UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE SOFTWARE PARA ATUALIZAÇÃO AUTOMÁTICA DE VERSÃO DE ARQUIVOS

AIRISON AMBROSI

BLUMENAU
2004

2004/2-01

AIRISON AMBROSI

PROTÓTIPO DE SOFTWARE PARA ATUALIZAÇÃO AUTOMÁTICA DE VERSÃO DE ARQUIVOS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Francisco Adell Péricas - Orientador

BLUMENAU
2004

2004/2-01

PROTÓTIPO DE SOFTWARE PARA ATUALIZAÇÃO AUTOMÁTICA DE VERSÃO DE ARQUIVOS

Por

AIRISON AMBROSI

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Fransisco Adell Péricas, MSc. – Orientador, FURB
Membro:	Prof. Sérgio Stringari, FURB
Membro:	Prof. Jomi Fred Hubner, FURB

Blumenau, 03 de novembro de 2004

Dedico este trabalho a todas as pessoas que me deram força, especialmente à minha família que me ajudou diretamente na realização deste.

Os bons livros fazem “sacar” para fora o que a pessoa tem de melhor dentro dela.

Lina Sotis Francesco Moratti

AGRADECIMENTOS

À Deus, pelo seu imenso amor e graça.

À minha família, que esteve presente a todo o momento.

Aos meus amigos de faculdade que foram sempre uma turma unida e divertida.

Ao meu orientador, Francisco Adell Péricas, por ter acreditado na conclusão deste trabalho.

À uma pessoa muito especial que está distante agora, mas acreditava na realização pessoal e na conclusão deste curso.

RESUMO

Este trabalho apresenta a especificação e implementação de um protótipo para fazer atualização automática de arquivos, auxiliado por um esquema de controle de versão e utilizando criptografia e algoritmos de resumo de mensagem para garantir a segurança e a integridade dos arquivos. Utilizou-se o protocolo FTP com criptografia SSL para conexão e autenticação, o algoritmo *Message Digest* 5 (MD5) para integridade e o um arquivo na linguagem *eXtensible Markup Language* (XML) para controle de versão.

Palavras chaves: Atualização de Arquivos; Resumo de Mensagem; Criptografia; Controle de Versão.

ABSTRACT

This work presents the specification and implementation of an archetype to make automatic update of files, assisted for a project of version control and using cryptography and algorithms of message digest to guarantee the security and the integrity of the files. It uses the protocol FTP with cryptography SSL for connection and authentication, the algorithm Message Digest 5 (MD5) for integrity and a file in the language eXtensible Markup Language (XML) for version control.

Key-Words: File update; Message Digest; Cryptography; Version Control.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A Internet tornou-se o meio mais rápido e eficiente para atualizações de versões de software, antivírus, documentos, sistema operacional ou qualquer documento eletrônico baseado em arquivos. Em função da necessidade de manter esses arquivos sempre atualizados, surgiram ferramentas que fazem esta atualização freqüentemente e automaticamente.

A atualização automática é feita por um software que fica instalado no computador onde existe a necessidade da atualização. Este computador deve estar conectado à Internet para que o software possa conectar-se a um servidor contendo um repositório dos novos arquivos que serão atualizados. A atualização de arquivos é uma tarefa que exige um controle de versões onde é importante que nada venha a ser atualizado com versões anteriores ou incompatíveis.

Recentemente a segurança dessa atualização passou a ser tratada com muito mais cuidado. Muitas vezes esses arquivos podem percorrer grandes distâncias e ficam vulneráveis a ataques de vírus ou de *hackers*, que podem alterar estes arquivos para propagar vírus e programas mal intencionados. Medidas de segurança devem ser tomadas para que estes casos não venham a tornar este mecanismo não confiável.

O *Transport Control Protocol/Internet Protocol* (TCP/IP) é um conjunto de protocolos utilizados na Internet que garantem a chegada do conteúdo ao seu destino. A solução apresentada neste trabalho é a utilização de algoritmo de *hashing* sobre esses arquivos para criar uma espécie de assinatura do arquivo. Esse algoritmo gera um código que deverá ser comparado com o código gerado a partir do arquivo depois de transferido. Caso o código não seja o mesmo, o software deve refazer a atualização do arquivo, pois o mesmo sofreu alterações durante o seu trajeto na Internet.

1.1 OBJETIVOS DO TRABALHO

O trabalho tem como objetivo criar um protótipo de aplicativo de atualização automática de versões através da Internet, utilizando um repositório de arquivos e um algoritmo de *Hashing* para garantir a integridade do arquivo.

Os objetivos específicos do trabalho são:

a) criar de um protótipo de aplicativo de atualização automática;
b) utilizar um algoritmo de *Hashing* para autenticação;
c) atualizar um arquivo demonstrando o funcionamento do dispositivo de segurança.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está subdividido em capítulos que serão explicitados a seguir.

O primeiro capítulo apresenta a contextualização e justificativa para o desenvolvimento da proposta do trabalho.

O segundo capítulo aborda segurança em redes, conceitos como: ataques, sigilo, autenticação e integridade de dados.

O terceiro capítulo apresenta a transferência de dados, contextualizando a atualização de arquivos com controle de versão. São abordados assuntos como protocolos de transferência de arquivos e utilização de XML para controle de versão.

O quarto capítulo aborda o desenvolvimento do trabalho, mostrando a especificação com diagramas de casos de uso, diagrama de classes e diagrama de seqüência. A implementação e resultados e discussão também são abordados neste capítulo.

2 SEGURANÇA EM REDES

2.1 INTRODUÇÃO

Nas primeiras décadas de sua existência, as redes de computadores eram usadas por pesquisadores universitários para uso de correio eletrônico, e por funcionários de empresas para compartilhar impressoas. Nessas circunstâncias a segurança nunca precisou de maiores cuidados. Hoje milhares de cidadãos estão utilizando grandes redes para as mais diversas operações, que vão de um simples acesso a uma página na internet até compras e transações bancárias de grandes valores.

A segurança é um assunto abrangente. Partindo de sua forma mais simples, a segurança preocupa-se em garantir que pessoas mal-intensionadas não leiam ou modifiquem mensagens enviadas a outros destinatários. Outra preocupação da segurança é quando pessoas que tentam ter acesso a serviços remotos, os quais não foram autorizadas. A maior parte dos problemas de segurança são intencionalmente causados por pessoas que tentam obter algum benefício ou prejudicar alguém (TANENBAUM, 1997). *Crackers* e *Hackers* tentam a todo instante invadir computadores na Internet, sendo assim torna-se fundamental a pesquisa de novas tecnologias na área de segurança (MARIANO, 2001).

Pode-se identificar como propriedades desejáveis da comunicação segura:

a) sigilo: somente o remetente e o destinatário pretendido devem ser autorizados a entender o conteúdo da mensagem transmitida. O fato de pessoas não-autorizadas poderem interceptar a mensagem exige, necessariamente, que esta seja cifrada de alguma maneira (que seus dados sejam disfarçados) para impedir que a mensagem interceptada seja decifrada (entendida) por um interceptador (KUROSE, 2003);
b) autenticação: o remetente e o destinatário precisam confirmar a identidade da outra parte envolvida na comunicação, confirmar que a outra parte é realmente quem alega ser. A comunicação pessoal entre seres humanos resolve facilmente esse problema pelo reconhecimento visual. Quando entidades trocam mensagens por um meio pelo qual não podem “ver” a outra parte, a autenticação não é tão simples assim (KUROSE, 2003);
c) integridade da mensagem: mesmo que o remetente e o destinatário consigam se autenticar reciprocamente, eles querem assegurar que o conteúdo de sua

comunicação não seja alterado, por acidente ou por má intenção, durante a transmissão (KUROSE, 2003).

2.2 SIGILO

O sigilo é o ato de manter as informações protegidas de indivíduos indesejáveis. Consiste em manter uma gerência total sobre as informações controlando quem tem acesso a elas. A solução mais adotada visando manter o sigilo das informações enviadas em uma rede de dados consiste no uso da criptografia, as informações são cifradas (embaralhadas) para que somente computadores autorizados consigam decifrar (restaurar) a informação na sua forma original.

Cifrar é o ato de transformar dados em alguma forma ilegível. Seu propósito é o de garantir a privacidade, mantendo a informação escondida de qualquer pessoa não autorizada, mesmo que esta consiga visualizar os dados criptografados.

Decifrar é o processo inverso, ou seja, transformar os dados criptografados na sua forma original, legível.

2.2.1 Criptografia

Criptografia é a ciência de escrever em cifra ou em código, ou seja, é um conjunto de técnicas que permite tornar incompreensível uma mensagem originalmente escrita com clareza, de forma a permitir normalmente que apenas o destinatário a decifre e compreenda. Quase sempre o deciframento requer o conhecimento de uma chave, uma informação secreta disponível ao destinatário (LUCCHESI, 1986).

As técnicas de criptografia permitem que um remetente disfarce os dados de modo que um intruso não consiga obter nenhuma informação com base nos dados interceptados (KUROSE, 2003).

A criptografia computacional protege o sistema quanto à ameaça de perda de confiabilidade, integridade ou não-repudiação, e é utilizada para garantir:

a) sigilo: somente os usuários autorizados têm acesso à informação;
b) integridade: garantia oferecida ao usuário de que a informação correta, original, não foi alterada, nem intencionalmente, nem acidentalmente;

c) autenticação do usuário: é o processo que permite ao sistema verificar se a pessoa com quem está se comunicando é de fato a pessoa que alega ser;
d) autenticação de remetente: é o processo que permite a um usuário certificar-se que a mensagem recebida foi de fato enviada pelo remetente, podendo-se inclusive provar perante um juiz, que o remetente enviou aquela mensagem;
e) autenticação do destinatário: consiste em se ter uma prova de que a mensagem enviada foi como tal recebida pelo destinatário;
f) autenticação de atualidade: consiste em provar que a mensagem é atual, não se tratando de mensagens antigas reenviadas.

A criptografia de dados é feita através de algoritmos que realizam o ciframento e o deciframento dos dados. O algoritmo pode ser amplamente conhecido, pois a segurança do sistema está em uma chave utilizada para cifrar e decifrar os dados. Os algoritmos de criptografia podem ser:

a) simétricos: também conhecidos como algoritmos de chave secreta, utilizam a mesma chave para realizar o ciframento e deciframento dos dados. Um exemplo de algoritmo simétrico é o DES (*Data Encription Standard*);
b) assimétricos: também conhecidos como algoritmos de chave pública, utilizam chaves diferentes para realizar o ciframento e o deciframento dos dados, porém sempre existe uma relação entre as duas chaves. Um exemplo de algoritmo assimétrico é o RSA (*Rivest, Shamir and Adleman*).

2.2.2 Criptografia com chave secreta

A criptografia simétrica ou de chave secreta faz uso de uma única chave criptográfica para realizar os processos de encriptação e decriptação. O termo simétrico refere-se ao fato de uma mesma chave ser utilizada nas duas operações, esta chave deve ser de conhecimento apenas das entidades que desejam trocar informações, sendo mantida secreta de todas as demais.

Conforme Anton (2001), um algoritmo simples capaz de gerar um criptograma indecifrável consiste na escolha de uma seqüência aleatória de bits do mesmo tamanho da mensagem a ser encriptada. Cada bit desta seqüência, que poderia ser, por exemplo, a representação ASCII dos caracteres formadores de uma frase, deve sofrer a operação de OU-EXCLUSIVO (XOR) com cada bit da mensagem, gerando um criptograma que não fornece

qualquer informação sobre a chave ou a mensagem. Para que o criptograma gerado por este algoritmo permaneça indecifrável, é importante que cada chave seja utilizada uma única vez, razão pela qual este algoritmo é chamado de “Bloco de Uma Só Utilização” (*One-Time Pad*). A utilização prática deste algoritmo é inviável devido à necessidade de compartilhamento de uma chave secreta muito grande, torna o processo impraticável. Para evitar este tipo de inconveniente, foram desenvolvidos algoritmos criptográficos que fazem uso de chaves de tamanho limitado utilizando a encriptação de blocos. A figura 1 demonstra de forma simples o uso da criptografia de chave simétrica.

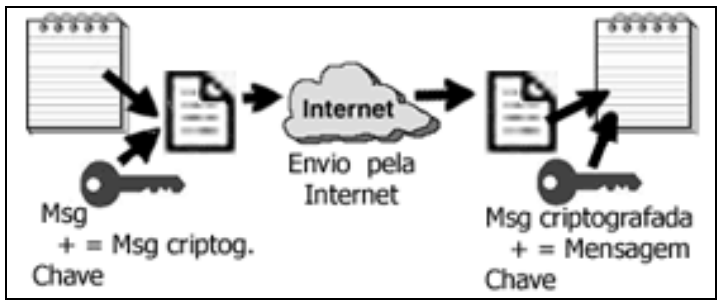

Figura 1 – Utilização da criptografia de chave simétrica

2.2.2.1 Data Encryption Standard (DES)

O *Data Encryption Standard* (DES) é um algoritmo que cifra blocos de 64 bits de texto normal, que foi publicado em 1977 pelo *National Bureau of Standards* (Escritório Nacional de Padrões dos Estados Unidos). O algoritmo de codificação do DES é parametrizado por uma chave *K* de 56 bits, embora aparente ter 64, pois um bit de cada byte é utilizado para paridade ímpar. Na época de seu desenvolvimento, o DES era eficiente para implementações em hardware, mas relativamente lento para implementações em software. Embora na época esta característica limitasse sua utilização a organizações capazes de arcar com o custo de hardware, atualmente, com os avanços tecnológicos, sua implementação em software tornou-se perfeitamente viável e bastante utilizada.

Um dos aspectos mais controversos do DES é a utilização de uma chave de 56 bits, o que torna o algoritmo menos seguro contra ataques exaustivos do que se fossem utilizados 64 bits. Em contrapartida, o DES apresenta a vantagem, segundo seus desenvolvedores, de permitir a detecção de chaves inválidas. O uso de paridade embora dificulte a utilização de uma chave inválida não a impede, pois chaves inválidas podem ser erroneamente tomadas como válidas (ANTON, 2001).

O gerenciamento de chaves é a tarefa de transmitir e armazenar chaves em um sistema de segurança. Os algoritmos de critografia simétricos exigem que o transmissor e o receptor de uma mensagem conheçam a chave secreta e única utilizada na codificação e decodificação. A transmissão da chave não é um problema comum, pois nem sempre é possível garantir que essa transmissão não seja interceptada. Caso seja interceptada, o responsável pelo ataque pode ler todas as mensagens que serão criptografadas utilizando a referida chave "secreta". Um complicador para o problema é o fato de que em um sistema com n usuários comunicando-se dois a dois, são necessárias n^2 chaves secretas.

2.2.3 Criptografia com chave pública

A criptografia de chave pública ou criptografia assimétrica foi criada em 1970. Esse método funciona com uma chave para criptografar, e outra para descriptografar a mesma mensagem. O método de criptografia de chave pública baseia-se na utilização de chaves distintas: uma para codificação (*E*) e outra para decodificação (*D*), escolhidas de forma que a derivação de *D* a partir de *E* seja senão impossível, pelo menos muito difícil de ser realizada. Nestas condições não há razão para não tornar a chave *E* pública, simplificando bastante a tarefa de gerenciamento de chaves (SOARES, 1995).

Os algoritmos de criptografia assimétrica, devido às suas operações, demandam mais tempo para serem processados que os de criptografia simétrica, tornando ineficiente seu uso na encriptação de longas seqüências de dados.

Tabela 1 – Notações para criptografia assimétrica

Notação 1	**Notação 2**	**Operação**
C = E(M)	$C = E_{K\ pública}(M)$	Encriptação com chave pública
M = D(C)	$M = D_{K\ privada}(C)$	Decriptação com chave pública
S = D(M)	$S = E_{K\ privada}(M)$	Assinatura com chave privada
M = E(S)	$M = D_{K\ pública}(S)$	Verificação de assinatura com chave privada

2.2.3.1 Rivest, Shamir e Adleman (RSA)

O método mais importante de criptografia assimétrico é o RSA, cujo nome deriva das inicias dos autores Rivest, Shamir e Adleman. O RSA implementa a idéia de que é fácil multiplicar dois números primos para obter um terceiro número, mas muito difícil recuperar os dois primos a partir daquele terceiro número. Isto é conhecido como fatoração. Gerar a chave pública envolve multiplicar dois primos grandes. Derivar a chave privada a partir da chave pública envolve fatoração de um grande número. Se o número for grande o suficiente e

bem escolhido, então ninguém pode fazer isto em uma quantidade de tempo razoável. O RSA baseia-se na dificuldade de se fatorar números muito grandes (SOARES, 1995).

O RSA apresenta a vantagem de permitir a utilização de chaves criptográfica de tamanho variável, permitindo a utilização de chaves grandes, favorecendo a segurança, ou pequenas, favorecendo a eficiência, sendo 256 bits o tamanho geralmente adotado. O tamanho do bloco de dados a ser encriptado é também variável, desde que seu tamanho seja menor que o da chave utilizada. O criptograma gerado é sempre do mesmo tamanho que a chave.

É importante enfatizar que o sigilo da chave privada é muito importante, pois a criptografia assimétrica se baseia no fato de que a chave privada é realmente privada, por isso, somente seu detentor deve ter acesso (Figura 2).

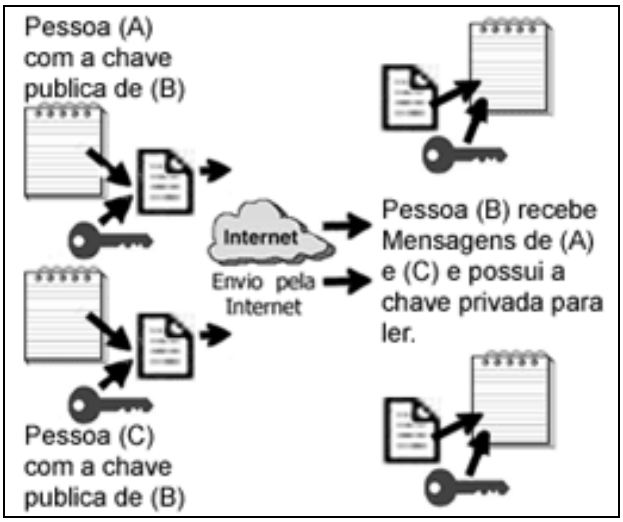

Figura 2 – Criptografia com chave pública

2.3 AUTENTICAÇÃO

Segundo Kurose (2003), a autenticação é o processo de provar a própria identidade a alguém. Como seres humanos, autenticamos mutuamente por meio visual ou até pela voz. Ao fazer a autenticação em uma rede, as partes comunicantes não podem confiar em informações biométricas, como a aparência visual ou a voz característica. Em uma rede a autenticação é feita somente na base de mensagens e de dados trocados como parte de um protocolo de autenticação. O protocolo de autenticação deve ser executado primeiramente antes de rodar qualquer outro protocolo a fim de estabelecer as identidades das partes de maneira satisfatória para ambas.

A escolha do mecanismo de autenticação apropriado depende do ambiente onde se dará a autenticação. Podemos citar as seguintes situações:

a) os parceiros e os meios de comunicação são todos confiáveis: a identificação pode ser confirmada por uma senha. Autenticação mútua pode ser implementada com a utilização de uma senha distinta em cada direção da comunicação;
b) cada identidade confia no seu parceiro, porém não confia no meio de comunicação: a proteção contra ataques pode ser fornecida com o emprego de métodos de criptografia, como por exemplo, utilização da criptografia por chave pública;
c) as entidades não confiam nos seus parceiros (ou sentem que não poderão confiar no futuro) nem no meio de comunicação: devem ser usadas técnicas que impeçam que uma entidade negue que enviou ou recebeu uma unidade de dados. Devem ser empregados mecanismos de assinatura digital (SOARES, 1995).

2.4 INTEGRIDADE DOS DADOS

Os mecanismos de integridade de dados atuam em dois níveis: controle da integridade de unidade de dados isolada e controle de integridade de uma conexão, isto é, das unidades de dados e da seqüência de unidades de dados transmitidas no contexto da conexão (SOARES, 1995).

Para garantir a integridade das unidades de dados (pacotes), podem ser usadas técnicas de detecção de modificações, que são normalmente associadas com a detecção de erros em bits, blocos, ou erros de seqüência e redes de comunicação, são usados para garantir a integridade dos dados. Porém, caso os cabeçalhos não sejam protegidos contra modificações, podem ser contornadas as verificações de integridade desde que sejam conhecidas as técnicas utilizadas. Portanto a garantia de integridade está mesmo nas informações de controle que devem ser mantidas confidenciais e íntegras para serem utilizadas nas detecções das modificações.

Para controlar modificações na seqüência de unidades de dados transmitidos em uma conexão, ou seja, pacotes transmitidos em uma conexão, são necessárias técnicas que garantam a integridade desses pacotes, de forma a garantir que as informações de controle não sejam corrompidas, em conjunto com informações de controle de seqüência. Esses cuidados,

apesar de não evitarem a modificação da cadeia de pacotes, garantem a detecção e notificação dos ataques (SOARES, 1995).

2.4.1 Assinatura digital

No mundo digital, freqüentemente se quer indicar o dono ou o criador de um documento ou deixar claro que alguém concorda com o seu conteúdo. A assinatura digital é a técnica criptográfica usada para cumprir essa tarefa (KUROSE, 2003).

Alguns algoritmos de criptografia de chave-pública permitem que sejam utilizados para gerar o que se denomina de assinaturas digitais. Estes algoritmos têm a característica de cifrar com a chave-pública e decifrar com a chave-privada, resultando na recuperação da mensagem. Obviamente esta forma de uso não assegura o sigilo da mensagem, uma vez que qualquer um pode decifrar o criptograma, dado que a chave-pública é de conhecimento público. Entretanto, se esta operação resulta na “mensagem esperada”, e pode-se ter a certeza de que somente o detentor da correspondente chave-privada poderia ter realizado a operação de cifração. Assim, uma assinatura digital é o criptograma resultante da cifração de um determinado bloco de dados (documento) pela utilização da chave-privada de quem assina em um algoritmo assimétrico.

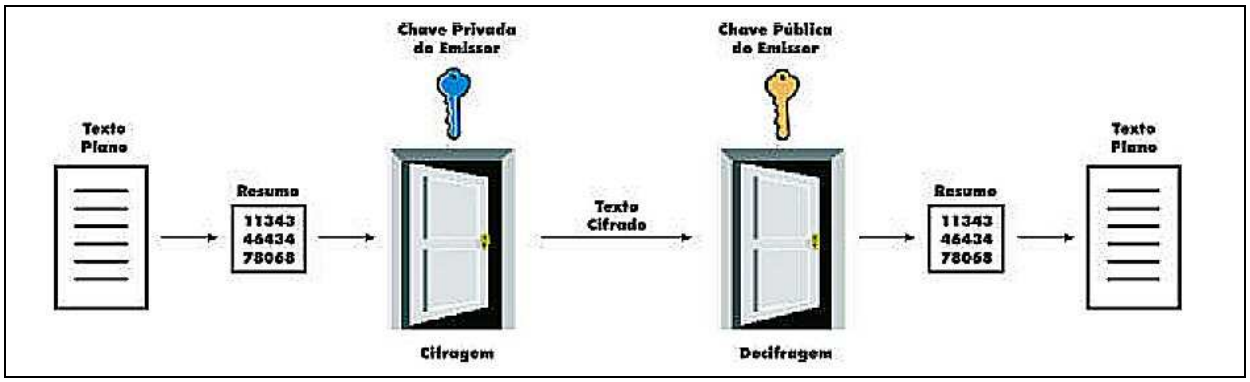

Figura 3 – Geração de Assinatura Digital de um documento

O processo de geração da assinatura utiliza a função *hash* que será abordada na sessão 2.4.2, para a obtenção do resumo do documento, que, em seguida, cifra-o com a chave privada do emissor e envia-o ao receptor. Este utilizará a chave pública do emissor para decifrar a mensagem e a função hash para recalcular o resumo do documento, comparando-o com o resumo recebido, segundo a Figura 3. Isto garante a integridade do documento (VASQUES e SCHUBER, 2002).

2.4.2 *Hash*

Os “resumos de mensagem”, também conhecidos com *Hash* ou *Message Digest*, são utilizados para garantir a integridade de documentos eletrônicos, protegendo-os de alterações por falha técnica ou fraude. Resumo de Mensagem ou Função Hash é a técnica permite que, ao ser aplicada a uma mensagem de qualquer tamanho, seja gerado um resumo criptografado de tamanho fixo e bastante pequeno, como por exemplo 128 bits. Algumas das propriedades da função *Hash*:

a) não é possível fazer a operação reversa, ou seja, dado um resumo é impossível obter a mensagem original;
b) duas mensagens diferentes, quaisquer que sejam, estatisticamente não podem produzir um mesmo resumo.

Uma boa função *hash* tem uma característica chamada de “efeito avalanche”, que significa que uma pequena mudança no arquivo de entrada acarreta uma grande e imprevisível mudança no sumário. Para uma mudança de um bit qualquer de entrada, levará a uma mudança de, em média, metade dos bits na saída, de maneira imprevisível. A funções *hash* tem grande emprego na criação de códigos para autenticação ou para verificação da integridade de arquivos (ou mensagens). Essas funções podem ser implementadas juntamente com uma chave. Neste caso, só pode ser utilizada por alguém que conheça a chave (ANTON, 2001).

Uma função *hash* simples pode ser obtida com o operador XOR (ou exclusivo) ou “shift” (deslocamento) circular à direita ou à esquerda.

Um outro exemplo de função *hash* pode ser obtido utilizando-se o valor ASCII dos caracteres que compõem um arquivo, o valor *hash* é obtido após os seguintes passos:

a) lê-se os dois primeiros caracteres e efetua-se um XOR com os valores lidos;
b) lê-se o terceiro caractere e realiza-se um XOR com o resultado do XOR anterior, e assim por diante, a cada caractere lido faz se um XOR com o resultado do último XOR, até o final do arquivo.

A tabela 2 representa o cálculo do valor hash de um o arquivo contenha o texto “HOJE”.

Tabela 2 – Exemplo simples de cálculo do valor "*hash*"

	Caractere	Código ASCII (em decimal)	Código ASCII (em binário)	XOR
1º Caractere	H	72	01001000	-
2º Caractere	O	79	01001111	00000111
3º Caractere	J	74	01001010	01001101
4º Caractere	E	69	01000101	**00001000**

O *hash* é o valor 00001000, obtido após o último XOR. Neste exemplo empregou-se apenas o operador XOR em blocos de 8 bits. Melhores funções *hash*, geralmente, são obtidas combinando-se operações modulares com operadores lógicos, de deslocamento e manipulando blocos maiores, de 128, 256 ou 512 bits (ANTON, 2001).

Muitas funções *hash* podem ser obtidas gratuitamente (Figura 4). As mais famosas são as da família MD, especialmente a MD4 e a MD5 que serão abordadas na sessão 2.4.4. A SHA-1 e a RipeMD-160 são dois exemplos que ainda são consideradas perfeitas.

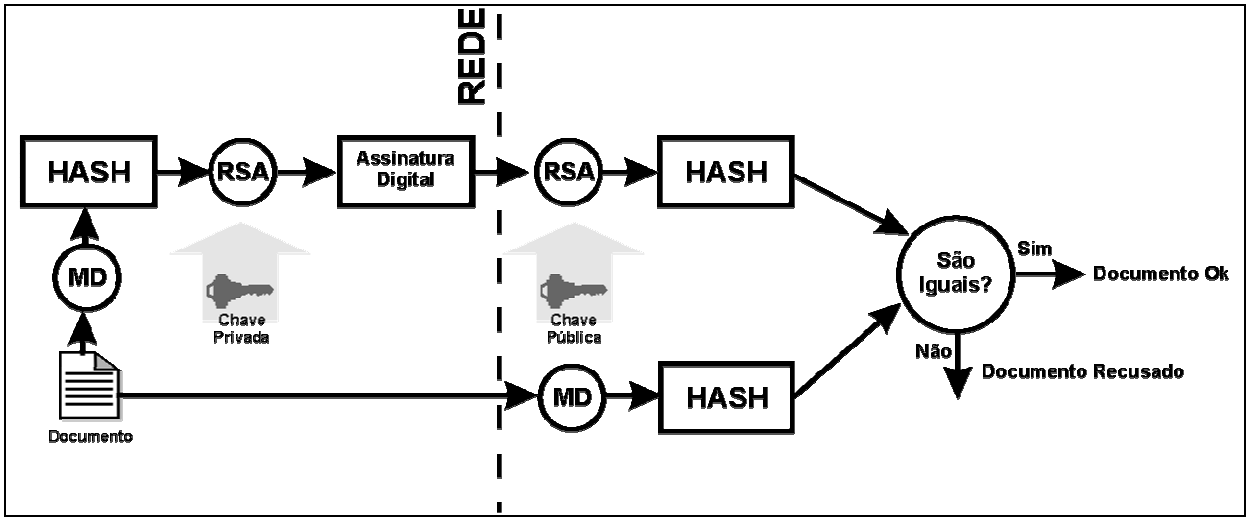

Figura 4 – Verificação da integridade de um documento através da função *hash*

2.4.3 Secure Hash Algorithm (SHA-1)

O *Secure Hash Algorithm* (SHA-1) é um algoritmo definido para ser utilizado no padrão *Digital Signature Standard* (DSS). A função deste algoritmo é gerar um resumo de uma mensagem de tamanho fixo, que seja único e inviolável, ou seja, dada uma mensagem M, o algoritmo gera uma assinatura para esta mensagem *A(M)* que demonstrará se a mensagem original foi alterada de alguma forma.

Quando uma mensagem qualquer de tamanho menor que 264 bits é introduzida no algoritmo, este gera uma saída de 160 bits chamada *message digest* (resumo da mensagem). Esse resumo pode ser incorporado à mensagem original e transmitido num bloco, formando assim uma mensagem assinada. Quando esta mensagem chega ao receptor, este retira a assinatura do bloco, reaplica nesta o algoritmo e compara o resultado com a assinatura recebida. Caso estas sejam iguais, pode-se garantir que a mensagem não sofreu alterações no seu trajeto (ANTON, 2001).

O SHA-1 é chamado seguro porque é computacionalmente inviável encontrar uma mensagem correspondente a um determinado resumo de mensagem, ou encontrar duas mensagens que produzem resumos idênticos. É resultante do aperfeiçoamento do SHA, onde uma operação de deslocamento circular para a esquerda foi adicionada ao algoritmo original para incrementar a segurança.

O SHA trabalha sobre uma mensagem transformada em uma string de bits, cujo tamanho deve ser múltiplo de 512 bits. O SHA processa sequencialmente blocos de 512 bits para computar o resumo da mensagem, e caso o tamanho total da mensagem não seja múltiplo de 512 bits é necessária a realização de uma complementação da mesma. Supondo uma mensagem de comprimento L menor que 2^{64}, o processo de complementação é o seguinte:

a) um bit "1" é adicionado ao final da mensagem;
b) "0"s são adicionados de acordo com o número de bits da mensagem.Os últimos 64 bits do bloco de 512 são reservados para o valor do comprimento da mensagem original;
c) adiciona-se ao final da mensagem a representação do comprimento da mensagem original em forma de duas palavras de 32 bits.

2.4.4 Message Digest 5 (MD5)

O *Message Digest 5* (MD5) é um algoritmo definido para ser utilizado no padrão *Digital Signature Standard* (DSS). Assim como o SHA, a função deste algoritmo é gerar um resumo de uma mensagem de tamanho fixo, que seja único e inviolável. Quando uma mensagem de comprimento qualquer é introduzida no algoritmo, este gera uma saída de 128 bits correspondente à *message digest* (ANTON, 2001).

O MD5 é derivado do MD4 e pretende ser mais robusto que este, apesar do primeiro ser bem mais rápido no processamento. O algoritmo MD5 tem a propriedade de que cada bit do código *hash* é uma função de cada bit da entrada. O autor do MD5 conjectura que a dificuldade de se gerar duas mensagens com o mesmo sumário é da ordem de 2^{64} operações, e a dificuldade de gerar uma mensagem que produza um dado sumário é da ordem de 2^{128} operações. Ambos os algoritmos MD5 e MD4 foram desenvolvidos pelo professor Ron Rivest.

2.5 CERTIFICADO DIGITAL

Segundo Dias (2003), em 1976, Diffie e Helmann propuseram o conceito de criptografia assimétrica, tornando possível pela primeira vez o estabelecimento de uma associação entre um documento e seu autor. Entretanto, a proposta de Diffie e Helmann não apresentava uma solução prática para a ligação entre a chave pública e seu proprietário, dando margem a fraudes. Em 1978, Loren Kohnfelder propôs o uso de uma terceira entidade confiável, denominada autoridade certificadora, que garantiria a ligação entre um usuário e sua chave pública. A criação de uma estrutura de dados assinada contendo uma identificação e a chave pública chamou-se de certificado digital.

O certificado digital pode ser entendido como sendo a identidade digital, e permite comprovar de forma digital a identidade do usuário. O certificado é emetido por uma autoridade certificadora digital CA (*Certificate Authority*), que pode ser uma empresa, organização ou indivíduo, público ou privado que atual como tabelião para verificar e autenticar a identidade de usuários de um sistema criptográfico de chave pública. A CA responsabiliza-se pela distribuição de chaves públicas e pela garantia de que uma determinada chave pública esteja seguramente ligada ao nome de seu dono.

Dentre os dados de um certificado digital, as seguintes informações estão presentes (Figura 5): chave pública do usuário, número de série do certificado, nome da CA que emitiu o certificado, a assinatura digital da CA, entre outras. A recomendação mais aceita e utilizada para a produção de certificados digitais é a X.509v3, formulada pela ITU-T (*International Telecommunication Union – Telecommunication Standardization Sector*) (VASQUES e SCHUBER, 2002) .

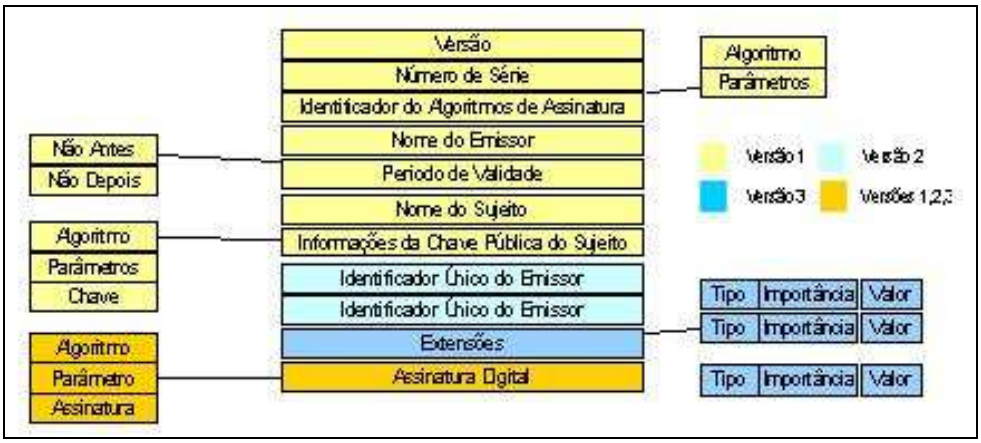

Figura 5 – Versões da estrutura de um certificado digital X.509

O X.509v1 tinha um número muito pequeno de campos o que limitava a sua utilização. Já o X.509v2 foi revisado e incluído novos campos possibilitando a reutilização de nomes iguais em diferentes certificados digitais. O X.509v3 resultado de novas experiências, foram adicionados campos de extensão, o que torna o certificado mais flexível possibilitando um leque maior de utilização.

3 TRANSFERÊNCIA DE DADOS

A transferência de arquivos é a etapa principal do processo de atualização. A necessidade da transferência de um arquivo para uma atualização é verificada através de um controle de versão que está incluído em uma *tag* XML que será abordada na sessão 3.3.

3.1 TRANSFERÊNCIA DE ARQUIVOS

O serviço *File Transfer Protocol* (FTP) é o serviço padrão da Internet para a transferência de arquivos entre computadores. A partir dele, usuários podem obter ou enviar arquivos de ou para outros computadores da Internet.

O funcionamento do FTP se baseia no estabelecimento de uma sessão limitada entre o cliente FTP local e o servidor FTP do equipamento remoto. Esta sessão é autenticada de forma semelhante ao serviço telnet, possuindo apenas comandos referentes à manipulação de diretórios e arquivos. O usuário pode pesquisar a estrutura de arquivos do equipamento remoto antes de executar as transferências desses arquivos.

A utilização mais comum do serviço FTP na Internet é a de obtenção de programas ou informações a partir de servidores de domínio público ou comercial, serviço conhecido como FTP Anônimo (*Anonymous FTP*). Para esse serviço, o servidor FTP oferece uma conta especial (conta anônima como nome de login) com autenticação flexível (normalmente a senha é apenas o endereço de correio eletrônico do usuário). Uma sessão de FTP anônimo possui acesso restrito aos arquivos que podem ser consultados ou transferidos para o computador do usuário.

O conjunto de arquivos referentes a um programa ou assunto oferecidos por um servidor FTP Anônimo é o que se chama de repositório. Por medida de segurança e disponibilidade, existe um processo conhecido como *FTP mirroring* que oferece cópias de um dado repositório em mais de um equipamento da Internet.

A operação do FTP baseia-se no estabelecimento de duas conexões entre o cliente (módulo que está solicitando o acesso a seus arquivos remotos) e o servidor (módulo que fornece acesso a seus arquivos locais). Uma conexão denominada “conexão de controle”, é usada para transferência de comandos; e a outra, denominada “conexão de transferência de dados”, é usada para a transferência dos dados. A conexão de controle permanece ativa

durante toda a sessão no FTP, e permite que sejam feitas várias conexões para transferência de dados (SOARES, 1995). O FTP permite que sejam transferidos tanto arquivos do tipo texto quando binários.

A arquitetura TCP/IP define um outro protocolo que fornece um serviço mais simplificado somente para transferência de arquivos, chamado de *Trivial File Transfer Protocol* (TFTP). O TFTP não implementa mecanismos de autenticação e opera em uma única conexão utilizando o UDP para o transporte de blocos de dados de tamanho fixo. Como o serviço UDP não garante a chegada dos pacotes de dados, o TFTP utiliza o protocolo de bit alternado para transmitir seus blocos (SOARES, 1995).

A necessidade da transferência de um arquivo durante o processo de atualização exige um mecanismo de controle de versão que será abordado na sessão 3.2.

3.2 CONTROLE DE VERSÃO

O sistema operacional Microsoft® Windows® adota um sistema de controle de versão em arquivos executáveis, bibliotecas dinâmicas DLLs e outros arquivos. Esse é um sistema chamado VERSIONINFO (Figura 6) que é um controle adicionado a esses arquivos para evitar conflitos de arquivos já instalados e arquivos sendo atualizados por um programa instalador ou um programa de atualização (MICROSOFT WINDOWS TEAM, 2003).

Figura 6 – Versão de um arquivo executável

O sistema de controle de versão permite evitar os seguintes problemas:

a) instalar versões mais antigas sobre versões mais atuais;
b) atualizar versões de componentes de um sistema com versões incompatíveis;
c) atualizar uma versão de outra língua em um sistema multilíngüe.

O controle de versão além de guardar informações sobre a versão e a linguagem também armazena informações sobre autor, data de compilação, direitos autorais e descrição do arquivo.

Conforme a figura 6, a versão do arquivo é divida em 4 dígitos conforme descritos a seguir:

a) *major version* (versão maior): primeiro dígito, tem como objetivo identificar uma grande alteração no aplicativo, implicando ás vezes em mudança de todo o seu funcionamento;

b) *minor version* (versão menor): segundo dígito, identifica uma alteração menor no aplicativo geralmente indica que uma nova funcionalidade foi anexada;
c) *release* (revisão): terceiro dígito, geralmente identifica uma alteração para arrumar um erro no aplicativo;
d) *build* (compilação): quarto dígito, indica a compilação diária do arquivo, quantas vezes o mesmo foi compilado naquela versão.

A informação de versão é utilizada no controle de versão, deve ser incluída no próprio arquivo, ou para pode ser utilizado uma estrutura XML abordada na sessão 3.3, e que conterá a informação da versão relacionada ao arquivo.

3.3 EXTENSIBLE MARKUP LANGUAGE (XML)

A *eXtensible Markup Language* (XML) é uma nova linguagem de marcação desenvolvida pelo *World Wide Web Consortium* (W3C) principalmente para solucionar as limitações da HTML. Embora se baseie na mesma tecnologia da linguagem HTML, é projetada para executar melhor a tarefa de gerenciamento das informações, exigida pelo crescimento atual da Internet.

Em vez de funcionar apenas como uma linguagem para a Web, na XML é possível criar marcadores personalizados para formatar determinados documentos para determinada necessidade. A linguagem de marcação personalizada é chamada de aplicação XML, que irá conter *tags* que descrevem de fato os dados contidos nela. Se uma *tag* identifica algum dado, os dados ficam disponíveis para outras tarefas onde é possível escrever um programa para extrair apenas as informações necessárias (MARCHAL, 2000).

4 DESENVOLVIMENTO DO TRABALHO

Este capítulo aborda o desenvolvimento do trabalho, mostrando a especificação com diagramas de casos de uso, diagrama de classes e diagrama de seqüência.

4.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O protótipo desenvolvido neste trabalho utiliza a criptografia e a assinatura digital para garantir a integridade de um arquivo que foi transferido pela internet. A transferência do arquivo é feita pelo *File Transfer Protocol* (FTP). Esta conexão é criptografada através do *Secure Sockets Layer* (SSL) que implementa chave secreta e a integridade dos dados é confrontada através do algoritmo *hash*.

O usuário detentor de uma nova atualização de um arquivo ou software deve entrar com a informação da versão que será postada no repositório, e selecionar o arquivo desta versão, e como saída terá um arquivo XML contendo a versão e o código *hash* do arquivo.

O usuário final deverá solicitar uma atualização e como resposta deverá receber um aviso que o arquivo foi atualizado ou não.

Através deste protótipo é possível fazer atualizações de softwares ou arquivos digitais garantindo que o arquivo que foi transferido é idêntico ao que foi postado no repositório.

Os requisitos funcionais (RF) e não funcionais (RNF) do protótipo proposto são:

a) disponibilizar um mecanismo de segurança para não permitir que outros usuários alterem o arquivo sendo atualizado (RF);
b) conectar automaticamente ao site (que será desenvolvido para testes do protótipo) e fazer *download* dos arquivos (RF);
c) ser de fácil utilização. O usuário somente precisa executar o aplicativo de atualização e aguardar o término da operação (RNF);
d) mostrar o percentual de progresso do *download* do arquivo (RNF).

O desenvolvedor disponibilizará em um servidor os arquivos a serem atualizados juntamente com o arquivo XML de controle de versão.

Para fazer a especificação deste protótipo foi utilizada a *Unified Modeling Language* (UML), e foi utilizado como ferramenta o Rational Rose.

4.2 ESPECIFICAÇÃO

Nesta sessão será apresentada a especificação do protótipo com diagramas da UML.

4.2.1 CASOS DE USO

O protótipo possui 2 casos de uso (Figura 7):

a) atualiza servidor: o desenvolvedor fica responsável por atualizar no servidor FTP as novas versões dos arquivos e o arquivo XML de controle de versão;
b) solicita atualização: o usuário deverá solicitar uma atualização.

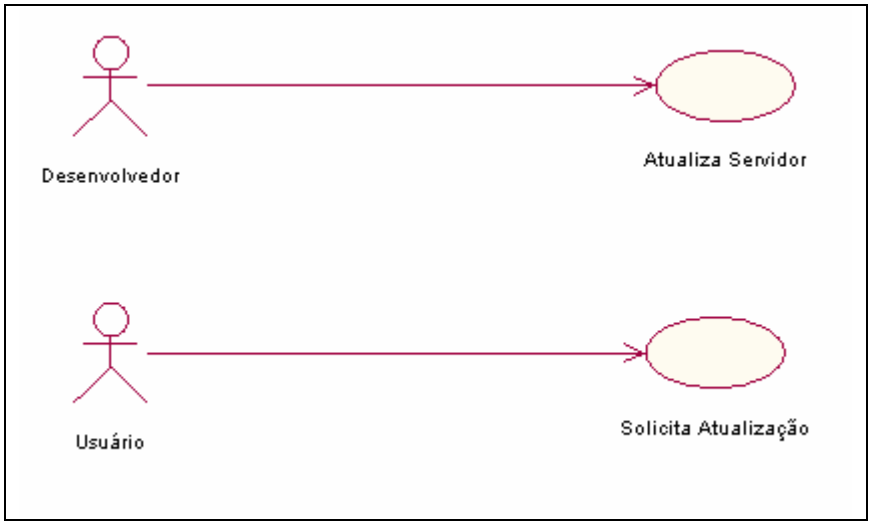

Figura 7 – Casos de uso

4.2.2 Diagrama de Classes

Foram criadas 3 classes:

a) TFileTransfer: classe responsável pela conexão FTP com suporte SSL e pela transferência dos arquivos. Utiliza mecanismos da biblioteca de componente *Internet Direct* (INDY) desenvolvida por terceiros (HOWER, 2004);
b) TXMLControl: cria e gerência o arquivo de controle de versão XML. É possível incluir vários registros de arquivos com versões diferentes. Cada registro contém informações como: nome do arquivo, versão, data de criação e o código *Hash* do arquivo físico;
c) TMD5HashCode: rotinas para geração e exibição do código *Hash*, utilizando o algoritmo MD5. Utiliza mecanismos da biblioteca de componente *Internet Direct* (INDY) desenvolvida por terceiros (HOWER, 2004).

O diagrama de classes é demonstrado na figura 8.

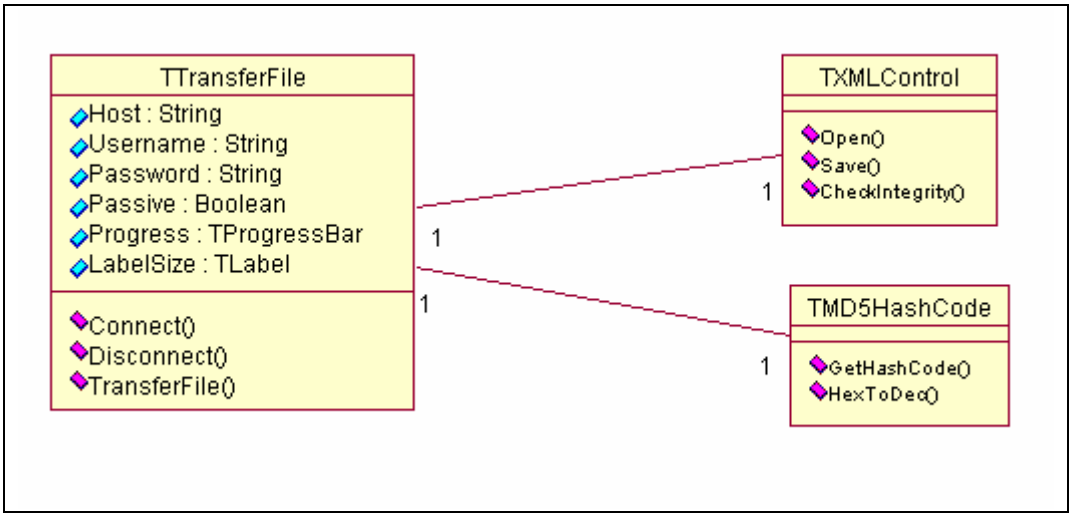

Figura 8 – Diagrama de Classes

4.2.3 Diagramas de Seqüência

Os diagramas de seqüência demonstram como são executadas as tarefas dentro do protótipo e também as mensagens trocadas entre as classes.

4.2.3.1 Gerar Atualização

Este diagrama de seqüência mostrado na figura 9, é executado pelo protótipo de geração do arquivo XML de controle de versão. Durante a inicialização do mesmo é instanciada a classe TXMLControl. Essa classe gera e controla um arquivo XML para controle de versão e de integridade dos arquivos na atualização. Em seguida é chamado o método “Open” que faz a leitura do arquivo e caso o arquivo não exista será criado. O método “Save” salva todas as informações do formulário.

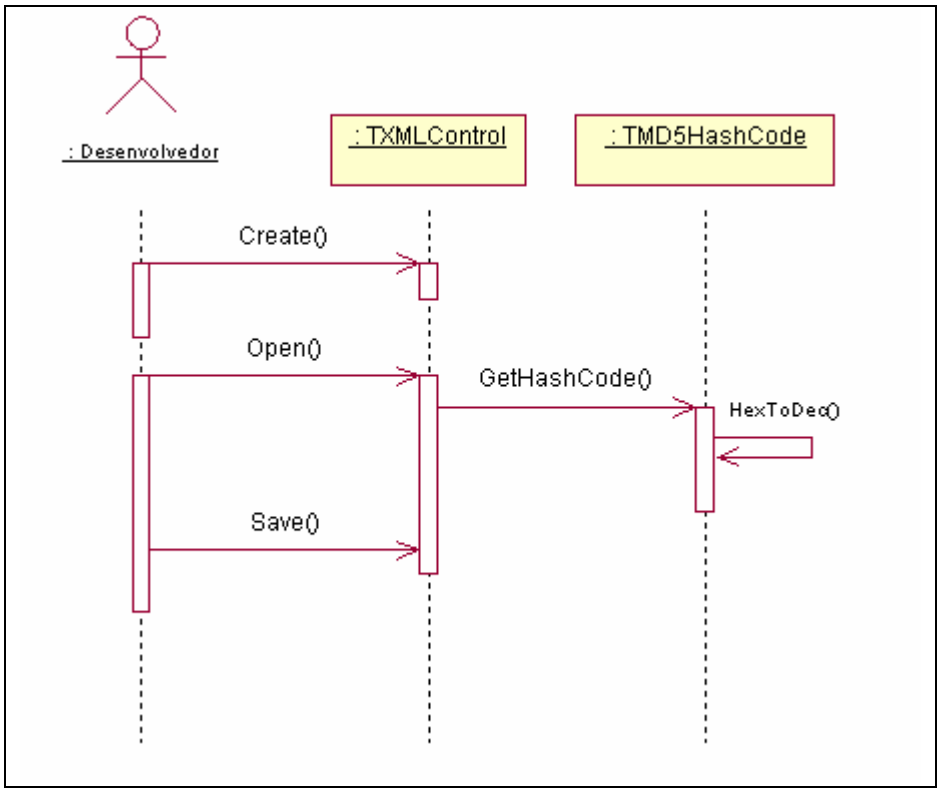

Figura 9 – Diagrama de Seqüência “Gerar Atualização”

4.2.3.2 Solicitar Atualização

Este diagrama de seqüência mostrado na figura 10, ocorre quando o usuário final solicita uma atualização através da execução do protótipo. Na inicialização são criadas as classes TXMLControl buscando as informações do servidor para atualização dos arquivos. Em seguida é instanciada a classe TTransferFile que faz o *download* do novo arquivo de controle de versão que contém a informação das novas versões disponíveis, onde é instanciada outra classe TXMLControl que será utilizada pelo novo arquivo XML de controle de versão. Caso exista atualização é chamado novamente o método “TransferFile”. Por último é chamado o método de “GetHashCode” da classe TMD5HashCode utilizado na verificação da integridade e o método Disconnect() da classe TTransferFile finaliza a conexão FTP e o protótipo é encerrado.

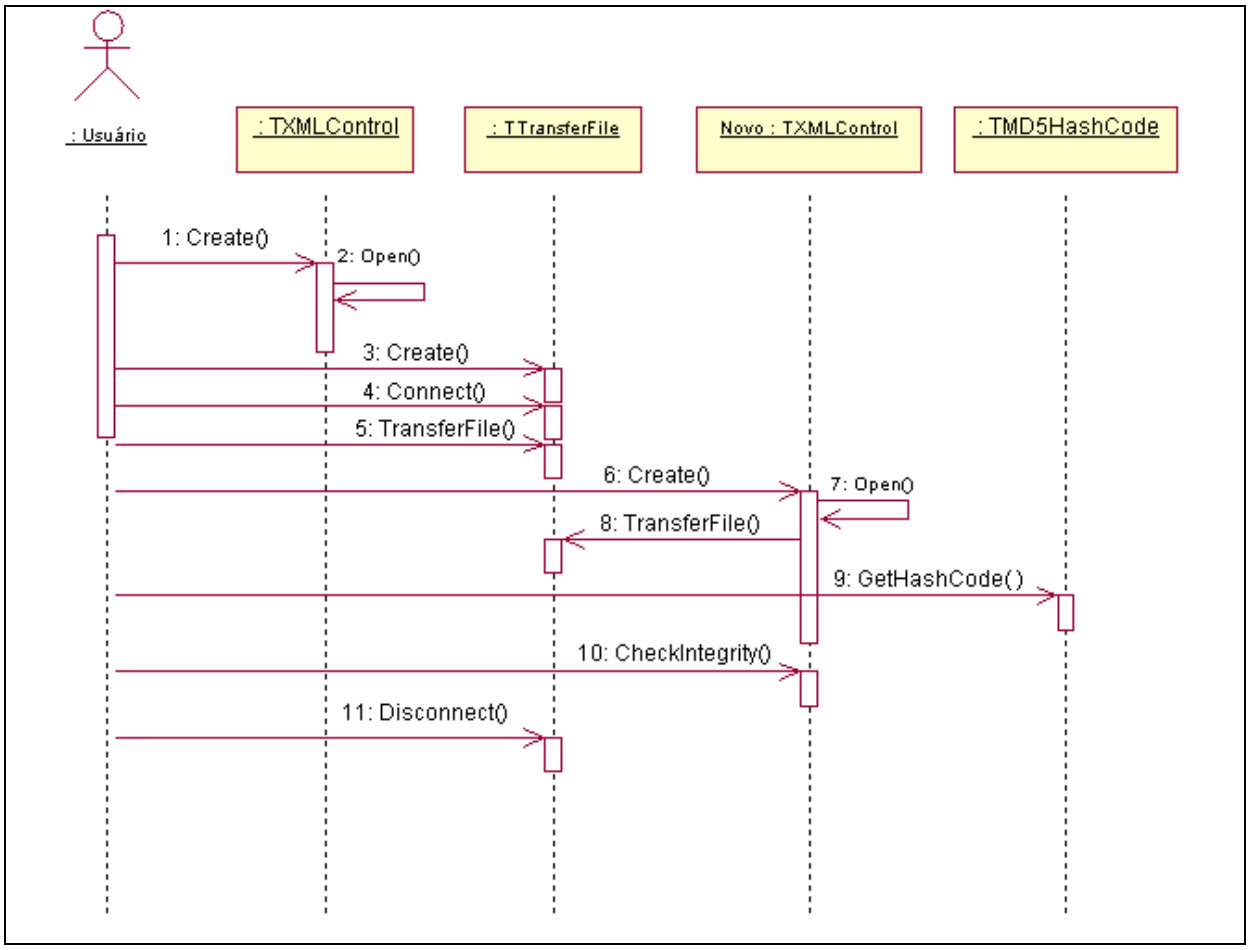

Figura 10 – Diagrama de Classes “Solicitar Atualização”

4.3 IMPLEMENTAÇÃO

Nesta sessão será apresentada a implementação do protótipo desenvolvido no ambiente de programação *Borland Delphi 7.0*.

4.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

A implementação dividida em dois protótipos:

a) gerar: este protótipo gera o arquivo XML de controle de versão contendo as novas versões dos arquivos que serão postados no servidor FTP. Neste protótipo são entradas informações de cada arquivo como: nome do arquivo, versão, data de criação e o código *Hash* que é gerado automaticamente por este protótipo. São também armazenadas informações como endereço do servidor FTP, usuário e senha para acesso para atualização;

b) atualizar: este protótipo busca o novo arquivo de atualização no servidor FTP para comparar com as versões existentes e faz *download* dos arquivos que contenham uma versão mais atual e por fim faz a verificação de integridade destes arquivos.

O arquivo XML de controle de versão é gerenciado por uma classe que é herança do componente TClientDataSet que faz parte da palheta de componentes do Delphi 7.0. Essa classe já possui vários tratamentos para arquivos XML, desde a criação, campos, armazenamento e gravação do arquivo. Na figura 11 é apresentado um exemplo do arquivo XML de controle de versões.

```
<?xml version="1.0" standalone="yes"?>
<DATAPACKET Version="2.0">
  <METADATA>
    <FIELDS>
      <FIELD attrname="File" fieldtype="string" required="true" WIDTH="100"/>
      <FIELD attrname="Version" fieldtype="string" required="true" WIDTH="13"/>
      <FIELD attrname="Date" fieldtype="date" required="true"/>
      <FIELD attrname="MD5" fieldtype="string" required="true" WIDTH="32"/>
      <FIELD attrname="HOST" fieldtype="string" required="true" WIDTH="40"/>
      <FIELD attrname="PASSIVE" fieldtype="boolean" required="true"/>
      <FIELD attrname="USERNAME" fieldtype="string" WIDTH="30"/>
      <FIELD attrname="PASSWORD" fieldtype="string" WIDTH="30"/>
    </FIELDS><PARAMS/>
  </METADATA>
  <ROWDATA>
    <ROW File="calculadora.exe" Version="01.00.00.0000"
         Date="20041020" MD5="FF2DB66B40AC1B9AC5A5C99FDF7C9829"
         HOST="ftp.inf.furb.br" PASSIVE="TRUE" USERNAME="airison"
         PASSWORD="******"/>
  </ROWDATA>
</DATAPACKET>
```

Figura 11 – Arquivo “update.xml” para controle de versão e integridade

O arquivo “update.xml” estará no mesmo diretório do protótipo de atualização e do software ou arquivos que serão atualizados. É a partir da informação HOST que o programa de atualização cria uma conexão FTP com um servidor na internet e faz a transferência de um novo arquivo “update.xml” onde este conterá informações da última versão daquele aplicativo ou dos arquivos em questão. Após a transferência do novo arquivo “update.xml” é feita a comparação das versões dos arquivos, e os que possuírem uma nova versão são solicitados para transferência. A cada arquivo transferido é obtido seu código *hash* pelo algoritmo MD5 e feita a comparação em seguida com o campo MD5 do arquivo XML para verificar a integridade do arquivo.

O protótipo para gerar o arquivo “update.xml”, que pode ser visualizado na figura 12, possui fácil usabilidade. O primeiro dado a ser informado são as configurações do servidor para atualização. Em seguida são relacionados os arquivos que compõem a atualização, ao relacionar um arquivo o protótipo já alimenta seu código *hash*.

Figura 12 – Tela do Protótipo para gerar atualização

O protótipo de atualização, que pode ser visualizado na figura 13, também possui fácil usabilidade. A sua tela principal posicionada no canto superior esquerdo tem o objetivo de indicar as tarefas que serão realizadas e as já compridas. Ao ser executado a primeira tarefa é realizar uma conexão FTP, porém essa conexão tem que ser segura desde a sua autenticação, para isso é utilizado uma autenticação SSL. Tudo isso se faz necessário quando se quer alcançar o objetivo de garantir a integridade dos arquivos atualizados.

A conexão é realizada utilizando o componente TIdFTP da biblioteca de componente *Internet Direct* (INDY) desenvolvida por terceiros (HOWER, 2004). Através do método TransferFile() da classe TFileTransfer é solicitada a transferência de um arquivo.

A versão dos arquivos é comparada diretamente nas duas classes de controle de versão XML, uma utiliza a estrutura atual antes da atualização e a outra é o novo arquivo XML.

Por final, a tecnologia de segurança que garante que o arquivo não sofreu nenhum dando durante a sua transmissão e recepção é o algoritmo MD5. O código *Hash* é retornado na classe TMD5HashCode utilizando rotinas da biblioteca de componente *Internet Direct* (INDY) desenvolvida por terceiros (HOWER, 2004).

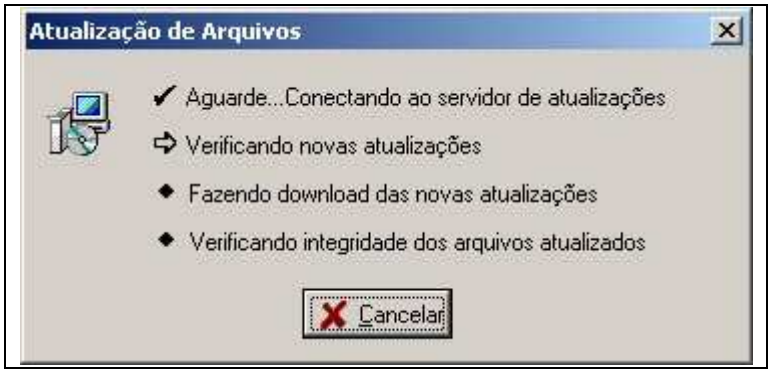

Figura 13 – Protótipo de atualização de arquivos

4.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Para ser útil, o protótipo deve estar instalado no mesmo diretório do software ou arquivos a serem atualizações e conter nesse mesmo diretório um arquivo de controle de versão contendo as informações sobre o servidor FTP e os arquivos inicialmente instalados. Os arquivos podem ter qualquer tamanho ou extensão, terão o mesmo tratamento para todos.

Para obter uma atualização, o servidor deverá conter arquivos com versões maiores e o arquivo de controle de versão atualizado para esta versão. O arquivo de controle de versão é gerado pelo protótipo “gerar”.

Para testar o tratamento de integridade do arquivo, o protótipo de atualização deve ser executado passando o parâmetro “teste”, isso fará com que o próprio protótipo force uma alteração do arquivo depois que ele foi transferido. Essa alteração é a gravação do caractere “A” no final do arquivo, e com esta pequena alteração o código MD5 ficará completamente diferente.

4.3.3 CASO DE USO “ATUALIZA SERVIDOR”

O caso de uso “Atualiza servidor” ocorre quando o desenvolvedor do software deseja publicar uma nova versão dos arquivos para atualização em um servidor. O desenvolvedor deverá executar o protótipo “gerar” para criar o novo arquivo de controle de versão.

O primeiro passo é informar os campos “Endereço do servidor FTP”, “Usuário”, “Senha” e o modo de conexão “Modo passivo”, este último é utilizado para servidores que estão protegidos por um *firewall*.

Utilizando o navegador, ao clicar no botão “+” é possível incluir um novo arquivo para esta atualização. No botão selecionar arquivo, é selecionado o arquivo físico que fará parte da atualização. Para cada arquivo deve-se informar a sua versão e a data de geração.

Para confirmar as alterações o usuário deverá usar a opção de menu “salvar”. A opção “limpar” para limpar e entrar com todas as informações novamente.

4.3.4 CASO DE USO “SOLICITA ATUALIZAÇÃO”

O caso de uso “Solicitar atualização” ocorre quando o usuário do protótipo de atualização deseja verificar se aquele software ou quaisquer outros arquivos que façam parte da atualização possuem uma nova versão. Neste momento, o usuário executa o protótipo de atualização, este tentará uma conexão com o servidor FTP configurado no arquivo de controle de versão, obtendo êxito na conexão é feito *o download* de um novo arquivo de controle de versão onde o protótipo irá obter a informação da existência de uma nova atualização.

Caso existam novas atualizações é questionado ao usuário se deseja prosseguir com a atualização e em seguida é apresentada uma barra de progresso demonstrando o progresso do *download* de cada arquivo.

Se o arquivo que foi transferido não estiver integro, será exibida uma mensagem e o protótipo fará o *download* novamente. Se todos os arquivos foram transferidos e estiverem íntegros é exibida a mensagem que a atualização foi concluída com sucesso.

4.4 RESULTADOS E DISCUSSÃO

Para ficar mais claro o funcionamento deste protótipo e como uma de suas importâncias está ligada a área de segurança em redes, serão discutidos testes e resultados obtidos por esse esquema de segurança.

Supondo um arquivo chamado “teste.txt” que possua o conteúdo “ABC”, este arquivo possui apenas 3 bytes. Alterando o conteúdo do arquivo para “abc” e extraindo o código h*ash* nas duas etapas é possível concluirmos que por menor que seja a diferença nos arquivos ou

dos seus tamanhos o código *hash* é totalmente diferente. A tabela a seguir demonstra os resultados obtidos na realização deste teste utilizando o algoritmo MD5 que foi utilizado no protótipo.

Tabela 3 – Resultados obtidos em teste com o MD5

Conteúdo do arquivo "teste.txt"	**MD5**
"ABC"	902FBDD2B1DF0C4F70B4A5D23525E932
"abc"	900150983CD24FB0D6963F7D28E17F72

Considerando o desenvolvimento de um software cujo nome do único arquivo e "caculadora.exe". Este software foi finalizado na versão 01.00.00.000, gerado o seu arquivo de controle de versão e instalado em um usuário qualquer juntamente com o protótipo de atualização e o arquivo de controle de versão.

Posteriormente o software "calculadora.exe" ganhou uma nova versão 01.01.00.0000 e da mesma maneira foi gerado seu arquivo de controle de versão atualizado e postado no servidor FTP. Com o controle de versão foi possível ao protótipo verificar para o usuário que existia uma nova atualização do software e garantir que ele tenha uma progressão e não uma regressão de versão.

Durante a atualização do software "calculadora.exe" foi utilizado o parâmetro "teste" no protótipo assim obtendo os seguintes resultados demonstrados em uma mensagem do protótipo que pode ser visualizada na figura 14.

Figura 14 – Mensagem de resultado do teste aplicado

O tempo da conexão FTP utilizando a autenticação SSL, foi um pouco lento nos testes onde foi utilizado servidor acadêmico da Universidade Regional de Blumenau. Porém para ema atualização não se requer grande *performance* nesta tarefa, pois o usuário não vai precisar verificar atualizações a cada instante.

5 CONCLUSÕES

Foram apresentados e estudados vários algoritmos de criptografia e algoritmos de resumo de mensagem (*hashing*) como o MD5, seus resultados, eficiências e deficiências na utilização sobre arquivos. Destaca-se também a utilização de arquivos XML como arquivo de controle de versão.

Estabelecendo uma conexão segura e autenticada com o uso da criptografia simétrica em conjunto com a utilização do resumo de mensagem torna-se muito seguro, praticamente impossível de violar a transferência de qualquer arquivo que será atualizado.

Com um arquivo XML é possível controlar versão de qualquer arquivo de qualquer extensão, e não somente executáveis ou DLL`s como o sistema operacional Windows controla hoje.

O tempo de processamento do algoritmo MD5 não é um caso preocupante nos dias de hoje, pois temos computadores com boa capacidade de processamento e esse tarefa de atualização não deverá ser utilizado com arquivos muito grandes. Em comparação com o tempo de conexão e autenticação, foi o mais demorado mas não chegou a ser uma grande preocupação.

Não foi utilizado algoritmo de verificação de erro como o *Cyclical Redundancy Checking* (CRC) nos arquivos transmitidos, pois a verificação através do código *hash* é muito seguro e eficiente garantindo que o arquivo que foi transferido é igual ao que está no servidor.

A biblioteca de componente INDY é uma ferramenta completa para soluções em rede e internet, é uma biblioteca de código aberto e distribuição gratuita. Possui implementação para os principais protocolos e também algoritmos de segurança prontos.

Um das deficiências do protótipo apresentado é que a atualização é permitida apenas para arquivos do mesmo diretório que o protótipo está executando.

A maior dificuldade encontrada foi no levantamento bibliográfico sobre controle de versão. Não foi encontrada uma padronização a nível mundial.

5.1 EXTENSÕES

Como extensão deste trabalho, pode-se estudar outros algoritmos de criptografia como o SHA-1. Estudar novas técnicas de atualização, ou sincronização de dados entre servidores, como os servidores FTP espelhos.

Com a importância dessa segurança poderia ser criado um novo protocolo de transferência de arquivos, como uma extensão ao FTP e ao TFTP. Após a sincronização dos arquivos em ambos os lados o protocolo faria a verificação de integridade sem a necessidade de um arquivo de controle.

REFERÊNCIAS BIBLIOGRÁFICAS

ANTON, Eric Ricardo. **Projeto e implementação de um protocolo para transmissão multidestinatária segura**. 2001. 121 f. Trabalho de Conclusão de Curso (Engenharia Elétrica) – Departamento de Eletrônica, Escola de Engenharia da UFRJ – Universidade Federal do Rio de Janeiro, Rio de Janeiro.

DIAS, Júlio da Silva. **Confiança no documento eletrônico**. 2003. 141 f. Tese (Doutorado em Engenharia de Produção e Sistemas) – Programa de Pós-Graduação em Engenharia de Produção e Sistemas, Universidade Federal de Santa Catarina, Florianópolis.

HOWER, Chad. Z. **The indy project**. [s.l.], 2004. Disponível em: <http://www.indyproject.org>. Acesso em: 10 jul. 2004.

KUROSE, James F; ROSS, Keith W. **Redes de computadores e a internet :** uma nova abordagem. São Paulo: Pearson Brasil, 2003. 548 p.

LUCCHESI, Claudio Leonardo. **Introdução à criptografia computacional**. Campinas: Papirus/UNICAMP, 1986. xiii, 132 p.

MARCHAL, Benoit. **XML:** conceitos e aplicações. São Paulo: Berkeley Brasil, 2000. xv, 548 p.

MARIANO, Ismael da Silva. **IPSec e DDoS,** aspectos de segurança em redes TCP/IP. 2001. 78.f. Monografia (Mestrado em Engenharia de Sistemas) – Coordenação do Programa de Pós-Graduação da UFRJ – Universidade Federal do Rio de Janeiro, Rio de Janeiro.

MICROSOFT WINDOWS TEAM. **Microsoft Windows XP professional resource kit**. 2.ed. Redmond, Wash: Microsoft Press, 2003. xxxiii, 1706 p.

SOARES, Luiz Fernando G. et al. **Redes de computadores**: das LANs, MANs e WANs às redes ATM. 2.ed. Rio de Janeiro: Campus,1995. 705 p.

TANENBAUM, Andrew S. **Redes de computadores**. Rio de Janeiro: Campus, 1997. 786 p.

VASQUES, Alan Tamer; SCHUBER, Rafael Priante. **Implementação de uma VPN em Linux utilizando o protocolo IPSec**. 2002. 72 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Área de Ciências Exatas e Tecnológicas, Centro Universitário do Estado do Pará, Belém.