3](#_Toc288536580)

[Tabela 7 – *S-Boxes* 3](#_Toc288536581)

[Tabela 8 – Permutação final para obter a função *f ( )*. 3](#_Toc288536582)

[Tabela 9 – O último passo do algoritmo DES: a permutação que resulta no bloco cifrado. 3](#_Toc288536583)

1. Introdução

Este trabalho tem por objetivo comparar e estudar quatro algoritmos de criptografia de chave simétrica bastante conhecidos e padronizados: DES (*Data Encryption Standard*), 3DES (*Triple Data Encryption Standard*), AES (*Advanced Encryption Standard*) e *Blowfish.* Como ambiente para realização dos experimentos utilizou-se uma aplicação cliente-servidor de fluxo de vídeo contínuo.

Mesmo antes do surgimento dos computadores, o homem sempre preocupou-se em proteger informações sigilosas para que pessoas não autorizadas não tivessem acessos a tais dados. Amores proibidos, comunicações em tempo de guerra, transações financeiras e declarações de imposto de renda são algumas motivações para desenvolver e aperfeiçoar a segurança na troca de mensagens. Com o advento da Internet, proteger essa troca de informações tornou-se mais complexo e necessário.

Estima-se que hoje existam dois bilhões de pessoas conectadas à Internet. O que representa 30% da população mundial. Com a popularização da internet pelo mundo, a demanda por transmissões de vídeo vêm aumentando. Transmissões em tempo real, sob demanda, vídeo conferências, *peer-to-peer*, são alguns dos exemplos do seu uso.

Essas transmissões podem conter informações sigilosas ou possuir um conteúdo que não deve ser público. Então faz-se cada vez mais necessário o controle para sua distribuição através da Internet.

No capítulo 4, explica-se como funciona a abordagem de algoritmos de chaves públicas e dois importantes algoritmos são apresentados: *Diffie-Helkman* e *RSA.* No primeiro é apresentado um exemplo através de um ataque conhecido uma possível quebra deste protocolo, mas que com uma ligeira modificação produz-se um protocolo aparentemente inatacável. No segundo algoritmo demonstra-se como são feitas a criptografia e a decriptografia, como o par de chaves deve ser calculado de maneira eficiente e porquê, baseado no problema da fatoração de números grandes em primos, não existe nenhum algoritmo que fatore as chaves deste protocolo em tempo polinomial.

A aplicação cliente-servidor é um exercício de programação do capítulo 7 do livro “Redes de Computadores e a Internet” [KUROSE, 2006] e seu código é parcialmente implementado e disponível para *download* no site do autor. Esta aplicação utiliza-se da visão de camadas de protocolos. O cliente usa o protocolo RTSP (*Real-Time Streaming Protocol*) para controlar ações sob o servidor. O servidor usa o protocolo RTP (*Real-Time Transfer Protocol*) para empacotar e enviar o vídeo em datagramas. Todos os pacotes trocados trafegam sob o protocolo de transporte UDP (*User Datagram Protocol*). Uma visão geral da camada de protocolos será apresentada no Capítulo 5**.**

Este trabalho foi desenvolvido utilizando-se a linguagem de programação JAVA e afim de facilitar a implementação dos algoritmos de criptografia utilizou-se o *framework* JCE (*Java Cryptography Extension*). A descrição técnica desta implementação e os desempenhos de cada algoritmo apresentados estão detalhados nos Capítulos 6 e 7**.**

Por fim nos Capítulos 8 e 9 apresenta-se possíveis trabalhos futuros para continuação deste trabalho e uma conclusão acerca de tudo o que foi apresentado e desenvolvido.

No Capítulo 2 apresentar-se-á um breve histórico da evolução da criptografia. Estes algoritmos serão melhores apresentados no Capítulo 3. Antes, os modos de operação, que basicamente são substituições monoalfabéticas utilizando-se de caracteres grandes grandes são explicados.

1. Criptografia

O uso da criptografia sempre esteve associado à proteção de segredos nacionais e estratégias. Sua história data desde o Antigo Egito, passando por duas grandes guerras mundiais e aplicando-se nos dias atuais à redes de computadores e os sistemas de comunicação, por exemplo, a Internet.

2. 1. ****Histórico****

**A palavra criptografia possui origem grega que significa “escrita secreta” e possui uma história de milhares de anos.**

**Em [KAHN, 1996] cobre-se os aspectos da criptografia utilizada desde os Egípcios, 4.000 anos atrás, até o Século XX, quando teve papel importante durante as duas guerras mundiais. Com a proliferação dos computadores e a crescente demanda por proteção das informações digitais, em 1977 adota-se um padrão de criptografia para informações não-classificadas** [FIPS, 1977]. Este algoritmo de criptografia tornou-se bastante conhecido e utilizado em transações comerciais e financeiras por todo o mundo [MENEZES, 1996].

**Em 1976 dois autores publicam um artigo que sugere um novo conceito para algoritmos criptográficos** [DIFFIE, 1976]. A segurança deste método é baseado no problema do logaritmo discreto.

**É importante fazer uma distinção entre cifras e códigos. Uma cifra é uma transformação de caractere por caractere ou de *bit* por *bit*, sem levar em conta a estrutura linguística da mensagem. Em contraste, um código substitui uma palavra por outra palavra ou símbolo. Atualmente não faz-se mais sentido o uso de códigos [TANENBAUM, 2003].**

Técnicas criptográficas permitem que um remetente embaralhe os dados de modo que um intruso não consiga obter nenhuma informação dos dados interceptados. O destinatário, é claro, deve estar habilitado a recuperar os dados originais a partir dos dados embaralhados [KUROSE, 2006].

* 1. A evolução

As mensagens a serem criptografadas, conhecidas como texto claro (*plain text*), são transformadas por uma função que é parametrizada por uma chave. Após este processo gera-se uma saída, conhecida como texto cifrado (*chiper text*). E esta mensagem é transmitida. Pressupõe-se que o inimigo, ou intruso, consiga interceptar essa mensagem cifrada. No entanto, ao contrário do destinatário pretendido, ele não conhece a chave para decriptografar o texto e, portanto, não pode fazê-lo com muita facilidade **[TANENBAUM, 2003].**

Até o fim da década de 70, todos os algoritmos criptográficos eram secretos, principalmente aqueles utilizados pela diplomacia e pelas forças armadas de cada país. A famosa máquina de criptografia ENIGMA [TERADA, 2000], usada pelos alemães até a Segunda Guerra Mundial era totalmente secreta. A sua chave era definida através da posição de rotores.

Até hoje não se conhece nenhum método matemático para provar que um algoritmo criptográfico é seguro. A maneira mais próxima deste ideal que se conhece é publicá-lo em conferências ou na Internet e tê-lo analisado pelos pesquisadores especializados que conheçam os métodos mais sofisticados para atacá-lo. Se o algoritmo passa por tal avaliação, a comunidade aceita-o como seguro em relação ao seu estado-da-arte [TERADA, 2000].

De forma geral, todos os algoritmos criptográficos, funcionam aplicando-se uma função matemática em um texto claro, tornando-o um texto cifrado. Para decifrar este texto basta-se aplicar a inversa desta função. Deve-se ressaltar que a função de codificação é conhecida, padronizada, publicada e disponível na Internet. Então o que garante a sergurança de um algoritmo é a sua chave secreta compartilhada apenas entre as duas entidades envolvidas. Os algoritmos criptográficos podem ser divididos em sistemas de chaves simétricas ou sistemas de chaves públicas.

O estudo de algoritmos de chaves públicas está fora do escopo deste trabalho. Em algoritmos de chave simétrica, ambas as chaves são idênticas e secretas.

A apresenta os componentes usados em uma criptografia simétrica. Aplicando-se uma função de criptografia *E ( )*, parametrizada com uma chave *K* ao texto claro *P*, gera-se um texto cifrado *C = EK(P)*. Este texto cifrado *C*, submetido à função matemática de decriptografia *D ( )*, com o parâmetro *K*, gera novamente o texto claro inicial, ou seja, *DK(EK(P)) = P*.

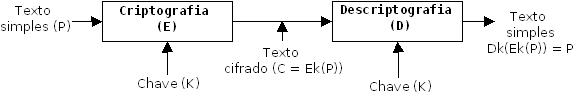


Figura - O modelo de criptografia, utilizando chaves simétricas [TANENBAUM, 2003].

Historicamente, os métodos de criptografia de chave simétrica são subdivididos em duas categorias: as cifras de substituição e as cifras de transposição. Cada uma dessas técnicas serviu de base para a criptografia moderna.

* 1. Cifras de substituição

Em uma cifra de substituição, cada letra ou grupo de letras é substituído por outra letra ougrupo de letras, de modo a criar um “disfarce”. Um exemplo prático é a Cifra de César.

Esta Cifra funciona substituindo-se cada letra da mensagem pela *k*-ésima letra sucessiva do alfabeto (a letra “*z*” é seguida pela letra “*a*”). Por exemplo, se *k = 3*, então “*a*” deve ser substituída por “*d*” e “*b*” transforma-se em “*e*”, e assim por diante. Por exemplo a mensagem “*estudar é importante*” tornar-se-á “*hvxgdu h lpsruxdqxh*”. Embora o texto cifrado pareça não ter nexo, não levar-se-ia muito tempo para quebrar o código se soubesse que foi usada a cifra de César, pois há somente 25 valores possíveis para as chaves. Existe uma série de variações a partir deste modelo, mas todos são facilmente quebrados através do método da força bruta e da análise estatística da linguagem (ou contagem de frequência) do texto aberto, por exemplo, na língua portuguesa a frequência das letras A, E, (geralmente mais frequentes), e Y, W (tipicamente menos frequentes) são particularmente distintas.

* 1. Cifras de transposição

As cifras de substituição preservam a ordem dos símbolos no texto claro, mas disfarçam esses símbolos. Por outro lado, as cifras de transposição reordenam as letras, mas não as disfarçam. A Tabela 1 mostra uma cifra de transposição mui­to comum, a transposição de colunas.

A cifra se baseia em uma chave que é uma palavra ou frase que não contém letras repetidas. Nesse exemplo, “*hamlet*" é a chave. O objetivo da chave é numerar as colunas de modo que a coluna 1 fique abaixo da letra da chave mais próxima do início do alfabeto e assim por diante. O texto claro “*estudar é importante*” é escrito horizontalmente, em linhas. O texto cifrado é lido em co­lunas, a partir da coluna cuja letra da chave seja a mais baixa. Obtêm-se assim o texto cifrado “*set dpt err umn tia aoe*”.

Tabela - Aplicação de cifras de transposição em uma mensagem

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **H** | **A** | **M** | **L** | **E** | **T** |
| 3 | 1 | 5 | 4 | 2 | 6 |
| e | s | t | u | d | a |
| r | e | i | m | p | o |
| r | t | a | n | t | e |

Embora a criptografia moderna utilize as mesmas ideias básicas da criptografia tradicional (transposição e substituição), sua ênfase é diferente. Tradicionalmente os algoritmos eram simples. Hoje em dia, acontece o inverso: o objetivo é tornar o algoritmo de criptografia tão complexo e emaranhado, mesmo que a mensagem cifrada seja interceptada totalmente, sem a chave não será capaz de captar qualquer sentido na mensagem.

Os algoritmos de criptografia podem ser implementados em *hardware* (obtendo-se maior velocidade) ou em *software* (aumentando sua flexibilidade) **[TANENBAUM, 2003].**

1. Criptografia de Chave Simétrica
2. 1. Modos de operação

Algoritmos de cifra de blocos realizam basicamente uma cifra de substituição monoalfabética que utiliza uma cadeia grande de caracteres (caracteres de 128 *bits*, por exemplo). Sempre que o mesmo bloco de texto claro é processado o mesmo bloco de texto cifrado é gerado. Caso codifique-se o mesmo texto, usando a mesma chave, o mesmo texto cifrado será obtido [TANENBAUM, 2003].

Existem quatro modos de operação para cifrar um bloco de texto claro. Estes modos são [TERADA, 2000] [FIPS, 1981]:

* Modo ECB – *Eletronic Code Book Mode*
* Modo CBC – *Cipher Block Chaining Mode*
* Modo s-CFB – *s-bit Cipher Feedback Mode*
* Modo s-OFB – *s-bit Output Feedback Mode*



### Modo ECB

O modo ECB é o mais natural dos quarto modos. Inicialmente divide-se o texto claro em blocos de comprimento fixo *t* da entrada do algoritmo, obtendo-se *n* blocos, *x1, x2,...,xn*, sendo o último bloco completado com brancos, se necessário, em seguida cifra-se cada bloco *xj* separadamente.

Nestas condições, se houver blocos repetidos no texto claro, haverão blocos correspondentes cifrados idênticos. Blocos de texto cifrados podem ser interceptados e sua ordem de transmissão alterada, por exemplo, se dois blocos cifrados repetidos correspondem ao salário de duas pessoas, pode-se substituir um dos blocos por outro bloco ilegível que talvez corresponda a um salário maior ou menor.

Vulnerabilidades deste tipo tornam este modo o pior dos quatro apresentados, e portanto raramente usado.

### Modo CBC

Este modo de cifras, permite evitar o problema apresentado no modo anterior, pois mesmo que ocorram blocos repetidos no texto claro, os bloco cifrados não serão idênticos.

Na Figura 2 (a), apresenta-se um valor inicial *VI* (que pode ser sempre fixo) é submetido a um *XOR* () com o primeiro bloco claro *P0*. Em seguida o bloco cifrado *C0 = E(P0 ⨁ VI)* é calculado e enviado ao destinatário e assim por diante.



Percebe-se que caso algum bloco seja alterado, digamos *P2*, para *P’2*, então *C2* será *C’2*. Portanto uma alteração mesmo que pequena de pelo um *bit* em *C2* acarreta alterações nos valores dos blocos *C2, C3*,.... A partir deste fato recomenda-se o uso do valor do último bloco *Cn* como valor de *hash* (usar em modo CBC como função de espalhamento).

A decriptografia é feita de maneira análoga como mostra-se na Figura 2 (b).

O encadeamento de blocos possui uma vantagem em relação ao modo ECB: o mesmo bloco de texto claro não resultará no mesmo bloco de texto cifrado. Assim, a criptoanálise será difícil. De fato essa é a principal razão para o seu uso [TANENBAUM, 2003].

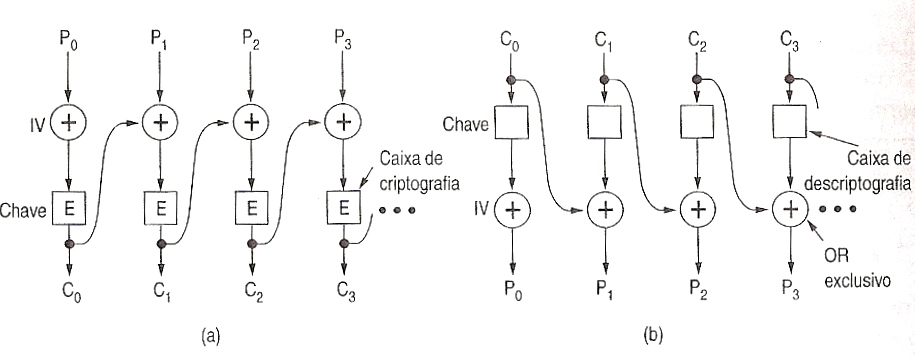


Figura - Encadeamento e blocos de cifras. (a) codificação. (b) decodificação [TANENBAUM, 2003].

### Modo CFB

O modo CFB permite cifrar blocos de texto claro de comprimento relativamente menor que no modo CBC. É possível definir tamanhos de blocos de um *byte* [TERADA, 2000]. Este é modo é utilizado em terminais interativos, por exemplo, onde menos de oito caracteres podem ser enviadas.

Na Figura 3 (a), o estado da máquina de criptografia é mostrado após os *bytes* 0 a 9 terem sido cifrados e enviados. Ao chegar o *byte* l0 do texto claro, o algoritmo opera sobre o registrador de deslocamento de 64 *bits* para gerar um texto cifrado de 64 *bits*. O *byte* mais à esquerda desse texto cifrado é extraído e submetido a uma operação *XOR* com *P10*. O registrador de deslocamento (*shift register*) é deslocado 8 *bits* à esquerda, fazendo *C2* ficar fora da extre­midade esquerda, e *C10* é inserido na posição que acabou de ficar vaga na extremidade direita, logo depois de *C9.* O conteúdo do registrador de deslocamento depende de todo o histórico anterior do texto claro; assim, um padrão que se repetir várias vezes no texto claro será cifrado de maneira diferente do texto cifrado a cada repetição. Como ocorre no encadeamento de blocos de cifras, é necessário um vetor de inicialização para dar início ao processo [TANENBAUM, 2003].

A decriptografia funciona exatamente como a criptografia como exibe-se na Figura 3 (b).

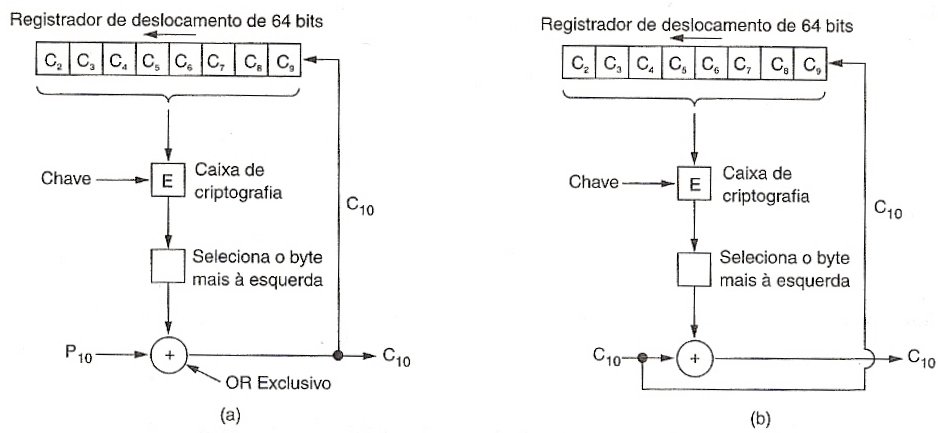


Figura - Modo CFB. (a) codificação. (b) decodificação [TANENBAUM, 2003].

Uma vantagem do modo CFB em relação ao CBC é que poucos *bits*, *s* *bits*, são cifrados de cada vez. Porém uma desvantagem deste modo é caso um *bit* do texto cifrado for invertido acidentalmente, os 8 *bytes* decodificados enquanto o *byte* defeituoso estiver na função de deslocamento serão danificados. Após as rodadas com este *byte* defeituoso termina, o texto claro será corretamente gerado [TANENBAUM, 2003].

### Modo OFB

Existem aplicações em que um erro de transmissão de 1 *bit* alteran­do 64 *bits* de texto claro provoca um impacto grande demais. Para essas aplica­ções, há o modo OFB, que possui uma operação muito parecida com o modo CFB. Ele funciona codificando um vetor de inicialização, com uma chave para obter um bloco de saída. O bloco de saída é então codificado, usando-se a chave para se obter um segundo bloco de saída. Em seguida, esse bloco é codificado para se obter um terceiro bloco e assim por diante. A sequência (arbitrariamente grande) de blocos de saída, chamada fluxo de chaves, é tratada como uma chave única e submetida a uma operação *XOR* com o texto claro para se obter o texto cifrado, como mostra a Figura 4 (a). O vetor de inicialização só é utilizado na primeira etapa. O fluxo de chaves é inde­pendente dos dados, e portanto pode ser calculado com antecedência, se neces­sário, e é completamente insensível a erros de transmissão.

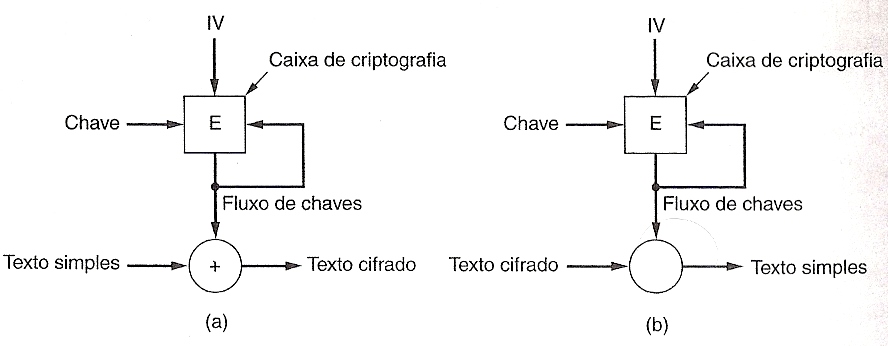


Figura - Modo OFB. (a) codificação. (b) decodificação [TANENBAUM, 2003].

A decodificação ocorre gerando-se o mesmo fluxo de chaves no lado receptor, Figura 4 (b). Como o fluxo de chaves só depende do vetor de inicialização e da chave, ele não é afetado por erros de transmissão no texto cifrado. Desse modo, um erro de 1 *bit* no texto cifrado transmitido gera apenas um erro de 1 *bit* no texto claro decodificado.

* 2. Algorimos simétricos

### XOR

Ou-exclusivo chamada também “disjunção exclusiva”, conhecido geralmente por *XOR*, é uma operação lógica em dois operandos que resulta em um valor lógico verdadeiro se e somente se exatamente um dos operandos tem um valor verdadeiro.

Em criptografia, esta operação pode ser aplicada em um texto claro, a fim de criar uma versão cifrada. Através de um algoritmo básico:

*(textoSimples ⨁ chave) = textoCifrado*

Pode-se dizer que o *XOR* realiza uma operação reversível, pois se aplicarmos *A ⨁ B* e reaplicarmos o *XOR* no resultado com o mesmo *B*, teremos *A*, como vemos a seguir:

*( A ⨁ B ) ⨁ B = A*

É baseado na reversibilidade da operação *XOR* que a técnica é usada em algoritmos criptográficos. Esta é uma técnica extremamente usada como um componente, ou uma etapa, em algoritmos de criptografia mais complexos. Usada isoladamente, com a mesma chave repetidas as vezes, torna-se vulnerável a ataques do tipo texto claro conhecido, já que *textoSimples* ⨁ *textoCifrado* = chave. A criptografia *XOR* é um tipo de algoritmo usado quando a segurança não é realmente importante. Porém, caso use-se uma chave randômica, que nunca se repita e com mesmo tamanho da mensagem, esta criptografia torna-se mais segura. O algortimo O*ne-time pad* é um exemplo.

Em criptografia, O*ne-time pad* (OTP), inventado em 1917, ou cifra de chave única, é um algoritmo de criptografia onde o texto claro é combinado com uma chave aleatória ou uma “*pad*” que seja tão grande quanto o texto claro e é usado somente uma vez. Uma adição modular (por exemplo *XOR*) é usada para combinar o texto claro com a *pad*. Se a chave for verdadeiramente aleatória, nunca reutilizada, e mantida em segredo, a *one-time pad* pode ser inquebrável. Também provou-se que toda a cifra teórica inquebrável deve usar chaves com as mesmas exigências que chaves de OTP. Não foi atendida como padrão de mercado porque é trabalhoso gerar sempre chaves aleatórias grandes e manter em sigilo.

### DES

O DES foi um padrão de criptografia de chaves simétricas desenvolvido em 1977 pelo *US Nation Bureau of Standards*. Este algortimo codifica um texto claro em porções de 64 *bits*, usando uma chave de 64 *bits*. Porém, para cada grupo de 8 *bits* há um *bit* de paridade, sendo assim a chave tem efetivamente 56 *bits* de comprimento, e assim é citado o tamanho da sua chave.[KUROSE, 2006]

Em [RIVEST, 1990] esboça-se o algoritmo DES da seguinte forma: Seja *M* a representação binária de uma mensagem em texto claro, de tamanho 64 *bits* e *K* uma chave de 56 *bits* aos quais são adicionados os *bits* de paridade, de forma que *K* tenha 64 *bits*. Inicialmente a chave é permutada de acordo com a Tabela 2. Nesta tabela os *bits* de paridade foram excluídos (os *bits* 8, 16, 24, 32, 40, 48, 56 e 64 estão ausentes), portanto, esta operação só é efetuada depois da verificação de integridade da chave. Como a primeira entrada da tabela é “57”, isto significa que o 57º *bit* da chave original *K* torna-se o primeiro *bit* da chave permutada. O 49º *bit* da chave original transforma-se no segundo *bit* da chave permutada. O 4º *bit* da chave original é o último *bit* da chave permutada. Apenas 56 *bits* da chave original aparecem na chave permutada. Obtém-se então a chave *K+*, resultado da permutação, que possui 56 *bits*.

Tabela – Permutação a ser aplicada na chave *K*.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PC-1** | | | | | | |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 |
| 1 | 58 | 50 | 42 | 34 | 26 | 18 |
| 10 | 2 | 59 | 51 | 43 | 35 | 27 |
| 19 | 11 | 3 | 60 | 52 | 44 | 36 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 |
| 7 | 62 | 54 | 46 | 38 | 30 | 22 |
| 14 | 6 | 61 | 53 | 45 | 37 | 29 |
| 21 | 13 | 5 | 28 | 20 | 12 | 4 |



**Formação das Subchaves**

A chave *K+* é dividida em duas metades, esquerda *C0* e direita *D0*, de 28 *bits*. Com *C0* e *D0*definidas criam-se 16 blocos *Cn* e *Dn*, para *1 <= n <= 16*, onde cada par é formado pelo par anterior *Cn-1* e *Dn-1*usando o esquema de deslocamentos a esquerda da Tabela 3.

Tabela - As sub-chaves são obtidas com deslocamentos a esquerda a aplicados ao bloco anterior de acordo com a tabela.

|  |  |
| --- | --- |
| Número da iteração | Número de deslocamentos à esquerda |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |
| 7 | 2 |
| 8 | 2 |
| 9 | 1 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |
| 13 | 2 |
| 14 | 2 |
| 15 | 2 |
| 16 | 1 |

As 16 sub-chaves *Kn*, para *1<=n<=16*, são obtidas a partir de cada bloco *CnDn*de acordo com a Tabela 4.

Tabela – A permutação PC-2 é aplicada aos blocos que foram deslocados à esquerda para obter-se as sub-chaves.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PC-2** | | | | | | |
| 14 | 17 | 11 | 24 | 1 | 5 |  |
| 3 | 28 | 15 | 6 | 21 | 10 |  |
| 23 | 19 | 12 | 4 | 26 | 8 |  |
| 16 | 7 | 27 | 20 | 13 | 2 |  |
| 41 | 52 | 31 | 37 | 47 | 55 |  |
| 30 | 40 | 51 | 45 | 33 | 48 |  |
| 44 | 49 | 39 | 56 | 34 | 53 |  |
| 46 | 42 | 50 | 36 | 29 | 32 |  |

**Criptografar bloco de 64 *bits***

A partir da representação em 64 *bits* *M*, da mensagem em texto claro, obtém-se a permutação *M’* segundo a Tabela 5. Então *M’*  é dividida em 2 blocos de 32 *bits*, *L0* e *R0*. Para cada par *LnRn*, utiliza-se a sub-chave *Kn*e a função *f ( )* para aplicar 16 iterações até obter o bloco cifrado *L16R16* de acordo com a fórmula:

*Ln = Rn-1*

*Rn = Ln-1 + f(Rn-1,Kn)*

Tabela – Permutação que é aplicada ao bloco de 64 *bits* *M* para se obter *M’*.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| IP - *Initial Permutation* | | | | | | | |
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

Para calcular *f ( )*, primeiramente expande-se cada bloco *Rn-1* de 32 para 48 *bits*, para se ajustar ao tamanho das sub-chaves. Isto é feito de acordo com a Tabela 6, uma tabela de seleção que repete alguns dos *bits* em *Rn-1*. Sendo *E ( )* a operação de expansão, denotará-se por *E(R)* o bloco *R* original expandido em 48 *bits*.

Tabela – Regra usada para expandir um bloco de 32 *bits* para 48 *bits*.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Expansion (E)* | | | | | | | |
|  | 32 | 1 | 2 | 3 | 4 | 5 |  |
|  | 4 | 5 | 6 | 7 | 8 | 9 |  |
|  | 8 | 9 | 10 | 11 | 12 | 13 |  |
|  | 12 | 13 | 14 | 15 | 16 | 17 |  |
|  | 16 | 17 | 18 | 19 | 20 | 21 |  |
|  | 20 | 21 | 22 | 23 | 24 | 25 |  |
|  | 24 | 25 | 26 | 27 | 28 | 29 |  |
|  | 28 | 29 | 30 | 31 | 32 | 1 |  |

A seguir, no cálculo de *f ( )*, fazemos um *XOR* na saída *E (Rn-1)* com a chave *Kn*. O motivo de se utilizar o *XOR* lógico é porque este é reversível. Se *A ⨁ B = C*, então *A ⨁ C = B* e *B ⨁ C = A*. A reversibilidade é importante para poder reverter-se o processo quando ocorrer a decodificação da mensagem cifrada. A operação *XOR* resulta em um bloco de 48 *bits* que será quebrado em 8 blocos de 6 *bits* cada. Cada grupo de 6 *bits* será usado como endereço em tabelas denominadas “*S boxes”*. Os *bits* de 1 a 6 são *B1*, *bits* de 7 a 12 são *B2*, e assim por diante com *bits* de 43 a 48 sendo *B8*, como descrito abaixo:

*Kn + E(Rn-1) = B1B2B3B4B5B6B7B8*

Seja *Sn* é a função definida de acordo com a Tabela 7 e *B* é um bloco de 6 *bits*, então *Sn (B)* é determinada da seguinte maneira: O primeiro e o último *bit* de *B* representam um número na base 2 com valor decimal entre 0 e 3 (ou binário 00 a 11). Que este número seja *i*. Os 4 *bits* centrais de *B* representam um número na base 2 com valor decimal entre 0 e 15 (ou binário de 0000 a 1111). Que este número seja *j*. Procura-se o número na tabela localizado na *j*-ésima coluna e na *i*-ésima linha. Esse número varia de 0 a 15 e é unicamente representado por um bloco de 4 *bits*. Este bloco é a saída *Sn (B)* de *Sn* para a entrada *B*.

Tabela – *S-Boxes*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S1 | | | | | | | | | | | | | | | |
| 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |
| S2 | | | | | | | | | | | | | | | |
| 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |
| S3 | | | | | | | | | | | | | | | |
| 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |
| S4 | | | | | | | | | | | | | | | |
| 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |
| S5 | | | | | | | | | | | | | | | |
| 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |
| S6 | | | | | | | | | | | | | | | |
| 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |
| S7 | | | | | | | | | | | | | | | |
| 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |
| S8 | | | | | | | | | | | | | | | |
| 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

Finalmente obtém-se *f* *( )* pela permutação da concatenação da saída das *S-boxes* tendo os blocos *Bn* como entrada. A permutação é dada pela Tabela 8.

*f= P(S1(B1)S2(B2)S3(B3)S4(B4)S5(B5)S6(B6)S7(B7)S8(B8))*

Tabela – Permutação final para obter a função *f ( )*.

|  |  |  |  |
| --- | --- | --- | --- |
| Permutação P | | | |
| 16 | 7 | 20 | 21 |
| 29 | 12 | 28 | 17 |
| 1 | 15 | 23 | 26 |
| 5 | 18 | 31 | 10 |
| 2 | 8 | 24 | 14 |
| 32 | 27 | 3 | 9 |
| 19 | 13 | 30 | 6 |
| 22 | 11 | 4 | 25 |

O passo final do algoritmo DES é realizar a permutação *IP-1*, como na Tabela 9, sobre o bloco *L16R16* para obter o bloco cifrado.

Tabela – O último passo do algoritmo DES: a permutação que resulta no bloco cifrado.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| IP-1 | | | | | | | |
| 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 |
| 39 | 7 | 47 | 15 | 55 | 23 | 63 | 31 |
| 38 | 6 | 46 | 14 | 54 | 22 | 62 | 30 |
| 37 | 5 | 45 | 13 | 53 | 21 | 61 | 29 |
| 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 |
| 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 |
| 34 | 2 | 42 | 10 | 52 | 18 | 58 | 26 |
| 33 | 1 | 41 | 9 | 49 | 14 | 57 | 25 |

O projeto do DES é inspirado em outro algoritmo da IBM chamado LUCIFER, que possui entrada e saída também de 64 bits mas chave mais longa: 128 bits. Acredita-se que o encurtamento da chave tenha sido proposital, a pedido do *National Security Agency* (NSA) nos EUA. Antes da sua adoção como padrão, DES foi alvo de fortes críticas que persistem até hoje. A primeira crítica é o decréscimo do comprimento da chave de 128 *bits* para 56 *bits*: isso torna muito mais economicamente viável o cálculo da chave secreta, mesmo por força bruta, i.e*.,* enumerando as 256 chaves. A segunda crítica é o segredo mantido quanto aos critérios de projeto da estrutura interna, os *S-boxes*. Os usuários do DES não possuem qualquer garantia de ausência, de pontos vulneráveis nos *S-boxes* que permita à NSA decriptografar sem conhecer chave em uso.

Em 1977, dois pesquisadores de criptografia de Stanford, Diffie, W. e Hellman, M.E., projetaram uma máquina para decifrar o DES e estimaram que ela poderia ser montada por um custo de 20 milhões de dólares. Com base em um pequeno trecho de texto simples e no texto cifrado correspondente, essa máquina poderia descobrir a chave através de uma pesquisa exaustiva do espaço de chaves de 256 entradas em menos de um dia. Atualmente, essa máquina custaria bem menos de um milhão de dólares.

Até 1999 o governo norte-americano exigia que o DES usado internacional­mente fosse restrito ao uso de 40 *bits* na chave. Mas uma chave de 40 *bits* corresponde a cerca de 1 trilhão de chaves possíveis, e em 18 minutos à razão de 1 bilhão de tentativas de chaves por segundo, pode-se calcular a chave correta de 40 bits.

Um computador distribuído na Internet chamado *DeepCrack* possui ve­locidade média de 90 bilhões de chaves por segundo. Em 4,5 dias o *DeepCrack* consegue achar uma chave correta de 56 *bits*. No seu pico de desempenho, *DeepCrack* consegue velocidade de 250 bilhões de chaves por segundo [DISTRIBUTED, 2011].

3DES

AES

Blowfish

1. Criptografia de Chave Pública

Até a década de 70, a comunicação cifrada exigia que as duas partes comunicantes compartilhassem a chave simétrica para criptar e decriptar. Para haver o compartilhamento dessa chave era preciso uma comunicação, mas caso essa comunicação fosse interceptada a chave não seria mais secreta.

1. 1. Protocolo Diffie-Hellman

Em 1976 dois autores Diffie, W. e Hellman, M.E. publicam um artigo [DIFFIE, 1976] que possui uma abordagem segura e radicalmente diferente dos algoritmos de chave simétrica.

Neste artigo foi proposto um modelo de criptografia chamado Modelo de Chave Pública em que cada usuário possui um par de chaves (*S, P*) sendo *S* a sua chave particular (secreta e guardada de forma segura) e *P* pública (facilmente acessada e disponibilizada publicamente). Estas chaves são relacionadas matematicamente de tal forma que:

* Se *x* denota um texto claro, e *S( )* a aplicação da chave *S*, que transforma *x* em *S(x) = y* então *P(y) = x*, onde *P( )* denota a aplicação da chave *P*. Ou seja *S* é a chave inversa da chave *P – P(S(x)) = x*;
* cálculo do par de chaves *(S, P)* é de tempo polinomial, ou seja, computacionalmente fácil;
* Para calcular *S* a partir do conhecimento de *P*, é computacionalmente difícil, ou seja, não se conhece um algoritmo para o cálculo em tempo polinomial;
* Os cálculos de *P( )* e *S( )* são computacionalmente fáceis para quem conhece as chaves;
* É computacionalmente difícil calcular *S( )* sem conhecer a chave *S*.

A seguir lista-se os passos deste protocolo para que seja possível combinar-se uma chave secreta *KCS* entre um cliente e um servidor. Previamente ambos conhecem publicamente dois inteiros *g* e *p*, sendo *p* um primo longo e *g* tal que 0 < *g* < *p* gerador de Z\*p [TERADA, 2000].

1. O cliente escolhe um número aleatório *SC*, 1 < *SC* < *p* – 2;
2. O servidor escolhe um número aleatório *SS*, 1 < *SS* < *p* – 2;
3. O cliente calcula *tC* = *gSc* mod *p* e o envia para o servidor;
4. O servidor calcula *tS* = *gSs* mod *p* e o envia para o cliente;
5. O cliente calcula *(tS)Sc* mod *p* = *KCS*. Nota-se que *(tS)Sc* = [*gSs* mod *p*]*Sc*= *gSsSc*mod *p*.
6. O servidor calcula *(tC)Ss* mod *p* = *KCS*. Nota-se que *(tC)Ss* = [*gSc* mod *p*]*Ss*= *gScSs*mod *p*.

Este protocolo é baseado na dificuldade computacional do Problema do Logaritmo Discreto, que é “Dado um primo *p* e inteiros *g*, *t* : 0 < *g*, *t* < *p*, calcular um inteiro *s* tal que *t* = *gs* mod *p.”* [TERADA, 2000].

Entretanto, este protocolo não garante a autenticidade nem do cliente e nem do servidor, portanto pode ser quebrado por um intruso ativo (ataque conhecido como *man-in-the-middle*), como apresenta-se a seguir [TERADA, 2000]:

1. Quando o cliente envia *tC*, o intruso bloqueia *tC* e envia para o servidor outro inteiro *tI* = *gSi* mod;
2. Cliente 🡪 *tC* 🡪 Intruso 🡪 *tI* 🡪 Servidor
3. O servidor recebe *tI* como se fosse *tC* e faz os cálculos necessários de acordo com o protocolo, estabelecendo uma chave *KSI* com o Intruso, pensando falsamente que esta é a chave entre ele e o cliente;
4. O intruso bloqueia *tC* e envia outro inteiro no seu lugar que pode ser o *tI* usado antes;
5. O cliente recebe *tI* como se fosse *tS* e faz os cálculos estabelecendo uma chave *KCI* com o Intruso, pensando falsamente que esta é a chave entre ele e o servidor.
6. A partir deste ponto tem-se a seguinte interceptação de pacotes por parte do Intruso
7. Cliente ⬄ *KCI* ⬄ Intruso ⬄ *KSI* ⬄ Servidor.

Porém uma ligeira modificação produz um protocolo útil e aparentemente inatacável [TERADA, 2000].

* 1. Protocolo Diffie-Hellman modificado

Com esta modificação evita-se o ataque *man-in-the-middle*, para tanto, deve-se incluir um terceiro valor inteiro à chave pública, tornando-se (*p, g, T*) onde *T = gS* mod *p* [TERADA, 2000].

Portanto, o cliente publica previamente a sua chave pública (*pC, gC, TC*) e mantém sua chave secreta *SC* tal que *TC* = (*gC*)*Sc* mod *pC*. (analogamente o servidor faz o mesmo). O protocolo modificado fica como a seguir, quando o cliente desejar enviar uma mensagem *x* para o servidor [TERADA, 2000]:

1. O cliente escolhe *SC* e calcula (*gS*)*Sc* mod *pS*= *UC* e *KCS* = (*TS*)*Sc* mod *pS*; usa *KCS* para cifrar: *KCS*(*x*) = *y* e envia *y* e *UC* para o servidor:
2. Cliente 🡪 *UC* = (*gS*)*Sc* mod *pS*, *y* = *KCS*(*x*) 🡪 Servidor
3. O servidor calcula [*UC*]*Ss* mod *pS*= [(*gS*)*Sc* mod *pS*]*Ss* mod *pS* = *KCS* e usa *KCS* para decifrar *y* e obter *x*.

Desta forma não há mais necessidade de um diálogo entre cliente e servidor [TERADA, 2000].

* 1. RSA

O algoritmo RSA, publicado em 1978 e cujo os inventores são Ron Rivest, Adi Shamir, Leonard Adlemar, é atualmente o mais usado em aplicações comerciais. A seguir apresenta-se as propriedades que as suas chaves públicas e particulares devem possuir, como é realizada a criptografia e decriptografia e a sua segurança.

Inicialmente cliente e servidor devem calcular um par de chaves da seguinte maneira

[TERADA, 2000] [RSA, 1978]:

1. Calcular dois números inteiros primos e longos (centenas de *bits*) chamados *q* e *r;* e calcular o seu produto *n* = *q \* r.* Recomenda-se que o comprimento de *q* seja próximo do comprimento de *r* para tornar inviável a fatoração rápida de *n* em primos;
2. Calcular um terceiro número chamado s relativamente primo a (*q - 1*) *\** (*r - 1*) (que é igual a Φ(*n*) = Φ(*q*) \* Φ(*r*) )e calcular um inteiro *p* que satisfaça *p \* s* = 1 mod(*q* - *1*) *\** (*r - 1*) *i.e., p* = *s-1* mod *n,* através do Algoritmo de Euclides estendido;
3. A chave secreta *S* = *(s, n)* é guardada com cuidado e a chave pública *P* = *(p, n)* é publicada.

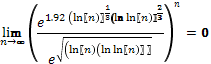
Para realizar a criptografia de um texto aberto *m* expresso como um número inteiro *0 <= m <= n – 1,* usa-se a seguinte função: *c = mp* mod *n*, onde *c* é o texto cifrado. A decriptografia é feito o seguinte processamento: *m = cs* mod *n*, onde *m* é o texto original.

Este algoritmo é baseado na dificuldade computacional de fatorar um número inteiro em primos. Caso seja possível fatorar o número *n* em primos *q* e *r,* então ele pode-se calcular *s* que satisfaça *p \* s = 1* mod(*q - 1*) \* (*r - 1*) como efetuado no cálculo de um par de chaves. É por isso que recomenda-se que *q* e *r* tenham tamanhos próximos para dificultar a fatoração de *n* = *q \* r.* Até hoje não se conhece um algoritmo em tempo polinomial para fatorar um número *n* “longo” em primos [TERADA, 2000].

Para ilustrar a dificuldade de fatorar um inteiro, quando o número *n* pos­sui 129 algarismos decimais gastaria-se 5 mil MIPS-anos, um MIPS-ano significando usar um computador por um ano executando um mil­hão de instruções por segundo, utilizando um dos algoritmos mais rápido que se conhece, chamado QS (*Quadratic Sieve*) [POMERANCE, 1985]. O algoritmo QS possui tempo de execução proporcional a [TERADA, 2000].



Outro algoritmo para fatoração chamado NFS (*Number Field Sieve*) [LENSTRA, 1993] com tempo de execução proporcional a é mais rápido que QS para números com mais de 350 *bits*, por ser mais rápido assintoticamente que QS.



Em 1999, Adi Shamir apresentou um computado específico para fatoração baseado em dispositivos opto-eletrônicos. Com uma implementação do algoritmo QS (ou do NFS) o TWINKLE consegue analisar 100 milhões de inteiros e determinar em menos de 10 milissegundos quais destes são fatoráveis completamente sobre uma base dos primeiros 200 mil primos. Shamir afirma que assim torna viável a fatoração de inteiros de comprimento entre 565 a 665 *bits*; isso torna vulnerável os sistemas de comércio eletrônico que na sua maioria utiliza RSA com chaves de 512 *bits*.

Em 1996, 512 *bits* para o módulo *n* do RSA era considerado razoavelmente seguro. Mas devido aos eventos mencionados e devido aos eventos mencionados e aos algoritmos QS e NFS, hoje recomenda-se no mínimo 768 *bits*. Para longo prazo, recomenda-se desde já adotar 1024 *bits*. É importante também mencionar que até hoje os pesquisadores não conseguiram descobrir qualquer outro ponto fraco neste sistema de criptografia.

A exponenciação exigida é um processo que consome um tempo considerável. Como comparação, o DES, ao contrário, no mínimo, cem vezes mais veloz em software e entre mil e dez mil vezes mais veloz em hardware [RSA FAST, 2011] [KUSORE, 2006].

Portanto o RSA é frequentemente usado em combinação com o DES ou AES. Quando deseja-se transferir uma grande quantidade de *bits*, alta velocidade, por exemplo, uma transmissão de vídeo, primeiramente o servidor escolhe uma chave simétrica aleatória e denominada chave de sessão. Esta chave é cifrada usando um algoritmo de chave pública e é enviado ao cliente, este decifra a mensagem e obtém a chave de sessão. A partir desse momento a troca de dados passa a ser feita usando um algoritmo de chave simétrica, utilizando-se a chave de sessão, para cifrar e decifrar os pacotes.

1. Aplicações multimídia
2. 1. Arquitetura da Internet

A Internet é um sistema extremamente complicado e que possui muitos componentes: inúmeras aplicações e protocolos, vários tipos de sistemas finais, conexões e meios físicos de enlace. Visando uma simplificação deste sistema, a Internet é dividida em camadas, modularizando e simplificando a implementação do serviço oferecido em cada camada. Cada camada contém seus protocolos padronizados e desenvolvidos para atender as necessidades específicas de cada nível. A Figura 5 apresenta as cinco camadas de protocolos: aplicação, transporte, rede, enlace e física.



Figura - A camada de protocolos da Internet [KUROSE, 2006].

O escopo deste trabalho concentra-se nas camadas de aplicação e transporte, portanto somente os protocolos destas camadas serão apresentados.

A Internet comporta uma grande variedade de aplicações de multimídia interessantes – vídeo em tempo real, telefonia IP (*Internet Protocol*), rádio por Internet, teleconferência, ensino à distância – são somente alguns exemplos dessas aplicações.

Aplicações de multimídia são altamente sensíveis a atraso, porém em sua maioria são tolerantes à perda – perdas ocasionais causam somente pequenas perturbações na recepção de áudio e vídeo, e essas perdas podem ser parcial ou totalmente encobertas pela aplicação [KUROSE, 2006].

Devido a grande variedade de aplicações multimídia presentes na Internet, pode-se classificar em três classes essas aplicações: áudio e vídeo de fluxo contínuo armazenado, áudio e vídeo de fluxo contínuo ao vivo e áudio e vídeo interativos em tempo real [KUROSE, 2006].

A aplicação apresentada neste trabalho classifica-se na primeira classe, por este motivo menciona-se a seguir três características fundamentais desta classe:

1. Mídia armazenada. O conteúdo multimídia está armazenado completamente em um servidor. Por esta razão um usuário que tenha acesso a esse servidor pode voltar, avançar ou parar a transmissão;
2. Fluxo contínuo (*streaming*). Numa aplicação cliente-servidor armazenado, o cliente inicia a reprodução do áudio e vídeo alguns segundos após começar a receber o arquivo do servidor. Desta forma o arquivo não necessita ser descarregado inteiro, para começar a ser reproduzido;
3. Reprodução contínua. Ao iniciar-se a reprodução do áudio e vídeo, ela deve prosseguir de acordo com a temporização original da gravação. Isso impõe sérias restrições ao atraso na entrega de dados.
   1. Protocolo UDP

O UDP (*User Datagram Protocol*) é um protocolo da camada de transporte não orientado a conexões, ou seja, oferece um meio para as aplicações enviarem datagramas encapsulados sem que seja necessário estabelecer uma conexão.

Implementado na camada de transporte, este protocolo exige que os datagramas sejam encapsulados, na camada de rede, usando o protocolo IP.

Na Figura 6, é apresentado o cabeçalho do UDP. Formado por quatro campos, cada um consistindo de 2 *bytes*, são eles: um campo para informar o número da porta de origem e outro para a porta de destino, um campo que informa a soma de verificação ou *checksum* e o campo de comprimento da mensagem, que especifica o comprimento do segmento UDP, incluindo o cabeçalho, em *bytes*. Os dados da aplicação ocupam o campo de dados do segmento UDP.

Por possuir um cabeçalho extremamente simples o UDP, não provê nenhuma garantia de entrega ou duplicidade de pacotes até o seu destino. Tudo isso cabe aos processos do usuário. Os dados são transmitidos apenas uma vez e a integridade é verificada pelo sistema de CRC (*Cyclic redundancy check*) de 16 *bits* (informação presente no campo *checksum*). Os pacotes corrompidos simplesmente são descartados sem que o transmissor tenha conhecimento. Portanto ele é recomendado para transmissão de dados pouco sensíveis, tais como fluxos de áudio e vídeo, serviços de tradução nome-número (DNS) ou transferência de arquivos simples (protocolo TFTP) [INS, 1979], [RFC, 1350], [RFC, 768].

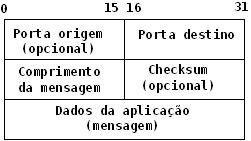


Figura - Estrutura do protocolo UDP [RFC, 768]

Comparando-se com o protocolo TCP (*Transmission Control Protocol*), que também é implementado na mesma camada de rede, porém orientado para conexão, seu tempo de latência de rede é menor. Facilmente notado já que não é necessário garantir retransmissão, detecção e correção de pacotes corrompidos, congestionamento do fluxo ou espera de dados. O UDP provê suporte a *broadcasting* e *multicasting*. Além disto não é necessário estabelecer uma conexão ponto-a-ponto otimizando dessa forma a troca de pacotes.

* 1. Protocolo RTP

O RTP (*real-time protocol*), é um padrão de domínio público para o encapsulamento de dados de áudio e vídeo. Este protocolo possui campos de cabeçalho extremamente úteis para aplicações deste tipo como: números de sequência, marcas de tempo e campos para dados de áudio e vídeo [RFC, 1889].

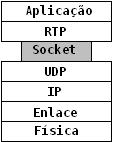


Figura - Posição do RTP na pilha de protocolos [KUROSE, 2006].

O RTP opera da maneira descrita a seguir: a aplicação de multimídia consiste em vários fluxos de áudio, vídeo, texto e possivelmente outros fluxos. Esses fluxos são armazenados na biblioteca RTP, que se encontra no espaço do usuário, juntamente com a aplicação. Essa biblioteca efetua a multiplexação dos fluxos e os codifica em pacotes RTP, que são então colocados em um soquete. Na outra extremidade do soquete (no núcleo do sistema operacional), os pacotes UDP são gerados e incorporados a pacotes IP. Se o computador estiver em uma rede *Ethernet*, os pacotes IP serão inseridos em quadros *Ethernet* para transmissão. A pilha de protocolos para essa situação é mostrada na Figura 7 [TANENBAUM, 2003].

Como conseqüência dessa estrutura, é um pouco difícil dizer em que camada o RTP está. Como ele funciona no espaço do usuário e está vinculado ao programa aplicativo, certamente parece ser um protocolo de aplicação. Por outro lado ele é um protocolo genérico e independente das aplicações que apenas fornecem recursos de transporte, e assim também é semelhante a um protocolo de transporte. Talvez a melhor descrição do RTP seja como um protocolo de transporte implementado na camada de aplicação [TANENBAUM, 2003].

A função básica do RTP é multiplexar diversos fluxos de dados de tempo real sobre um único fluxo de pacotes UDP. O fluxo UDP pode senviado a um único destino (unidifusão) ou a vários destinos (multidifusão) [TANENBAUM, 2003].

Cada pacote enviado em um fluxo recebe um número uma unidade maior que seu predecessor. Essa numeração permite ao destino descobrir se algum pacote está faltando. Como conseqüência, o RTP não tem nenhum controle de fluxo, nenhum controle de erros, nenhuma confirmação e nenhum mecanismo para solicitar retransmissões [TANENBAUM, 2003].

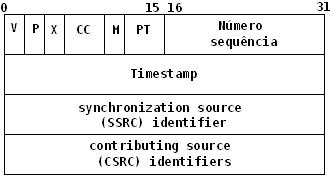


Figura - Campos de cabeçalho do pacote RTP [RFC, 1889].

O cabeçalho do RTP é ilustrado na Figura 8. Ele consiste em três palavras de 32 *bits* e, potencialmente, algumas extensões. A primeira palavra contém o campo v*ersion* (*V*), que atualmente já está em 2. O *bit* *P* indica que o pacote foi completado até chegar a um múltiplo de 4 *bytes*, o último *byte* de preenchimento informa quantos *bytes* foram acrescentados. O *bit* *X* indica que um cabeçalho de extensão está presente. O formato e o significado do cabeçalho de extensão não são definidos. O único detalhe definido é que a primeira palavra da extensão fornece o comprimento. Essa é uma válvula de escape para quaisquer exigências imprevistas. O campo *CC* informa quantas origens de contribuição estão presentes, de 0 a 15. O *bit* *M* é um marcador específico da aplicação, ele pode ser usado para marcar o começo de um quadro de vídeo, o começo de uma palavra em um canal de áudio ou qualquer outro elemento que a aplicação reconheça. O campo *PT* (*payload type*) informa que algoritmo de codificação foi usado (por exemplo, áudio não-compactado de 8 *bits*, MP3, etc). Tendo em vista que todo pacote apresenta esse campo, a codificação pode mudar durante a transmissão. O campo número de sequência é apenas um contador incrementado em cada pacote RTP enviado. Ele é usado para detectar pacotes perdidos.

O timbre de hora (*timestamp*) é produzido pela origem do fluxo para anotar quando a primeira amostra no pacote foi realizada. Esse valor pode ajudar a reduzir a flutuação no receptor, desacoplando a reprodução no momento da chegada do pacote. O *synchronization source identifier* informa a que fluxo o pacote pertence. Esse é o método usado para multiplexar e demultiplexar vários fluxos de dados em um único fluxo de pacotes UDP. Finalmente, os campos c*ontributing source identifiers*, se estiverem presentes, serão usados quando houver misturadores (*mixers*) de áudio no estúdio. Nesse caso, o misturador será a origem de sincronização, e os fluxos que estão sendo mixados serão listados nesse campo [RFC, 1889].

Diferentemente dos protocolos HTTP e FTP, o RTP não realiza um *download* completo de todos os dados para o cliente. Ao invés disso, ele realiza um fluxo contínuo de transmissão de áudio e vídeo para o cliente. Por essas razões utilizou-se estes protocolos nesta aplicação cliente-servidor.

* 1. Protocolo RTSP

Para permitir que seja possível controlar o fluxo contínuo em tempo real, o cliente e o servidor precisam de um protocolo para que ocorra esta troca de informações de controle. O RTSP (*Real-Time Streaming Protocol*), definido em [RFC, 2326], é este protocolo.

É importante que faça-se uma distinção entre os protocolos RTP e RTSP. O RTSP permite uma comunicação bidirecional, isto é, o cliente pode comunicar-se com o servidor de mídia e realizar pedidos de iniciar, pausar, avançar ou parar uma reprodução. O servidor recebe estas requisições e trata cada uma devolvendo uma resposta ao cliente. Já o RTP é um protocolo utilizado para envio de fluxo contínuo do servidor para o cliente. Desta forma é possível enumerar o que o RTSP não faz:

1. Não define esquemas de compressão para áudio e vídeo;
2. Não define como áudio e vídeo são encapsulados em pacotes para uma transmissão por uma rede; o encapsulamento pode ser fornecido por RTP ou por um protocolo proprietário;
3. Não restringe o modo como a mídia de fluxo é transportada; pode ser por UDP ou TCP;
4. Não restringe o modo como o cliente armazena o áudio e vídeo. Ele pode ser reproduzido logo que começar a chegar, após um atraso de alguns segundos ou pode ser descarregado integralmente antes de ser reproduzido.

O RTSP é um protocolo “fora da banda”. Em particular, as mensagens RTSP são enviadas “fora da banda”, ao passo que o fluxo de mídia, cuja estrutura não é definida pelo RTSP, é considerado “dentro da banda”. Mensagens RTSP utilizam a porta 544 e o fluxo de mídia utiliza um número diferente [KUROSE, 2006].

Devido à especificação do RTSP permitir que suas mensagens sejam enviadas por TCP ou UDP, neste trabalho as mensagens são enviadas através do UDP.

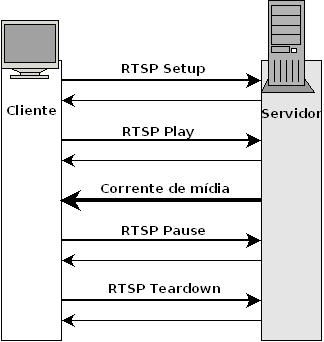


Figura - Integração entre cliente-servidor usando RTSP [KUROSE, 2006]

A Figura 9, apresenta um exemplo simples de uma interação entre cliente/servidor. Como ilustra a figura o cliente e o servidor enviam um ao outro uma série de mensagens RTSP. O cliente envia uma requisição RTSP *setup* e o servidor responde com uma mensagem RTSP *ok*. O cliente envia uma requisição RTSP *play* e recebe uma resposta com uma mensagem RTSP *ok*. Nesse ponto, o servidor de fluxo contínuo bombeia vídeo para dentro de seu próprio canal “dentro da banda” e a execução do vídeo inicia-se. Mais tarde, o cliente envia uma requisição RTSP *pause* e o servidor responde com uma mensagem RTSP *ok*. Quando o usuário termina, o cliente envia uma requisição RTSP *teardown* e o servidor confirma com uma resposta RTSP *ok*.

Agora apresenta-se as mensagens RTSP propriamente ditas entre um cliente (C:) e um servidor (S:).

C: SETUP movie.Mjpeg RTSP/1.0

C: CSeq: 1

C: Transport: RTP/UDP; client\_port= 25000; mode=PLAY

S: RTSP/1.0 200 OK

S: CSeq: 1

S: Session: 123456

C: PLAY movie.Mjpeg RTSP/1.0

C: CSeq: 2

C: Session: 123456

S: RTSP/1.0 200 OK

S: CSeq: 2

S: Session: 123456

C: PAUSE movie.Mjpeg RTSP/1.0

C: CSeq: 3

C: Session: 123456

S: RTSP/1.0 200 OK

S: CSeq: 3

S: Session: 123456

C: TEARDOWN movie.Mjpeg RTSP/1.0

C: CSeq: 5

C: Session: 123456

S: RTSP/1.0 200 OK

S: CSeq: 5

S: Session: 123456

Todas as mensagens de requisição e resposta são em texto ASCII, o cliente emprega métodos padronizados (*setup*, *play*, *pause* e assim por diante) e o servidor responde com códigos padronizados de resposta. O servidor RTSP monitora o estado do cliente para cada sessão em curso. Por exemplo, o servidor moni­tora se o cliente está em um estado de inicialização (*init*), de reprodução (*play*) ou em um estado de pausa (*pause*). Os números de sessão e de sequência, que fazem parte de cada requisição e resposta, ajudam o servidor a monitorar o estado da sessão. O número da sessão é fixo durante toda a comunicação, o cliente incrementa o número de sequência cada vez que envia uma nova mensagem e o servidor devolve um eco com a sessão e o número de sequência corrente.

Como demonstra-se o cliente inicia a sessão com a requisição *setup*, fornecendo o url do arquivo que deverá ser transmitido e a versão do RTSP. A mensagem de estabelecimento (*setup*) inclui o número de porta do cliente para o qual a mídia deve ser enviada. Essa mensagem também indica que a mídia deve ser enviada por UDP usando o protocolo de empacotamento RTP.

Portanto outra característica do RTSP é que ele é capaz de manter um estado para cada sessão criada. O cliente é capaz de mudar seu estado, quando recebe do servidor uma resposta à sua requisição, de acordo com o diagrama de estados da Figura 10. Estes estados serão detalhados no item 7.1.

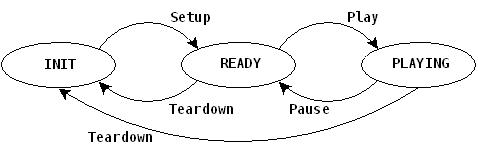


Figura - Diagrama de estados para um cliente usando o RTSP [KUROSE, 2006].

1. Aplicação cliente-servidor

Tomando-se como base o exercício prático do Capítulo 7, do Livro “Redes de Computadores e a Internet” (*5*), implementou-se alguns métodos que não estavam presentes nas classes JAVA que simulavam um servidor e um cliente, numa aplicação de fluxo contínuo de vídeo. Essas mesmas classes também foram modificadas para permitirem o uso de algoritmos de criptografia, dessa forma simulando no servidor a criptografia do pacote de dados e no cliente a decriptografia deste pacote.

Há quatro importantes classes que menciona-se a seguir:

1. 1. Cliente

Apresenta a interface do usuário. Exibe o vídeo e permite controlar o fluxo de dados, através do envio de comandos RTSP ao servidor.

Esta classe deve ser invocada da seguinte maneira:

java Client [host] [porta RTSP] [arquivo vídeo] [algoritmo criptografia]

Os parâmetros, respectivamente, são: endereço de *host* do servidor de mídia, porta RTSP na qual o servidor irá realizar a comunicação, o nome do arquivo de vídeo no servidor e qual algoritmo de criptografia será utilizado.

No momento em que o cliente é inicializado, cria-se um *socket*, através de uma porta para comunicação com o servidor. É através deste *socket* que os dados de comunicação RTSP serão enviados e recebidos.

O cliente reserva uma área de memória de 15.000 *bytes* para armazenar cada pacote recebido do servidor.

Implementou-se nessa classe as ações do cliente, quando os botões de controle são acionados. Estes botões e seus tratamentos são descritos a seguir:

* *Setup*

Cria um *socket* para receber os dados RTP e possui um tempo de expiração no *socket* para 5 milisegundos. Caso o pacote não chegue nesse tempo um aviso é levantado, mas a execução continua, quando o pacote for recebido. Quando os dados não são criptografados este tempo é suficiente, porém aplicando-se algoritmos de criptografia, necessitou-se aumentar esse tempo para 500 milisegundos. Envia uma requisição *setup* para o servidor. No cabeçalho de transporte (UDP), especifica-se a porta para o socket de dados RTP criado. Aguarda a resposta do servidor e analisa o cabeçalho de sessão na resposta para obter o ID da sessão. Caso a resposta seja 200, o RTSP muda para o estado *ready*. Inicializa o número de sequência do RTSP com o valor 1.

* *Play*

Envia uma requisição *play*. Incrementa o número de sequência do RTSP em mais um. No cabeçalho desta requisição encontra-se o ID de sessão fornecido na resposta ao *setup*. Se houver uma resposta do tipo 200, vinda do servidor, o RTSP passa para o estado *playing*. Inicializa a variável de tempo “*timer*” e seu valor será sempre usado no cabeçalho dos pacotes UDP.

* *Pause*

Similar à requisição *play*. Com exceção para a mudança de estado do RTSP para *ready*, caso a resposta seja do tipo 200. Além disso, a variável de tempo “*timer*” é parada.

* *Teardown*

Possui o mesmo comportamento das requisições *play* e *pause*. Após uma resposta válida do servidor, seu estado é alterado para *init*, a variável de tempo “*timer*” é parada e o programa é encerrado.

Nesta classe encontra-se o método “*timerListener*” que manipula os pacotes UDP recebidos. Este método é apresentado no Apêndice A e é nele que os pacotes criptografados são decriptografados.

Nota-se que os pacotes RTP recebidos estão criptografados, então o primeiro passo é descartar os dados que estão presentes somente para preencher o vetor de 15.000 *bytes*. Em seguida chama-se o algoritmo de decriptografia passado como argumento.

* 1. Servidor

Responde às requisições RTSP, realiza o tratamento destas requisições através de classes Java, que encapsulam os pacotes RTP que em seguida são enviados ao cliente, contendo os dados de um quadro de vídeo. Executa-se esta classe da seguinte maneira:

java Server [porta RTSP] [algoritmo criptografia]

Onde, tem-se respectivamente a porta RTSP que estabelecera-se a comunicação com o cliente e o algoritmo de criptografia para esta comunicação.

Implementou-se o empacotamento dos dados de vídeo em pacotes RTP. Criado o pacote necessário, ajustou-se os campos no cabeçalho do pacote e copiou-se a carga útil (o *frame* do vídeo) dentro do pacote.

Quando o servidor recebe a requisição *play* do cliente, aciona-se um temporizador que é ativado a cada 100ms. Nesses tempos, o servidor lerá um quadro de vídeo do arquivo e o enviará para o cliente. O servidor cria um objeto *RTPpacket*, que é o encapsulamento RTP do quadro de vídeo.

O servidor chama o primeiro construtor da classe *RTPpacket* para realizar o encapsulamento.

Nesta classe o método “*actionPerformed*” é responsável por realizar o envio dos pacotes contendo os *frames* de vídeo. Portanto é nesta classe que ocorre a criptografia dos pacotes. Todo o pacote RTP (incluindo seu cabeçalho) é armazenado no vetor “*packetBits*”, este pacote será criptografado por um dos algoritmos selecionados. Seu novo tamanho é calculado, para em seguida incluir-se no protocolo de transporte UDP e enviar ao cliente.

* 1. RTPpacket

Realiza o tratamento dos pacotes RTP. Responsável por empacotar e descompactar os pacotes RTP e serve tanto ao cliente como o servidor.

Esta classe é idêntica para o cliente e para o servidor, pois é a unidade de comunicação entre ambas.

Na Figura 11, demonstra-se como é o cabeçalho deste pacote. O campo *RTP-version* (*V*) é 2, os campos *padding* (*P*), *extension* (*X*), *number of contributing sources* (*CC*) e *marker* (*M*) são todos 0.

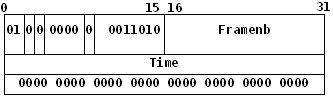


Figura – Esquema do cabeçalho RTP usado na aplicação Servidor

O campo carga útil (*PT*), possui o valor 26, já que o tipo de codificação utilizado é o MJPEG. O servidor fornece o número de sequência como argumento *Framenb* para o construtor desta classe. Da mesma forma preenche-se o campo *timestamp* com o argumento *Time.* O identificador da fonte (*SSRC*) identifica o servidor. Optou-se por utilizar o valor 0, como forma de simplificação. Como não há nenhuma outra fonte de contribuição (campo *CC* = 0), o campo *CSRC* não existe. Então, o comprimento do cabeçalho do pacote é de 12 *bytes*.

Estes valores foram utilizados na disposição “*header*” da classe *RTPpacket*. A carga útil (fornecida como argumento “*data”*) é copiada para a variável “*payload*”. O comprimento da carga útil é dado no argumento “*data\_length*”.

Este diagrama encontra-se na ordem de *byte* de rede (também conhecido como *big-endian*). A *Java* *Virtual Machine* usa a mesma ordem de *byte*, então não precisa-se transformar o cabeçalho de pacote na ordem de *byte* de rede.

* 1. VideoStream

Esta classe é usada para ler os dados de vídeo do arquivo em disco. Não foi necessário implementar ou modificar nenhum método nesta classe.

O formato de vídeo usado nesta aplicação é o tipo .Mjpeg (*MotionJPEG*) que basicamente consiste numa sequência de imagens JPEG, ou seja cada quadro é uma imagem comprimida, codificada numa sequência de *byte*s. Cada quadro é enviado individualmente em um pacote RTP e que pode ser recebido por um cliente que descodifique tal formato. O uso deste formato é comum em câmeras digitais, câmeras IP e é nativo no *browser* Firefox.

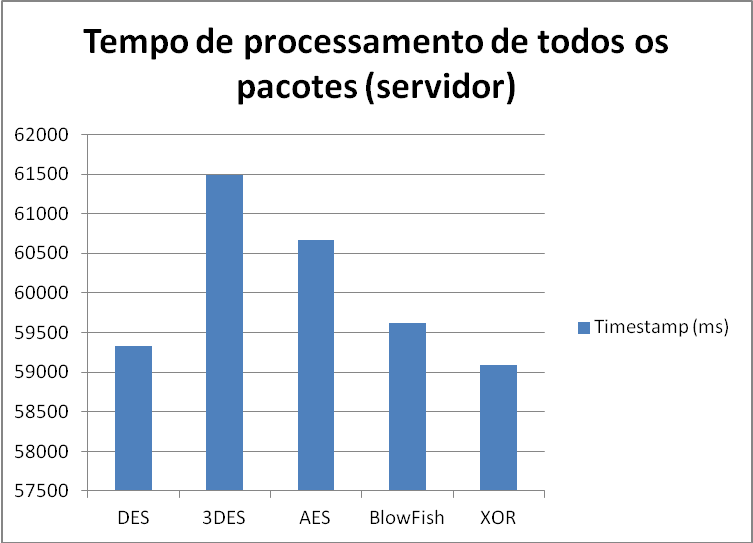
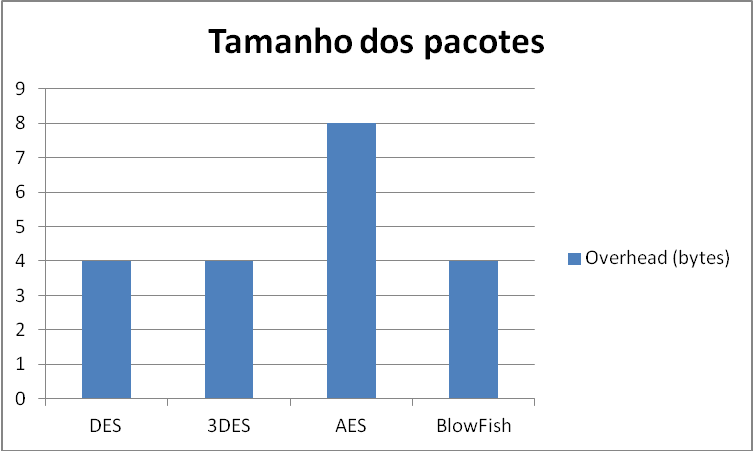
1. Resultados

* 1. O ambiente

Os experimentos foram realizados no Laboratório de Ciência da Computação, do Instituto de Computação da UFF, utilizando dois computadores, ambos com as seguintes configurações: Processador AMD Sempron 2800+ 1.61 GHz, 512 MB de memória RAM, sistema operacional Windows XP e máquina virtual Java versão 6 atualização 22. A topologia da rede é do tipo *ethernet* com velocidade de 100 Mbps.

Execução da transmissão completa do arquivo do vídeo divididos em 500 *frames*. Cada *frame* era cifrado de maneira independente usando a mesma chave. O tempo de uso gasto pela CPU para cifrar cada *frame* foi registrado e calculamos uma média aritmética para chegar aos resultados obtidos no Gráfico. *Esperava-se que o 3DES obtivesse um tempo de processamento três vezes maior em comparação ao DES mas não foi isso que notou-se.*

O mesmo método foi aplicado para registramos no Gráfico a média de *overhead* dos *frames* após o uso de cada um dos algoritmos. Os algoritmos DES e 3DES geraram o mesmo *overhead*, o que já era esperado, pois a diferença entre estes algoritmos está no fato de que o 3DES é uma execução tripla do DES e isto não gera um *overhead* adcional, mas influenciando diretamente num tempo maior de processamento de CPU.



1. Conclusão e Trabalhos futuros

Com a implementação desse trabalho, podemos perceber a dificuldade em se acessar um dado criptografado e a importância de uma comunicação segura entre cliente-servidor. Também podemos perceber a importância da segurança da informação utilizando-se em algoritmos de criptografia que impedem em um tempo computacionalmente viável que um usuário não autorizado acesse ao conteúdo do dado.

* 1. Criptografia Quântica
  2. Autenticação do cliente num servidor KDC

Apêndice A

Neste apêndice são apresentados trechos do código implementado no servidor e no cliente para realizar a criptografia e decriptografia dos pacotes de dados.

Código 1: Cliente

//Constrói um datagrama para receber os dados do socket UDP

rcvdp = **new** DatagramPacket(buf, buf.length);

//Recebe o pacote UDP do socket

RTPsocket.receive(rcvdp);

//Os dados estão criptografados. A primeira coisa é decriptografá-los

//Não estou interessado nos 15000 *bytes*. Só naquilo que efetivamente //foi preenchido

**final** ***byte***[] encrypted*Bytes* = Arrays.*copyOfRange*(rcvdp.getData(), 0, rcvdp.getLength());

//Decriptografa

***byte***[] decrypted*Bytes* = **null**;

**switch** (Client.*algorithm*) {

**case** *AES*:

decrypted*Bytes* = decriptaAES(encrypted*Bytes*);

**break**;

**case** *BlowFish*:

decrypted*Bytes* = decriptaBlowfish(encrypted*Bytes*);

**break**;

**case** *DES*:

decrypted*Bytes* = decriptaDES(encrypted*Bytes*);

**break**;

**case** *DESede*:

decrypted*Bytes* = decriptaDES3(encrypted*Bytes*);

**break**;

**case** *XOR*:

decrypted*Bytes* = criptaDecripta(encrypted*Bytes*);

**break**;

**default**:

**throw** **new** IllegalArgumentException("Algoritmo não previsto");

}

//Constrói o pacote com os dados decriptografados.

**final** RTPPacket rtpPacket = **new** RTPPacket(decrypted*Bytes*, decrypted*Bytes*.length);

Código 2: Servidor

//Constrói um pacote RTP contendo um frame do vídeo

**final** RTPPacket rtpPacket = **new** RTPPacket(Server.*MJPEG\_TYPE*, imagenb, imagenb \* Server.*FRAME\_PERIOD*, buf, imageLength);

//Descobre o tamanho total do pacote a ser enviado

**int** packetLength = rtpPacket.getLength();

//recupera o pacote de streaming e armazena num vetor de *bytes*

**final** ***byte***[] packet*Bits* = **new** ***byte***[packetLength];

rtpPacket.getPacket(packet*Bits*);

System.*out*.println("Tamanho do texto claro: " + packetLength);

//Criptografa

***byte***[] encrypted*Bits* = **null**;

**switch** (Server.*algorithm*) {

**case** *AES*:

encrypted*Bits* = encriptaAES(packet*Bits*);

**break**;

**case** *BlowFish*:

encrypted*Bits* = encriptaBlowfish(packet*Bits*);

**break**;

**case** *DES*:

encrypted*Bits* = encriptaDES(packet*Bits*);

**break**;

**case** *DESede*:

encrypted*Bits* = encriptaDES3(packet*Bits*);

**break**;

**case** *XOR*:

encrypted*Bits* = criptaDecripta(packet*Bits*);

**break**;

**default**:

**throw** **new** IllegalArgumentException("Algoritmo não previsto");

}

packetLength = encrypted*Bits*.length;

System.*out*.println("tamanho do texto criptografado: " + packetLength);

//Envia o pacote como um datagrama através do protocolo UDP

senddp = **new** DatagramPacket(encrypted*Bits*, packetLength, clientIPAddr, rtpDestPort);

rtpSocket.send(senddp);

Bibliografia

[AES, 2010] ORACLE. *Using AES with Java Technology*. Disponível em: http://java.sun.com/developer/technicalArticles/Security/AES/AES\_v1.html. Acessado em 11 outubro 2010.

[INS, 1979] POSTEL, J. *Internet Name Server*. Disponível em: http://www.networksorcery.com/enp/ien/ien116.txt. Acessado em 01 dezembro 2010.

[ITU, 2010] INTERNATIONAL TELECOMUNICATION UNION. Disponível em:

http://www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf. Acessado em 10 janeiro 2011.

[KUROSE, 2006] KUROSE, James F; ROSS, Keith W. *Redes de Computadores e a Internet*: *uma abordagem top-down*. 3. ed. São Paulo: Pearson Addison Wesley, 2006.

[RFC, 2326] SCHULZRINNE, H. Real Time Streaming Protocol. Abril 1998. RFC 2326.

[RFC, 2435] BERC, L. RTP Payload Format for JPEG-compressed Video. Outubro 1998. RFC 2435.

[RFC, 768] POSTEL, J. User Datagram Protocol. Agosto 1980. RFC 768.

[RFC, 1350] SOLLINS, Karen R. The TFTP Protocol. Julho 1992. RFC 1350

[SCHNEIER, 1994] SCHNEIER, B. Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," Fast Software Encryption, Cambridge Security Workshop Proceeding, Londres: Springer-Verlag, p. 191-204, 1994.

[TANENBAUM, 2003] TANENBAUM, Andrew S. *Redes de Computadores*. 4. ed. São Paulo: Campus, 2003

[TERADA, 2000] TERADA, Routo. *Segurança de Dados: criptografia em redes de computador*. São Paulo: Edgard Blücher, 2000.

[RFC, 1889] RTP: A Transport Protocol for Real-Time Applications. Janeiro 1996. RFC 1889

[FIPS, 1977] U.S. FEDERAL INTERNATIONAL PROCESSING STANDARDS, 46*,* 1977, Washington D.C.

[POMER­ANCE, 1985] POMER­ANCE, C. The Quadratic Sieve Factoring Algorithm, *Proceeding of the EUROCRYPT 84 workshop on Advances in cryptology: theory and application of cryptographic techniques.* Nova Iorque: Springer-Verlag, p. 169-182, set. 1985.

[LENSTRA, 1993] LENSTRA, A.K. et al. The number field sieve. *Lecture Notes in Mathematies,* Nova Iorque: Springer- Verlag, v. 1554, p. 11-12, 1993.

[SHAMIR, 1999] SHAMIR, A. Factoring Large Numbers with the TWINKLE Device, *Lecture Notes in Computer Seience*, Nova Iorque: Springer-Verlag, v. 1717, 1999)

[RSA, 1978] RIVEST, R.L; SHAMIR, A; ADLEMAN, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystem*s.* *Communications of the ACM*, Nova Iorque: ACM, v. 21, n. 2, p 120-126, fev. 1978.

[RIESEL,1994] RIESEL, H. *Prime Numbers And Computer Methods Of Factorization (Progress in Mathematics).* 2. ed. Boston: Birkhäuser Boston, 1994. cap. 6, p. 174-177.

[COUTINHO, 2003] COUTINHO, S. C. *Números Inteiros e Criptografia RSA*. Rio de Janeiro: IMPA, 2003

[DES, 2011] Disponível em: http://www.numaboa.com/criptografia/bloco/313-des2. Acessado em 22 março 2011.

[RSA, 2011] RSA LABORATORIES. *How fast is the RSA algorithm?*. Disponível em:

http://www.rsa.com/rsalabs/node.asp?id=2215. Acessado em 22 março 2011.

[DISTRIBUTED, 2011] Disponível em: http://www.distributed.net/Main\_Page. Acessado em 22 março 2011

[MENEZES, 1996] MENEZES, A; OORSCHOT, P. Van; VANSTONE, S. *Handbook of Applied Cryptography*. Nova Iorque: CRC Press, 1996

[STALLINGS, 2003] STALLINGS, **William. *C****ryptography and Network Security: Principles and Practice*. 3. ed.  Prentice Hall, 2003

[KAHN, 1966] KAHN, David. *The Code-breakers: The Comprehensive History Of Secret Communication From Ancient Times To The Internet*. 2. ed. Nova Iorque: Scribner, 1996

[DIFFIE, 1976] DIFFIE, W; HELLMAN, M.E. New Directions in Cryptography. In: IEEE TRANSACTIONS ON INFORMATION THEORY, 22, pp. 644-654)