GUSTAVO YAMASAKI MARTINS VIEIRA

Projeto de um dispositivo de autenticação e assinatura

Dissertação de mestrado apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do Título de Mestre em Engenharia

São Paulo

2007

GUSTAVO YAMASAKI MARTINS VIEIRA

Projeto de um dispositivo de autenticação e assinatura

Dissertação de mestrado apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do Título de Mestre em Engenharia

Área de concentração:

Sistemas Digitais

Orientador:

Professor Dr. Wilson Vicente Ruggiero

São Paulo

2007

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 14 de novembro de 2007.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Vieira, Gustavo Yamasaki Martins
Projeto de um dispositivo de autenticação e assinatura / G. Y.M. Vieira. -- ed.rev. -- São Paulo, 2007.
175 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Segurança na Internet 2.Autenticação de mensagens (Dispositivos) I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

Aos meus pais, minha irmã e meu irmão

AGRADECIMENTOS

Este trabalho é o fruto de um trabalho de três anos e teve a contribuição de muitas pessoas até a sua versão final. Gostaria de agradecer a todos que de alguma forma me ajudaram ou contribuíram para a sua conclusão. A primeira pessoa é o Professor Wilson Ruggiero, afinal, foi ele quem orientou, deu sugestões, acompanhou o desenvolvimento, leu e releu esse texto.

Outro professor importante, foi o Paulo Barreto. Ele ajudou na implementação de algoritmos, na medição de tempos e além de dúvidas que tive ao longo do trabalho.

Além da ajuda dos professores, dentro do LARC, muitas pessoas gastaram um pouco do seu tempo para a construção desse trabalho. Um deles foi o Reinaldo que me ajudou na etapa de integração com o Tidia-Ae (inclusive em alguns finais de semana). Não posso me esquecer do Ian, uma pessoa bastante importante afinal, eu precisava de alguém para soldar e retirar as soldas dos meus circuitos impressos! Finalmente gostaria de agradecer a Christiane que me manteve sempre informado dos prazos da pós-graduação e me ajudou na reta final desse trabalho.

Gostaria de agradecer a todos no LARC que de alguma forma me ajudaram, a Fundação de Apoio à Universidade de São Paulo (FUSP) pela concessão de bolsa e auxílio financeiro, a todo o pessoal do Tidia-Ae, pelo código fonte e ajuda no ambiente de programação e finalmente a todos na Scopus que de alguma forma ajudaram, deram sugestões ou contribuíram para a conclusão desse projeto.

RESUMO

Atualmente o uso de senhas, método comum para efetuar autenticação em páginas da internet, mostra-se uma alternativa com problemas de segurança devido ao aumento de ataques baseados em *spyware* e *phishing*. O objetivo desses ataques é obter a senha do usuário, isto é, sua identidade digital sem que o usuário perceba o ocorrido. Para conter esse tipo de ataque, instituições financeiras começaram a adotar a autenticação forte, técnica que emprega o uso simultâneo de múltiplos autenticadores. A combinação das vantagens dos diferentes autenticadores resulta em uma atenuação mútua de suas vulnerabilidades e, em conseqüência, um método mais seguro de verificação de identidade. Esse trabalho apresenta o projeto e a implementação de um dispositivo de autenticação, permitindo combinar o uso de senhas e autenticadores baseados em objeto. As principais características do dispositivo são o seu custo reduzido e o uso de algoritmos criptográficos com código aberto. Algoritmos de código aberto possuem a sua segurança averiguada de forma ampla e independente, característica que dá maior confiabilidade ao sistema, permitindo a qualquer pessoa avaliar o código executado pelo dispositivo.

Palavras-Chave: Segurança, Autenticação, Token, *Phishing, Scam.*

ABSTRACT

Currently, password-based authentication is the most widespread identity verification method for web pages access. However it presents security issues due to the growth of attacks based on spywares and phishing. The main purpose of both techniques is the digital identity theft, that is, stealing users' passwords in an unnoticed way. In order to counter this type of attack, many financial institutions have adopted strong authentication, a technique that employs a simultaneous use of different authentication factors. By synergistically combining the advantages of distinct factors, such arrangement results in the mutual mitigation of the vulnerabilities of each one, yielding an architecturally safer identity verification method. This work presents the design and implementation of an authentication device, which combines password-based and object-based authenticators. Its main distinguishing features are the reduced cost and the use of open sourced cryptographic algorithms. Open source algorithms have their security widely and independently verified, a characteristic that helps increase the system's reliability, since third parties may check the source code running on the device.

Keywords: Security, Authentication, Token, Phishing, Scam.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Sigla	**Significado**
AC	Autoridade Certificadora
ANSI	*American National Standards Institute*
CMAC	*Cipher-based MAC*
DNS	*Domain Name System*
DoS	*Denial-of-Service*
EEPROM	*Electrically Erasable Programmable Read Only Memory*
FRA	*False Acceptance Rate*
FRR	*False Rejection Rate*
HMAC	*Hash-based MAC*
HTML	*Hypertext Markup Language*
IDS	*Intrusion Detection System*
IETF	*The Internet Engineering Task Force*
INT	Código de Integridade
LAN	*Local Area Network*
LCD	*Liquid Crystal Display*
LED	*Light Emitting Diode*
MAC	*Message Authentication Code*
MMV	Monitor de máquinas virtuais
OTP	*One Time Password*
PC	*Personal Computer*
PCB	*Printed Circuit Board*
PDA	*Personal Digital Assistant*
PIN	*Personal Identification Number*
POR	*Power on-reset*
RADIUS	*Remote Authentication Dial In User Service*
RAM	*Random Access Memory*

Sigla	**Significado**
RG	Registro Geral (Documento de Identificação utilizado no Brasil)
ROM	*Read Only Memory*
SMS	*Short Message Service*
SO	Sistema Operacional
VPN	*Virtual Private Network*

1. INTRODUÇÃO

A internet vem se tornando um meio de comunicação cada vez mais importante devido ao aumento do uso dessa ferramenta para realizar determinadas operações, tais como acessar contas bancárias ou efetuar compras de bens. Realizar tais operações na internet oferece benefícios como receber encomendas na porta de casa ou evitar longas filas em bancos. Evidências desse crescimento podem ser obtidas através de trechos de reportagens extraídas de publicações de grande circulação:

> A 13ª edição do *Web* Shoppers, estudo do realizado pela e-bit com o apoio da Câmara Brasileira de Comércio Eletrônico - Camara-e.net, apontou crescimento de 43% de 2004 para 2005 no comércio eletrônico brasileiro. O setor teve um faturamento de R$ 2,5 bilhões (...) O estudo também prevê que o setor terá um crescimento constante nos próximos anos(...) (BARBOSA, 2006)

> (...) Segundo o banco, o número total de acessos à conta bancária pela *web* também atingiu uma marca histórica: 12,5 milhões por mês. Isso indica que os clientes concluem, em média, 4,4 operações a cada acesso à conta bancária pelo site do BB. Os novos números apontam que mais de 17% do atendimento do Banco do Brasil a pessoas físicas já é feito pela internet. Quando as operações realizadas por pessoas jurídicas também são consideradas, o índice se eleva para mais de 30%. O BB diz que, nos últimos seis anos, o uso da internet para transações bancárias pelo (*sic*) seus clientes pessoa física aumentou 1.128% (GREGÓRIO, 2006)

Como se pode notar a partir dos números apresentados acima, a internet está sendo utilizada para movimentar grandes volumes de dinheiro e essa é uma tendência que vai aumentar com o passar do tempo.

O uso crescente da internet veio acompanhado de potenciais problemas causados por pessoas em busca de vulnerabilidades na segurança dos sistemas *on-line* para obter ganhos de forma ilícita. A segurança de um sistema de informação opera como uma corrente contendo múltiplos elos. Burlar esse esquema de segurança significa romper pelo menos um elo dessa corrente, em geral, o elo mais fraco.

Hoje já está claro que um sistema *on-line* não é formado apenas pelos servidores de uma instituição. A Figura 1 mostra a estrutura de um sistema atual e nela percebe-se que o sistema é formado por 3 partes (elos) principais, os servidores da instituição, onde as operações são efetivamente executadas, a internet que serve como via de acesso e finalmente o computador através do qual o cliente interage com o sistema.

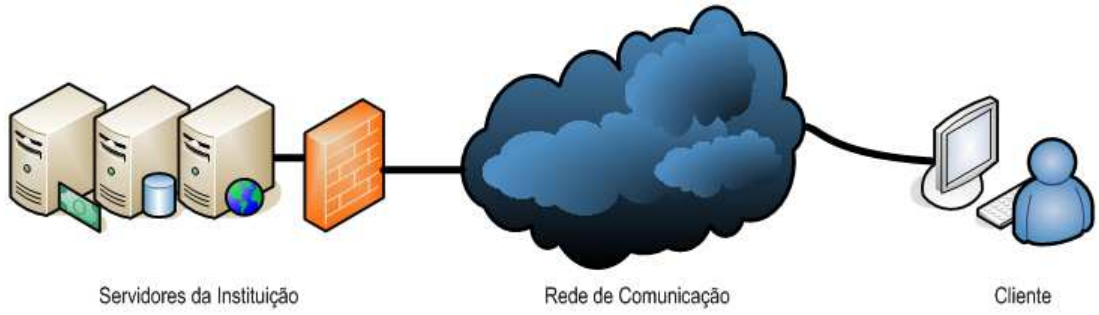


Figura 1 - Arquitetura cliente/servidor

Dado que a fonte de informações valiosas está concentrada nos servidores, uma primeira abordagem para atacar essa estrutura é através da exploração de vulnerabilidades diretamente nos servidores de empresas. Estudo realizado entre 2003 e 2004 nos sistemas de internet *banking* da Noruega mostra como esses sistemas poderiam ser atacados (HOLE; MOEN;TJØSTHEIM, 2006). No entanto, ataques a servidores passam a ser dificultados com a adoção de ferramentas como *firewalls* e IDS[1], assim como através da aplicação de rotinas freqüentes de atualização de segurança aos servidores.

O gráfico[2] a seguir (CERT.BR, 2007) apresenta os três principais tipos de ataques ocorridos no Brasil nos últimos anos e ajuda a ilustrar como o foco dos ataques deixou de ser os servidores e passou a ser as máquinas dos clientes (representadas por fraudes e *worms*[3]).

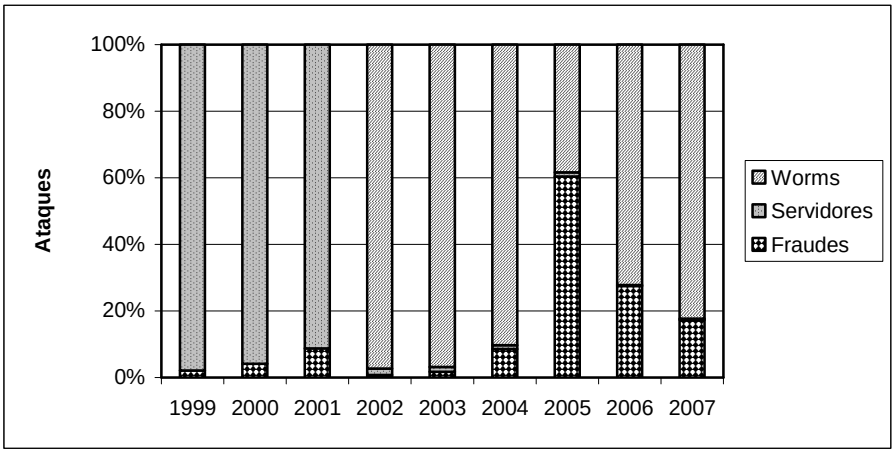


Figura 2 - Ataques reportados no Brasil entre 1999 e 2007

[1] *Intrusion Detection System* – Sistema que analisa constantemente a rede em busca de sinais que possam indicar uma invasão no sistema.
[2] Dados atualizados até o primeiro trimestre de 2007. Até 2002 *worms* não apareciam nos dados.
[3] Programa malicioso capaz de se propagar automaticamente de computador para computador

Muitos usuários têm uma "falsa" sensação de segurança ao acessarem a internet de dentro de suas casas. Isso acontece porque fisicamente eles se sentem protegidos e acreditam que a possibilidade de "estar a qualquer hora em qualquer lugar" de forma "anônima" os protege de qualquer problema.

Utilizar a internet é quase como andar na rua, onde se toma cuidado para atravessar a rua ou visitar determinados lugares. A diferença é que a pessoa não é afetada fisicamente, porém pode ser lesada financeiramente ou em algum outro ativo virtual. Dependendo das páginas visitadas ou dos programas copiados e executados em seu computador, *spywares* podem ser instalados e dados pessoais passam a ser enviados para fraudadores em outras partes do país ou do mundo sem o conhecimento do usuário.

Ainda no lado do cliente existe um ataque onde emprega-se a *Engenharia Social*, ou seja, o trabalho do atacante consiste em persuadir o próprio usuário a entregar suas informações de "livre e espontânea vontade" (SCHNEIER, 2004). Para isso páginas falsas, conhecidas como *phishing*, solicitam informações como senhas e o usuário pode passá-los acreditando estar os fornecendo às páginas originais.

O sucesso de ataques ao cliente, destacados pelos números da Figura 2, ilustra a necessidade de um processo de aculturamento às tecnologias da internet para tornar a sociedade mais preparada para o convívio com este tipo de tecnologias.

Aproveitar-se da ingenuidade, da falta de atenção ou da boa-fé dos usuários da rede é a base para a construção dos roubos de identidade digital. Como pôde ser visto no gráfico anterior, esses ataques tornaram-se mais freqüentes nos últimos 3 anos e isso fez com que instituições financeiras investissem em novas formas de autenticação num esforço para melhorar a segurança de seus sistemas:

> O Bradesco comprou 470 000 *token*s - espécie de chaveiro eletrônico que carrega a senha do correntista - para clientes pessoa jurídica e mais de 14 milhões de cartões de senhas para pessoa física. (...) Já no ABN Amro Real (...) optou pela tripla autenticação (*sic*). São usadas duas senhas - a tradicional e outra presente em uma tabela que é exibida aleatoriamente a cada novo acesso - e também um *software* que bloqueia a ação de programas espiões. Para pessoas jurídicas, o ABN recorreu a um *token* que apresenta um código novo a cada 2 minutos e é sincronizado com os servidores do banco. No último ano, os gastos com segurança do ABN cresceram cerca de 18%. (TERZIAN, 2006)

Segundo pesquisa de 2005, realizada pelo FBI (PATEL-PREDD, 2006), estima-se que crimes na internet representaram uma perda de US$ 67 bilhões a empresas nos EUA. Essa mesma pesquisa aponta que 79% das empresas tiveram problemas com "*spywares*".

Tais perdas não se significam apenas prejuízos financeiros diretos. Elas também incidem negativamente na imagem da instituição e na credibilidade de ferramentas *on-line,* de um modo geral. É preciso evitar que esses problemas continuem ocorrendo visto que o potencial de crescimento da internet é ainda muito grande. Só no Brasil, 79% dos brasileiros (CETIC.BR, 2007) ainda não tiveram oportunidade de acessar a internet.

1.1. Objetivo e contribuição

Qualquer transação financeira na internet exige que o cliente realize algum tipo de autenticação antes de finalizá-la. É nesse ponto em que os ataques de engenharia social procuram agir e obter a identidade digital de um usuário. Tendo como base esses ataques, esse trabalho faz uma revisão sobre autenticação e seus mecanismos, destacando que a autenticação pode ser melhorada com a combinação de múltiplos fatores como uma solução para reduzir esses ataques.

Para mostrar a viabilidade dessa combinação, esse trabalho ainda propõe o projeto e implementa um dispositivo de segurança com baixo custo para autenticação de entidades ou objetos.

Esse projeto tem a preocupação de criar um dispositivo seguro com custos reduzidos de produção. Esses requisitos se refletem no projeto do *hardware* e na escolha e programação do *software* para o sistema. Como *hardware* são adotados basicamente "componentes de prateleira", ou seja, componentes de emprego comum e, em vista disso, com custo reduzido por unidade.

A implementação do dispositivo utiliza algoritmos de código aberto que trazem consigo importantes vantagens. Além de reduzir custos ao torná-lo livre do pagamento de *royalties*, mostra a segurança do algoritmo executado por ser bem conhecido e amplamente testado.

Ponto essencial a ser ressaltado é a transparência que isso traz ao sistema, pois se sabe que o dispositivo faz tudo e somente aquilo que foi especificado.

1.2. Outras soluções

A mudança na forma de autenticação é uma solução que admite que o ambiente onde o usuário está trabalhando já esteja sendo atacado por *phishing/scam*. Abaixo são listadas algumas medidas com o intuito de evitar esses ataques, cada uma com seus prós e contras. No entanto, elas estão fora do escopo desse trabalho:

- Educar o usuário a navegar a internet de forma mais segura
- Empregar extensões nos navegadores dos usuários para detectar e alertar o usuário sobre *phishing* (KIRDA;KRUEGEL, 2005)
- Adotar a autenticação de e-mails (LAWTON, 2005) para ignorar mensagens não autenticadas
- Garantir que o teclado do usuário não será monitorado por aplicações maliciosas (LINDA DAILEY, 2005)
- Identificar previamente páginas de *phishing* (LIU et al., 2005)

1.3. Organização do trabalho

O capítulo 2 é utilizado para definir autenticação, introduzir os mecanismos de autenticação e analisar o autenticador baseado em conhecimento. Esse tipo de autenticador é importante por ser o mais aplicado nas páginas eletrônicas atualmente. O capítulo termina explicando as vantagens do uso da autenticação forte.

No capítulo 3 explica-se os autenticadores baseado em objeto, principalmente aqueles baseados em algoritmos desafio-resposta. São examinadas as formas mais comuns de implementação e apresentados alguns modelos comerciais de autenticação forte.

Com o embasamento construído nos capítulos anteriores, os capítulos 4 e 5 apresentam a proposta deste trabalho. O capítulo 4 apresenta o projeto de um *token* baseado em *hardware*, onde se analisa o *hardware* e o *software* empregados em sua implementação para atender os objetivos deste trabalho.

O capítulo 5 apresenta a verificação de uso do dispositivo desenvolvido em diversos aspectos. Também é apresentado um exemplo de página modificada para suportar o dispositivo aqui desenvolvido.

O capítulo 6 encerra o trabalho apresentado conclusões, suas contribuições e sugestões de projetos futuros.

Os apêndices são utilizados pra complementar o projeto apresentado ao longo do capítulo 4.

2. AUTENTICAÇÃO: IDENTIDADE E INTEGRIDADE

A autenticação é um serviço de segurança que verifica se algo ou alguém é o que afirma ser. O ato de autenticar verifica a integridade, a origem ou a identidade de um objeto ou uma entidade.

A integridade tem dois papéis importantes. É constatada através da verificação uma mensagem não foi adulterada, assim como por meio da confirmação de que a mensagem foi realmente gerada pela fonte da qual acredita-se que tenha sido. A primeira função é chamada de integridade de dados e a segunda é conhecida como autenticação de mensagem ou autenticação de origem dos dados.

A identificação é o processo pelo qual um requerente afirma possuir uma determinada identidade perante uma entidade. A autenticação de entidades se refere ao processo pelo qual o requerente prova sua identidade através de uma verificação feita pela entidade (OPPLIGER, 2005). A identificação está sempre associada a um protocolo de autenticação.

O serviço de autenticação é básico e importante, pois é freqüentemente pré-requisito para prover os demais serviços de segurança como controle de recursos, confidencialidade, etc.

Segundo Smith (SMITH, 2002), qualquer sistema de autenticação, computacional ou não, possui alguns elementos sempre presentes:

- Requerentes – grupo de pessoas autorizadas a utilizar o sistema
- Identidade – representação de uma característica única para um dado requerente que permite a sua diferenciação dos demais membros do grupo
- Proprietário do sistema – responsável por controlar o sistema
- Mecanismo de autenticação – meio através do qual a pessoa é associada à sua identidade no sistema.
- Mecanismo de controle de acesso – regras que garantem o acesso aos recursos ao requerente através de sua identidade.

Esses conceitos estão ilustrados na tabela a seguir, adaptada de Smith (SMITH, 2002). Associam-se os elementos à sua presença em alguns sistemas típicos:

Sistema / Elemento	Clube	Caixa automático	Página Web
Requerente	Membro do Clube	Cliente do banco	Usuário autorizado
Identidade	Carteira de membro	Cartão e senha	Usuário/Senha
Proprietário	Donos do clube	Banco	Empresa proprietária da página
Mecanismo de autenticação	Carteira pertence ao requerente?	Cartão é válido e senha é correta?	Usuário e senha estão corretos?
Mecanismo de controle de acesso	Guarda na portaria	*Software* de controle do banco	Caixa de *login* e lista de controle de acesso

Tabela 1 - Elementos de um sistema de autenticação

2.1. Objetivos da autenticação

O principal objetivo da autenticação é verificar a identidade ou a integridade de uma entidade ou objeto. Uma de suas aplicações importantes é prover autorização, isto é, dar ao usuário de um sistema acesso a todos os recursos ou funcionalidades aos quais ele tem direito.

Outras aplicações incluem:

- Evitar a personificação. Personificação é o emprego de uma identidade digital que não pertence ao requerente. Uma vez autenticado como se fosse o requerente verdadeiro, um usuário ilegítimo tem acesso aos recursos do usuário pelo qual ele está se passando.
- Tarifação. A cobrança de recursos somente é possível se o usuário que está sendo tarifado é conhecido. Cobra-se então por recursos como consumo de CPU e banda, limite de tráfego, alocação de espaço de armazenamento, disponibilidade do sistema, etc.
- Garantia de procedência. Nem sempre é necessário ou possível estabelecer comunicação com um usuário de forma confidencial. Entretanto, ainda assim é possível garantir a origem de uma mensagem.

- Auditoria. A auditoria permite através de registros (*logs*) rastrear atividades realizadas no sistema e por quem foi feito. Com isso, usuários legítimos que tentaram realizar operações ilegais no sistema podem ser identificados e advertidos de forma adequada.
- Reconhecimento de padrões. Ao autenticar um usuário, pode-se passar a conhecer os seus hábitos e antecipar suas necessidades (personalização). Além de personalizar, pode-se identificar um comportamento anormal do usuário e alertar para um possível roubo de identidade.

2.2. Identidade

Identidade é a representação única de um requerente para um dado sistema. A autenticação associa o requerente à sua representação de identidade interna ao sistema. Cada sistema tem a sua própria forma de expressar essa representação, e todas as decisões de acesso e alocação de recursos assumem que essa associação está correta (BISHOP, 2003).

A *representação da identidade* é feita através de um identificador pessoal, facilmente reconhecível e único para um sistema. Esse *identificador pessoal* consiste de um identificador ou um conjunto de identificadores. Cada um desses *identificadores*, também chamados de *autenticadores* ou *fatores de autenticação*, é um *atributo com a propriedade de diferenciar entidades ou objetos* podendo ser secreto ou simplesmente de difícil reprodução ou alteração.

Uma pessoa pode possuir diferentes identidades digitais em diferentes sistemas, porém única para um dado sistema. Em um banco na internet a identidade digital de um correntista pode ser dada, por exemplo, pelo par Agência/Conta. Em um sistema de *webmail* é dada pelo par e-mail/senha.

2.3. Integridade e autenticação de mensagens

A integridade de uma mensagem deve ser vista sob dois aspectos diferentes. Primeiro, existe o que é conhecido como integridade de dados (*data integrity*) que

permite garantir que a mensagem não foi modificada acidentalmente ou intencionalmente durante a sua transmissão. O outro aspecto se refere à comprovação de que ela tenha realmente sido enviada por quem afirma ter sido, ou seja, a autenticação de origem dos dados (*data-origin authentication*) ou autenticação de mensagem (*message authentication*) (MAO, 2004). Seja qual for o aspecto, a integridade é verificada através de um código de integridade associado à mensagem.

A integridade de dados não envolve necessariamente transmissão de mensagens e pode ser conferida independentemente de quando um código de integridade foi gerado.

Já a autenticação de mensagens somente faz sentido quando existe a transmissão de dados. Dependendo do uso, o instante em que a mensagem foi gerada também se torna importante.

Em protocolos de autenticação, mensagens antigas, mesmo que possuam um código que validem a origem, não podem ser autenticadas já que podem ser uma tentativa de repetição por parte de um atacante. Esse tipo de mensagem normalmente possui uma chave de sessão criada e descartada quando a sessão é encerrada.

Documentos guardados por longos períodos podem ter as respectivas origens averiguadas, independente da época em que foram geradas.

2.4. Mecanismos de autenticação

Os proprietários de um sistema apóiam-se em mecanismos de autenticação que são as ferramentas que irão efetivamente validar a integridade, origem ou identidade de uma entidade ou objeto.

A verificação de integridade de dados é normalmente realizada através de algoritmos de *hash*. *Hash,* ou resumo criptográfico, é uma função determinística com a finalidade de mapear uma cadeia de bits de qualquer tamanho em uma quantidade fixa de bits. O mapeamento é tal que não pode ser invertido, ou seja, a partir do

hash é inviável[4] encontrar a mensagem que o gerou. Também é inviável encontrar duas mensagens com o mesmo *hash*.

A garantia de origem e integridade é feita através de algoritmos MAC ou Códigos de Autenticação (*Message Authentication Code*). Assim como o *hash*, ele mapeia uma cadeia arbitrária de bits em uma quantidade fixa, porém impõe duas entradas: a mensagem em si juntamente com uma chave criptográfica (por isso também é chamado de *keyed-hash*). O MAC pode ser construído a partir de uma função de *hash* ou a partir de algoritmos de criptografia simétrica.

Assinaturas digitais garantem integridade, origem e irretratabilidade. A assinatura digital aplica criptografia assimétrica em conjunto com funções de *hash*. A partir de uma mensagem, gera-se o seu *hash* que, por sua vez, é assinado pela chave privada da entidade que o gerou. A mensagem é então enviada juntamente com a sua assinatura. A verificação é feita aplicando-se a chave pública da entidade na assinatura. O resultado é então comparado com o *hash* da mensagem e se ambos os resultados forem idênticos, atesta-se a origem e a integridade da mensagem. Como não existe segredo compartilhado, atesta-se também a irretratabilidade.

Mecanismos para verificar a identidade adotam senhas, certificados, objetos ou biometria para esse fim. O funcionamento desses mecanismos é detalhado entre as seções 2.7. e 2.9. Os principais ataques a esses mecanismos são apresentados na seção 2.10. A seção 2.11. apresenta as possibilidades de combinação dos fatores de autenticação.

2.5. Mecanismos de controle de acesso

Depois de corretamente autenticado, um requerente tem acesso a dados e recursos através de mecanismos de controle de acesso. Esses mecanismos devem ser implementados de forma que o usuário tenha acesso apenas ao que ele tenha direito.

[4] Neste texto, a palavra "inviável" sempre se refere a algo que não é computacionalmente factível com a tecnologia atual devido ao elevado tempo de processamento que seria necessário para realizar a tarefa.

Apesar dessa premissa, ao criar um mecanismo de controle de acesso, às vezes é inviável separar os recursos de forma suficientemente granular para permitir esse tipo de acesso. Por exemplo, se um usuário tem a chave de uma sala, o mecanismo de controle de acesso (fechadura) dá acesso a todos os itens da sala e não apenas aqueles em sua mesa de trabalho.

Atribuir permissões individuais a muitos usuários também não é fácil e por isso estabeleceu-se o conceito de papéis ou grupos de usuários, segundo os quais permissões são associadas aos mesmos. Eventualmente, pode-se encontrar casos de usuário com permissões em excesso em um grupo e permissões faltantes em outro.

Mecanismos de acesso são freqüentemente implementados com o uso de ACL ou Listas de Controle de Acesso (*Access Control Lists*). Essas listas contêm instruções permitindo acesso apenas àquilo que é apresentado de forma explícita e bloqueando as demais. Outra possibilidade é negar acesso a tudo que é apresentado de forma explícita e permitindo as demais. As instruções aparecem de forma individual para cada entidade ou para grupos de entidades. As ACL estão presentes na implementação de sistemas de arquivos, sistemas operacionais de múltiplos usuários, *firewalls* etc.

2.6. Etapas da autenticação

Com o surgimento de sistemas digitais e a sua interligação através de redes, a autenticação passou a ser feita através de sistemas baseados na arquitetura cliente/servidor.

Nessa arquitetura empresas estendem seu sistema até o computador do usuário de modo que este se torna capaz de se autenticar e realizar operações remotamente. Esse efeito é obtido enviando o "*software* cliente" e desta forma incorporando a máquina do usuário final ao sistema, disponibilizando assim, todas as suas facilidades.

Em virtude dessa arquitetura, o processo de autenticação foi separado em duas etapas. Em um primeiro passo o sistema deve autenticar o "*software* cliente", realizando assim o que pode ser chamado de autenticação de máquina.

A segunda etapa é chamada de autenticação de usuário. O usuário entra com suas credenciais no "*software* cliente" e este por sua vez as envia ao servidor onde elas são verificadas e o devido nível de acesso é concedido. A Figura 3 resume o processo descrito anteriormente.

Figura 3 - Etapas da Autenticação

2.7. Autenticação de máquina

Como citado anteriormente, a autenticação entre máquinas é a primeira etapa do processo de autenticação. No caso de sistemas baseados na internet, essa etapa é tipicamente realizada através da troca de certificados digitais. Com o uso de certificados é estabelecido um canal seguro através do qual o usuário envia suas credenciais ao sistema. Além de autenticar, o canal pode ainda ser utilizado para garantir a confidencialidade da comunicação.

Um servidor ligado à internet é identificado através de seu certificado digital. Uma vez que um certificado é validado, o dono desse certificado é autenticado. A geração dos certificados e o estabelecimento de canal seguro são atualmente realizados através de protocolos já bem conhecidos e seguros para os padrões computacionais atuais (MAO, 2004).

2.7.1. Certificado Digital

O certificado digital é um documento em um formato padronizado (X.509) (HOUSLEY et al., 1999), gerado a partir das técnicas de assinatura digital, para

distribuir e comprovar a autenticidade da chave pública de uma entidade, equipamento ou indivíduo.

Os certificados digitais emitidos por Autoridades Certificadoras[5] (AC) atestam que a identidade da entidade que vai receber o certificado foi adequadamente verificada. Para isso, a AC se vale de documentos que comprovem a identidade real dessa entidade e procedimentos para validar que a entidade realmente possui a chave privada correspondente à chave pública que será apresentada no certificado.

Confiar em uma AC implica em concordar na forma como é feita a validação da identidade real do sistema. Em outras palavras, pode-se dizer que quem confia na AC acredita que ela realmente validou a identidade de um sistema e que documentos e chaves apresentados são suficientes para fazer essa verificação.

Os principais campos de um certificado digital são: o dono, AC que o assina, validade e a chave pública do dono. Uma vez preenchidos, estes dados passam por um *hash* que por sua vez é criptografado com a chave privada da entidade certificadora.

O resultado é a assinatura do certificado. A Figura 4 mostra esses passos:

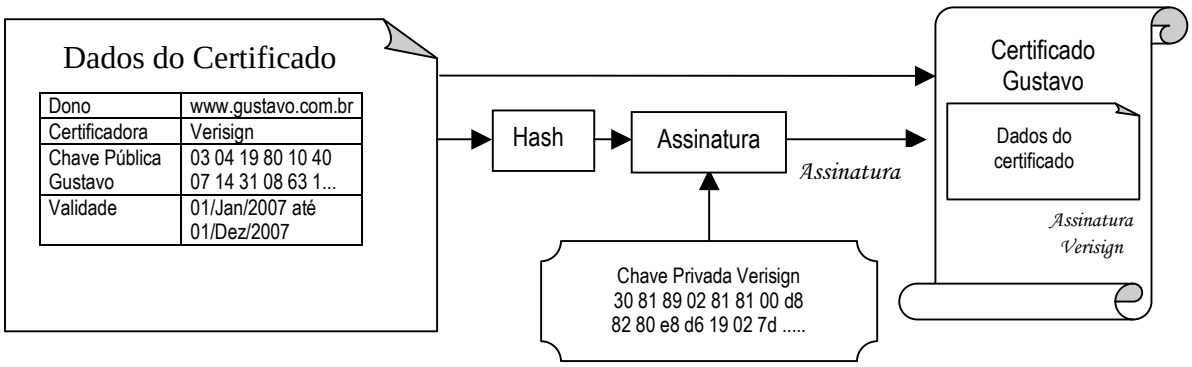


Figura 4 - Criação de um certificado

2.7.2. Uso do certificado

Quando estabelece uma comunicação com um sistema, o usuário recebe uma cópia do certificado e deve prosseguir se, e somente se, considera o certificado válido. Em

[5] Autoridade Certificadora é uma entidade que emite certificados e é considerada confiável pelos usuários do certificado.

caso positivo, é estabelecido um túnel seguro através do qual o usuário troca mensagens com essa empresa.

Os seguintes pontos são verificados pelo usuário ao validar um certificado[6]:

- Certificado realmente pertence a quem o está apresentando
- Validade do certificado (temporal e revogação)
- Integridade do certificado (assinatura)
- Confiança na AC

Quando todos os itens são válidos, o certificado é aceito como autêntico e tem-se a certeza de que realmente a comunicação está sendo realizada com o dono do certificado.

2.7.3. SSL/TLS

O SSL é um protocolo desenvolvido originalmente para o navegador Netscape e em 1996 foi publicada a sua terceira versão. Esse se tornou o protocolo mais utilizado na internet para realizar autenticação entre computadores e por isso é brevemente descrito nessa seção. O SSL se tornou um padrão *de facto* entre os desenvolvedores de navegadores para a internet e evoluiu para o TLS tornando-se o protocolo padrão desenvolvido pelo IETF.

O funcionamento do TLS pode ser descrito em uma etapa de autenticação e outra de comunicação. A fase de autenticação consiste basicamente de uma troca de certificados entre as máquinas e o estabelecimento de uma chave de sessão. A fase de comunicação utiliza a chave de sessão como entrada para algoritmos de criptografia, provendo confidencialidade e autenticidade na troca de mensagens.

A autenticação é feita pelo *TLS Handshake Protocol* e, resumidamente, uma negociação típica constitui-se através dos seguintes passos:

[6] Normalmente essa tarefa cabe ao navegador do usuário. Se detectado algum problema, o mesmo é reportado e cabe ao usuário decidir se prossegue com a comunicação.

- Cliente e Servidor trocam mensagens (*Client Hello, Server Hello*) para definir os protocolos (algoritmos simétricos, assimétricos, MAC, compressão e versão do SSL/TLS) que serão utilizados na comunicação.
- Acertados os protocolos, o servidor envia o seu certificado e o cliente gera um segredo (*master_secret*) cifrado com a chave do certificado enviado. Nessa etapa todos os dados do certificado são validados pelo cliente.
- O protocolo termina com o cliente enviando uma mensagem *ClientFinished* ao servidor autenticada com um MAC gerado a partir do segredo enviado anteriormente. O servidor envia a mensagem de *ServerFinished* também autenticada com o MAC.

Na descrição acima, admitiu-se que o cliente não utiliza um certificado para se autenticar, situação muito freqüente no uso cotidiano[7] da internet. Se o cliente também utilizasse um certificado, ele teria sido enviado após o certificado do servidor e uma mensagem a mais teria sido assinada pelo cliente para o servidor verificar a sua identidade. Após a autenticação, a etapa da comunicação é gerenciada pelo *TLS Record Protocol*. Uma descrição mais profunda do TLS pode ser encontrada em (DIERKS;ALLEN, 1999) e (MAO, 2004)[8].

Além do TLS, citam-se como exemplos de protocolos com finalidade semelhante SSH (YLONEN;LONVICK, 2006), IPSec (STALLINGS, 2003) e S/MIME (STALLINGS, 2003).

2.8. Autenticação humano/máquina

A seção anterior descreveu a criação de um caminho seguro entre duas máquinas que se comunicam. Uma vez estabelecido esse caminho, na próxima etapa o usuário envia suas credenciais para o sistema que ele vai utilizar.

Na autenticação humano/máquina, o usuário recebe uma identidade digital, através da qual, na maioria dos sistemas, associa-se ao usuário um nome simbólico chamado de "Nome de Usuário" e uma senha. Normalmente, esses dados são

[7] Pensa-se aqui em acesso a páginas de compras, de e-mail, acesso a bancos, etc.
[8] Não é objetivo desse texto detalhar o *TLS Record Protocol* ou conhecer mais a fundo o TLS.

fornecidos no computador apenas quando uma comunicação segura é estabelecida, isto é, em cima de algum protocolo como o TLS.

O problema desse método é que muitos ataques não são feitos através da escuta do caminho seguro, mas através da escuta do caminho entre o teclado do usuário até o instante em que a senha aparece (mascarada) na tela ou então através da simples "adivinhação" da senha do usuário.

Como será mostrado mais adiante, os usuários não escolhem senhas com entropia suficiente para garantir a segurança do algoritmo adotado. Adicionado a esse problema, também será visto que, nos últimos anos, os usuários têm sido alvos de novas técnicas de roubo de senha. Tais técnicas empregam o uso de páginas falsas, com o fim de obter dados de usuários que acreditam que tais páginas sejam autênticas, ou ainda é persuadido a instalar programas aparentemente úteis que, na verdade, passam a monitorar dados que os usuários digitam no computador.

Para tornar a autenticação mais segura, outros tipos de autenticadores vêm sendo cada vez mais pesquisados e adotados com os objetivos de facilitar, para o usuário, a identificação em um sistema e aumentar a entropia das senhas utilizadas.

2.9. Tipos de autenticadores

Quando é preciso utilizar um sistema, a identidade digital de um usuário é validada através de um autenticador apresentado ao sistema. Esses autenticadores são classificados segundo o tipo de identificador pessoal adotado.

O autenticador mais comum é baseado em conhecimento ("algo que se sabe"). Outros autenticadores utilizados são baseados em objeto ("algo que se tem") e em biometria ("algo que se é"). Eventualmente, referências citam autenticadores baseados em localização, isto é, algum lugar que apenas aquele usuário tem autorização para estar, ou não autenticar usuários de uma determinada localização; entretanto, localização pode ser reduzida a um dos três autenticadores citados anteriormente. Quando mais de um autenticador é necessário em um mesmo sistema eles são referenciados como "fatores de autenticação" (*authentication factors*).

Nos sistemas em que a autenticação é baseada em conhecimento, a identidade de um requerente é verificada através de um segredo compartilhado entre ele e o sistema. O exemplo mais simples desse tipo de autenticação é a senha. Nessa categoria também pode ser incluída a identificação positiva, isto é, o uso de informações que, apesar de públicas, requerem um conhecimento um pouco mais profundo da vida da pessoa, como por exemplo, o número do RG, ou o nome do animal de estimação.

Em sistemas nos quais a autenticação é baseada em um objeto, a pessoa precisa mostrar a posse de um determinado objeto que lhe foi dado pelo sistema. Como exemplo, pode-se citar o controle remoto de uma garagem ou a carteira de identificação de um clube. Uma vulnerabilidade desse fator é que se o mesmo for perdido, qualquer um que encontrá-lo poderá utilizá-lo como se fosse o requerente original. Por isso, esse método é freqüentemente acompanhado de outros fatores de autenticação (o cartão do banco tem a senha, a carteira do clube tem a foto). O exemplo do controle remoto não possui outro fator, porém tão logo se perceba que o objeto foi comprometido, uma providência pode ser tomada imediatamente. No caso do comprometimento de uma senha, o seu dono normalmente percebe o evento apenas quando ela é empregada indevidamente, provavelmente na ocorrência de uma autenticação fraudulenta.

Com os sistemas onde a autenticação é baseada em biometria, a autenticação ocorre com o uso de identificadores baseados em características físicas das pessoas. Nessa categoria, a escolha do autenticador é baseada na dificuldade de sua reprodução visto que dados biométricos não são secretos.

2.9.1. Autenticação baseada em conhecimento

Autenticação baseada em conhecimento é a forma mais conhecida e implementada em sistemas digitais. Nesse método, o usuário escolhe ou recebe um nome de usuário e uma senha para se identificar no sistema. Uma senha típica varia entre 4 e 10 caracteres dentro de um universo que pode ser numérico ou alfanumérico[9],

[9] Considera-se aqui 62 caracteres diferentes entre 0-9, a-z e A-Z

equivalente a uma senha de aproximadamente 60 bits, caso a senha possua 10 caracteres.

Apesar desse grande universo, a escolha de senhas não costuma ser aleatória. As pessoas precisam utilizar senhas para vários sistemas diferentes e por isso muitas vezes escolhem senhas fáceis de serem lembradas posteriormente tais como datas especiais, nomes de amigos ou familiares, entre outros.

Na melhor das hipóteses, são escolhidas palavras comuns do dia-a-dia. Admitindo-se um universo de 500.000 palavras diferentes (veja que esse é um número razoável, já que o dicionário Aurélio (FERREIRA, 2004) possui pouco mais de 435.000 verbetes diferentes), tem-se uma senha de aproximadamente 19 bits[10]. Além do problema das senhas fracas, as mesmas raramente são trocadas e a reutilização de senhas em sistemas diferentes é bastante comum.

Como se pode ver, as senhas criadas pelos usuários tendem a ser simples e por isso diz-se que elas possuem baixa entropia[11], isto é, não apresentam aleatoriedade suficiente para se garantir a segurança do recurso protegido por elas. Ataques baseados em dicionários podem facilmente quebrar essas senhas.

Aumentando a segurança das senhas

Para tentar aumentar a segurança nas senhas costuma-se "forçar" o aumento da entropia na hora da sua escolha. Pode-se, por exemplo, solicitar que a senha possua um tamanho mínimo, caracteres numéricos ou especiais, utilizar letras maiúsculas e minúsculas, verificar se a senha não possui uma seqüência de fácil adivinhação (ex: "1234", "asdfg") e nenhuma relação com os dados cadastrais do usuário (ex: gustavo01). Outros métodos pedem para o usuário digitar frases ao invés de uma palavra, o uso de mais de uma senha para acessar o recurso, ou forçar a mudança periódica da senha.

---

[10] Para se ter idéia da diferença entre 19 e 60 bits, se um dispositivo for capaz de testar mil senhas por segundo, com força bruta a senha de 19 bits pode ser encontrada em menos de dez minutos. Um programa capaz de testar um milhão de senhas por segundo levaria mais de 26 mil anos para quebrar o sistema de 60 bits.

[11] A entropia de uma variável aleatória X é a medida da quantidade de informação provida pela observação de X. De forma equivalente, é a incerteza sobre o resultado antes da observação de X (MENEZES; OORSCHOT;VANSTONE, 2001).

A página *Diceware* (REINHOLD, 2007) possui um interessante sistema para ajudar um usuário a criar uma frase secreta (*passphrase*) "memorizável" porém mais segura que uma senha comum. Esse sistema associa palavras a números obtidos a partir de um dado. Joga-se um dado 5 vezes para obter uma palavra. Uma senha pode ter quantas palavras se desejar, desta forma, para uma frase de 5 palavras, joga-se o dado 25 vezes. As palavras vêm de uma lista criada para cada idioma e cada palavra tem no máximo 6 caracteres. Esse sistema associa aleatoriedade a palavras simples.

Gerenciadores de senhas como o *KeePass* (KEEPASS, 2007) permitem gerar senhas aleatórias para cada sistema que se deseja utilizar. As senhas são protegidas em um arquivo criptografado com senha mestre. Em uma combinação de dois sistemas, a senha para páginas da internet pode ser criada pelo gerenciador de senhas e a chave mestra pode ser gerada pelo método *Diceware*.

MacKenzie (MACKENZIE, 2006) mostra que o uso de senhas ainda é viável se forem tomados os cuidados adequados[12] e também mostra o conceito de *domain hashing*, para o qual a senha é combinada com o domínio em um *hash* e só então é enviada pela rede. Dessa forma o sistema não irá conhecer a senha do usuário e uma mesma senha, quando utilizada em sistemas diferentes, possuirá um *hash* diferente, visto que é combinada com o nome do domínio.

Apesar das melhorias sugeridas, ataques bem conhecidos ainda podem burlar a segurança adicional trazida pelo aumento da entropia das senhas. Senhas ainda podem ser vulneráveis a ataques como *replay*, em que o atacante utiliza a informação de uma execução anterior do protocolo, ou *DoS*, em que o atacante torna o serviço indisponível para usuários legítimos de um sistema, já que muitos sistemas bloqueiam o acesso após um certo número de tentativas sem sucesso. *Keyloggers* e *phishing* ainda podem capturar a senha de um sistema comprometido.

[12] Uso de algoritmos que criam os túneis seguros como o TLS ou o SSH.

2.9.2. Autenticação baseada em objeto

Na autenticação baseada em objeto, a identidade de um requerente é validada quando esse mostra a posse de um objeto previamente fornecido pela entidade que o requerente deseja utilizar.

A prova de posse do objeto pode ser feita de duas formas. Na primeira, o requerente passa o objeto por um leitor especialmente criado para esse fim. Exemplos desse tipo de objeto incluem *smartcards* e cartões magnéticos.

Existem também objetos com processadores e sistemas para realizar entrada e saída de dados próprios. Com esses objetos a prova de posse é feita ao solicitar que o objeto gere algum tipo de sinal e esse sinal então é lido pela entidade. Exemplo desse tipo de objeto incluem *smartcards* com leitores especiais e *token*s capazes de gerar senhas de uso único.

Esse tipo de autenticação é explorada no capítulo seguinte onde serão apresentados dispositivos de autenticação e a suas formas de utilização.

2.9.3. Autenticação baseada em biometria

A biometria é o tipo de autenticador mais antigo que existe. Os identificadores utilizados por esse método são baseados em atributos físicos ou comportamentais do requerente. Por exemplo, o documento de identidade brasileiro apresenta a foto do rosto e a impressão digital como atributos físicos, além da assinatura como um atributo comportamental.

Existem atributos biométricos que são constantes ao longo do tempo como por exemplo impressões digitais, face, íris dos olhos e veias da mão. Outros atributos podem se alterar com o tempo ou mesmo intencionalmente entre uma autenticação e outra, permitindo a implementação de algoritmos do tipo desafio-resposta. Nessa classe de algoritmos se encaixam o reconhecimento de voz e o reconhecimento de escrita.

O uso de atributos biométricos tem a vantagem de que dificilmente são esquecidos ou perdidos e por isso tornam-se mais fáceis de serem utilizados no dia-a-dia.

No entanto, a biometria tem um custo mais elevado quando comparada com os demais autenticadores. A leitura de dados biométricos exige o uso de equipamentos especializados e ainda caros. Atualmente, o tipo de dispositivo mais acessível é o leitor de impressão digital com modelos mais simples podendo ser encontrados em *notebooks* mais avançados.

Como se pode perceber, diferentemente dos outros fatores de autenticação, a biometria não é baseada em um dado secreto. A sua segurança é baseada na dificuldade de duplicação dos atributos utilizados. Para ser utilizado em sistemas digitais, o atributo que será utilizado é medido e posteriormente comparado com o padrão armazenado.

Um fator importante a ser considerado na adoção da biometria é a taxa[13] de falso-negativo (FRR) e falso-positivo (FAR). A biometria é o único autenticador que apresenta esse parâmetro e ele existe devido a imprecisões na leitura dos atributos biométricos. As imprecisões ocorrem devido a erros de leitura, posicionamento incorreto, estado de saúde pessoa, ou tentativas de fraude no sistema. Por exemplo, o reconhecimento de voz pode negar acesso a uma pessoa que se encontra gripada no dia. Normalmente a taxa de falso-negativo pode ser ajustada às custas do aumento de falso-positivos e vice-versa.

Maiores detalhes sobre a biometria podem ser encontrados em diversos artigos e livros como (O'GORMAN, 2003), (SMITH, 2002) entre outros. Palavras chaves nessa área são: forma de armazenamento, custo do reconhecimento, métodos para detectar falsificações, FAR, FRR, balanço entre FAR e FRR e tamanho de uma "chave biométrica".

2.10. Ataques a protocolos de autenticação

Ataques a protocolos de autenticação não têm o objetivo de encontrar falhas nos algoritmos de criptografia no qual eles se baseiam, mas encontrar falhas na forma como os algoritmos são utilizados. O ataque a um protocolo costuma ter um dos seguintes objetivos:

[13] Em inglês: *false rejection rate* (FRR) e *false acceptance rate* (FAR).

- Impedir que a comunicação entre duas entidades seja estabelecida
- Fazer um atacante se passar por uma entidade válida sem que o requerente perceba o ataque
- Obter a chave utilizada pelas entidades

O ataque mais simples consiste em adivinhar a senha do usuário no sistema (ataque dicionário). Isso pode ser feito tentando-se palavras de uso comum, ou então se fixando uma senha e adivinhado o nome dos usuários para evitar que o sistema bloqueie o acesso. Existem sistemas que mostram dicas de senha e isso também pode ser utilizado para inferir uma senha.

Eavesdropping consiste em escutar o canal por onde a comunicação é feita e tentar descobrir algo a partir dessa escuta. Outro ataque, o *Replay*, baseia-se na repetição de sessões anteriores de autenticação para tentar burlar o sistema de autenticação. O *eavesdropping* e o *replay* podem ser considerados complementares.

Man-in-the-middle é um ataque em que o atacante fica entre as entidades que desejam se comunicar replicando mensagens de um lado para o outro. Nesse ataque o atacante estabelece uma sessão com o servidor passando-se pelo cliente e uma sessão com o cliente passando-se pelo servidor. Quando o cliente envia uma mensagem contendo a senha, o atacante a repassa ao servidor, conseguindo autenticar-se como se fosse o cliente.

Outro tipo comum de ataque é o DoS, para o qual o principal objetivo é tornar o sistema indisponível para usuários legítimos. Isso pode ser feito ao realizar mais conexões ao servidor do que ele é capaz de suportar para derrubá-lo. Outra possibilidade é aproveitar o fato de que os sistemas costumam bloquear o acesso se um excessivo número de tentativas inválidas é realizado.

A irretratabilidade é outro problema que deve ser tratado. A princípio, um usuário é responsável por todas as ações que acontecem sob o seu nome, porém um usuário legítimo e mal intencionado pode alegar que teve a sua identidade roubada.

Outros ataques e respostas a ataques podem ser encontrados em (SMITH, 2002), (MAO, 2004) ou (MENEZES; OORSCHOT *et al.*, 2001).

2.10.1. Engenharia Social

Como explicitado na introdução desse trabalho, *phishing*, *scam* e *spywares* tornaram-se ataques comuns na internet. Esses ataques são bem conhecidos na literatura (BERGHEL, 2006; SCHNEIER, 2004) e utilizam-se de Engenharia Social (SMITH, 2002), isto é, convencem, de uma maneira astuta, os usuários de que eles precisam compartilhar a sua identidade com o atacante.

O *Phishing* consiste em criar mensagens conhecidas pelo nome de *Scam*; que induzem usuários a visitar páginas falsas e entregar as informações necessárias para que um atacante possa utilizá-las posteriormente. A palavra surgiu a partir do termo *fishing* (pescaria) e a metáfora se baseia no "lançamento" de uma "isca" e na espera de que alguém seja "pego" pela armadilha.

As páginas falsas são muito semelhantes às originais[14] de instituições e têm a finalidade de roubar os dados pessoais dos clientes da instituição. Ao clicar no link de *scams*, os usuários acreditam estar visitando a página da instituição. Ao utilizar a página, digitando senhas ou preenchendo cadastros, acabam entregando dados para que fraudadores possam utilizá-los posteriormente. Após obter os dados da vítima, a página apresenta algum erro ou simplesmente redireciona o usuário à verdadeira página da instituição.

O artigo (BERGHEL, 2006) apresenta uma interessante análise sobre *phishing*. Segundo Berghel, para um *phishing* ter chances de sucesso, a mensagem criada deve parecer real, mostrando-se como algo interessante através de um conteúdo razoável e que não levante desconfianças. As páginas de *phishing* costumam desaparecer depois de coletar os dados.

Os *scams* podem ainda persuadir o usuário a baixar programas para o seu computador. Esses programas são conhecidos como programas espiões ou em inglês *Spywares* (SUDHINDRA;FIONA FUI-HOON, 2005) (AMES, 2004). Uma vez instalados, monitoraram tudo o que o usuário vê e digita, além de se propagarem

[14] Parte do sucesso desse tipo de ataque acontece pois é fácil copiar páginas inteiras da internet e disponibilizá-las em servidores que podem estar em qualquer parte do mundo.

para outros computadores. Depois de coletados, esses dados são enviados a fraudadores.

A maior parte dos *scams* utiliza e-mails como principal forma para propagação, porém também podem utilizar programas de mensagens instantâneas (LEAVITT, 2005) e páginas de relacionamentos como o Orkut[15]. Quando o atacante utiliza métodos em que se passa por um amigo ou familiar, tenta-se dar maior credibilidade à mensagem para induzir uma vítima a instalar algum tipo de *software* espião em seu computador. A Figura 5 reproduz, a título de exemplo, uma tentativa de ataque através da página de recados do Orkut:

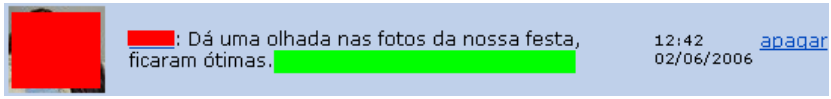


Figura 5 - *Scam* que surgiu no orkut

É importante notar que veio de uma pessoa conhecida (o nome e a foto foram escondidas intencionalmente), tem uma mensagem que chama a atenção e realmente pode ser verdadeira ("*Dá uma olhada nas fotos da nossa festa, ficaram ótimas.*") e aponta para um arquivo na internet (escondido intencionalmente). Apesar de o atalho já ter sido eliminado, até que isso fosse feito, muitos usuários podem ter acessado esse arquivo. Segundo a página "Linha Defensiva" do UOL[16], o arquivo tratava-se de um programa espião que se auto-envia para outros usuários do orkut e obtém senhas de banco.

Menos comum pelo seu maior nível de dificuldade, outro tipo de ataque também utilizado para forçar um usuário a visitar uma página falsa é o *pharming* (KNIGHT, 2005). Essa técnica consiste em atacar os servidores de DNS que quando consultados, enviam ao usuário o endereço IP da página falsa ao invés do endereço da página original. Alterar as respostas em trânsito também é uma possibilidade de exploração de falhas no sistema de DNS (CHANDRAMOULI;ROSE, 2006). Note-se que neste caso nem um usuário protegido com programas como antivírus e *firewall* poderá evitar cair em uma dessas páginas com *phishing*.

---

[15] http://www.orkut.com – Página de relacionamentos criada pela *Google Inc*. que se tornou um sucesso principalmente entre os brasileiros

[16] Praga que rouba dados se espalha pelo Orkut (Página acessada em 24/Jul/2006) http://linhadefensiva.uol.com.br/2006/05/orkut-festa/

2.11. Combinando autenticadores

Como visto, ataques atuais (como o *phishing*) baseiam-se em vulnerabilidades da autenticação baseada em conhecimento ou em vulnerabilidades nos sistemas utilizados para ler a senha do usuário.

Combater esses ataques tem se mostrado complicado pois não dependem de uma ação direta da empresas que utilizam a internet pois eles acontecem nos computadores dos clientes que utilizam os serviços da empresa.

Esses ataques costumam ser bem especializados e específicos para o roubo de senhas. Com isso, a possibilidade de alterar o autenticador adotado torna-se interessante. Evidentemente, a simples troca de forma de autenticação irá mudar apenas a forma dos ataques. Ao invés de trocar o autenticador, passou-se a empregar simultaneamente diferentes autenticadores de forma a amenizar suas vulnerabilidades e combinar as vantagens que eles apresentam.

O uso combinado de diferentes autenticadores ou fatores de autenticação é conhecido como *autenticação forte* ou *strong authentication*. O termo autenticação de múltiplos fatores ou *multi-factor authentication* também é largamente utilizado.

Combinar fatores significa que a identidade passa a ser dada por um fator **E** pelo outro fator. Ela não deve ser utilizada para substituição em caso de perda de um dos fatores. O uso de três fatores combinados é mais raro e ocorre em situações bem específica onde se precisa de um grau muito elevado de segurança (aplicações militares).

A tabela a seguir, adaptada do artigo de O'Gorman (O'GORMAN, 2003), expõe os três tipos de autenticadores com suas características, vantagens, problemas e a proteção adicional em caso de comprometimento quando combinados com o segundo fator de autenticação.

Autenticador	Conhecimento	Objeto	Biometria
Como a Identidade é reconhecida	Segredo	Posse	Unicidade
Como a Segurança é obtida	Deve ser mantida secreta	Deve ficar sempre com o dono	Difícil de ser copiado
Problema se utilizado sozinho	Pode ser compartilhado e enfraquece com o uso	Não pode ser perdido	Difícil de ser substituído
Proteção conferida pelo Conhecimento	-	Protegido pela senha em caso de perda	É preciso a senha para se autenticar
Proteção conferida pelo Objeto	Objeto pode gerar senhas	-	Segurança garantida pelo outro fator
Proteção conferida pela Biometria	É preciso forjar o dado biométrico	Segurança garantida pelo outro fator	-

Tabela 2 - Autenticação de múltiplos fatores

A necessidade de utilizar mais de um autenticador depende da importância das mensagens trocadas, dos serviços prestados ou do valor dos bens protegidos pelo sistema. O emprego de múltiplos autenticadores, quando realizado adequadamente, aumenta a segurança da autenticação, porém aumenta também o custo e muitas vezes a dificuldade de uso do sistema.

2.12. Sumário

Este capítulo apresentou o conceito de autenticação, serviço utilizado para verificar a integridade ou a identidade de um objeto ou uma entidade. Foram apresentados os mecanismos de autenticação bem como o seu emprego na autenticação entre máquinas e na autenticação de usuários.

Discutiu-se sobre autenticação baseada em conhecimento, e que ela é burlada porque os dados estão sendo roubados sem que o usuário tenha consciência do que está acontecendo.

Foi introduzido o funcionamento de autenticadores baseados em objetos e em biometria. O uso de mais de um autenticador cria o conceito de **autenticação forte**, segundo o qual as vulnerabilidades de um autenticador são amenizadas pelas vantagens do segundo autenticador e vice-versa.

3. DISPOSITIVOS DE AUTENTICAÇÃO

Este capítulo descreve o funcionamento de dispositivos para autenticação (*token*s). Mostra-se como são geradas senhas de uso único e justifica a utilização dos dispositivos para assinatura/autenticação de transações. O capítulo ainda apresenta a descrição das formas de implementação e alguns sistemas de autenticação comerciais baseados nesse tipo de dispositivo.

3.1. Definição

Um *token* é um objeto com a função de auxiliar a identificação de um usuário em um sistema. Pode-se citar como exemplo de *token* a carteira de sócio de um clube. Com ela uma pessoa é identificada como membro do local. O cartão bancário, necessário para se utilizar os caixas automáticos, é outro exemplo de *token*. Como se pode perceber, *token*s já estão presentes no dia-a-dia de quase qualquer pessoa.

Quando se fala em *token* na literatura de criptografia, usualmente refere-se ao *token* de segurança. Sua implementação pode ser feita tanto em *software* quanto em *hardware*. A autenticação do usuário perante um sistema que se vale desse método de autenticação é feita através da resolução de um problema em que apenas o *token* de segurança desse usuário é capaz de resolvê-lo. O *token* também pode ser aplicado na autenticação de mensagens. Deste ponto em diante, o *token* de segurança será referenciado simplesmente como "*token*".

O *token* é um autenticador baseado em objeto e raramente é utilizado como único identificador pessoal. Freqüentemente é acompanhado de algum outro método de autenticação como senha ou biometria.

3.2. Senhas Descartáveis - OTP

Como visto no capítulo anterior, senhas são reutilizadas a cada novo acesso ao sistema e por isso podem ser capturadas através de *spywares* ou *phishing*. Para contornar esse problema, pode-se empregar uma senha descartável ou OTP (*One-Time Password*). Como o nome sugere, a cada novo acesso ao sistema, uma nova

senha é gerada tornando um acesso sempre diferente do acesso seguinte. Nunca há a repetição de uma mesma OTP.

Uma OTP é gerada por um *token* a partir de um desafio proposto ao cliente pelo servidor. Dependendo da implementação do sistema, o desafio pode ser gerado a partir da hora atual, a partir de uma seqüência de números ou a partir de um número aleatório fornecido pelo servidor.

De forma simplificada, a resolução do problema consiste em utilizar o desafio juntamente com a chave do usuário como entradas de um algoritmo criptográfico e o valor resultante é a OTP utilizada na sessão. O algoritmo adotado pode se basear em cifras de chave simétrica ou assimétrica.

A idéia original de utilizar OTP aparece em Lamport (LAMPORT, 1981). Esse sistema apresenta uma etapa de registro, na qual são estabelecidos os parâmetros responsáveis por gerar as OTP, e uma etapa de autenticação, em que as OTP são geradas e utilizadas.

Na etapa de registro escolhe-se um número randômico inicial (SEED) e a quantidade de vezes (X) que o usuário fará a autenticação no sistema. Os dois números são armazenados pelo servidor e pelo cliente. O cliente recebe um *token* para armazenar esses valores.

Na etapa seguinte quando o cliente vai se autenticar, ele executa uma função de hash H() por X vezes tendo como entrada inicial SEED e como entradas subseqüentes o resultado do hash anterior. O valor resultante é a OTP utilizada nessa sessão. Feito isso, cliente e servidor subtraem 1 de X e armazenam o novo X.

- Passo 1[17]: $OTP = H^X(SEED)$
- Passo 2: $X = X - 1$

Como a função H() não é inversível, torna-se impossível descobrir o valor de $H^{X-1}(SEED)$ conhecendo apenas $H^X(SEED)$.

Quando X finalmente chegar a 1, é preciso executar o protocolo de registro novamente.

[17] A notação $H^x(SEED)$ significa H(H(H(...(H(SEED))...))), isto é, chama-se recursivamente a função H() por X vezes com parâmetro inicial SEED.

O desafio para o sistema de Lamport é valor atual de X e a chave de criptografia é representada por SEED.

Pode-se citar outros exemplos com idéias semelhantes como (KU, 2004), o sistema S/Key (HALLER, 1995) e o Kerberos (NEUMAN;TS'O, 1994).

3.3. Algoritmos de Desafio-Resposta

Como visto, *token*s utilizam algoritmos do tipo desafio-resposta (*challenge-response*) para autenticar o usuário. Nesse tipo de algoritmo o usuário mostra o conhecimento de um segredo sem o revelar explicitamente. Isto é feito através do envio de um desafio que, quando utilizado em conjunto com o segredo em um algoritmo de criptografia, gera a resposta para o desafio.

A seguir mostra-se de forma genérica as etapas necessárias para que uma entidade A se autentique perante uma entidade B. Admite-se que A já enviou uma mensagem inicial para que B suponha que está se comunicando com A.

	Dados Transferidos		Comentários
A	← D	B	Desafio 'D' é enviado
A	→ R = f(S, D)	B	'A' envia a resposta 'R' que é uma função do segredo 'S' e do desafio 'D'

Tabela 3 - Forma geral de um algoritmo desafio-resposta

Na transformação também é comum o uso de um sal que pode ser incluído na transformação 'f', em que sal é uma cadeia de bits aleatória aplicada juntamente com o desafio antes que ele passe pela função de criptografia. Com isso se um desafio for igual ao outro, devido ao sal, a saída será diferente a cada execução.

Algoritmos de desafio-resposta também podem ser utilizados para garantir autenticação mútua, isto é, garantia de que ambas as partes estão se comunicando com a entidade correta. Ambos os lados mostram o conhecimento de um segredo através de desafios gerados por ambas as partes. Como será mostrado, autenticação mútua não é a simples execução do algoritmo de autenticação unilateral duas vezes.

3.3.1. Desafio-resposta baseado em criptografia simétrica

Quando um desafio utiliza criptografia simétrica, a transformação 'f' anterior é o algoritmo simétrico 'E' e o segredo 'S' é uma chave 'K' previamente compartilhada entre as partes. A resposta 'R' é validada se for possível recuperar o desafio 'D' a partir da chave 'K'.

Sentido			Conteúdo
A	←	B	D
A	→	B	R = E(K, D)

Tabela 4 - Autenticação unilateral em desafio-resposta baseado em cifra simétrica

Para obter autenticação mútua, é necessário um passo a mais onde são trocados dois desafios 'D1' e 'D2':

Sentido			Conteúdo
A	←	B	D1
A	→	B	R1 = E(K, [D1, D2])
A	←	B	R2 = E(K, D2)

Tabela 5 - Autenticação mútua em desafio-resposta baseado em cifra simétrica

A entidade 'B' autentica 'A' se e somente se encontra 'D1' dentro de 'R1'. Por sua vez, 'A' continua a comunicação com 'B' se e somente se a resposta 'R2' contém o valor 'D2' devidamente cifrado. O Kerberos é um exemplo de protocolo que utiliza esse tipo de autenticação.

3.3.2. Desafio-resposta baseado em MAC

Quando o algoritmo é baseado em um código de autenticação de mensagens, a transformação 'f' anterior é o algoritmo MAC 'H' e o segredo 'S' é uma chave 'K' previamente compartilhada pelas partes. A resposta 'R' é validada se for igual ao MAC do desafio enviado.

O protocolo para autenticação unilateral vira:

Sentido			Conteúdo
A	←	B	D
A	→	B	R = H(K, D)

Tabela 6 - Autenticação unilateral em desafio-resposta baseado em MAC

Para obter autenticação mútua, é necessário um passo a mais onde são trocados dois desafios 'D1' e 'D2':

Sentido			Conteúdo
A	←	B	D1
A	→	B	R1 = [D2, H(K, [D1, D2])]
A	←	B	R2 = H(K, D2)

Tabela 7 - Autenticação mútua em desafio-resposta baseado em MAC

A entidade 'B' autentica 'A' se e somente se a resposta 'R1' está correta. Por sua vez, 'A' continua a comunicação com 'B' se e somente se a resposta 'R2' está correta.

3.3.3. Desafio-resposta baseado em criptografia assimétrica

Quando um desafio utiliza criptografia assimétrica, a transformação 'f' anterior é o algoritmo assimétrico 'E'. Não existe um segredo 'S' compartilhado entre as entidades. São utilizados dois pares de chaves públicas KU e privadas KR onde KU_A e KR_A pertencem à entidade 'A' e KU_B e KR_B pertencem à entidade 'B'. A função 'h(x)' representa o hash de 'x'.

O protocolo para autenticação unilateral vira:

Sentido			Conteúdo
A	←	B	[E(KU_A, D), h(D)]
A	→	B	R = E(KR_A, E(KU_A, D))

Tabela 8 - Autenticação unilateral em desafio-resposta – Criptografia Assimétrica

Neste caso o desafio 'D' vai cifrado com a chave pública de 'A'. O hash é necessário para 'B' demonstrar que conhece 'D', sem revelá-lo durante a transmissão. 'A' responde com o desafio 'D' original.

Quando se deseja obter autenticação mútua, existe um passo extra, no qual são trocados dois desafios 'D1' e 'D2', como na tabela a seguir.

Sentido			Conteúdo
A	←	B	E(KU_A, D1)
A	→	B	E(KU_B, [D1, D2])
A	←	B	D2 ou h(D2)

Tabela 9 - Autenticação mútua em desafio-resposta – Criptografia Assimétrica

Na primeira etapa 'B' envia o desafio 'D1' cifrado com KU_A. 'A' decifra 'D1', gera 'D2' e cifra os dois valores com a chave KU_B. 'B' autentica 'A' quando recebe o desafio 'D1' de volta. 'A' autentica 'B' se receber o desafio 'D2' de volta.

3.4. Tipos de desafios

Algoritmos desafio-resposta utilizam desafios implícitos ou explícitos. Desafios implícitos baseiam-se na data e hora atual, ou um conjunto de números de alguma forma seqüenciais. Desafios explícitos demandam do servidor o envio do desafio para o cliente.

Para desafios baseados em tempo (*time-based*) é definido um intervalo mínimo de tempo, tipicamente na ordem de 1 minuto, no qual uma nova OTP é gerada. O sistema tem a vantagem de não precisar transmitir o desafio do servidor para o cliente (desafio implícito). A desvantagem é que os relógios do cliente e do servidor perdem o sincronismo com o passar do tempo, principalmente quando o *token* não é utilizado por longos períodos. Sempre que o servidor autentica um cliente, o sincronismo é refeito com o cliente, armazenando o deslocamento entre o relógio do cliente e o relógio do servidor.

Para contornar o problema da perda de sincronismo, o protocolo exige que o servidor considere válidos todos os desafios dentro de uma determinada janela de tempo. Uma janela típica tem o tamanho de 3 minutos. A Figura 6 mostra um exemplo de como funciona essa janela.

<table>
<tr><td></td><td colspan="6">◄ Janela de 3 minutos ►</td><td></td></tr>
<tr><td></td><td>17:57</td><td>17:58</td><td>17:59</td><td>18:00</td><td>18:01</td><td>18:02</td><td>18:03</td><td></td></tr>
<tr><td>Inválidos</td><td colspan="3">Mínimo</td><td>Relógio Servidor</td><td colspan="3">Máximo</td><td>Inválidos</td></tr>
</table>

Figura 6 - OTP Baseado em Tempo

Na Figura 6, apesar do horário do servidor estar marcando exatamente 18:00, é preciso validar qualquer desafio válido que esteja entre às 17:57 e às 18:03. Essa janela reduz a segurança do sistema, pois mais de uma OTP é válida em um determinado instante. No sistema acima são consideradas válidas 7 OTP diferentes. Uma OTP com 1 milhão de desafios diferentes (6 dígitos) torna-se equivalente a um

sistema de aproximadamente 143 mil desafios. Por essa razão é preciso balancear o tamanho da janela e a freqüência de geração das OTP.

A implementação baseada em eventos (*event-based*) adota um contador[18] como desafio. A cada nova autenticação, o contador é incrementado e uma OTP é gerada a partir desse número. O sistema tem a vantagem de ser implícito, porém também necessita de uma janela, visto que um usuário pode gerar senhas e nunca utilizá-las[19]. Nesse caso a janela é apenas progressiva (nunca serão analisados desafios que já foram utilizados). Esse tipo de desafio aparece em (M'RAIHI et al., 2005).

◄ Janela de 10 números ►		
1000	1005	1010
Esperado	Gerado pelo *token*	Máximo

Figura 7 - OTP baseado em seqüências

Na Figura 7 o servidor autenticará o usuário que apresentar uma resposta válida até 10 números acima da resposta esperada.

O problema da janela e do sincronismo pode ser resolvido de forma mais fácil que no caso do relógio. Sempre que detectado um desafio fora de ordem, o sistema pode solicitar uma nova OTP e confirmar se o usuário realmente possui o *token*[20].

Baseado no método citado acima de restabelecimento do sincronismo, nota-se que o desafio baseado em seqüência ainda é vulnerável a *phishing*. Se um atacante obtivesse múltiplos desafios de um cliente, o atacante teria acesso a diferentes OTP válidas até que o usuário voltasse a acessar o sistema.

Finalizando o assunto sobre os tipos de OTP, chega-se ao momento de explicar o funcionamento da OTP com desafio explícito. Nesse modo, o servidor apresenta explicitamente um número que serve como desafio ao cliente. O cliente entra com o número no *token* e esse responde com a OTP correspondente.

Como se pode perceber, a sincronia do sistema é estabelecida assim que um novo desafio é proposto pelo sistema. Esse sistema não sofre com o problema de

[18] Uma seqüência não precisa ser óbvia (como em 1, 2, 3, 4...). Números podem ser preparados por algo como um *hash* antes do seu uso como desafio.
[19] Não é difícil imaginar uma situação onde uma OTP não é utilizada. Ao apresentar o sistema a amigos ou mesmo com toques acidentais, OTPs podem ser geradas e nunca utilizadas no sistema.
[20] Idéia semelhante pode ser adotada para o sistema baseado em tempo, porém, é preciso fazer o usuário aguardar o intervalo em que é gerada a próxima OTP.

múltiplas OTP válidas em um mesmo instante de tempo e nem com os demais problemas que o erro de sincronismo pode acarretar.

A desvantagem é a forma como o *token* é utilizado no dia-a-dia. A implementação do *token* vai depender de um método de entrada de dados mais complexo que os demais sistemas. Pensando na implementação de um *hardware* para os 3 casos, enquanto nos casos anteriores a interface pode se basear apenas em um ou dois botões, a interface desse modelo deve ser mais complexa exigindo a adoção de algo como um teclado.

Nos 3 casos acima, consegue-se apenas obter autenticação unilateral pois o cliente não envia um desafio para autenticar o servidor. Os aparelhos de desafio explícito podem ser facilmente modificados para essa função.

3.4.1. Uso

Como visto, independentemente do tipo de desafio utilizado, e do tipo de algoritmo desafio-resposta, quando o usuário fica responsável por digitar a resposta, dificilmente todos os bytes resultantes da resposta são enviados.

Isso é feito, pois, além da possibilidade de erros de digitação, é incômodo para um usuário digitar longas cadeias de números. Nesse caso, trunca-se parte da resposta escolhendo-se um tamanho que apresente um balanço entre segurança e facilidade de uso.

Mais a frente será apresentado como o dispositivo proposto trata desafio, algoritmo e a questão segurança/facilidade de uso.

3.5. Integridade de transações

Schneier (SCHNEIER, 2005) mostra que *token*s apenas no modo de autenticação de entidade não garantem a segurança de um sistema, pois eles ainda são vulneráveis a ataques do tipo *man-in-the-middle*[21]. No começo de julho de 2006 foi

[21] O ataque não invalida totalmente o uso de OTP, por ser condicionado ao acesso do recurso pelo cliente. Muitos sistemas não permitem mais de uma autenticação simultânea pelo mesmo usuário e o cliente perceberia que algum problema está acontecendo.

divulgada a notícia de que a autenticação baseada em dois fatores utilizada por um grande banco americano foi burlada (KREBS, 2006).

O funcionamento do ataque baseou-se no envio de um e-mail que redireciona o cliente para uma página falsa (*phishing* tradicional). Esse por sua vez, através de um *proxy*, redireciona as entradas do usuário (inclusive a OTP) para a verdadeira página do banco, autenticando assim o atacante e não o verdadeiro cliente do banco[22].

Assim sendo, um *token* apenas com a função de OTP não é suficiente para evitar ataques do tipo *man-in-the-middle*. Por essa razão começam a surgir *token*s que além de realizar autenticação de usuário também começam verificar também a integridade[23] de transações. Pode-se fazê-lo através de assinatura de uma transação (criptografia assimétrica) ou através de autenticação de transação (uso de segredo compartilhado).

O uso desse sistema condiciona a conclusão de qualquer transação à geração de um código de integridade da mesma pelo *token*. O fornecimento dos dados identificadores da transação deve ser feito de forma que o cliente associe visualmente a transação desejada com a transação efetivamente assinada ou autenticada.

Como o código de integridade está sempre diretamente relacionado à transação, esta não pode ser forjada por um atacante, sem o conhecimento das chaves de segurança envolvidas no processo. O emprego de um código de integridade garante ainda que o cliente esteja ciente de qualquer transação realizada em seu nome, decidindo assim se ele deve ou não prosseguir com a operação.

Neste sistema, deve-se fazer verificação de integridade nos dois sentidos, isto é, o banco assina/autentica a transação e a envia para o usuário. O usuário verifica se a transação é a solicitada e o *token* verifica o código de integridade gerado pelo banco. Como o código de integridade garante tanto os dados como a origem, percebe-se erros de digitação e/ou tentativas de fraude.

---

[22] Conhecendo esse tipo de ataque, uma instituição pode tomar precauções. Quando uma mesma origem estabelece múltiplas conexões a diferentes contas em um espaço curto de tempo, a atividade deve ser considerada suspeita.

[23] Aqui se refere à integridade de dados e autenticação de mensagem.

Depois de validada a integridade da mensagem enviada pelo banco, o *token* pode validar a transação com um novo código de integridade e o cliente pode enviá-la ao banco. O banco, por sua vez, verifica se um novo código de integridade foi adequadamente gerado pelo *token*. A Figura 8 sumariza as etapas para a geração de uma assinatura/autenticação digital.

	Sentido	Mensagem	Descrição
1	Cliente ➔ Banco	Transação	Cliente deseja completar uma transação
2	Banco	Alg(TransaçãoS)=INT_B	Banco gera INT_B da transação
3	Cliente ➔ Banco	(TransaçãoS | INT_B)	Banco envia Transação e INT_B ao cliente
4	Cliente (*Token*)	Alg(TransaçãoS) = INT_B?	Cliente verifica se INT_B é válido
5	Cliente (*Token*)	Alg((TransaçãoS|INT_B))=INT_C	Cliente assina/autentica transação e gera INT_C
6	Cliente ➔ Banco	INT_C	Cliente envia INT_C
7	Banco	Alg((TransaçãoS|INT_B))=INT_C?	Banco verifica INT_C
8	Banco	OK/Erro	Banco aceita transação se INT_C está correto

- Alg() pode ser um algoritmo de assinatura ou autenticação. Chave utilizada está implícita em Alg()
- TransaçãoS é Transação adicionado com um sal para evitar que INT_B seja igual para transações iguais realizadas em instantes diferentes
- Passos 4 e 5 são realizados pelo *token*
- INT_B representa o código de integridade da transação, gerado pelo banco
- INT_C representa o código de integridade da transação, gerado pelo cliente

Figura 8 - Passos para Assinatura/Autenticação de uma transação

Na Figura 8, a transação é garantida, pois um atacante não pode gerar INT_B ou INT_C sem o conhecimento da(s) chave(s) envolvida(s). Um sal é utilizado para evitar o reaproveitamento de uma mensagem anterior.

Pode-se utilizar a mesma chave para autenticação e para a assinatura/autenticação desde que se garanta que as mensagens trocadas nunca sejam iguais. Isso é obtido adicionando-se um tipo de sal para autenticação e outro para assinatura/autenticação.

3.6. Formas de Implementação

Tokens são implementados de diferentes maneiras, existindo versões em *software* rodando em PCs ou dispositivos portáteis, assim como em *hardware* seja no formato de um chaveiro, de uma pequena calculadora portátil ou ainda no formato de cartões de papel.

3.6.1. *Token* em PC

*Token*s implementados no PC, funcionam como qualquer outro programa no computador, porém antes de serem executados exigem que o usuário digite uma senha antes de fornecer acesso às suas funcionalidades.

Essa abordagem apresenta a vantagem de não ser necessário portar um objeto a mais para obter autenticação baseada em múltiplos fatores. *Tokens* em *software* utilizam os recursos de máquinas com maior poder computacional e por isso podem utilizar algoritmos assimétricos sem maiores complicações. Isso facilita a distribuição de chaves e permite operações computacionalmente mais pesadas, como a assinatura de transações.

Por outro lado, o *token* fica sujeito a todas as vulnerabilidades deste ambiente como vírus e programas espiões. Com isso, o uso da autenticação forte pode ser comprometida se o arquivo de chaves do cliente é roubado por estes programas maliciosos.

Para evitar as desvantagens de um PC, uma possível solução é utilizar o *token* com o auxílio de máquinas virtuais. Em poucas palavras, essa solução seria possível, pois as máquinas virtuais permitem a um mesmo computador executar simultaneamente diferentes sistemas operacionais de forma independente entre eles.

A criação e o gerenciamento de máquinas virtuais fica a cargo do Monitor de Máquinas Virtuais (MMV). Esse programa controla os recursos da máquina hospedeira e os distribui entre os múltiplos SO de forma que pareçam estarem sendo executados em computadores diferentes. A Figura 9 ilustra esse conceito.

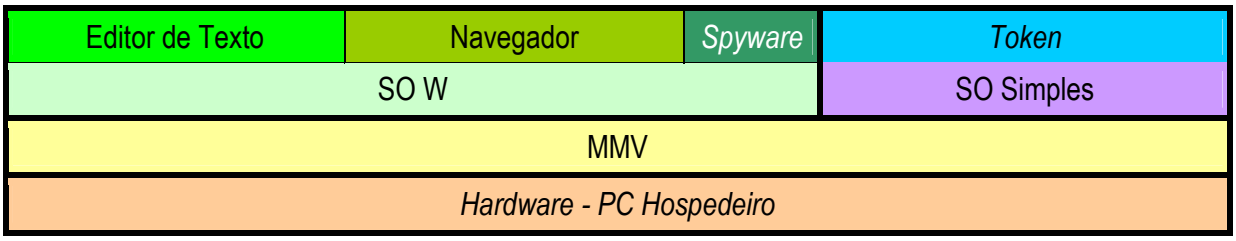


Figura 9 - Funcionamento de Máquinas Virtuais

Como no sistema acima os SO estão isolados entre si, um programa espião ou um vírus executado em SO W, não afeta o SO vizinho. Com isso, um *token* pode ser executado em segurança sem que nada interfira no seu funcionamento ou em seus dados.

O SO Simples (onde o *token* está sendo executado) deve ser o mais simples possível para minimizar possíveis vulnerabilidades em seu código fonte.

O modelo com MMV combina as vantagens de um *token* em *software* com o *token* em *hardware* que já é naturalmente isolado de outros sistemas. Entretanto os MMV atuais não são capazes de completamente virtualizar recursos como placas de captura de vídeo ou aceleração gráfica[24]. Usuários de aplicativos de escritório como editores de texto, emuladores de terminais, entre outros, são menos penalizados pelo uso de máquinas virtuais. Todavia, usuários que necessitam de recursos mais avançados não poderão desfrutar dessa vantagem de máquinas virtuais.

O custo de MMV vem caindo e diversas empresas[25] vêm lançando versões de seus MMV de uso livre mesmo para em ambiente corporativo. Essa versões funcionam de forma um pouco diferente do que foi demonstrado na Figura 9. O MMV é executado em cima de um SO hospedeiro, para fornecer os recursos para os SO hospedados.

Outra possibilidade para contornar as vulnerabilidades do PC é utilizar *Trusted Computing* , (TRUSTEDCOMPUTINGGROUP, 2007). Esse é um assunto extenso e será brevemente resumido aqui. A Intel chama essa plataforma de "La Grande" (INTEL, 2006). Essa plataforma possui as seguintes características:

[24] No segundo trimestre de 2007 foi lançado um novo MMV capaz de aproveitar recursos 3D, como OpenGL e DirectX, dentro de máquinas virtuais (PARALLELS, 2007).

[25] Empresas com versões gratuitas de seus MMV:
VMware – http://www.vmware.com
Microsoft - http://www.microsoft.com/windows/virtualpc/default.mspx

- Execução protegida: Todos os programas são executados em área de memória protegida que não sofre interferências de outros aplicativos
- Atestação: prova baseada em *hardware* de que o ambiente em que se vai trabalhar é confiável
- Armazenamento Selado: Proteção aos dados do usuário
- E/S Protegida: Garante comunicação segura entre os dispositivos de entrada/saída

Um dos problemas dessa solução é que ela precisa que toda a arquitetura de um PC seja recriada. Cada componente (teclado, memória, *chipset*, etc) deve possuir módulos especiais para estabelecer qualquer comunicação. Isso envolve custos, mudanças de *hardware* e *software*, problemas de compatibilidade e tempo até todos adotarem essa solução.

3.6.2. *Token* em dispositivos portáteis

Os *token*s em dispositivos portáteis funcionam como os *token*s rodando em PC porém com o programa especificamente desenvolvido para essa categoria. Os dispositivos portáteis considerados aqui são telefones celulares, PDAs e *smartphones*.

A idéia de se utilizar tais dispositivos é uma decorrência natural, já que o seu uso é bastante disseminado entre a população: as pessoas já têm o hábito de carregá-los consigo. Outro ponto a seu favor é que a capacidade computacional deles vem aumentando a cada dia. A solução implementada em dispositivos portáteis quando comparada a uma solução em PC, também tem vantagem de ainda não ser o principal alvo de pragas virtuais.

Existem diferentes ferramentas para desenvolvimento de programas em plataformas móveis como BREW ou Java. A tecnologia Java é particularmente interessante, uma vez que hoje 40% da base mundial de celulares (FELITTI, 2006) já suporta essa plataforma e os processadores mais recentes já incluem a tecnologia Jazelle

(STEELE, 2001). Com essa tecnologia as instruções Java mais comuns[26] são diretamente executadas em *hardware* aumentando o desempenho desses dispositivos.

Existem trabalhos que propõem o uso de celulares como token que recebem mensagens de SMS como forma de enviar as OTP ao cliente (CLAESSENS; PRENEEL;VANDEWALLE, 2002). Isso é interessante por utilizar canais separados, e por isso mais complicados de serem controlados por um atacante, No entanto, isso representa um custo adicional para cada autenticação, além do fato de que mensagens SMS nem sempre são enviadas de forma instantânea.

O poder computacional de dispositivos portáteis pode ser utilizado de forma que eles façam o papel do objeto na autenticação (ME; PIRRO;SARRECCHIA, 2006). Neste caso programas em Java rodando nesses ambientes são responsáveis por gerar as OTP para o usuário.

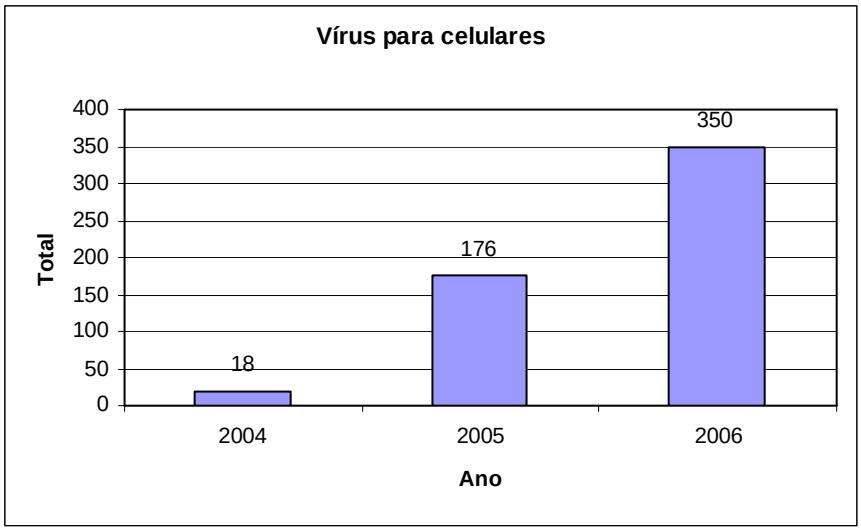


Figura 10 - Crescimento da quantidade de vírus para celulares

Freqüentemente esses trabalhos baseiam-se na hipótese de que esses dispositivos são considerados isolados do computador, isto é, seguros para serem adotados como tokens. Apesar de ainda pouco freqüente, já existem vírus (TERRA, 2006) e *keyloggers (IDG, 2006)* criados para celulares. Há também vulnerabilidades que exploram problemas no envio de mensagens SMS, no sistema de *bluetooth* ou senhas fracas utilizadas por seus usuários (HONG, 2005).

[26] As demais instruções são emuladas (são instruções que aparecem 5% do tempo em aplicações típicas).

Uma pesquisa feita pela McAfee (FELITTI, 2006) é mostrada através do gráfico da Figura 10, no qual é possível visualizar a evolução dos vírus para celulares com o passar do tempo. Dado o aumento no número desses tipos de vírus, espera-se que os *tokens* para dispositivos móveis, no futuro, irão apresentar as mesmas vulnerabilidades dos *tokens* para PC.

3.6.3. *Token* em *hardware*

Os *token*s em *hardware* implementam os algoritmos para geração de OTP, autenticação ou assinatura em pequenos dispositivos semelhantes a chaveiros ou calculadoras de mão.

A grande vantagem do *token* em *hardware* é a sua isolação. Por não possuir nenhuma forma de comunicação – além do *display*/teclado – a chave de criptografia, os algoritmos, a memória, a entrada de dados bem como a exibição de dados não está sujeita a ataques com a função de furtar dados. Tanto as operações quanto os dados críticos estão isolados de qualquer ataque ao computador do usuário.

*Token*s no formato de um chaveiro, normalmente apresentam apenas um botão acionado quando se deseja gerar uma OTP. As OTP geradas são baseadas em tempo ou em uma seqüência numérica. Eles não possuem um PIN[27] para evitar a geração da OTP. A proteção é feita pela senha digitada quando o usuário acessa o sistema.

*Token*s com teclado são protegidos através de um PIN e além de gerar a OTP baseada em um dos sistemas descritos anteriormente, também podem implementar a função de geração e verificação de códigos de integridade para transações.

A sua maior desvantagem é a necessidade de se portar um novo dispositivo. O problema cresce à medida que mais entidades começam a utilizar esse meio de autenticação. Daí decorre que cada uma dessas entidades necessitará de um novo dispositivo associado.

A diferença entre dois dispositivos de autenticação é basicamente a chave utilizada para suas operações. A princípio um mesmo dispositivo poderia armazenar mais de

[27] Personal Identification Number – Senha de 4 a 6 dígitos para proteger um objeto em caso de perda.

uma chave e desta forma, ser utilizado por diversas entidades, porém isso pode acarretar problemas políticos e de logística que não serão discutidos ao longo desse texto.

3.6.4. *Token* em papel

Durante a Segunda Guerra Mundial, alemães distribuíam livros códigos em suas bases com as senhas diárias que seriam utilizadas no mês por sua máquina Enigma. Isso dificultou consideravelmente o trabalho dos aliados (SINGH, 2001), (MAO, 2004).

Diversos bancos brasileiros já modificaram o seu sistema de autenticação e vêm distribuindo cartões com função semelhante à dos livros códigos alemães. Neste caso, os cartões possuem números impressos e são utilizados conforme a necessidade.

Para uso nos dias atuais, o sistema tem características interessantes, pois é barato e cumpre a função de fazer com que o cliente utilize uma OTP. No entanto, como no caso alemão (e de qualquer senha anotada em um pedaço de papel), uma pessoa com acesso físico ao cartão pode obter uma cópia e conseqüentemente os dados necessários para realizar a autenticação em alguma página na web.

Diferentemente dos bancos, os alemães não tiveram que enfrentar o problema do *phishing* (pelo menos não *on-line*). Esse sistema não impede que uma página falsa solicite para que o usuário digite várias senhas do cartão comprometendo, assim, a sua segurança.

3.6.5. Outros sistemas de autenticação forte

Personalização – nesse sistema a página do banco é personalizada através de cores ou então figuras escolhidas pelo próprio usuário. Desta forma o usuário percebe que entrou em uma página falsa assim que as suas configurações padrões não são carregadas (TUOHEY, 2006).

Localização – nesse método a entidade usa o computador do usuário como *token* e não permite que o acesso seja realizado através de outros computadores (ANAKAN, 2006).

CD – nesse sistema cria-se um CD com um SO especial carregado sempre que se deseja acessar uma página. Programas espiões não serão carregados juntamente com o SO, porém ao utilizá-lo é preciso reiniciar o computador antes de acessar a página da instituição.

3.7. Limitações

Um dispositivo de autenticação aumenta a segurança da autenticação de usuários, porém não impede que um computador ainda seja alvo de pragas virtuais como vírus, *spywares*, ou outros. Mesmo com o emprego dos dispositivos esses softwares maliciosos continuarão em execução no hospedeiro e podem ainda transmitir dados para atacantes, destruir dados ou ainda atrapalhar o uso do computador.

O combate a esses problemas é preventivo e deve ser realizado através de métodos tradicionais como o uso de programas antivírus, *firewall* e *anti-spyware*. Algumas recomendações de programas gratuitos podem ser encontradas nos trabalhos de Harrison (HARRISON;BOLLINGER, 2004), Ames (AMES, 2004) entre outros.

Essas ferramentas de segurança não tem sido tão adotadas quanto se deveria (LEE;KOZAR, 2005). Pesquisa apresentada pelo CGI.BR (CETIC.BR, 2007) mostra que no Brasil o programa mais utilizado é o antivírus, e apesar de estar presente em pelo menos 70,2% dos computadores brasileiros (10,9% não sabe se usa alguma ferramenta) a atualização semanal é feita em no mínimo 49,7% dos casos (19,4% não sabem se é atualizado).

Não apenas ferramentas de segurança, como também hábitos saudáveis devem ser adotados quando se trata da internet. Isso é feito através de campanhas de empresas de segurança e bancos. Esse tipo de iniciativa deve ajudar a formar e disseminar a cultura de segurança digital na sociedade.

3.8. Produtos comerciais de autenticação forte

Atualmente, existem diferentes produtos comerciais disponíveis no mercado. Essas soluções são implementadas tanto em *hardware* quanto em *software* para PC e computadores portáteis.

Normalmente o principal foco desses produtos é a autenticação de usuário, porém há soluções que também realizam autenticação de mensagens. Com o objetivo de comparar os diferentes produtos, foi montada para esse trabalho a Tabela 10 que apresenta a principal característica dessas soluções.

Empresa	Identidade	Integridade	*Hardware*	*Software*	Chave	Corporativo	Banco
Aladdin	X	X	X	X	?	X	
Authenex	X		X		Simétrica	X	
CryptoCard	X	X	X	X	Simétrica	X	
Kobil Systems	X		X	X	?	X	
RSA SecureID	X	X	X	X	Simétrica	X	
SafeNet iKey*	X	X	X		?	X	
Secure Computing	X		X	X	?	X	X
Vasco	X	X	X	X	Simétrica	X	X
Anakam	X			X	?	X	
PassMark Security[28]	X			X			
WiKID Systems	X	?		X	?	X	

Tabela 10 - *Token*s Comerciais

A Tabela 10 foi construída apenas a partir das páginas web[29] de seus fabricantes. O único dispositivo ao qual foi possível ter acesso é apresentado no quinto capítulo deste texto e o mesmo foi utilizado como base de comparação para o dispositivo proposto nesse trabalho.

A coluna "identidade" marca as soluções que empregam alguma forma de autenticação forte. A coluna "integridade" indica se a solução também valida a integridade de documentos, as colunas "*hardware*" e "*software*" indicam o tipo de implementação do dispositivo de autenticação e a coluna "chave" indica o tipo de chave adotada para realizar a autenticação.

[28] Implementação que usa 2 fatores, porém não é baseado em OTP
[29] Dados atualizados até Julho de 2006

A coluna "corporativo" indica que o sistema foi criado para ser utilizado em redes corporativas com serviços como VPN, LAN, RADIUS. A coluna "banco" indica se a solução já foi adotada por algum banco no mundo.

As células com um sinal "?" indicam que o atributo do produto em questão não pode ser verificado com absoluta certeza a partir dos dados disponíveis na página do fabricante.

A quantidade de produtos e fabricantes disponíveis indica que esse tipo de autenticação vem ganhando espaço no mercado. Em breve, o uso se tornará mais comum e as pessoas irão se acostumar com essa forma alternativa para representar a sua identidade.

3.9. Sumário

Este capítulo apresentou como são utilizados algoritmos de desafio-resposta com implementações em chaves simétricas e chaves assimétricas.

Posteriormente, apresentou-se os três tipos de desafios (baseados em tempo, seqüenciais e randômicos) utilizados nos algoritmos de autenticação de usuário. Foi visto que, além da autenticação de usuário, a autenticação de mensagens também é fundamental para garantir a segurança de transações.

A seguir, foram abordadas as principais formas de implementação de *token*s com suas vantagens e desvantagens.

Foi citada a limitação do uso de autenticação baseada em objeto. Apesar de aumentar a segurança da autenticação, ela não previne que pragas virtuais que se instalem no PC do usuário.

O capítulo termina apresentando uma tabela comparativa de diversos produtos de autenticação forte disponíveis no mercado.

4. PROJETO DE UM DISPOSITIVO PARA AUTENTICAÇÃO

O projeto apresentado neste capítulo tem o objetivo de demonstrar a viabilidade de se ter dispositivos seguros, com algoritmos abertos e de baixo custo. As questões de interface humano-computador foram simplificadas para que o projeto se concentrasse nos aspectos mais relevantes à segurança e se tornasse compatível com o esforço de desenvolvimento de uma dissertação de mestrado.

Descreve-se a seguir passos e decisões tomadas no projeto, requisitos básicos, a interface do sistema, *hardware* utilizado e finaliza-se com os algoritmos utilizados na programação do equipamento.

4.1. Requisitos de projeto

O projeto do dispositivo aqui apresentado apresenta os seguintes requisitos:

- Autenticação de usuário através de desafio-resposta explícito
- Autenticação de mensagens
- Adoção de algoritmos seguros
- Custo de produção máximo de cinco dólares
- Tempo de resposta para o usuário em torno de 4 segundos
- Duração da bateria em torno de cinco anos

As seções a seguir apresentam os passos necessários para que os requisitos aqui propostos fossem atendidos.

4.2. Interface com o usuário

Por definição, o *token* é um dispositivo isolado do computador onde uma comunicação é iniciada e por isso deve ter seus próprios meios para comunicar-se com o usuário. Com o requisito de baixo custo, as possibilidades de interface desse sistema tornam-se bastante limitadas.

Com isso, utilizando o computador como um guia para a interface do *token* indicando o que deve ser digitado e qual a resposta esperada, mas sem executar suas principais funções, uma interface simplificada pode ser adotada pelo dispositivo.

Utilizando o computador como um guia, o *token* foi baseado em um *display* simples, como aqueles utilizados em calculadoras de bolso. Esses *displays* possuem 8 caracteres de 7 segmentos cada. Apesar de limitados, eles ainda permitem que mensagens sejam exibidas desde que obedeçam as restrições de tamanho e segmentos, isto é, nem todas as letras estão disponíveis e por isso as mensagens devem ser escolhidas de forma a não confundir o usuário.

Com a finalidade de executar autenticação de usuário e de mensagens, um teclado numérico com teclas de controle Entra/Cancela é necessário para servir como entrada de dados pelo usuário.

4.2.1. Interface para geração de OTP

Como será explicado a seguir, adotou-se nesse trabalho uma OTP com tamanho de 9 dígitos. O *display* empregado é numérico e capaz de exibir até 8 caracteres simultaneamente. Por isso a entrada é dividida em 3 etapas de 3 dígitos cada.

<table>
<tr><th>Token</th><th colspan="2">Mensagem do guia no computador</th></tr>
<tr><td>1 ⌋ 503 ⌊</td><td>Entrada do bloco 1
🧍 ⇨ 🖩: Digite 503 e pressione Entra</td><td rowspan="3">Erros de digitação podem ser corrigidos com a tecla Cancela</td></tr>
<tr><td>2 ⌋ 041 ⌊</td><td>Entrada do bloco 2
🧍 ⇨ 🖩: Digite 041 e pressione Entra</td></tr>
<tr><td>3 ⌋ 980 ⌊</td><td>Entrada do bloco 3
🧍 ⇨ 🖩: Digite 980 e pressione Entra</td></tr>
<tr><td>---------
(barra de progresso)</td><td colspan="2">Aguarde um momento...</td></tr>
<tr><td>⌋ 104015 ⌊</td><td colspan="2">🖩 ⇨ 💻 Digite a resposta exibida no campo abaixo:
_____</td></tr>
<tr><td>Erro</td><td colspan="2">Caso a mensagem Erro tenha sido exibida, repita o processo.
Em caso de novo erro entrar em contato com o suporte.
Possibilidade de se estar acessando uma página falsa!</td></tr>
</table>

Tabela 11 - Uso do dispositivo para geração de OTP

Enquanto o computador exibe na tela o procedimento que deve ser seguido no *token*, isto é, digitar cada um dos três blocos separadamente, o *token* exibe o bloco

atual e a área onde deve ser digitado o desafio. A Tabela 11 mostra um exemplo de entrada do desafio assistida pelo computador.

4.2.2. Interface para autenticação de mensagens

Para poder guiar o usuário, o servidor deve montar a transação de forma que o usuário consiga facilmente associar a transação que ele está realizando com a seqüência de números que deve ser digitada no *token*. Abaixo, mostra-se um exemplo de como seria uma possível transação financeira e a forma como ela poderia ser digitada no *token*. Primeiro deve-se notar que qualquer transação pode ser representada facilmente apenas por números:

- Favorecido: João da Silva
- Tipo de transação: 8 (Depósito)
- Banco: 034
- Conta Corrente: 01978-9
- Valor: R$ 1.500,00
- INT: 2914 (Código de integridade)

Para esta transação[30], pode-se ter o seguinte número associado:

Transação	Banco	Conta[31]		Valor[31]		INT
8	034	0197	89	1	500	2914

Tabela 12 - Exemplo de transação

No exemplo anterior, o identificador da transação que deve ser digitado no *token* é 8 034 0197 89 1 500 2914. A informação foi dividida em blocos de no máximo 4 dígitos para facilitar a entrada no *token*. Com a informação visualmente clara para o usuário, ele pode ter certeza da informação que está sendo autenticada.

Neste ponto pode surgir uma dúvida de como garantir que o usuário é capaz de identificar que a operação está sendo guiada por uma página realmente legítima. Além do guia, a página gera um código de integridade (INT) associado a cada

[30] Nome não é necessário dado que banco e conta já identificam o cliente. Esse exemplo não leva em consideração o uso de um sal
[31] Conta e valor estão divididos em duas partes

transação. A função do INT é evitar que uma transação seja forjada como, por exemplo, com o uso dos mesmos números, porém utilizados de forma invertida para autenticar algum outro tipo de transação. O INT é o autenticador da primeira parte da transação gerado pelo próprio banco e posteriormente checado pelo *token*. Qualquer transação inválida é imediatamente detectada pelo *token* e esse alerta ao usuário da possibilidade de ele estar acessando uma página falsa.

Outras interfaces

Apesar de um teclado ser uma interface conhecida, foi mostrado que o usuário deve digitar uma grande quantidade de números para autenticar uma transação. Esta é uma tarefa cansativa e sujeita a erros, apesar de serem detectáveis com o uso do INT.

Uma possível mudança na interface de entrada seria dotar o dispositivo de autenticação de um sensor capaz de ler variações de luz e utilizar o monitor do computador ou os *leds* de um teclado (*Num Lock* ou *Caps Lock*) como fonte de luz. Esse sensor passa a fazer o papel do dispositivo de entrada de dados, poupando o usuário de digitar longas transações no dispositivo.

O uso de um *led* comum para essa função aparece no trabalho de Dietz (DIETZ; YERAZUNIS;LEIGH, 2003). Essa implementação permite que o *led* detecte a presença ou ausência de luz emitida por alguma fonte externa.

No curso desse projeto experimentou-se ligar um *led* comum, como mostra o artigo, e o sistema mostrou-se capaz de detectar a presença de luz no ambiente, ao custo de apenas duas portas extras do microcontrolador. Todavia, não se chegou a realmente implementar um protocolo capaz de estabelecer uma comunicação com o uso do *led*. Esse recurso foi relegado ao plano das possíveis melhorias futuras do projeto.

O *display* ainda é necessário, pois, é preciso que o usuário seja capaz de ver exatamente a transação que está sendo autenticada, para que ele escolha entre aceitar ou rejeitar a transação em questão. O valor de INT ainda continua necessário para garantir que uma transação forjada não foi criada por algum atacante.

4.3. *Hardware*

A parte eletrônica do dispositivo veio a tornar-se bem simples e consiste de um microcontrolador com poucos circuitos externos, além dos teclado e *display* citados anteriormente.

Microcontrolador é um circuito integrado que possui em uma única pastilha um processador para uso geral, memória e periféricos como, por exemplo, uma porta serial ou um conversor A/D. Para esse projeto, o microcontrolador é a principal peça do *hardware* do *token*. Além de executar o algoritmo de criptografia, ele é responsável por controlar os demais hardwares (essencialmente LCD e teclado) conectados a ele.

Um microcontrolador é um dispositivo genérico podendo ser empregado em diferentes projetos e com isso é produzido em grandes quantidades, tornando-o barato. Por essa razão ele foi adotado no projeto. A escolha do modelo de microcontrolador foi baseada em seus Periféricos Integrados e em seu Ambiente de Desenvolvimento.

4.3.1. Periféricos Integrados

O *token* é um dispositivo portátil e isso significa que ele deve ser pequeno e possuir alimentação própria. Minimizar o tamanho do *token* implica em escolher um microcontrolador que possua todos os periféricos necessários para o funcionamento do *token* integrados além de ter modos de gerenciamento de energia eficientes.

Para o projeto, um dos periféricos obrigatórios é um controlador de LCD. Um LCD não funciona por meio de níveis lógicos como os componentes digitais, mas sim pela tensão RMS em suas entradas (STMICROELETRONICS, 2005). O controlador de LCD permite que sinais não convencionais (para dispositivos digitais) possam ser gerados facilmente evitando um circuito complexo com o acréscimo de componentes específicos para essa função.

Outro periférico importante é o oscilador interno. Além de eliminar o dispositivo externo, um oscilador interno permite que sua velocidade possa ser facilmente

controlada pelo *software* do microcontrolador. Isso permite que tarefas com pouca necessidade de processamento, como a leitura de um teclado, sejam realizadas em menor velocidade. Tarefas mais exigentes, como a execução do algoritmo de criptografia, são executadas com maior velocidade para que o tempo de resposta do sistema seja razoável para um ser humano comum.

Um modo de *stand-by*, no qual todas as atividades do microcontrolador são suspensas quando o dispositivo não é utilizado, juntamente com o controle de velocidade do oscilador, são necessários para que a vida útil da bateria do dispositivo seja maximizada tanto quanto possível.

Com controlador de LCD e oscilador integrados, o circuito do sistema se reduz ao microcontrolador, alguns resistores e capacitores, *display* e teclado. Isso permite o uso de uma placa de circuito impresso pequena com no máximo duas faces. O tamanho final de todo o dispositivo é importante já que o *token* é um dispositivo que deve sempre ser carregado pelo usuário.

4.3.2. Ambiente de desenvolvimento

Com a complexidade do projeto e a quantidade de linhas de códigos que ele apresenta, o emprego da linguagem de montagem do microcontrolador (comum quando se trabalha com esse tipo de dispositivo) torna o código-fonte mais difícil de ser gerado, documentado, depurado e posteriormente analisado.

O projeto tornou importante a adoção de uma linguagem de alto nível como C (uma das poucas linguagens de alto nível disponíveis para microcontroladores) e de um ambiente de desenvolvimento com ferramentas que permitam editar o código e depurar erros facilmente. Além de simplificar a codificação, também tornam mais simples realizar qualquer manutenção que seja necessária posteriormente.

Ferramentas para carregar o programa no microcontrolador para testá-lo também são extremamente importantes, pois apesar de se contar com simuladores quando se trabalha com um microcontrolador, são raras as ferramentas para a simulação de certos comportamentos como o funcionamento do teclado e do *display* utilizados no projeto.

4.3.3. Sistema adotado

Dados os requisitos de *hardware* apresentados, foi adotado o microcontrolador PIC18F6490 da Microchip para os primeiros protótipos[32]. O ambiente de desenvolvimento utilizado foi o MPLAB 7 em conjunto com o compilador ANSI C MPLAB C18 2.3 ambos da Microchip. Apesar da escolha, há plataformas de outros fabricantes que podem ser utilizadas, como os microcontroladores da AVR que possuem modelos com os recursos citados.

4.3.4. Circuito elétrico

O diagrama elétrico do sistema pode ser encontrado no apêndice A. Pode-se perceber pela leitura do esquema que o circuito completo é simples e, como conseqüência, a PCB (placa de circuito impresso) apresenta dimensões reduzidas e necessita de apenas duas camadas, minimizando assim o seu custo de produção. O tamanho final da placa protótipo ficou em 28mm por 38mm.

Enquanto o projeto do circuito foi descrito no módulo ISIS do Proteus, o projeto das vias da PCB foi feito com o módulo ARES. O aplicativo ELECTRA ficou responsável por otimizar o traçado das trilhas na placa. O tamanho final da placa foi reduzido até que não fosse possível mais traçar trilhas na PCB.

Apesar do tamanho reduzido, para facilitar a compreensão do funcionamento da parte eletrônica do dispositivo, o mesmo foi dividido nas seguintes seções:

- Alimentação
- POR (*power on-reset*)
- Teclado
- *Display*
- Programação/Debug

[32] O projeto se iniciou com o PIC18F252, porém por serem da mesma empresa, possuem capacidades semelhantes e o código fonte não precisou ser muito alterado ao fazer a migração para o microcontrolador mais adequado.
Um novo microcontrolador, o PIC18F64J90, foi lançado após o término do primeiro protótipo e apresenta novos recursos e custo reduzido em relação ao adotado. A migração pode ser feita alterando-se os *headers* do código original e recompilando o projeto.

- Interface Analog

Alimentação

A alimentação é feita através de uma bateria "tipo botão" de 3V. Em cada entrada de alimentação do microcontrolador foram colocados capacitores cerâmicos de 15pF. A sua função é prover um caminho de baixa impedância para correntes de alta freqüência para o terra e reduzir o ruído irradiado pelas trilhas de alimentação. Outras considerações para reduzir a corrente de alimentação, sugeridas pelo fabricante (MICROCHIP, 2001), também foram levadas em consideração.

POR

O circuito POR inicia o dispositivo em um estado válido. O microcontrolador adotado permite um circuito externo simples que consiste apenas de um resistor de 10 kΩ ligado entre o pino de reset (MCLR) do microcontrolador e a alimentação Vdd.

Teclado

O teclado utiliza a configuração matricial (HIRAKAWA;CUGNASCA, 2002) para realizar varredura do teclado. No diagrama elétrico os pinos correspondentes ao teclado são LINx e COLx.

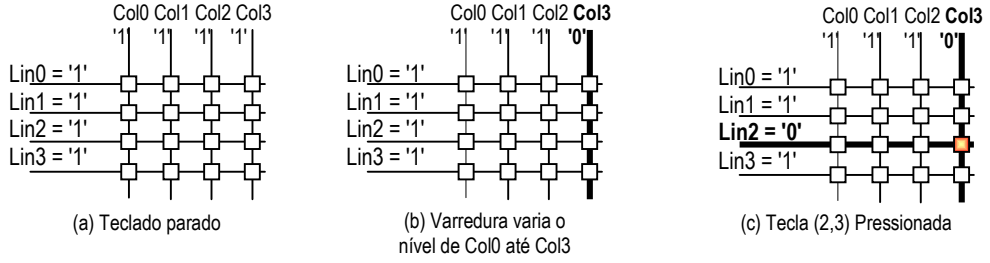


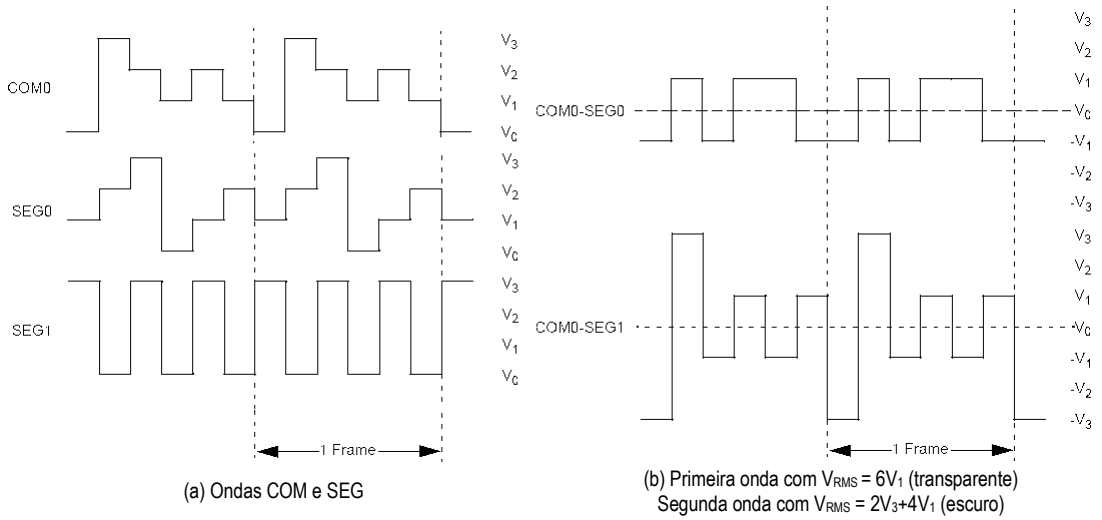
Figura 11 - Funcionamento do teclado

Explicando-se rapidamente, uma varredura consiste em aplicar o nível lógico '0' em cada um dos pinos COLx um de cada vez, enquanto os demais pinos (LIN e COL) ficam em '1'. Se algum pino LINx, também possuir o nível lógico '0', isso significa que

a tecla correspondente à linha e à coluna com o nível lógico '0' foi pressionada[33]. A Figura 11 ilustra o processo de varredura de um teclado.

Display

Com relação ao *display*, o seu funcionamento pode ser rapidamente explicado pela forma como os seus segmentos são controlados. Um segmento[34] é controlado pelo par COM/SEG. Se a diferença de tensão entre COM e SEG possuir uma tensão RMS[35] inferior a um valor mínimo, o segmento não escurece. A Figura 12 (MICROCHIP, 2004) torna mais claro o funcionamento de um *display*: o segmento controlado por COM0 e SEG0 se mantém transparente e o segmento controlado por COM0 e SEG1 fica escuro.



(a) Ondas COM e SEG

(b) Primeira onda com $V_{RMS} = 6V_1$ (transparente)
Segunda onda com $V_{RMS} = 2V_3+4V_1$ (escuro)

Figura 12 - Funcionamento de um *Display*

Para gerar a onda de COM e SEG são utilizados 3 resistores ligados aos pinos BIASx na forma de divisores de tensão. São utilizados resistores de 1MΩ para minimizar a corrente de fuga nesses pinos. Na Figura 12 quando $V_3=V_{DD}=3V$ e $V_0=V_{SS}=0V$, as demais tensões ficam: $V_2=2V$ e $V_1=1V$.

[33] Ao pressionar uma tecla, o usuário conecta o pino COLy ao pino LINx. Quando não conectados, LINx mantém o nível lógico '1'. Quando conectados ambos ficam com o mesmo nível lógico, indicando o pressionamento de uma tecla. Resistores de pull-up internos no microcontrolador são ativados antes da varredura para evitar que aconteça um curto quando os pinos são conectados.
[34] Equivalente a um pixel para um display gráfico.
[35] Apenas recordando: a tensão RMS, nesse caso, pode ser rapidamente obtida calculando-se a área da função dentro de um período.

O *display* do projeto impõe o uso de 3 pinos para a interface COM e mais 3 pinos SEG por caractere do *display*, perfazendo um total de 3+(3x8)=27 pinos de controle. No anexo, os pinos SEG estão agrupados por dígitos do *display*.

Programação/Debug

A programação do microcontrolador é feita através da interface ICSP (*In Circuit Serial Programming*), formada pelos pinos V_{PP}, PGD, PGC, V_{SS} e V_{DD}. Essa interface permite que o microcontrolador seja reprogramado mesmo estando conectado a um circuito.

Além de reprogramar, essa interface facilita a depuração do sistema, pois permite que o programa do microcontrolador seja executado no próprio circuito em modo de debug pela IDE. O resistor de 10 kΩ entre V_{PP} e V_{DD} também tem a função de isolá-los quando em modo de programação/debug.

Expansões

A interface Analog foi deixada para que expansões posteriores pudessem ser adicionadas ao protótipo. Ela está conectada a pinos que podem funcionar tanto como uma interface digital quanto uma interface analógica. No projeto atual, essa interface não seria utilizada em um sistema final.

4.4. *Software*

O *software* do sistema é dividido em três partes básicas. A primeira parte corresponde à implementação do algoritmo de criptografia adotado. O segundo tipo de *software* são os *drivers* que controlam o *hardware* do sistema e o terceiro tipo é composto pelos programas que implementam as funções de interface com o usuário.

4.4.1. Algoritmo de criptografia

O *token* baseado em um microcontrolador deve executar um algoritmo de criptografia que atenda a critérios de baixo consumo de memória e pouca necessidade de processamento.

Para atender a critérios de segurança, o algoritmo deve ser preferencialmente público. Dessa forma, é mais difícil que vulnerabilidades conhecidas estejam presente nesse algoritmo. Sugere-se também que a escolha seja livre do pagamento de *royalties* para que seu uso não implique em um acréscimo no custo final do sistema.

Segurança dos algoritmos

O critério de algoritmo público foi utilizado para que o sistema atendesse ao requisito de segurança.

Quando um algoritmo tem o seu código exposto e existe uma documentação explicando a razão da escolha de todos os parâmetros, isso permite que toda a comunidade de segurança possa verificar se a base matemática a partir da qual o algoritmo foi construído não apresenta vulnerabilidades, se os parâmetros escolhidos estão adequados para o algoritmo, ou mesmo se a implementação não apresenta algum defeito não percebido pelo programador.

Neste sentido esse trabalho verificou o uso tanto do CMAC-AES128 quanto do HMAC-SHA1 por serem considerados seguros na época em que esse texto foi escrito.

Algoritmos avaliados

Durante o desenvolvimento do protótipo foram experimentados dois algoritmos de MAC. O algoritmo final adotado foi o CMAC com o AES 128. O outro algoritmo testado foi o HMAC utilizando o SHA1.

O CMAC é baseado em cifras de blocos e é descrito originalmente com o nome de OMAC em (IWATA;KUROSAWA, 2003). Seu nome foi oficializado em (NIST, 2005).

Na implementação desse projeto, o algoritmo foi baseado na cifra de blocos AES com chave e blocos de 128 bits.

O HMAC é um MAC baseado em funções de hash. Sua descrição pode ser encontrada em (NIST, 2002) e a implementação aqui utilizada é baseada no SHA1. O nome CMAC é a abreviação de *cipher-based MAC* e HMAC é a abreviação para *hash-function-based MAC*.

A primeira implementação no projeto se baseou no HMAC-SHA1 e ele apresenta a característica de ser otimizado para um sistema de 32 bits, o que não é muito interessante para o microcontrolador adotado.

A mudança de algoritmos ao longo do projeto permitiu também criar um tabela para comparar o desempenho de ambos. Nessa comparação é preciso deixar claro que o microcontrolador utilizado é baseado em um sistema de 8 bits. Enquanto o SHA1 é otimizado para sistemas de 32 bits, o AES pode ser otimizado para uma grande quantidade de plataformas. Naturalmente, aqui foi empregada uma implementação otimizada para o sistema de 8 bits.

Além do problema da plataforma, é preciso lembrar que SHA1 foi enfraquecido por ataques apresentados por Wang, Yin e Yu (WANG; YIN;YU, 2005). Apesar dos ataques, o algoritmo não pode ser considerado "quebrado".

Comparações

O HMAC foi utilizado no começo do projeto e foi implementado em um microcontrolador PIC18F252 da Microchip[36]. Esse algoritmo consumiu um total de 16 kbytes de memória de programa do microcontrolador e operando a 4 MHz verificou a validade de um desafio e gerou a resposta do mesmo em 2,84±0,04 segundos.

[36] Apesar do microcontrolador ser de modelo diferente em cada um dos casos, a comparação é possível pois o compilador se manteve o mesmo (o que significa que o número de instruções gerada seria equivalente para ambos os casos) e o ciclo por instrução é o mesmo já que a arquitetura de ambos os microcontroladores é a mesma.

O CMAC foi implementado no PIC18F6490. O CMAC ocupou 1,6 kbytes da memória de programa do microcontrolador. Rodando a 1 MHz para verificar a validade de um desafio e gerar a resposta do mesmo, é necessário 1,52±0,05 segundo.

Note-se que o consumo de memória do CMAC é aproximadamente um décimo do consumo do HMAC e o processador está funcionando com ¼ de sua velocidade para realizar a mesma função com algoritmos diferentes.

Algoritmo	Tempo de execução	Memória
HMAC	2,84±0,04s (4 MHz)	16.056 bytes
CMAC	1,52±0,05s (1 MHz)	1.674 bytes
Observações	CMAC 7,4 vezes mais rápido	CMAC 9,5 vezes menor

Tabela 13 - Comparação entre HMAC e CMAC

Independente da escolha do algoritmo, a tabela acima além de comparar o desempenho de dois algoritmos também permite afirmar que, para o caso de um sistema de chaves simétricas, um algoritmo seguro é suficientemente rápido para ser executado em um microcontrolador sem a necessidade de qualquer tipo de *hardware* dedicado para essa função. Como rápido, entende-se um sistema que apresente uma resposta ao usuário sem causar desconforto no uso do sistema.

É preciso lembrar que não é objetivo dessa dissertação comparar o desempenho de diferentes algoritmos de MAC e por isso foi mantida a comparação entre algoritmos de 8/32 bits em uma plataforma de 8 bits. O objetivo é encontrar um algoritmo que se mostre adequado para o dispositivo implementado. A possibilidade de variar a velocidade do dispositivo apenas quando necessário viabiliza a escolha de ambos os algoritmos.

Em geral, funções dedicadas de hash são indicadas por possibilitarem alta velocidade, porém a diferença de desempenho observada nessa seção pode ser explicada pelas seguintes razões[37]:

- O código AES utilizado é especifico para a plataforma de 8 bits
- Enquanto somas no AES são feitas através da operação binária XOR, as somas do SHA-1 são feitas com números inteiros, o que requer um

[37] Também não é escopo do trabalho otimizar o algoritmo e por isso não se procurou otimizar o algoritmo do SHA-1 para o ambiente.

tratamento mais complicado na plataforma adotada, como por exemplo o tratamento do vai-um (*carries*)

- Os deslocamentos (*shifts*) do AES são mais simples de serem implementados

Uso dos algoritmos de MAC

Os algoritmos de MAC são utilizados como descrito no artigo de Vieira e Ruggiero (VIEIRA;RUGGIERO, 2007). Esses algoritmos são utilizados para garantir integridade e prover autenticação unilateral. No entanto, o resultado de um MAC não é utilizado diretamente, pois os dados devem ser digitados por uma pessoa e por razões de ordem prática é preciso utilizar um MAC reduzido, isto é, o valor de MAC truncado.

	Cliente		Servidor	Descrição
1			Gera RND	O servidor gera um número aleatório de 6 dígitos (RND)
2			INT = MAC(0xFF | RND)	Servidor calcula o MAC de 0xFF (sal) concatenado com RND. Pega-se os 3 últimos algarismos do MAC para se gerar o código de integridade do desafio (INT).
3	Cliente	←	Desafio = (RND | INT)	O desafio de 9 dígitos composto RND e INT é enviado ao cliente.
4	Cliente (*Token* ← Desafio)			O cliente digita o desafio no *token*.
5	*Token*: INT=MAC(RND)?			O *token* verifica se INT corresponde ao RND e pára o processo em caso de erro.
6	*Token*: RSP=MAC(0x01 | RND)			O *token* utiliza o 0x01 (sal) concatenado com RND como entrada para um novo MAC, gerando RSP.
7	*Token*: RSP			O *token* pega os 6 últimos algarismos de RSP e exibe o resultado para o usuário.
8	Cliente	→	Servidor	O usuário envia os dados para o servidor.
9			MAC(0x01 | RND) = RSP?	O servidor verifica RSP e finaliza a autenticação se o valor estiver correto.

* O uso da chave está implícito nas chamadas do MAC

Tabela 14- Uso do algoritmo de MAC para autenticação de usuário

Para truncar um MAC, o mesmo é calculado e então convertido para o seu formato decimal. Descarta-se então os dígitos mais significativos e fica-se apenas com o número de dígitos desejado em cada caso. A escolha dos bits a ser utilizada como MAC reduzido foi feita de forma arbitrária já que os algoritmos garantem que os bits

aparecem de forma equiprovável dentro do valor do MAC. A RFC4226 (M'RAIHI; BELLARE *et al.*, 2005) apresenta outra forma para a geração de um MAC reduzido.

A implementação final permite que tanto o tamanho do desafio, o tamanho da resposta e o tamanho do código de integridade de mensagens possam ser configurados. A sua escolha visa balancear segurança e facilidade de uso.

	Cliente		Servidor	Descrição
1			Recebe TRS	O servidor recebe a solicitação de uma transação (TRS)
2			Calcula INT: INT = CMAC(0xFE | TRS)	Servidor calcula o CMAC de 0xFE (sal) concatenado com TRS. Tomam-se os 4 últimos algarismos do CMAC para se gerar o código de integridade da transação (INT).
3	Cliente	← TRSI	TRSI = TRS | INT	A transação é formatada de forma que o cliente associe a transação com o número que será digitado.
4	Cliente (*Token* ← TRSI)			O cliente digita a transação no *token*.
5	*Token*: INT=CMAC(0xFE | TRS)?			O *token* verifica se INT corresponde ao TRS e pára o processo em caso de erro.
6	*Token*: RSP=CMAC(0x02 | TRS)			O *token* calcula CMAC de 0x02 (sal) concatenado com TRS.
7	*Token*: RSP			Tomam-se os 6 últimos algarismos do CMAC (RSP) e exibe o resultado para o usuário.
8	Cliente	→ RSP	Servidor	O usuário envia o código de autenticação para o servidor.
9			CMAC(0x02 | TRS) = RSP?	O servidor verifica RSP e autentica a transação se o valor estiver correto.

* O uso da chave está implícito nas chamadas do CMAC

Tabela 15- Uso do algoritmo de MAC para autenticação de transações

No protótipo foi definido que tanto desafio quanto resposta podem assumir 10^6 valores diferentes, isto é, um número decimal de 6 dígitos. O valor do desafio é um número de 9 dígitos decimais, dos quais 6 dígitos são gerados de forma aleatória pelo servidor e outros 3 dígitos são produzidos pelo MAC reduzido dos outros 6 dígitos com o intuito de que o cliente possa fazer verificação de integridade e detectar erros de digitação. O algoritmo de MAC empregado para autenticação de usuário é descrito na Tabela 14. O algoritmo garante integridade de dados e provê autenticação unilateral.

Como tanto desafio quanto resposta nascem a partir do MAC de 6 dígitos e uma mesma chave, a distinção entre ambos é feita através de um sal anexado aos 6 primeiros dígitos do desafio de forma transparente ao usuário. O sal do desafio é 0xFF e o sal da resposta é 0x01.

Para efetuar autenticação de transações, o algoritmo de MAC foi empregado como descrito na Tabela 15. O código de autenticação da transação (RSP) é um número de 6 dígitos e como é utilizada a mesma chave da geração de OTP, o sal para a geração da mesma é 0x02. O código de integridade da transação (INT) é o MAC reduzido de 4 dígitos e o sal para a geração do mesmo é 0xFE.

4.4.2. Programas Auxiliares

O funcionamento do *token* não depende apenas do algoritmo de criptografia executado, mas também de *drivers* e programas para realizar a interface com o usuário.

Os *drivers* têm a função de controlar os dispositivos conectados ao microcontrolador e os periféricos internos do microcontrolador. São eles que provêem o suporte, por exemplo, para gerenciamento de energia, exibição de mensagens e leitura de entradas de dados do usuário. O programa de interface é responsável por interagir diretamente com o usuário e passar os parâmetros adequados para o algoritmo de criptografia.

Esse conjunto formado pelos *drivers* e pelo programa de interface funciona como um sistema operacional exclusivamente projetado para suportar o algoritmo de criptografia.

Apesar de servirem apenas de suporte para o algoritmo de criptografia, esse conjunto precisa de uma atenção especial, pois os algoritmos de criptografia já são amplamente utilizados, enquanto os programas auxiliares são exclusivos para o sistema e precisam ser desenvolvidos desde o começo.

A Tabela 16 apresenta os principais *drivers* do sistema juntamente com a sua função principal.

Driver	Função
Controlador de LCD	Controla a geração dos caracteres exibidos no LCD. A principal função é transformar caracteres ASCII em um formato para a exibição no *display*.
Controlador de Mensagens	Armazenamento de mensagens ao usuário.
Configurações	Aloca áreas de memória para armazenar e consultar as configurações específicas de usuário e de estado do *token.*
Controlador de teclado	Leitura do teclado.
Controle de velocidade	Mudança de velocidade do oscilador quando necessário
Gerência de energia	Detecção e ativação do modo de economia de energia. Reativação do circuito para uso pelo usuário.
Gerência de tempo	Baseado no sistema interrupção, mantém um relógio interno para implementar funções dependentes de um relógio como timeouts, exibição de mensagens ou varredura do teclado.

Tabela 16 - *Drivers*

A Tabela 17 destaca e descreve brevemente os principais módulos do programa de interface.

Programa	Função
Gerenciador de chave de criptografia	Responsável por alterar a chave de criptografia sempre que necessário. Essa é a chave compartilhada previamente entre o servidor e o cliente
Gerenciador de PIN	Geração, leitura e alteração de um PIN. O PIN fica responsável por proteger o dispositivo contra acessos não autorizados. Fica a critério do usuário escolher e modificar o PIN. Na implementação realizada, se o PIN não é digitado corretamente em três tentativas o *token* é bloqueado por 5 minutos. Com mais três erros, todo o conteúdo da memória (PIN e chave) do microcontrolador é apagado
Programa de OTP	Responsável por executar o algoritmo de criptografia no modo de geração de uma OTP
Programa de Autenticação	Responsável por executar o algoritmo de criptografia no modo de Autenticação de Transações

Tabela 17 - Programas do *Token*

O apêndice B fornece uma descrição mais detalhada das funções empregadas no projeto e o código fonte está disponibilizado em http://www.larc.usp.br/~gymv/.

4.5. Atualização do dispositivo

Antes que um algoritmo de criptografia seja aceito como um padrão, o mesmo é amplamente avaliado com o fim de verificar a sua resistência a ataques conhecidos pela comunidade de segurança. Entretanto, essa avaliação muda com o tempo, pois novos ataques podem comprometer a segurança desse algoritmo ou então o

aumento do poder computacional de processadores torna viável a quebra por força-bruta. Por essa razão, aumenta-se o tamanho das chaves de segurança ou ainda novos padrões são adotados.

Não são previstos pelo projeto procedimentos para permitir a atualização em massa dos dispositivos, e isso também não é possível pela simples conexão do *token* ao computador, pois o sistema foi projetado exatamente para ser isolado de um sistema que possa estar comprometido.

Apesar disso, o acréscimo dessa característica de possibilidade de atualização é simples, do ponto de vista do *hardware* do dispositivo. Para isso, o projeto final precisa apenas manter a interface ICSP, permitindo assim que o *firmware* do sistema possa ser atualizado. É preciso enfatizar que tal procedimento não poderia ser realizado por usuários, dado que a reprogramação exige equipamento específico para a função.

4.6. Modificação para várias entidades

Como citado no capítulo anterior, em uma situação em que muitas entidades adotam o sistema de autenticação baseada em objeto, um usuário comum necessitaria trazer consigo dispositivos para diferentes sistemas.

Se essas entidades concordarem em utilizar o mesmo dispositivo, o que diferenciaria uma entidade da outra seria a chave de criptografia. Do ponto de vista de interface, as seguintes modificações são necessárias:

- Por se tratar de um *display* é numérico, é preciso que o usuário digite um código associado à entidade (exemplo: número do banco) para a seleção da chave correta.

Do ponto de vista de programação, as seguintes modificações são necessárias:

- Adicionar uma seção para solicitar que o usuário digite o código da entidade para que o dispositivo escolha a chave correta.
- As chaves são armazenadas em um vetor na RAM. O espaço pode ser facilmente aumentado para armazenar um número de identificação e a chave

correspondente. O microcontrolador adotado ainda tem espaço suficiente para pelo menos 8 chaves de 128 bits. Versões mais novas do microcontrolador já possuem mais memória disponível para essa função.

Como impacto dessas mudanças do ponto de vista de segurança, tem-se o seguinte quadro:

- Uma entidade ainda continua sem ter acesso à chave de outra entidade.
- Um ataque possível seria uma instituição induzir o usuário a colocar uma chave inválida para outra instituição, porém isso não forneceria acesso aos dados da instituição atacada.
- A escolha eventual de uma instituição incorreta seria detectada pelo código de integridade anexado a cada OTP.
- Um dispositivo associado a múltiplas instituições torna-se mais interessante de ser roubado, porém o mesmo ainda é protegido por um PIN e está programado para apagar os dados após 6 tentativas inválidas.

4.7. FIPS 140-2

O FIPS 140-2 (NIST, 2001) é um documento que apresenta requisitos de segurança para módulos de criptografia de forma a classificá-los em diferentes níveis de segurança. Os níveis de segurança variam entre 1 e 4, sendo o nível 4 o mais alto. Na época da redação dessa dissertação o FIPS 140-3 ainda é considerado um *draft*.

Essa seção realiza uma comparação entre o dispositivo apresentado neste capítulo e o FIPS 140-2. O projeto do dispositivo apresentado neste capítulo não visou atender ao FIPS 140-2, porém já que eles estão relacionados é interessante fazer a comparação. Abaixo estão listados os requisitos do FIPS 140-2 e o nível de segurança alcançado pelo dispositivo desenvolvido:

- Especificação do módulo criptográfico: Requisito é atendido pela documentação fornecida nesse capítulo e nos apêndices desse documento. Também é utilizado um algoritmo aprovado pelo NIST (AES) em um modo de operação também aprovado (CMAC).

- Portas e interfaces: Atende aos requisitos de nível 1 e 2, pois a entrada e saída de dados em aberto (Desafio), dados cifrados (Resposta) e chave de criptografia são feitas através de portas compartilhadas. Para os níveis 3 e 4 atende-se ao requisito que solicita que os dados devem ser fornecidos diretamente ao dispositivo.

- Papéis, serviços e autenticação: Requisito não atendido completamente, pois o tamanho do PIN é de 4 dígitos. Pode-se modificar facilmente para um PIN de 6 dígitos. O papel de usuário e gerente do módulo cabe apenas ao usuário do sistema.

- Modelo de Máquina de Estados: Todos os estados são detalhados na documentação no apêndice B, porém não foi implementado um estado de auto-teste e de erro no auto-teste. A inclusão desses estados exigiria pouco esforço adicional.

- Proteção Física: não foi levado em consideração nenhum tipo de proteção física para o *token*, o que atende ao nível 1. Para atender ao nível 2 seria preciso incluir uma proteção no chip que deixe claro qualquer tipo de tentativa de adulteração do processador (*tamper-evident*) e no gabinete.

- Ambiente operacional: refere-se à gerência do *hardware* e do *software*. O requisito não é atendido pois o dispositivo não faz uma verificação de integridade da memória de programa e de dados. Não foi estudada uma forma de atender a esse requisito.

- Gerência de chaves: são satisfeitos os requisitos de nível 1 e 2, segundo os quais a chave de criptografia deve ser digitada pelo usuário em um formato não criptografado. O método para zerar o *token* para o seu estado inicial é removendo a bateria do sistema. A chave de criptografia é armazenada em formato não criptografado.

- Interferência eletromagnética: Por tratar-se de um protótipo, houve pouca preocupação com a interferência eletromagnética.

- Auto-Teste: o dispositivo não implementa os autotestes sugeridos no FIPS 140-2. Os testes incluem teste do algoritmo de criptografia, teste de integridade

do *software* e teste de funções críticas. A implementação do primeiro teste é simples, porém os dois últimos são mais difíceis de serem implementados no sistema atual.

- Garantia de projeto: Diz respeito às boas práticas durante o projeto, desenvolvimento, posicionamento e operação do módulo criptográfico, garantindo que o módulo foi adequadamente testado, configurado, entregue, instalado e desenvolvido, e também que documentação ao operador é fornecida. Esse trabalho provê toda a documentação até o nível 2. Um manual do usuário, especificação dos algoritmos e diagrama lógico estão incluídos no apêndice C dessa dissertação. O projeto ainda chega ao nível 3, pois foi desenvolvido em linguagem de alto nível (C).

- Suavização de outros ataques: Tratam-se de cuidados com ataques de análise de energia (*power analysis*), análise tempo (*time analysis*), forçar estados de erro (*fault induction*) e detecção e obtenção de sinais eletromagnéticos emitidos (TEMPEST). Não houve preocupação com esse requisito no desenvolvimento do dispositivo.

4.8. Sumário

Esse capítulo apresentou a implementação de um dispositivo de autenticação de baixo custo e baseado em algoritmos de código aberto.

A implementação completa do sistema incluiu:

- Projeto de circuito elétrico
- Projeto de placa de circuito impresso
- Algoritmo de criptografia
- Programas para gerência de *hardware* e *software* de criptografia

O capítulo também faz análise da possibilidade do acréscimo de algumas características que não estão disponíveis, mas eventualmente podem se tornar desejáveis no dispositivo:

- Atualização do software executado pelo dispositivo

- Uso por mais de uma entidade
- Comparação dos requisitos do FIPS 140-2

5. VERIFICAÇÃO DE USO DO DISPOSITIVO

Este capítulo tem o objetivo de verificar a aderência do projeto do dispositivo realizado frente aos requisitos originais destacando-se os seguintes aspectos:

- Interface de uso
- Consumo de energia
- Aleatoriedade das OTP
- Desempenho
- Custo
- Transparência do código do dispositivo
- Integração com uma aplicação *web*

Sempre que possível e quando se encontravam disponíveis dados sobre dispositivos de uso já consagrados no mercado, estes foram utilizados como um "*benchmark*" de comparação com o intuito de posicionar o resultado do projeto alvo desta dissertação frente aos dispositivos já existentes.

5.1. *Benchmark*: *Token* de comparação

O dispositivo empregado para comparação[38] conta com dois tipos de OTP. O primeiro tipo é baseado em tempo e o segundo tipo é uma combinação de OTP baseada em desafios temporais com desafio-resposta. O mesmo ainda apresenta a funcionalidade de autenticação de transações.

As OTP temporais possuem uma janela de 36 segundos e cada OTP é representada por um número de 6 dígitos variando entre 0 e 999.999. Não foi possível encontrar uma documentação que indicasse qual a tolerância do dispositivo para desvios do relógio.

No caso da OTP temporal/desafio-resposta para um mesmo desafio, a mesma resposta é obtida desde que a mesma seja calculada dentro da mesma janela de

[38] *Token* Vasco modelo Digipass 250

252 segundos (7 x 36 segundos). No modelo analisado, desafios possuem 6 dígitos (sendo um dígito de verificação) e a resposta um número de 7 dígitos. Dentro de uma mesma janela de tempo, todas as respostas se iniciam com o mesmo dígito e esse dígito é incrementado em 1 quando a janela de tempo é alterada, por isso, considerou-se uma resposta com 6 dígitos.

O *token* utilizado para as análises é um modelo real e em utilização e por esta razão esse trabalho não divulgará os números referentes às OTP geradas por uma questão de sigilo e privacidade.

5.2. Interface de uso

Levando-se em conta a limitação imposta pelas características restritivas do *display* do dispositivo projetado, procurou-se definir a interface de uso do mesmo de forma intuitiva e coerente.

O dispositivo de autenticação tem a sua interface semelhante à de uma calculadora, porém apenas com o teclado numérico. O apêndice C apresenta o "Manual do usuário" do dispositivo com um esboço de como se encontra estruturada sua interface.

Resumidamente, ao ligar o dispositivo o usuário é levado a uma tela com a palavra "ESCOLHA". Nessa tela o usuário deve pressionar uma das teclas, identificáveis através de rótulos nas teclas do aparelho, para executar a função desejada.

Na entrada de dados, as teclas têm o comportamento de um teclado numérico e a tecla "Entra" tem a função de confirmar a operação corrente. A tecla "Cancela/Limpa" desempenha duas funções. Quando a tela está limpa, a função corrente é cancelada, e quando a tela possui dados digitados, a tecla limpa o conteúdo da tela, sem apagar eventuais dados anteriores.

Para a entrada de dados, uma tela com marcadores '⌋' e '⌊' é exibida para o usuário, indicando início e fim da entrada de dados. Esse mesmo comportamento é utilizado para receber dados de entrada do usuário, mantendo consistência entre todas as telas de entrada de dados.

5.2.1. Dispositivo de *benchmark*

O *token* de comparação possui interface semelhante ao modelo desenvolvido, porém apresenta apenas uma tecla de controle além das teclas numéricas.

Ao ligar o dispositivo, o usuário é levado para uma tela onde está escrito "APPLI". Nessa tela o usuário deve escolher qual operação executar. Não há indicações no aparelho de qual função é executada dependendo da tecla pressionada.

Independente da função escolhida, quando os dados são digitados, o aparelho inicia o processamento solicitado imediatamente.

A tecla de controle apresenta as seguintes funcionalidades:

- A tecla é utilizada para ligar e desligar o aparelho
- Na operação de desafio/resposta a tecla cancela a função atual
- Na operação de autenticação a tecla é utilizada para finalizar a entrada de dados

	***Token* Dissertação**	***Token* Comparativo**	**Comentários**
Teclado	Numérico	Numérico	Teclado simples e já conhecido pelo usuário.
Tecla de controle	2 (Entra e Cancela)	1 (<)	No *token* dissertação é possível corrigir dados, porém o usuário deve pressionar mais teclas para finalizar uma operação. No *token* comparativo a mesma tecla tem diferentes funções dependendo da aplicação.
Display	Numérico	Numérico	*Display* muito simples; usuário casual pode depender de ajuda do computador para entender o funcionamento.
Gabinete	Possui indicações visuais das funcionalidades de cada tecla	Apenas a indicação dos dígitos	No *token* dissertação usuário consegue se lembrar das funcionalidades mais rapidamente.
Entrada de dados	Em grupos de 3 ou 4 dígitos. Tamanho do grupo e quantidade de grupos depende da operação.	Tudo ao mesmo tempo. Na autenticação são grupos de 9 dígitos.	O *token* dissertação obriga o usuário a digitar 'entra' a cada grupo enquanto no outro dispositivo o grupo é aceito imediatamente. Apesar disso, o usuário ainda tem uma chance de ler o que foi digitado antes de confirmar alguma operação.

Tabela 18 - Comparação de interfaces

A interface de ambos é muito semelhante, porém a Tabela 18 faz a comparação das principais diferenças entre os dois sistemas.

Não foi um objetivo deste trabalho fazer um teste de campo preciso e estatisticamente correto para avaliar a facilidade da interface de uso do dispositivo. A comparação de interface de uso com um dispositivo de mercado serve simplesmente para ilustrar a similaridade e a simplicidade/complexidade do dispositivo desenvolvido nesta dissertação.

5.3. Consumo de energia

Todas as configurações do *token* (basicamente PIN e chave de criptografia) ficam armazenadas na memória RAM do microcontrolador. Por essa razão, o dispositivo deve ser continuamente alimentado por uma bateria[39]. O dispositivo foi projetado para funcionar por pelo menos um ano, admitindo-se, como base de cálculo um uso médio em todos os dias úteis por 1 minuto. Para garantir esse tempo, foi feita medição do consumo do dispositivo e foi escolhida uma bateria que pudesse alimentar o dispositivo nesse período.

Para medir o consumo de corrente, o dispositivo foi ligado a uma fonte de tensão continua de 3V e ligado um amperímetro conforme a Figura 13:

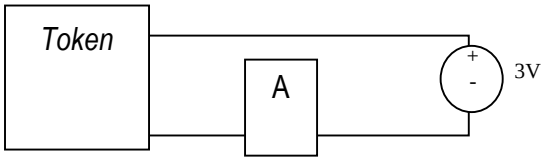


Figura 13 - Medição de consumo de energia

A medição foi feita nas seguintes situações: (i) dispositivo se encontra em *stand-by*, isto é, microcontrolador e periféricos desligados e alimentação apenas na memória, (ii) dispositivo ligado rodando a 250KHz e (iii) dispositivo rodando a 1MHz, isto é, rodando algoritmos de autenticação. Os resultados das medições se encontram na Tabela 19.

[39] O gabinete do sistema poderia aproveitar essa característica e sempre que o *token* for aberto, ele pode forçar a saída da bateria apagando o conteúdo da memória caso o sistema seja violado. Essa característica é interessante para atender parte do requisito "Proteção Física" de nível 3 do FIPS140-2, porém não foi estudado maiores detalhes de uma implementação desse tipo.

Estado	Consumo
Stand-by	1,0 μA
250 KHz (Uso típico)	0,33 mA
1 MHz (Autenticação)	0,67 mA

Tabela 19 - Consumo de energia

Admitindo-se um uso apenas em dias úteis por 1 minuto, em um ano, o sistema deve operar a 250 KHz por:

$$(1\min/dia \times (5 dias/semana \times 52 semanas)) \times \frac{1h}{60\min} \cong 4{,}33 horas$$

Durante o uso, considerou-se que ele fica 2 segundos operando a 1MHz:

$$(2\,seg/dia \times (5\,dias/semana \times 52 semanas)) \times \frac{1h}{3600 seg} \cong 0{,}15 horas$$

No resto do tempo, o sistema fica em *stand-by*:

$$(365 \times 24) - (4{,}33 + 0{,}14) \cong 8756 horas$$

Para obter o consumo do dispositivo em mAh, faz-se:

(0,33mA x 4,34h) + (0,67mA x 0,15h) + (0,001mA x 8756h) =

= 1,5mAh + 0,1mAh + 8,8mAh = **10,4mAh**

Uma bateria comum do tipo "botão", modelo CR2025, tem capacidade variando entre 150mAh e 170mAh, segundo especificação fornecida pelos fabricantes[40].

Como o consumo do dispositivo é bem inferior ao que a bateria pode fornecer e os resultados ainda foram arredondados para cima, garante-se operação do sistema por pelo menos 5 anos, dando ainda espaço para alguma eventualidade não prevista no uso típico do sistema.

Para o dispositivo de comparação, o fabricante informa que a sua bateria deve durar por aproximadamente 5 anos.

Com base no cálculo acima, percebe-se que o principal fator de consumo é o tempo em *standy-by*. O que garante o baixo consumo nesse período são os resistores de

[40] Foram analisadas as baterias das marcas Maxell, Sanyo e Renata.

1MΩ referentes ao circuito do *display*. Alterar o valor desses resistores pode aumentar o brilho do *display* mas também reduzir a vida útil da bateria.

A premissa de uso de um minuto diário é também um valor superestimado. O uso típico em uma situação de autenticação simples envolve uma operação entre 15 e 20 segundos de uso.

Com os dados obtidos pela medição de energia e pelo perfil de uso típico de um dispositivo dessa natureza, verificou-se que o tempo de uso de uma bateria é superior a 5 anos, valor este compatível com o uso típico do dispositivo e similar a demais dispositivos existentes no mercado.

A duração de uma bateria é importante, pois da forma como foi construído, o dispositivo perde as suas configurações quando reiniciado. Para evitar custos com distribuição de novas chaves, o dispositivo deve ser capaz de armazená-la por um longo período.

5.4. Característica estatística das OTP

Uma das formas de verificar a segurança do sistema é avaliar se as respostas dos desafios quando analisadas como um conjunto, seguem uma distribuição uniforme, isto é, a partir de respostas anteriores é inviável descobrir qualquer informação sobre a chave de criptografia ou qualquer relação entre respostas anteriores ou futuras.

A verificação dessa característica foi feita aplicando-se um teste que mostra, estatisticamente, se um dado conjunto de OTP distintas para uma mesma chave segue uma distribuição uniforme discreta. O teste aplicado para essa verificação chama-se "teste de qui-quadrado" (MONTGOMERY;RUNGER, 2003).

Esse teste foi implementado em linguagem C[41] e foi aplicado para o *token* de comparação e para o *token* desenvolvido. O teste foi aplicado para ambos os casos, e no caso do dispositivo proposto nesta dissertação, foi possível efetuar o teste para múltiplas chaves e uma quantidade maior de desafios.

[41] O compilador empregado nesses testes foi o Microsoft Visual C++ 6.0.

5.4.1. Teste de Qui-quadrado

O teste de qui-quadrado é empregado quando não se conhece a distribuição de uma determinada amostra e deseja-se avaliar a hipótese de que uma dada distribuição se encaixa como modelo para descrever essa amostra.

O procedimento exige *n* amostras arranjadas em um histograma de *k* intervalos de classe. Chamando de O_i a freqüência de amostras observadas dentro do *i*-ésimo intervalo e E_i a freqüência esperada de amostras para o *i*-ésimo intervalo na distribuição testada, define-se a estatística do teste como:

$$X_0^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

Se a amostra segue a distribuição analisada, então X_0^2 tem, aproximadamente, uma distribuição qui-quadrado com *v* graus de liberdade. Para realizar o teste, define-se um nível de significância *α* definindo assim o erro do Tipo I, ou seja, a probabilidade de se errar ao afirmar que a amostra não segue a distribuição testada. No caso de uma distribuição uniforme, E_i=n/k. O valor de E_i deve ser maior que 5.

O teste aceita a hipótese como correta se $X_{\alpha,\nu}^2 > X_0^2$, para a qual os valores de $X_{\alpha,\nu}^2$ referem-se à distribuição qui-quadrado, cujos valores podem ser encontrados em (NIST/SEMATECH, 2006).

5.4.2. Dispositivo de comparação

Para o dispositivo de *benchmark*, o teste das OTP temporais foram manualmente geradas 104 OTP e o teste de qui-quadrado foi aplicado considerando-se dez intervalos de classe (k=10) e o grau de significância (*α*) adotado foi `0,050`, `0,025` e `0,010`.

A execução do teste indicou que a amostra analisada encontra-se uniformemente distribuída em todos os graus de significância adotados.

Para o teste das OTP desafio/resposta, foram gerados 9 desafios dentro de uma mesma janela de tempo e os mesmos desafios foram reutilizados em janelas de

tempo diferente. Foram geradas 186 OTP e o teste aplicado utilizou como parâmetros k=10 intervalos de classe e com grau de significância *α* de `0,050`, `0,025` e `0,010`.

A execução do teste indicou que a amostra analisada se encontra uniformemente distribuída em todos os graus de significância adotados.

5.4.3. Dispositivo projetado

Para o dispositivo da dissertação é possível ter acesso ao algoritmo executado pelo mesmo e por isso o teste foi aplicado sobre diferentes amostras. As amostras foram geradas sorteando-se 100 mil chaves AES distintas e para cada uma das chaves testadas foram gerados 1000 desafios[42].

Como o teste é realizado para um universo de 100 mil amostras, construiu-se a Tabela 20 que resume os resultados dos testes. Ela apresenta a quantidade de amostras onde o teste indicou uma possível rejeição da hipótese de distribuição uniforme para diferentes valores de *k* e *α*. A partir dos dados de testes rejeitados, mostra-se entre parênteses a sua freqüência dentro do universo. Como os testes rejeitados estão dentro da faixa de erro *α* do teste considerou-se que o algoritmo empregado para geração OTP segue distribuição uniforme.

k \ α	0,050 (5,0%)	0,025 (2,5%)	0,010 (1%)
10	4921 (4,9%)	2419 (2,4%)	936 (0,9%)
20	5046 (5,0%)	2473 (2,5%)	979 (1,0%)
40	4780 (4,7%)	2450 (2,5%)	968 (1,0%)
50	4949 (4,9%)	2474 (2,5%)	991 (1,0%)

Tabela 20 - Resultados do teste de qui-quadrado

O teste de qui-quadrado para o dispositivo de *benchmark* foi aplicado para apenas duas amostras de tamanho reduzido se comparado com o teste aplicado para o dispositivo da dissertação. Estatisticamente, isso demonstra que para aquele caso, o dispositivo tem um comportamento dentro do esperado.

[42] A interpretação estatística do teste qui-quadrado é: Foram aplicados 100 mil testes diferentes. Cada teste apresenta uma população de tamanho 1.000.

Para o caso do dispositivo da dissertação, tem-se um caso raro no qual se tem acesso a diferentes populações. Por isso foi construída a tabela que demonstra que, dentro da margem de erro do teste, tem-se uma distribuição uniforme para os números gerados pelo algoritmo adotado.

Se fosse possível ter acesso ao código do *token* de comparação seria interessante fazer um confronto entre ambos no mesmo nível. Apesar disso, com os números obtidos até este ponto, pode-se afirmar que, estatisticamente, ambos apresentam distribuição uniforme para as suas OTP geradas.

O código fonte dos testes aplicados encontra-se disponível na internet no seguinte endereço: http://www.larc.usp.br/~gymv.

5.5. Desempenho

Para comparar o desempenho dos dispositivos, o tempo entre a solicitação da OTP e o tempo necessário para disponibilizá-la na tela do token foi medido. Foram tomadas 15 medidas e calculados o tempo médio e o desvio padrão dessa amostra. O tempo foi medido com auxílio de um cronômetro independente do *token*.

Para o *token* de comparação foi medido o tempo de geração de OTP baseada em tempo, OTP desafio/resposta e finalmente autenticação de transação.

Para o *token* da dissertação foi medido o tempo para geração de OTP desafio/resposta e tempo para efetuar autenticação.

	Modelo de *Benchmark*			*Token* Dissertação	
	OTP Temporal (segundos)	OTP Desafio/ Resposta (segundos)	Autenticação (segundos)	OTP Desafio/ Resposta (segundos)	Autenticação
Tempo Médio	1,14	1,14	1,18	1,52	1,34
Desvio Padrão	0,03	0,03	0,03	0,05	0,06

Tabela 21 - Comparação de desempenho

A diferença de tempo se justifica, pois o *token* desenvolvido executa o algoritmo MAC duas vezes. Na primeira execução, ele verifica se o desafio foi gerado por uma fonte válida e na segunda vez ele gera efetivamente a OTP. O *token* de *benchmark*

realiza uma verificação diferente, através da qual o desafio é considerado válido apenas através de um dígito de verificação.

O segundo ponto que deve ser ressaltado é que não foi possível ter acesso às especificações do processador do *token* de comparação e nem a sua freqüência de operação.

O *token* desenvolvido pode ter o desempenho aumentado modificando-se a execução do algoritmo AES. Cada chamada ao algoritmo executa o escalonamento da chave novamente, apesar de ser um valor constante para uma mesma chave. Outra possibilidade é modificar a velocidade do relógio interno do processador. No entanto, o tempo inferior a 2 segundos foi considerado aceitável para a aplicação em questão e ganhos adicionais não compensariam o maior consumo de memória e/ou energia.

5.6. Estimativa de custo de produção

O projeto considerou a produção de pelo menos 10.000 unidades para que os custos fixos com equipamentos de montagem, molde do gabinete e os fotolitos, sejam diluídos entre as unidades e não causem impacto no custo de um único dispositivo.

Quantidade	Componentes[43]	Custo (US$)
1	Microcontrolador[44]	2,87
4	Resistores	0,01
4	Capacitores	0,01
1	Placa de Circuito Impresso	0,50
1	*Display*	0,50
1	Gabinete	0,50
	Total	**4,39**

Tabela 22 - Custo do *Token*

[43] O preço do display e do gabinete foi estimado, com base no valor da calculadora utilizada para o protótipo.
[44] O preço desse componente vem baixando com o tempo e modelos semelhantes, porém mais baratos, já foram lançados desde a conclusão do primeiro protótipo. Valor obtido na página do fabricante.

A Tabela 22 apresenta o custo de cada componente necessário para se montar uma unidade do *token*. O custo final do sistema é influenciado principalmente pelo preço do microcontrolador ficando em menos de US$ 4,50 por unidade:

Os valores unitários da tabela foram obtidos através das páginas dos fabricantes ou estimados tomando como base dispositivos similares, todos considerados em pequenas quantidades. Isso contextualiza esta estimativa de custo como bastante conservadora.

Uma instituição financeira pode arcar com os custos ou repassá-los de forma suave para o cliente, pois a adoção de um *token* não deve ser vista como um custo adicional, mas como um investimento já que perdas por conta de ataques como *phishing*/*scam* devem se tornar menos freqüentes com a adoção gradual desse dispositivo.

Este levantamento inicial de custos, baseado em quantidades bem conservadoras, demonstra que mesmo com a inclusão de uma interface humano-computador mais adequada o custo final do dispositivo ficaria numa faixa bastante adequada demonstrando viabilidade dos requisitos originais deste trabalho.

5.7. Transparência do código

Um dos principais aspectos deste projeto é manter o código utilizado aberto para que se possa fazer uma análise precisa e completamente transparente com o intuito de demonstrar que o programa possui todas as funcionalidades especificadas e, principalmente, não possui funcionalidades adicionais não especificadas. Como citado anteriormente, essa é uma característica altamente desejável em um dispositivo na qual a finalidade é prover um serviço de segurança.

Esta seção mostra como o código está estruturado de forma a facilitar a sua leitura. Além da estrutura descrita abaixo, o apêndice B apresenta a descrição de cada uma das funções especificadas e o apêndice D complementa com diagramas de caso de uso para o *software* desenvolvido para o dispositivo.

A transparência do código fonte se torna ainda mais importante quando se deseja implementar este *software* em outros dispositivos como *smartphones*. A seguir, é

descrita a estrutura do programa implementado e com o auxílio dos casos de uso descrito no apêndice D é possível fazer um "*walk through*" para determinar todas as funcionalidades existentes na implementação.

5.7.1. Camadas

Para facilitar a programação, a manutenção e eventuais análises do código fonte, esse sistema foi organizado na forma de camadas, exatamente como exibido na Figura 14.

<table>
<tr><td colspan="2">Interface com o usuário</td></tr>
<tr><td>Núcleo Autenticação</td><td>Programas de Suporte</td></tr>
<tr><td colspan="2">Infra-Estrutura (Drivers)</td></tr>
</table>

Figura 14 - Programação em camadas

Na camada de Interface, estão os programas que fazem interação direta com o usuário. Na camada de Suporte estão os programas que transformam as instruções mais complexas da Interface em instruções mais simples que podem ser realizadas pelos *drivers* do sistema. No Núcleo estão os programas que fazem efetivamente cálculos realizando chamadas aos algoritmos de criptografia. Na Infra-Estrutura estão os *drivers* do sistema, responsáveis por controlar os periféricos acoplados e os nativos do microcontrolador.

5.7.2. Interface

Na interface estão os programas responsáveis pela interação com o usuário. A Figura 15 mostra pilha de execução dos programas. Programas em um mesmo nível não estão na pilha de execução simultaneamente. Quanto mais alto o nível de um programa, mais cedo ele é executado.

Todos os programas utilizam as mesmas rotinas de entrada de dados do usuário, para manter a consistência entre os mesmos.

Para o caso de OTP e autenticação, depois que a interface é executada, o controle é passado para o respectivo Programa do Núcleo. Para Chave e PIN são necessárias chamadas aos Programas de Suporte.

Ao fim da execução todos retornam ao Menu "ESCOLHA". Paralelamente e independente da vontade do usuário, existe um temporizador que desliga o sistema depois de algum tempo de inatividade (mais detalhas na camada de Infra-Estrutura).

Para executar suas funções os programas de Interface fazem chamadas aos Programas de Suporte e Infra-Estrutura. Esses não estão listados no esquema da Figura 15.

1	Entra PIN			
2	Menu Escolha			
3	OTP	Autenticação	Troca de Chave	Troca de PIN
4	Leitura de Dados em Blocos			
5	Leitura de Dados: Monta campo para entrada de dados Leitura de *String*			
6	Núcleo		Programas de Suporte	

Figura 15 - Camada de Interface

5.7.3. Núcleo Autenticação

No núcleo estão os programas responsáveis por calcular OTP e Autenticação. A Figura 16 mostra a pilha de execução dos programas. Programas de Suporte e Infra-Estrutura não estão listados nessa figura.

1	Geração de OTP	Autenticação de transações
2	Rotina de geração de MAC	
3	CMAC	
4	AES-128	
5	Exibição de MAC	

Figura 16 - Camada Núcleo Autenticação

Devido ao encapsulamento, se necessário, a troca por um novo algoritmo de criptografia diferente pode ser realizada facilmente.

5.7.4. Programas de Suporte

Os Programas de Suporte são utilizados pela Interface para ler, apresentar e gravar dados ao usuário. Por serem executados dentro dos programas de interface e de Núcleo, abaixo estão listados os tipos de programas bem como suas principais

funções. Para executar suas funções Programas de Suporte fazem chamadas aos Programas de Infra-Estrutura.

Teclado:

- Aguardar uma tecla
- Leitura de *strings*
- Aguardar uma tecla ou espera um certo tempo (também utiliza o *driver* de temporização)

Display:

- Exibe *string*
- Exibe *string* fazendo deslocar pelo *display*
- Limpa *display*
- Imprime caractere
- Envio de comandos

EEPROM:

- Gravação de dados
- Leitura de dados
- Limpeza de memória

Outros:

- Conversão de decimal (*string*) para hexadecimal (representação binária)
- Conversão de hexadecimal (representação binária) para decimal (*string*)

Os programas classificados como "outros" são utilizados pelas rotinas Núcleo e pelo Programa de Interface de Troca de Chaves.

5.7.5. Infra-Estrutura

Os Programas da Infra-Estrutura normalmente são acionados apenas pelos Programas de Suporte. Somente quando a função é bem simples os Programas de

Interface acionam os Programas da Infra-Estrutura; esse é o caso dos *drivers* de Controle de velocidade.

Drivers de Inicialização e Temporização são executados independente da vontade do usuário. Abaixo estão descritas as principais funções realizadas pelos *drivers* do sistema.

Teclado:

- Inicialização do teclado
- Leitura de teclas

Display:

- Inicializa *display*
- Controle de registradores do *display*
- ROM de caracteres

EEPROM:

- Leitura de byte na ROM
- Escrita de byte na ROM

Temporização:

- Interrupção
- Detecção de passagem de tempo

Energia:

- Verifica capacidade de bateria
- Modo de economia de energia (*stand-by*)
- Controle de entrada no modo de economia de energia

PIC:

- Inicializa do microcontrolador
- Verificação de primeiro uso do sistema
- Controle de velocidade

Esta seção mostra como está organizado todo o código fonte do sistema. A sua leitura não deve ser feita de forma individual, mas em conjunto com os apêndices B e D dessa dissertação, para prover uma visualização do sistema como um todo.

5.8. Integração com uma aplicação

Aqui é ilustrado o dispositivo de segurança integrado a um ambiente *web* no qual, originalmente, a identidade é baseada apenas em conhecimento.

O sistema foi acoplado ao sistema de autenticação do ambiente de aprendizado eletrônico Tidia-Ae (Tecnologia da Informação para o Desenvolvimento da Internet Avançada - Aprendizagem Eletrônica), projeto com o objetivo de desenvolver ferramentas para auxiliar no aprendizado eletrônico.

Apesar do sistema escolhido nesta integração inicial não demandar requisitos críticos de segurança, a mesma é adequada para um teste de validação pois possui estrutura de autenticação similar àquela adotada por sistemas que necessitam de segurança crítica.

O sistema atual de autenticação do Tidia-Ae recebe um nome de usuário e senha e verifica se o nome de usuário é válido. Uma vez feito isso, o sistema verifica qual é o contexto[45] inicial ao qual o usuário deve ser associado e então qual o seu papel[46] dentro do contexto. Após essa verificação, gera-se um número de identificação (tíquete) associando a sessão criada no servidor ao computador do usuário.

Apesar de atualmente o Tidia-Ae não ser um sistema que exige o nível de sofisticação de autenticação apresentado nesse trabalho, o mesmo pode ser aproveitado quando na autenticação de usuários com acesso a papéis com níveis de privilégio mais elevados como, por exemplo, o papel de administrador. Esta situação caracteriza uma aplicação típica, do ponto de vista de integração do *software* e da própria experiência de uso do dispositivo de segurança.

[45] Contexto é um ambiente que reúne as ferramentas de aprendizagem e os seus usuários. Um contexto pode ser equivalente virtual de uma disciplina, um departamento ou mesmo uma universidade.
[46] O papel representa as permissões que os usuários têm dentro do seu contexto.

5.8.1. Tecnologia empregada no Tidia-Ae

O servidor do Tidia-Ae utiliza o JBoss (STARK; FLEURY;RICHARDS, 2005) como servidor de aplicação para gerar páginas dinâmicas e o Hibernate[47] (BAUER;KING, 2006) para fazer interface com o banco de dados. Isso significa que para compreender as mudanças executadas no processo de autenticação é aconselhável que o leitor entenda como são geradas páginas que utilizam Sevlets e JSP.

A seguir, é apresentada uma explicação superficial do funcionamento desse sistema. Uma explicação mais detalhada está fora do escopo deste documento. Um leitor com o conhecimento dessa tecnologia pode ir diretamente para a seção seguinte.

O conceito de Servlets surgiu quando se começou a pensar no uso da tecnologia Java em servidores surgiu. Servlets são códigos em Java que basicamente possuem instruções do tipo "println" e que, quando executados pela JVM (*Java Virtual Machine*), geram dinamicamente uma página html que é enviada ao cliente.

Esse tipo de abordagem torna difícil a manutenção dos arquivos fontes pois o código em Java fica misturado com o código html em um único arquivo. Em uma situação mais adequada deve-se ter um código html editado apenas pelo *web designer* e um código Java mantido apenas pelo programador.

Para amenizar esse problema surgiu o JSP (JavaServer Pages). Essa abordagem introduz uma separação maior entre o código em Java e o html. Para tornar isso possível, essa solução introduziu um novo elemento conhecido pelo nome de servidor de aplicação.

Posto em poucas palavras, o servidor de aplicação transforma uma página JSP em um Servlet e esse então é executado seguindo todo o fluxo já descrito anteriormente. A Figura 17 ilustra o funcionamento de JSP.

[47] Resumidamente, o Hibernate é uma API que converte qualquer chamada à base de dados em comandos SQL reconhecíveis por qualquer base de dados do mercado (MS-SQL, Oracle, MySQL, PostGreSQL, etc)

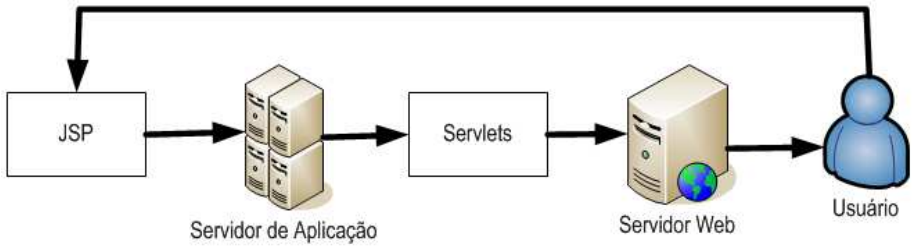


Figura 17 - Diagrama simplificado de geração de páginas com JSP

O JSP ainda mantém rótulos (*tags*) de programação em suas páginas e por isso a separação entre a camada de apresentação (html) e a camada de aplicação (Java) ainda não é completa. A separação aconteceu com o uso de Struts.

Struts (ROUGHLEY, 2006) é um *framework* baseado no conceito de MVC (*Model, View, Controller*). *Model* se refere à camada de aplicação, que é o que provê inteligência às páginas. A representação do *Model* é chamada de Ação (*Action*). *View* diz respeito à camada de apresentação. *Controller* faz a ligação entre a camada de aplicação e a camada de apresentação.

O resultado do processamento feito pelas ações (como, por exemplo, o acesso a uma base de dados) é enviado para uma JSP que possui entradas específicas para colocar o resultado fornecido pelas ações. A página JSP é então enviada para o servidor de aplicação que por sua vez converte a JSP em um Servlet e o resultado do processamento do Servlet é então enviado para o usuário. A Figura 18 ilustra essas etapas.

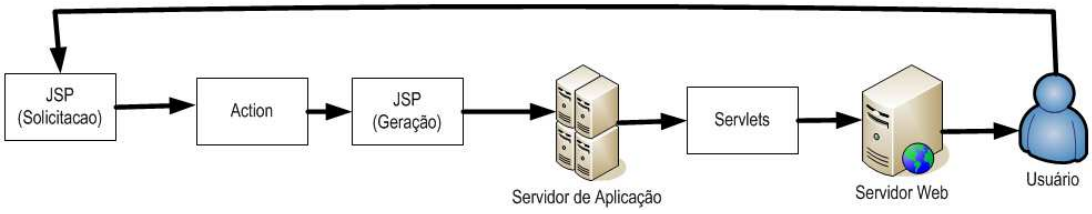


Figura 18 - Diagrama simplificado da Geração de páginas com uso de Struts

5.8.2. Cenário atual

O código que responde pelo processo de autenticação do Tidia-Ae é bem simples e, assim como grande parte dos sistemas *on-line* atuais, utiliza autenticação baseada em conhecimento.

A página de autenticação é um JSP com um formulário no qual um usuário fornece seu nome de usuário e senha. Os dados desse formulário são enviados ao servidor para uma ação de autenticação (AuthenticateAction). Essa ação verifica junto à base de dados o nome de usuário e senha digitados. Em caso de erro, o usuário é notificado e redirecionado para a página de autenticação novamente.

Com a identidade validada, é gerado um tíquete baseado no nome de usuário, senha e IP do usuário. Esse ticket é repassado para uma nova ação (AccessContextAction) com a função de associar o usuário ao seu contexto e papel iniciais.

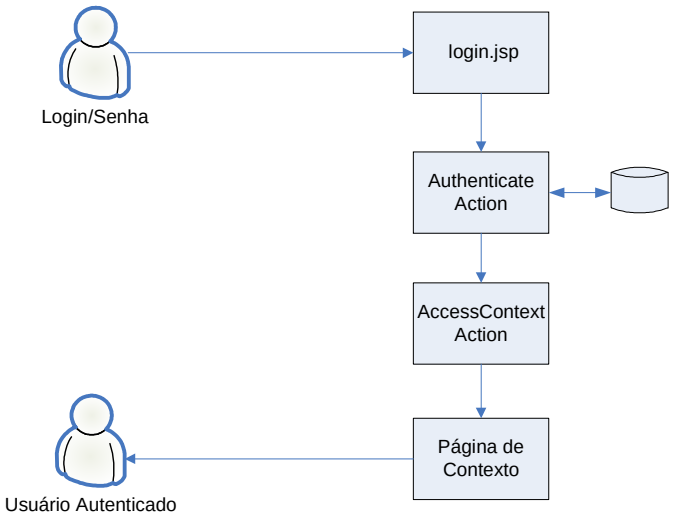


Figura 19 - Autenticação do Tidia-Ae

Essa ação gera um novo tíquete enviado para o usuário e utilizado como um autenticador da sessão atual. A Figura 19 ilustra o processo de funcionamento da rotina de autenticação do Tidia-Ae.

5.8.3. Cenário proposto

A modificação proposta para integração do dispositivo de autenticação no sistema Tidia-Ae altera a ação AuthenticateAction. Ao invés de processar a ação AccessContext, o servidor continua processando AuthenticateAction que também fica responsável por buscar no banco de dados a chave de criptografia do usuário, criar um desafio e gerar o código de integridade desse desafio.

O código de integridade é gerado por uma classe chamada StrongAuth. Essa classe possui métodos para gerar o MAC de um desafio, calcular a resposta do desafio e efetuar autenticação (apesar de suportar autenticação, ela não é tratada nessa implementação).

O MAC de StrongAuth é gerado pela classe OMAC, código disponibilizado em (BARRETO, 2004). Com o desafio preparado, AuthenticateAction armazena tanto o desafio e a resposta do desafio na seção do usuário.

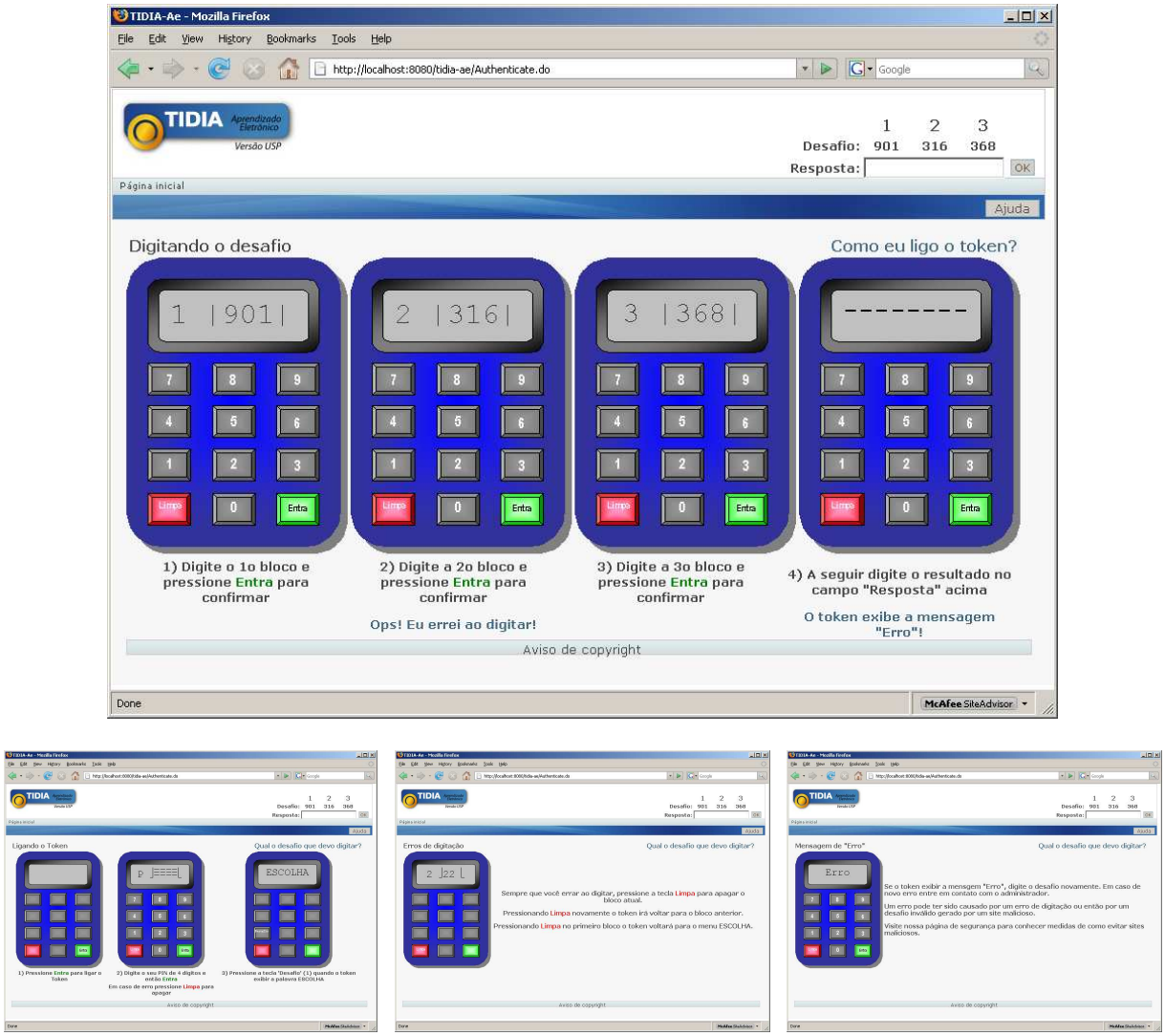


Figura 20 - Telas solicitando OTP ao usuário e demais telas de suporte

A etapa seguinte é feita por uma página JSP (*Login*OTP.jsp) que exibe o desafio ao usuário. A página, ilustrada na Figura 20, solicita o desafio ao usuário e procura fornecer informações sobre os passos que o usuário deve seguir para utilizar o seu dispositivo com sucesso. São basicamente quatro telas. A tela principal exibe o

desafio e solicita a sua resposta. As outras três telas são utilizadas por quem não ainda teve muito contato com o dispositivo. São instruções que mostram como ligá-lo e como proceder em caso de erro. A terceira tela ainda mostra a seguinte mensagem: "Visite nossa página de segurança para conhecer medidas de como evitar sites maliciosos". Apesar deste projeto não possuir essa página, em um sistema efetivamente em produção ela é importante por procurar casar o uso do dispositivo com práticas saudáveis de uso da internet.

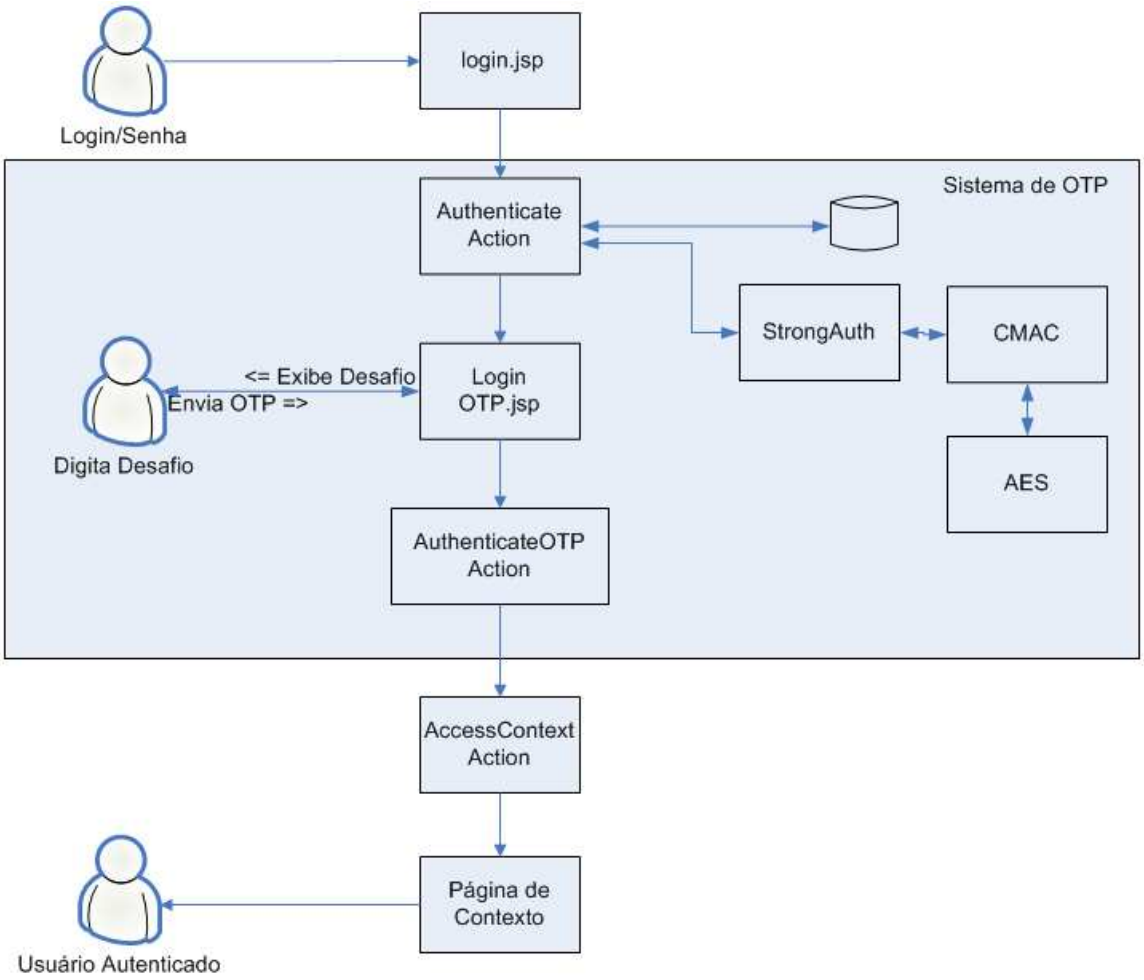


Figura 21 - Autenticação do Tidia-Ae com o uso de OTP

De forma semelhante à primeira autenticação, os dados do novo formulário são enviados para uma ação responsável por verificar se a OTP está correta (AuthenticateOTPAction). Um novo ticket baseado em nome de usuário, senha, OTP e IP do usuário é enviado para a ação AccessContext e o fluxo de informações prossegue como anteriormente.

A Figura 21 ilustra o ponto exato onde são feitas modificações para introdução da autenticação por OTP.

A integração com uma aplicação demonstrar as possibilidades de uso do dispositivo. A modificação de um sistema real mostra que um sistema tradicional pode ser modificado e que o escopo das modificações fica restrito à etapa de autenticação.

5.9. Sumário

Esse capítulo procurou demonstrar a adequação do *token* em diferentes requisitos para verificar o uso em diferentes situações. Com um dispositivo de comparação, confrontou-se aspectos referentes à interface, consumo de energia, aleatoriedade de OTP e desempenho.

O capítulo ainda provê recursos para que se possa entender o código executado pelo sistema, assim como analisá-lo.

Encerra-se com a apresentação de um sistema que originalmente se baseava apenas em um único autenticador. Mostra-se como ele foi modificado para que pudesse integrar um novo autenticador em seu sistema de autenticação.

6. CONCLUSÕES

O emprego da internet para realizar operações financeiras está crescendo e tornando-se comum com o passar do tempo. Em paralelo a esse crescimento, surgiram ataques visando obter a identidade dos usuários nesse ambiente digital. Tais ataques tornam-se possíveis, pois uma pessoa não é reconhecida por sua presença, mas por senhas, método adotado pela maioria dos sistemas atuais para autenticar seus usuários.

Este trabalho apresenta como foco a autenticação. Com a definição de autenticação e seus principais mecanismos, foram introduzidas as vulnerabilidades dos sistemas nos quais a identidade digital de seus usuários é representada apenas com identificadores associados a segredos compartilhados apenas entre usuário e servidor (senhas).

Proprietários de aplicações com necessidade de maior segurança, passaram a sentir que devem complementar o processo de autenticação com a adoção de múltiplos autenticadores, como, por exemplo, aqueles baseados em objeto que utilizam mecanismos de desafio/resposta. Esses autenticadores combinados para mutuamente maximizar suas vantagens e minimizar suas vulnerabilidades formam o que é conhecido como "autenticação forte".

Apesar dessa abordagem, ela ainda não é perfeita e em muitos casos também existe uma necessidade de uma funcionalidade para assinatura de transação para prevenir inclusive ataques "*man-in-the-middle*".

Com esse embasamento, nesse trabalho projetou-se e implementou-se um dispositivo de baixo custo de produção capaz de permitir o emprego da autenticação forte e assinatura em sistemas que demandam tal requisito. O desenvolvimento realizado resultou em contribuições importantes por apresentar uma solução para os problemas que demandam um processo de autenticação forte e assinatura de transações.

6.1. Contribuições

São várias as contribuições geradas por este trabalho. Uma delas apresentou o estudo dos mecanismos de autenticação empregados para identificar um usuário. Para isso, este estudo definiu autenticadores baseados em conhecimento, apontou suas vulnerabilidades e apresentou algumas soluções adotadas a fim de tornar o emprego de senhas mais seguro. O estudo mostrou que trabalhar apenas com senhas não é suficiente para garantir a identidade em sistemas críticos, como aqueles utilizados por instituições financeiras. Em ambientes vulneráveis a *phishing* ou *spywares* é necessário adotar uma solução conjunta com múltiplos autenticadores. É nesse contexto em que se aplicam as contribuições deste trabalho.

A principal contribuição deste trabalho foi a implementação de uma alternativa ao modelo de autenticação atual. É implementada a autenticação forte através de autenticadores baseados em conhecimento e em objeto. O objeto é um dispositivo seguro e de baixo custo capaz de executar autenticação de usuário e de mensagens. O dispositivo de autenticação alcançou a meta desejada através do uso de componentes de prateleira como microcontroladores e também algoritmos criptográficos de código aberto para que todos possam conhecer exatamente o seu funcionamento. Qualquer pessoa pode avaliá-lo, identificar vulnerabilidades e gerar correções ou eventualmente propor novos algoritmos.

Além do baixo custo, o dispositivo demonstrou a viabilidade de implementação de *tokens* com o uso de componentes comuns sem o comprometimento do seu tempo de resposta sendo bastante aceitável para um usuário comum. O trabalho ainda compara dois algoritmos MAC dentro da plataforma adotada analisando consumo de memória e tempo de execução dos algoritmos, itens particularmente importantes em sistemas em que recursos como memória e energia são escassos.

O desenvolvimento do dispositivo também gerou códigos utilizáveis em aplicações diferentes da original. Sendo composto de *drivers* para periféricos tipicamente utilizados em microcontroladores e a reutilização em projetos que envolvam esse

tipo de dispositivo é perfeitamente viável. Isso contribui claramente para projetos que se apóiam em plataformas semelhantes.

Por fim, o uso do dispositivo foi verificado através de uma série de ensaios e estudos comparativos. O dispositivo foi ainda integrado a uma aplicação web que passou a incorporar o mecanismo de autenticação forte. A aplicação em questão tem como finalidade o ensino a distância, apesar de o esquema de autenticação utilizado é próximo ao adotado em sistemas financeiros.

6.2. Trabalhos futuros

Este trabalho não esgota o assunto e deixa lacunas para serem preenchidas por novos trabalhos na área de autenticação.

Uma possibilidade seria avaliar a viabilidade do emprego de outras classes de algoritmos utilizando o *hardware* atual. Poderia-se pesquisar o uso de algoritmos de criptografia assimétrica para garantir a irretratabilidade de transações.

O emprego de algoritmos de conhecimento zero também se mostra como alternativa interessante principalmente com a possibilidade de tornar independente o processo de identificação do processo de autenticação (LYSYANSKAYA, 2007).

O trabalho não valida a autenticação de transações, pois o sistema adotado não apresentava essa necessidade. Uma nova etapa seria empregar o *hardware* em um sistema com esse requisito.

Um *token* com teclado não é complicado de ser utilizado, já que é uma interface encontrada no cotidiano das pessoas. Todavia, autenticar mensagens mais longas pode se tornar incômodo para alguns usuários. Modificar a interface do sistema para empregar o conversor A/D do microcontrolador pode permitir o acoplamento de sensores capazes de "ler" uma tela ou um *led* piscando (como sugerido no texto) ou então de "escutar" um som que é emitido pelo telefone ou por uma página web facilitando a entrada dos dados necessários ao processo de assinaturas e verificação de integridade de transações eletrônicas.

7. REFERÊNCIAS

AMES, W. Understanding spyware: risk and response. IT Professional, v.6, n.5, p.25-29. 2004.

ANAKAN. **A Unique Approach to Eliminating Online Fraud**. Disponível em: <http://www.anakam.com/GoVirt.asp>. Anakan LLC, 2006. Acesso em: 25/Jul/2006.

BARBOSA, A. **Comércio eletrônico faturou R$ 2,5 bilhões em 2006** Disponível em: <http://www.estadao.com.br/tecnologia/internet/noticias/2006/fev/02/234.htm>. Estado de São Paulo, 2006. Acesso em: 19/Jun/2006.

BARRETO, P. **PCS5734 - Segurança em redes de computadores - Notas de Aula**. São Paulo 2005.

BARRETO, P. S. L. M. **Paulo Barreto's Crypto Page**. Disponível em: <http://paginas.terra.com.br/informatica/paulobarreto/>. 2004. Acesso em: 04/Jan/2007.

BAUER, C. e KING, G. **Java Persistence with Hibernate**: Manning Publications. 2006

BERGHEL, H. **Phishing mongers and posers**. Communications of the ACM. 49 2006.

BISHOP, M. **Computer Security: Art and Science**. Boston, MA: Addison-Wesley. 2003

CERT.BR. **Estatísticas dos Incidentes Reportados ao CERT.br**. Disponível em: <http://www.cert.br/stats/incidentes/>. CERT.br, 2007. Acesso em: 10/Jun/2007.

CETIC.BR. **TIC DOMICÍLIOS e USUÁRIOS 2006**. Disponível em: <http://www.cetic.br/usuarios/tic/2006/index.htm>. CGI.BR, 2007. Acesso em: 10/Jun/2007.

CHANDRAMOULI, R. e ROSE, S. Challenges in securing the domain name system. Security & Privacy Magazine, IEEE, v.4, n.1, p.84-87. 2006.

CLAESSENS, J., PRENEEL, B. e VANDEWALLE, J. Combining World Wide Web and wireless security. Informatica, p.123-132. 2002.

DIERKS, T. e ALLEN, C. **RFC2246: The TLS Protocol Version 1.0**: RFC Editor United States 1999.

DIETZ, P., YERAZUNIS, W. e LEIGH, D. Very Low-Cost Sensing and Communication Using Bidirectional LEDs. **International Conference on Ubiquitous Computing**. October 2003, 2003. p.

FELITTI, G. **Vírus para celulares dobram durante 2006, segundo análise da McAfee**. Disponível em: <http://idgnow.uol.com.br/seguranca/2006/12/21/idgnoticia.2006-12-21.4137157066>. IDG Now, 2006. Acesso em: 27/Jul/2007.

FERREIRA, A. B. D. H. **Novo Dicionário Aurélio da Língua Portuguesa**: Positivo. 2004. 2120 p.

FIPS197. Announcing the Advanced Emcryption Standard (AES). Nov 26, 2001, p.47. 2001.

GREGÓRIO, D. **Transações no site do BB passam de 56 mi por mês**. Disponível em: <http://info.abril.com.br/aberto/infonews/062006/20062006-21.shl>. InfoOnline, 2006. Acesso em: 20/Jun/2006.

HALLER, N. RFC1760: The S/KEY One-Time Password System. 1995.

HARRISON, W. e BOLLINGER, T. **User Confidence and the Software Developer**. Software, IEEE. 21**:** 5-8 p. 2004.

HIRAKAWA, A. R. e CUGNASCA, C. E. **Interface com teclado e display**: Escola Politécnica 2002.

HOLE, K. J., MOEN, V. e TJØSTHEIM, T. **Case Study: Online Banking Security**. Security & Privacy Magazine, IEEE. 4**:** 14-20 p. 2006.

HONG, J. I. **Minimizing security risks in ubicomp systems**. Computer. 38**:** 118-119 p. 2005.

HOUSLEY, R.*, et al.* **Internet X. 509 Public Key Infrastructure: Certificate and CRL Profile (RFC 2459)**. T. I. Society 1999.

IDG. **F-Secure alerta para 1ª praga que rouba dados de celulares**. Disponível em: <http://idgnow.uol.com.br/seguranca/2006/03/30/idgnoticia.2006-03-30.0073893494>. IDG Now, 2006. Acesso em: 10/Jun/2007.

INTEL. **LaGrande Technology Preliminary Architecture Specification**: Intel Corporation 2006.

IWATA, T. e KUROSAWA, K. **OMAC: One-Key CBC MAC**: Springer**:** 24-26 p. 2003.

KEEPASS. **KeePass Password Safe**. Disponível em: <http://keepass.info/>. 2007. Acesso em: 06/Jun/2007.

KIRDA, E. e KRUEGEL, C. Protecting users against phishing attacks with AntiPhish, 2005. 517-524 Vol. 2 p.

KNIGHT, W. **Caught in the net**. IEE Review. 51**:** 26-30 p. 2005.

KREBS, B. **Citibank Phish Spoofs 2-Factor Authentication**. Disponível em: <http://blog.washingtonpost.com/securityfix/2006/07/citibank_phish_spoofs_2factor_1.html>. The Washington Post, 2006. Acesso em: 25/Jul/2006.

KU, W.-C. **A hash-based strong-password authentication scheme without using smart cards**. ACM SIGOPS Operating Systems Review. 38 2004.

LAMPORT, L. Password authentication with insecure communication. Communications of the ACM, v.24, n.11, p.770-772. 1981.

LAWTON, G. E-mail authentication is here, but has it arrived yet? Computer, v.38, n.11, p.17-19. 2005.

LEAVITT, N. Instant messaging: a new target for hackers. Computer, v.38, n.7, p.20-23. 2005.

LEE, Y. e KOZAR, K. A. **Spyware: Investigating factors affecting the adoption of anti-spyware systems**. Communications of the ACM. 48**:** 72-77 p. 2005.

LINDA DAILEY, P. **News Briefs**. 38**:** 22-24 p. 2005.

LIU, W.*, et al.* Phishing Web page detection, 2005. 560-564 Vol. 2 p.

LYSYANSKAYA, A. Authentication without Identification. Security & Privacy Magazine, IEEE, v.5, n.3, p.69-71. 2007.

M'RAIHI, D.*, et al.* RFC4226 - HOTP: An HMAC-Based One-Time Password Algorithm. 2005.

MACKENZIE, P. **Passwords Will Not Die: How Cryptography Can Help Deal With Them**. RSA Conference 2006 2006.

MAO, W. **Modern Cryptography: Theory and Practice**. Upper Saddle River, New Jersey: Prentice Hall PTR. 2004. 648 p.

ME, G., PIRRO, D. e SARRECCHIA, R. A mobile based approach to strong authentication on Web. **Computing in the Global Information Technology, 2006. ICCGI '06. International Multi-Conference on**, 2006. 67-67 p.

MENEZES, A. J., OORSCHOT, P. C. V. e VANSTONE, S. A. **Handbook of Applied Cryptography**: CRC Press. 2001

MICROCHIP. **Getting Started - Power Considerations**: Microchip Technology Incorporated 2001.

MICROCHIP. **PIC18F6390/6490/8390/8490 Data Sheet**: Microchip Technology Incorporated 2004.

MONTGOMERY, D. C. e RUNGER, G. C. **Estatística aplicada e probabilidade para engenheiros**. Rio de Janeiro: LTC Editora. 2003

NEUMAN, B. C. e TS'O, T. Kerberos: an authentication service for computer networks. Communications Magazine, IEEE, v.32, n.9, p.33-38. 1994.

NIST. FIPS140-2 Security requirements for cryptographic modules. May 25, 2001. 2001.

NIST. FIPS198 - The Keyed-Hash Message Authentication Code (HMAC). March 6, 2002, p.20. 2002.

NIST. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. May, 2005. 2005.

NIST/SEMATECH. **e-Handbook of Statistical Methods**. Disponível em: <http://www.itl.nist.gov/div898/handbook/eda/section3/eda3674.htm>. NIST, 2006. Acesso em: May/28/2007.

O'GORMAN, L. Comparing passwords, tokens, and biometrics for user authentication. Proceedings of the IEEE, v.91, n.12, p.2021-2040. 2003.

OPPLIGER, R. **Contemporary Cryptography**. Norwood, MA: Artech House. 2005. 503 p. (Computer Security Series).

PARALLELS. **Virtual pc, virtual machine and multiple operating system solutions by Parallels**. Disponível em: <http://www.parallels.com/en/products/desktop/>. Parallels, Inc, 2007. Acesso em: 20/Jul/2007.

PATEL-PREDD, P. **Cybercrime At A Glance**. IEEE Spectrum 2006.

REINHOLD, A. G. **The Diceware Passphrase Home Page**. Disponível em: <http://world.std.com/~reinhold/diceware.html>. 2007. Acesso em: 27/jun/2007.

ROUGHLEY, I. **Starting Struts 2**: C4Media. 2006

SCHNEIER, B. **Customers, passwords, and Web sites**. Security & Privacy Magazine, IEEE. 2**:** 88 p. 2004.

SCHNEIER, B. **Two-factor authentication: too little, too late**. Communications of the ACM. 48 2005.

SINGH, S. **O Livro dos Códigos**: Record. 2001. 512 p.

SMITH, R. E. **Authentication: From Passwords to Public Keys**: Addison Wesley. 2002. 550 p.

STALLINGS, W. **Cryptography and Network Security: Principles and Practice**: Prentice Hall. 2003. 681 p.

STARK, S., FLEURY, M. e RICHARDS, N. **JBoss 4.0 - The Official Guide**: Sams Publishing. 2005. 648 p.

STEELE, S. **Accelerating to meet the challenge of embedded Java**: ARM Limited 2001.

STMICROELETRONICS. AN1048 - ST7 Software LCD Driver. 17/03/2005. 2005.

SUDHINDRA, S. e FIONA FUI-HOON, N. Web browsing and spyware intrusion. Communications of the ACM, v.48, n.8, p.85-90. 2005.

TERRA. **Descoberto vírus que se transfere do PC para o celular**. Disponível em: <http://tecnologia.terra.com.br/interna/0,,OI901779-EI4805,00.html>. Terra, 2006. Acesso em: 01/Jun/2006.

TERZIAN, F. **Sua identidade digital corre perigo**. Disponível em: <http://info.abril.com.br/aberto/infonews/082006/25082006-6.shl>. Exame, 2006. Acesso em: 26/Ago/2006.

TRUSTEDCOMPUTINGGROUP. **Trusted Computing Group**. Disponível em: <https://www.trustedcomputinggroup.org>. Trusted Computing Group, 2007. Acesso em: 27/Jul/2007.

TUOHEY, S. Two-Way Authenticationfor Member Online Access. **RSA Conference 2006**, 2006. p.

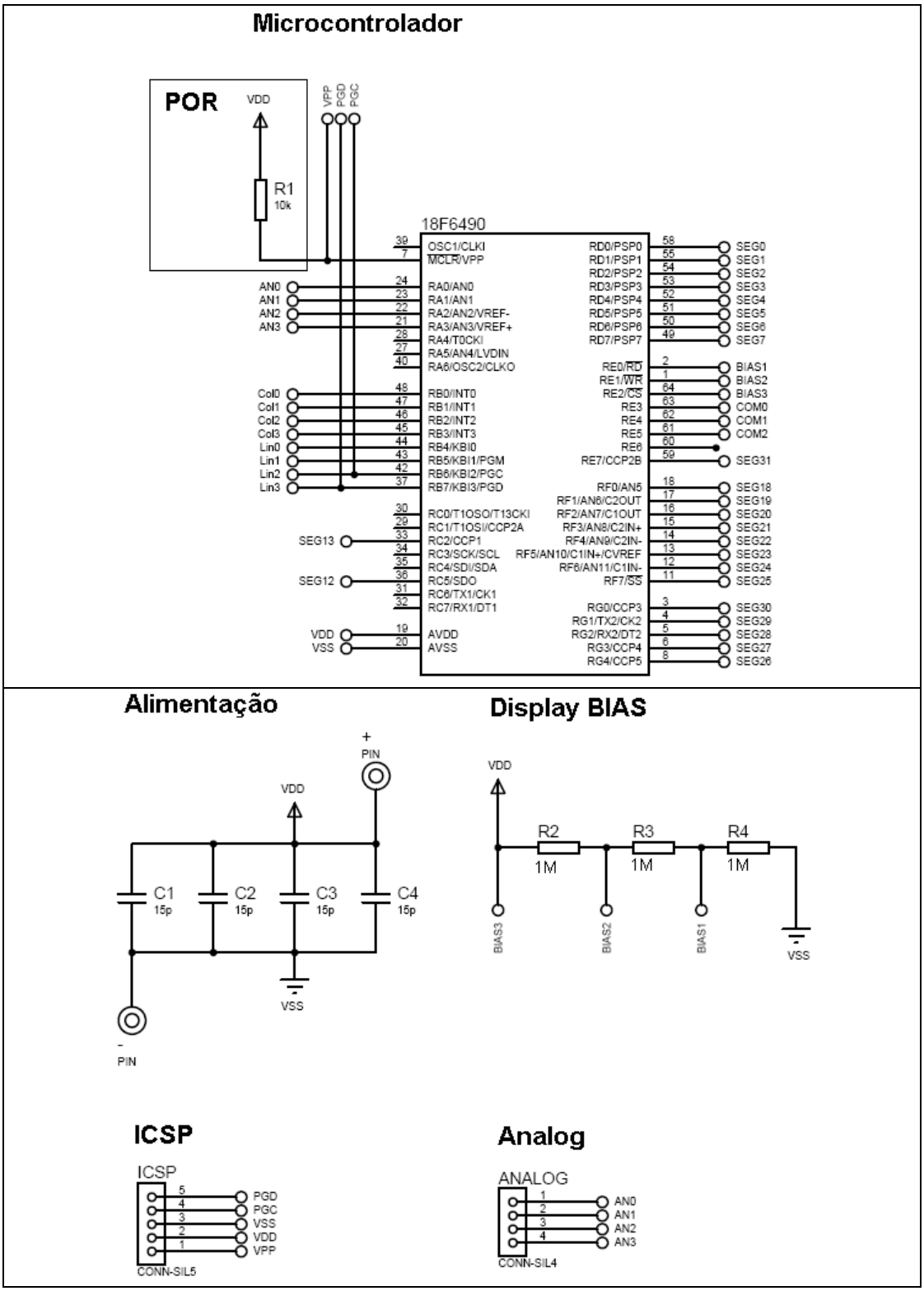
VIEIRA, G. Y. M. e RUGGIERO, W. V. Algoritmos para tokens de autenticação. **Conferência IADIS Ibero-Americana WWW/Internet 2007**. Portugal: IADIS, 2007. 7 p.

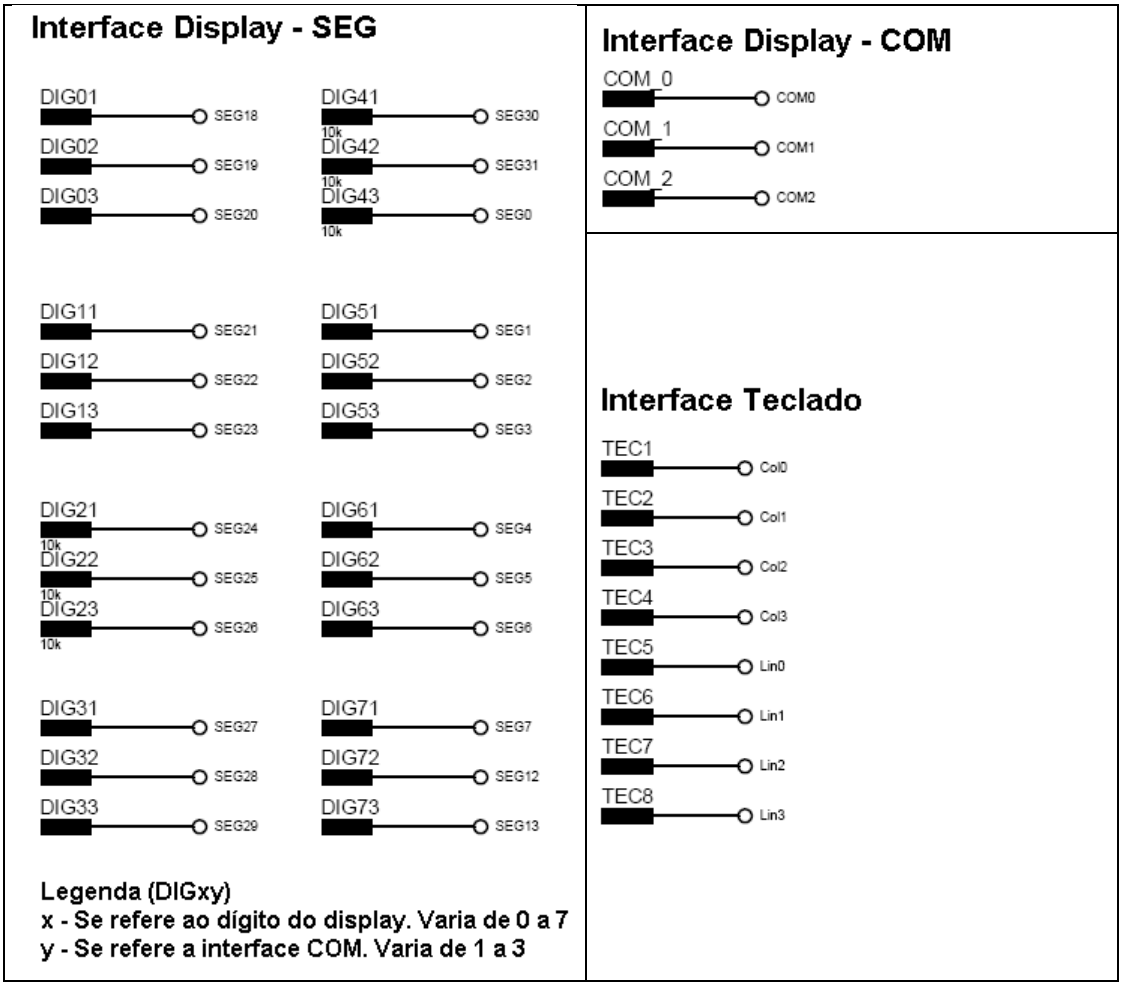
WANG, X., YIN, Y. L. e YU, H. Finding Collisions in the Full SHA-1. **Crypto'05**, 2005. p.

YLONEN, T. e LONVICK, C. **The Secure Shell (SSH) Protocol Architecture (RFC 4251)**: Network Working Group 2006.

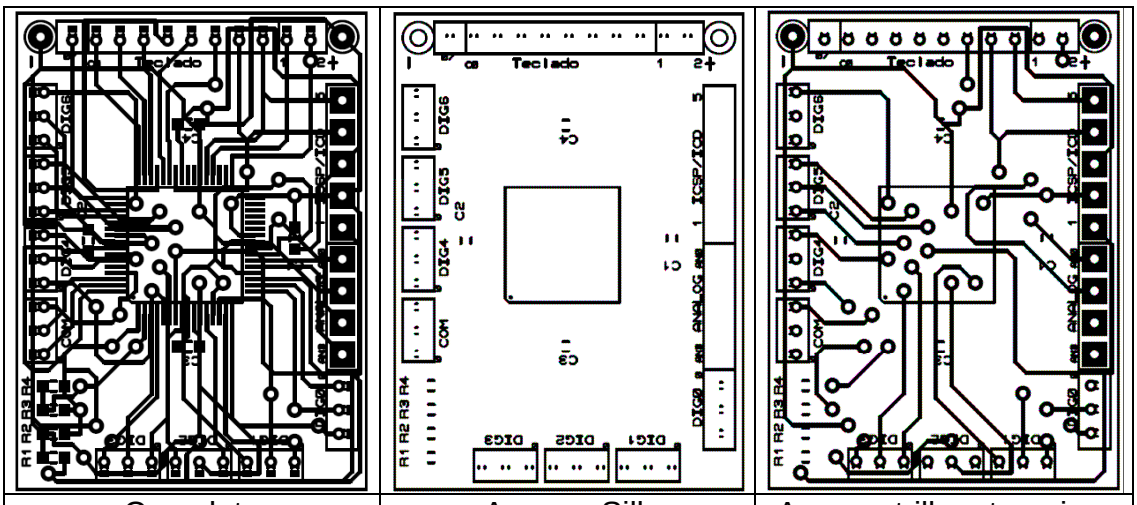
APÊNDICE A - HARDWARE

A.1. Diagrama Circuito Elétrico





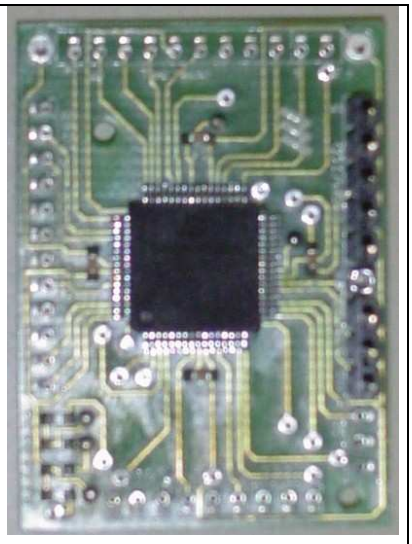
A.2. Placa de Circuito impresso



Completo	Apenas Silk	Apenas trilhas traseiras

Versão ampliada. Tamanho original: 28mm x 38mm

A.3. Fotos Protótipo

		
Frente Calculadora Adaptada	Traseira Calculadora Adaptada	PCB

APÊNDICE B - *SOFTWARE*

Este apêndice descreve o funcionamento de todas as rotinas implementadas no microcontrolador PIC18F para o funcionamento do *token*.

Este apêndice está dividido em 5 partes:

- Uso de variáveis Globais
- Rotinas de *driver*
  - ✓ Teclado
  - ✓ *Display*
  - ✓ EEPROM
  - ✓ Gerenciamento de Energia
  - ✓ Temporização
  - ✓ Inicialização
- Rotinas auxiliares
  - ✓ PIN
  - ✓ *Token*
- Rotinas de interface com o usuário
- Rotina **main()**

As rotinas que são executadas diretamente a partir de algum comando do usuário são chamadas pelo documento de programas e estão descritas na tabela abaixo:

Rotina Programa	Função
RunProgNewTicket()	Gerar uma OTP
RunProgSignTransaction()	Assinatura de transações
RunProgNewKey()	Alterar a chave de criptografia do dispositivo
RunProgNewPIN()	Alterar o PIN do usuário
NavigateLanguageMenu()	Alterar idioma padrão

Ao desenvolver todas as rotinas procura-se observar os seguintes cuidados:

- Sempre que possível, utilizar constantes (define) ao invés de colocar números diretamente no código, para facilitar a leitura.
- Os arquivos do projeto foram separados em diferentes diretórios conforme a sua afinidade.
- O nome dos arquivos fonte faz referência ao nome do diretório em que o arquivo se encontra.
- O nome das rotinas procura descrever o seu objetivo final. O nome da maioria está em inglês.
- Qualquer operação que aparentemente ficasse mais fácil com o uso de divisões foi evitada, já que seria preciso incluir uma rotina em *software* para realizar divisão, já que o microcontrolador não possui um módulo capaz de executar esse tipo de operação.

B.1. Variáveis globais

O sistema define as seguintes variáveis globais:

CancelPowerSave – Armazena quantos segundos já se passaram desde que o usuário tocou pela última vez no teclado do dispositivo. Seu valor máximo fica em SLEEP_TIME (PIC.H).

Segundos – Contador incrementado uma vez por segundo com auxílio da interrupção.

unsigned char Tempo

unsigned char MeioSegundo

unsigned char Segundos

B.2. Teclado

As rotinas do teclado consideram o uso de um teclado matricial com 3 colunas por 4 linhas (12 teclas).

As colunas são selecionadas como saídas e, mudando-se o valor das colunas, varre-se cada uma das linhas e verifica-se se alguma possui o mesmo valor da coluna. Cruzando-se o valor de linha e coluna, descobre-se a tecla pressionada.

Por padrão o teclado está ligado na PORTB do microcontrolador. Para alterar esse padrão, basta mudar o valor dos defines em **teclado.h**:

```
#define PORT_Teclado PORTB
#define TRIS_Teclado TRISB
```

PORTB foi escolhido por possuir resistores de *pull-ups* internos e por possuir interrupções que acordam o microcontrolador quando ele entra em estado de economia de energia (mais detalhes em B.5. Gerenciamento de energia).

B.2.1. TecladoInit.c

B.2.1.1. void InicializaTeclado(void)

Coloca os bits 7 a 4 da porta como entrada e os bits 3 a 1 como saída. O bit 0 não é alterado. Para conseguir esse efeito a configuração é feita em duas etapas (primeiro configura entradas e depois as saídas).

Os *buffers* do teclado são colocados em 1.

Por padrão considera-se que o teclado está ligado em PORTB, por isso a função desabilita as interrupções dessa porta e ativa os resistores de *pull-ups* internos.

B.2.2. Teclado*Driver*.c

B.2.2.1. unsigned char LeituraTeclado(void)

Essa função está preparada para ler um teclado de 4 linhas por 3 colunas. Inicialmente ela coloca valores 0 em cada uma das colunas e verifica se houve mudança em alguma linha.

Logo depois a função verifica qual foi a mudança ocorrida para determinar a tecla pressionada.

Retorna-se o valor da tecla pressionada. Ele pode variar entre 0 e 9. Há constantes definidas em **teclado.h** para determinar se foi pressionada também a tecla '-' (CANCELA), '+' (ENTER), 'MCR', '%', 'M-' ou se nenhuma tecla foi pressionada (NENHUMA).

A rotina de *debounce* deve ser aperfeiçoada.

B.2.3. TecladoProgs.c

B.2.3.1. unsigned char AguardarTecla(void)

Essa função chama a função **LeituraTeclado()** até que o valor retornado seja diferente de NENHUMA.

Logo a seguir, ela zera a variável global **CancelPowerSave**, que funciona como um contador de quantos segundo o dispositivo está inativo. Se essa variável ficar maior que o valor SLEEP_TIME o microcontrolador passa para o modo de economia de energia.

A seguir a rotina espera que a tecla pressionada seja liberada, chamando a função **LeituraTeclado()** até que o valor retornado seja NENHUMA.

B.2.3.2. unsigned char GetString(unsigned char *s, unsigned char Size, unsigned char Mask)

Essa função lê **Size – 1** dígitos digitado pelo usuário. Sempre que uma tecla numérica é pressionada, o valor correspondente é mostrado no *display*.

A leitura de teclas é feita através da função **AguardarTecla()**.

Essa função pode retornar 5 valores diferentes:

- CANCELOU_SEM_DIGITAR se foi pressionado CANCELA sem que nada tenha sido digitado
- ENTER_SEM_DIGITAR se foi pressionado ENTER sem que nada tenha sido digitado
- CANCELOU_MAS_DIGITOU se foi pressionado CANCELA após o usuário entrar com pelo menos 1 dígito

- TERMINOU_ANTES se foi pressionado ENTER antes da função ler todos os dígitos (Size - 1)
- TERMINOU se foi pressionado ENTER e a *string* estiver completa

Para finalizar a função é sempre necessário pressionar ENTER ou CANCELA.

A *string* lida é armazenada no parâmetro ***s**. Se a *string* tiver tamanho **Size – 1**, o caractere de fim de *string* é armazenado na última posição da *string*: **s[Size – 1] = 0**. Caso contrário o caractere de fim de *string* é armazenado depois da última posição digitada.

O parâmetro **Mask** é utilizado para ler uma senha, isto é, é utilizado caso se deseje que, ao invés de se mostrar os números que estão sendo digitados, mostre-se apenas uma máscara para encobrir esses números. O valor passado no parâmetro será a máscara escolhida (normalmente '*'). Para não mascarar o dígito o parâmetro deve permanecer em 0.

B.2.3.3. unsigned char AguardarTempoOuTecla(unsigned char EsperaTempo)

Essa função aguarda que uma tecla seja digitada por no máximo o tempo definido em **EsperaTempo**. Se uma tecla é digitada, o seu valor é retornado e a função é imediatamente finalizada. Senão é retornado **NENHUMA**.

EsperaTempo indica quantos 0,5 s a função deve esperar.

B.3. Display

Foram implementadas rotinas para tornar possível o controle do *display* de uma calculadora comum através do microcontrolador.

B.3.1. LCD*Driver*.c

O *driver* do *display* possui 3 variáveis básicas de controle:

*buffer*LCD[LCD_*BUFFER*_SIZE] – Possui tamanho padrão de 16 bytes e armazena a *string* de caracteres que está atualmente sendo exibida no *display*.

currentChar – próxima posição onde será escrito um caractere no ***buffer*LCD**

cursorControl – Bit 7 de **cursorControl** indica se o cursor deve mostrar o cursor piscante, Bit 6 de **cursorControl** indica se o cursor deve mostrar o cursor fixo, Bit 0 de **cursorControl** indica se no último 0,5s o cursor piscou ou não. Os demais são ignorados.

B.3.1.1. void LCDLimpa(void)

Limpa tudo o que está escrito na tela do *display*, bem como todas as variáveis de controle.

B.3.1.2. void DisplayData(unsigned char data)

Coloca o dado em **data** no *buffer* do *display*. Se for possível imprimir o caractere no *display* ele é impresso, senão ele fica apenas no *buffer*.

B.3.1.3. void DisplayCommand(unsigned char cmd)

Envia o comando em **cmd** para o *display*.

Os comandos disponíveis estão listados na tabela abaixo.

Comando	Função
CLEAR_DISPLAY	Limpa a tela do *display*
CURSOR_OFF	Não exibe cursor no *display*
FIXED	Exibe o cursor sem fazê-lo piscar
BLINK	Faz o cursor piscar
GO_START	Posiciona o cursor na posição do *display*. Esse comando também é utilizado para posicionar letras em outras partes do *display*. Para isso utiliza-se GO_START + x, no qual x representa quantas posições depois da posição inicial.
Os comandos abaixo não foram utilizados, apesar de implementados	
GO_RIGHT	Desloca o cursor uma posição à direita
GO_LEFT	Desloca o cursor uma posição à esquerda
DISPLAY_ROTATE_LEFT	Desloca o conteúdo do LCD repetindo a mensagem
DISPLAY_SHIFT_LEFT	Desloca o conteúdo do LCD sem repetir a mensagem
DISPLAY_MINUS	Exibe o sinal MINUS do display
DISPLAY_MEMORY	Exibe o sinal MEMORY do *display*
DISPLAY_ERROR	Exibe o sinal ERROR do *display*

B.3.1.4. void DisplayNextPosition(void)

Posiciona ponteiro do *buffer* do *display* na próxima posição. Somente é utilizado pela função **DisplayCommand().**

Em caso de *overflow* do *buffer* o ponteiro é posicionado no começo do *buffer*.

B.3.1.5. void DisplayPreviousPosition(void)

Posiciona ponteiro do *buffer* do *display* na próxima posição. Somente é utilizado pela função **DisplayCommand()**.

Em caso de *underflow* do *buffer* o ponteiro é posicionado no fim do *buffer*.

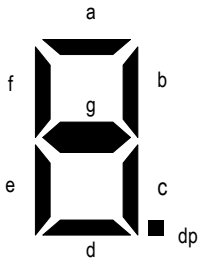
B.3.1.6. void DisplayCursorControl(void)

É chamada pela rotina de interrupção a cada meio segundo. Faz o cursor piscar se essa opção estiver ativa.

B.3.2. LCDROM.H

Armazena a ROM de caracteres do LCD. Um dígito do *display* possui 9 segmentos. A ROM trata apenas 8 deles. O segmento não tratado não afeta o funcionamento do *display*.

Para criar um desenho no *display* é preciso lembrar que ele possui os seguintes segmentos:



Cada uma das linhas de **romCharCalc[][]** tem o desenho de uma letra diferente. Um bit "setado" liga o segmento e um bit "resetado" desliga o segmento. Os bits controlam os segmentos na seguinte ordem:

.cdbgeaf

Desta forma, para o número 1 que utiliza apenas os segmentos 'b' e 'c', devemos gravar em **romCharCalc[]** a seguinte seqüência:

- Binário: 0b01010000 ou
- Hexadecimal: 0x50 ou
- Decimal: 80

Para tornar mais claro, todos os desenhos foram feitos utilizando notação binária.

B.3.3. LCDUpdate.c

B.3.3.1. void DisplayUpdateChar(unsigned char position, unsigned char character)

Essa rotina é responsável por gravar nos registradores do microcontrolador o caractere que será exibido no *display*.

Inicialmente ela converte o caractere em **character** para um endereço de **romCharCalc[][]**.

A **romCharCalc[][]** é lido e o conteúdo é dividido em 3 partes, cada uma das partes responsável por um sinal COM do *display*.

Em seguida, verifica-se onde deve ser impresso o caractere através do conteúdo de **position**. Baseado-se nisso, grava-se no registrador responsável por COM0, COM1 e COM2.

Os registradores foram alocados da seguinte forma:

Posição	Registradores responsáveis
0	LCDDATA18 a LCDDATA20
1	LCDDATA21 a LCDDATA23
2	LCDDATA24 a LCDDATA26
3	LCDDATA27 a LCDDATA29
4	LCDDATA30, LCDDATA31 e LCDDATA20
5	LCDDATA1 a LCDDATA3
6	LCDDATA4 a LCDDATA6
7	LCDDATA7, LCDDATA12 e LCDDATA13

A figura a seguir indica os registradores utilizados com cores para facilitar a localização dos registradores utilizados e os registradores livres.

LCDDATA7	**LCDDATA6**	**LCDDATA5**	**LCDDATA4**	**LCDDATA3**	**LCDDATA2**	**LCDDATA1**	**LCDDATA0**
LCDDATA15	LCDDATA14	**LCDDATA13**	**LCDDATA12**	LCDDATA11	LCDDATA10	LCDDATA9	LCDDATA8
LCDDATA23	**LCDDATA22**	**LCDDATA21**	**LCDDATA20**	**LCDDATA19**	**LCDDATA18**	LCDDATA17	LCDDATA16
LCDDATA31	**LCDDATA30**	**LCDDATA29**	**LCDDATA28**	**LCDDATA27**	**LCDDATA26**	**LCDDATA25**	**LCDDATA24**

B.3.4. LCDInicializa.c

B.3.4.1. void LCDInicializa(void)

Rotina executada quando se liga o dispositivo. Configura o controlador de *display* do microcontrolador. Segue as instruções descritas no *datasheet* do microcontrolador.

B.3.5. LCDProgs.c

São rotinas baseadas nas rotinas do *driver*. Elas tornam mais fácil a escrita de *strings* no *display*.

B.3.5.1. void DisplayString(const unsigned char *a)

Faz o *display* mostrar uma *string*.

Deve ser utilizada para mostrar *strings* que estão armazenadas em uma variável, isto é, na área de dados do microcontrolador.

B.3.5.2. void DisplayStringROM(const rom unsigned char *a)

Faz o *display* mostrar uma *string*.

Deve ser utilizada para mostrar *strings* que estão armazenadas na área de programa do microcontrolador.

B.3.5.3. unsigned char DisplayBanner(unsigned char waitKey)

Mostra todo o conteúdo do *buffer* no *display* fazendo um deslocamento à esquerda. Se **waitKey** estiver em '1' o conteúdo é exibido até que uma tecla seja pressionada, senão ele é exibido uma única vez.

A todo instante se uma tecla é pressionada a função é finalizada.

Ela retorna a tecla pressionada pelo usuário.

B.4. Uso da EEPROM

Diferentemente do primeiro microcontrolador adotado (PIC18Fx52) os microcontroladores PIC18Fxx90 não possuem EEPROM.

Neste projeto, foi alocada uma área da memória RAM do microcontrolador para simular a EEPROM. O tamanho dessa área é definido pelo mnemônico EEPROM_SIZE.

Os protótipos das funções que faziam acesso à EPROM foram mantidos. Dessa forma não foi necessário fazer mudanças nos trechos de código que faziam referência à EEPROM. As funções foram alteradas para referenciar a área de RAM que simula a EEPROM.

Para uso no *token* a memória foi dividida em 5 partes distintas, que são referenciadas por seus mnemônicos nas rotinas da EEPROM.

Mnemônico	Função	Faixa
AUTKEY_POSITION	Chave Secreta do *Token*	0 a 15
VALID_AUTKEY_POSITION	Indica se o *token* já foi utilizado	16
*TOKEN*_LOCKED_POSITION	Indica quantos erros foram cometido ao entrar o PIN	17
PIN_POSITION	PIN do usuário	18 a 21

B.4.1. EEPROM*Driver*.c

A EEPROM é acessada basicamente através de duas funções, uma grava e a outra lê a EEPROM. Todas as demais rotinas executam chamadas a essas funções.

B.4.1.1. void WriteEEPROM(unsigned char What, unsigned char Where)

Essa função grava o que foi passado no parâmetro **What** na posição de memória **Where**.

B.4.1.2. unsigned char ReadEEPROM(unsigned char Where)

Essa função lê a posição de memória dada pelo parâmetro **Where** e retorna o valor dessa posição.

B.4.2. EEPROMProgs.c

Esse arquivo é constituído de rotinas típicas para se utilizar a EEPROM. Há rotinas para ler e gravar *strings* na EEPROM e uma rotina para limpar toda a EEPROM.

B.4.2.1. void WriteDataEEPROM(unsigned char What[], unsigned char Size, unsigned char Where)

Essa função grava o vetor **What[]** de tamanho **Size** a partir da posição de memória **Where.**

B.4.2.2. void ReadDataEEPROM(unsigned char What[], unsigned char Size, unsigned char Where)

Essa função grava no vetor **What[]** de tamanho **Size** o que está a partir da posição de memória **Where.**

B.4.2.3. void CleanEEPROM()

Escreve zeros ('\0') em todas as posições da EEPROM.

B.5. Gerenciamento de energia

O sistema de gerenciamento de energia está dividido em duas partes. A primeira é responsável por verificar a tensão atual da bateria. A segunda parte passa o microcontrolador para o modo de economia de energia quando não utilizado por um tempo maior que **SLEEP_TIME** segundos.

A variável global **CancelPowerSave** está diretamente associada à rotina de economia de energia.

B.5.1. CancelPowerSave

CancelPowerSave é uma variável global instanciada em Main.c essa variável é incrementada uma vez por segundo dentro da rotina de interrupção.

Sempre que qualquer tecla é pressionada na rotina AguardarTecla() a variável **CancelPowerSave** é zerada.

Quando **CancelPowerSave** chega a **SLEEP_TIME**, isto é, em **SLEEP_TIME** segundos sem que alguém pressione qualquer tecla do dispositivo, a rotina de economia de energia é acionada.

O mnemônico **SLEEP_TIME** pode ser encontrado em **PIC.H**.

B.5.2. PowerBattery.c

Verifica a tensão atual da bateria.

B.5.2.1. void LowBattery(void)

Essa rotina é executada sempre que o dispositivo é ligado ou quando ele retorna da economia de energia. Ela verifica se a tensão da bateria é maior do que 2,43V. Em caso positivo uma mensagem pedindo a troca da bateria é exibida. Ela utiliza o circuito de *High*/*Low Voltage Detection* (HLVD) do microcontrolador.

Não é utilizado o bit de interrupção associado a esse circuito.

B.5.3. PowerPowerSave.c

Coloca o dispositivo em modo de economia de energia.

B.5.3.1. void PowerSave(void)

Esta rotina é ativada sempre que a variável global **CancelPowerSave** chega em **SLEEP_TIME**.

Inicialmente ela desabilita todas as interrupções do microcontrolador e prepara o microcontrolador para perceber qualquer toque na coluna 4 do teclado. Logo a seguir desliga o Timer0 (não obrigatório, mas desliga), certifica-se de que as portas, com exceção de **PORTB**, estão configuradas como saída para economizar energia (MICROCHIP, 2001) e ativa a economia de energia do microcontrolador.

Quando qualquer tecla é pressionada, o microcontrolador passa a executar as instruções imediatamente depois do **Sleep()**. Os 3 **Nop()** estão presentes apenas para garantir que nada de errado será executado. O *buffer* do teclado é limpo e o microcontrolador é reiniciado.

B.6. Rotinas de temporização

A temporização no microcontrolador é feita por meio de interrupções geradas pelo contador **TIMER0** do microcontrolador. Esse timer é configurado para executar 1 interrupção a cada décimo de segundo (quando funcionando a 250 kHz).

B.6.1. Rotina de interrupção (main.c)

Sempre que uma interrupção ocorre, o microcontrolador pára o que está fazendo e vai executar a rotina retornando exatamente ao ponto em que parou como se nada tivesse acontecido.

Mais detalhes sobre o funcionamento de interrupções no microcontrolador devem ser vistas no manual do compilador.

B.6.1.1. void interrupt_at_high_vector(void)

As interrupções no microcontrolador são gravadas sempre a partir do endereço 0008H. Nessa posição foi colocada a rotina **interrupt_at_high_vector** que simplesmente é um desvio para a verdadeira rotina de interrupção **void Interrupcao(void)**.

B.6.1.2. void Interrupcao(void)

Essa rotina é executada 1 vez a cada décimo de segundo. A rotina testa se o *flag* de interrupção do **TIMER0** (**TMR0IF**) está ativo. Em caso positivo ele é desabilitado, os segundos são incrementados de 1 e o **TIMER0** é reiniciado para que ele estoure dentro de 1 segundo novamente.

O Registrador **TMR0L** tem o seu valor subtraído de 17x4, pois até que ele seja atualizado, 17 instruções em *Assembly* são executadas antes dessa atribuição. A idéia da subtração é compensar esse atraso. A interrupção é executada exatamente a cada décimo de segundo e nesses instantes o contador **TIMER0** deve ser atualizado para que a próxima interrupção ocorra novamente em 1 décimo de segundo. No entanto, há tarefas que devem ser executadas antes dessa atualização e é preciso compensar o tempo que o contador não foi atualizado. Se o MPLAB for colocado no modo *Disassembly Listing* (Menu *View*), pode-se contar quantas instruções são executadas antes de se realizar a atualização. No caso, 17. A multiplicação por 4 é necessária pois o contador não é incrementado a cada ciclo de *clock* do cristal, mas a cada ciclo de instrução que equivale a 4 ciclos de *clock* do cristal.

Logo a seguir as variáveis para se atualizar o tempo atual são atualizadas.

Sempre que se passa 1 segundo, a variável global **CancelPowerSave** é incrementada. Ver mais informações sobre essa variável na parte de economia de energia.

Nessa rotina, o *flag* de interrupção de **PORTB**, não é executado em condições normais, mas como ele é utilizado na rotina de economia de energia, ele foi situado na rotina de interrupção para evitar qualquer problema.

B.6.2. TempoAguarda.c

B.6.2.1. unsigned char TempoHouveBorda(unsigned char ClockSource, unsigned char *old)

Verifica se houve alguma mudança em **ClockSource** desde a última vez que esta função foi chamada retornando 1 em caso positivo e 0 em caso negativo.

ClockSource pode ser CLOCK_MEIO_SEGUNDO e desta forma a função identifica se já passou meio segundo desde a última execução da função.

ClockSource pode ser CLOCK_UM_SEGUNDO e desta forma a função identifica se já passou um segundo desde a última execução da função.

Para tomar essa decisão a função grava em ***old** o instante em que a função foi executada pela última vez.

B.6.3. PICVelocidade.c

Esse arquivo possui as funções para aumentar e reduzir a velocidade do *clock* do sistema.

B.6.3.1. void PICSpeedUp(void)

Desliga as interrupções do sistema, evitando que o dispositivo seja desligado automaticamente.

Logo a seguir, o oscilador interno é configurado para operar a 1MHz. Instruções **Nop()** são utilizadas para evitar que o processador execute alguma instrução útil até que a velocidade do sistema se estabilize. O seu uso não é obrigatório.

B.6.3.2. void PICSpeedDown(void)

O oscilador interno é configurado para operar a 250KHz. Instruções **Nop()** são utilizadas para evitar que o processador execute alguma instrução útil até que a velocidade do sistema se estabilize. O seu uso não é obrigatório.

Logo a seguir reativa as interrupções do sistema.

B.7. Rotinas de Inicialização

São rotinas executadas apenas quando o dispositivo é ligado ou então retorna do modo de economia de energia. Elas configuram as portas de entrada e saída do microcontrolador, contadores internos e outros dispositivos.

B.7.1. PICInicializa.c

B.7.1.1. void PIC18FInicializa(void)

Essa rotina configura o oscilador interno do microcontrolador para operar a 250 KHz, coloca as interrupções no modo de compatibilidade com a família PIC16, ativa apenas a interrupção do **TIMER0** e desabilita todos os periféricos do microcontrolador:

- Suporte aos *timers*, com exceção do **TIMER0**
- Suporte aos canais de PWM
- Suporte à comunicação MSSP/SPI/I2C
- Suporte a EUSART
- Suporte a AUSART
- Suporte ao conversor A/D
- *Watchdog* Timer

Não são alterados os valores das portas **PORT**. Eles são alterados apenas pelas rotinas que as utilizam. Portas não utilizadas são configuradas como saída, pois o datasheet do microcontrolador (MICROCHIP, 2001) recomenda que elas sejam mantidas nesse estado e desconectadas para minimizar o consumo de corrente do microcontrolador.

B.7.2. PICFirstRun.c

B.7.2.1. void FirstRun(void)

Quando a chave do dispositivo é ajustada, na posição **VALID_AUTKEY_POSITION** da EEPROM, é gravada a seqüência 0x55 (0b01010101).

Essa rotina verifica se essa seqüência foi gravada e, em caso negativo, admite-se que o dispositivo nunca foi utilizado. Dessa forma a EEPROM é limpa para se ter certeza que qualquer conteúdo antigo seja apagado.

Logo depois é pedido um PIN e uma nova Chave de Criptografia para o dispositivo.

B.8. *Token*

Possui as rotinas que chamam as rotinas de criptografia do *token*.

B.8.1. *Token*Progs.c

B.8.1.1. unsigned char GeraOTP(...)

Essa rotina possui os seguintes parâmetros:

unsigned char *ticket – Valor do desafio digitado pelo usuário

unsigned char *notp – Resultado do desafio em formato numérico

Inicialmente são definidos os seguintes *buffers*:

```
unsigned char chaveUsuario[16];
unsigned char ticketCheck[4];
```

Logo a seguir, os *buffers* que serão utilizados serão zerados pela função padrão do C, **memset()**, para assegurar que eles não possuam qualquer lixo inicial.

A seguir a chave do usuário é lida da EEPROM. E como as próximas rotinas exigem maior velocidade, o microcontrolador passa a operar a 1 MHz através da rotina **PICSpeedUp()**.

A rotina que gera a OTP, **makeOTP()**, é chamada tendo como parâmetros os *buffers* declarados inicialmente. A rotina **calcMAC()** é chamada para verificar a validade do desafio.

O processamento de ambas as rotinas leva menos de 2 segundos a 1 MHz. Logo a seguir o microcontrolador passa a operar a 250 KHz através da rotina **PICSpeedDown().**

Antes de terminar a rotina o desafio é verificado. Ele é válido se os últimos 3 dígitos do desafio conferem com o valor de **ticketCheck[]**.

B.8.1.2. unsigned char AssinaTransacao(...)

A rotina de assinatura de transação é semelhante à rotina de geração de OTP.

Essa rotina possui os seguintes parâmetros:

unsigned char *Caracterizador – Número da transação digitada pelo usuário

unsigned char *nsignature – Assinatura da transação em formato numérico

B.9. Interface com o usuário

As rotinas anteriormente apresentadas remetem ao uso dos hardwares do microcontrolador e dos periféricos acoplados a eles.

Elas foram criadas para dar suporte às rotinas que realizam a interface com o usuário. Nessas rotinas, encaixam-se aquelas que mostram o menu principal, pedem o PIN do usuário, permitem a entrada de uma nova chave, entre outras.

As rotinas de interface com o usuário podem ser encontradas na pasta **User**.

B.9.1. UserMessages.h

Esse arquivo contém todas as mensagens que são mostradas ao usuário através do *display*. Todas as mensagens estão limitadas a 16 caracteres (tamanho do *buffer display*), mais o identificador de fim de *string*.

As vantagens em concentrar todas as mensagens em um só arquivo é que ações como encontrar erros de grafia, mudar mensagens que são mostradas e alterar o

idioma das mensagens tornam-se trabalhos triviais. Mensagens repetidas deixam de aparecer no código fonte economizando memória de programa.

A desvantagem ocorre na hora da programação, pois essas mensagens são referenciadas através de índices. Contudo, esse problema é compensado pelo fato dos índices serem tão mnemônicos quanto possível. O uso dos índices permite que a passagem de parâmetros em qualquer rotina seja feita através de índices e não de mensagens com 17 bytes. Enfim, esse tipo de organização traz muitas vantagens com quase nenhuma desvantagem. Os índices aparecem em **User.h**, logo à frente.

B.9.2. User.h

Além dos protótipos, esse arquivo armazena uma enumeração que aponta para cada uma das mensagens em **UserMessages.h**. A mensagem e a enumeração devem manter sempre a mesma ordem para que não sejam mostradas mensagens erradas aos usuários.

Nos programas dificilmente há números fixos, trabalha-se principalmente com constantes em todos os programas. As seguintes constantes são definidas nos programas de interface com o usuário:

Constante	Valor	Função
MAIN_MENU_ITENS	5	Número de itens do menu principal
ENTRA_TICKET	1	Chamada dos desafios
ENTRA_ASSINATURA	2	Chamada da rotina de assinatura
NOVA_CHAVE	3	Chamada da rotina para mudar a chave
NOVO_PIN	4	Chamada da rotina para mudar o PIN
IDIOMA	*5*	*Chamada da rotina para mudar o Idioma*
LANGUAGE_MENU_ITENS	*2*	*Número de itens no menu Idioma*
PORTUGUESE	*1*	*Idioma português*
ENGLISH	*2*	*Idioma inglês*

Note-se que nem todas as constantes são utilizadas. Algumas foram herdadas das primeiras implementações do sistema. Elas estão destacadas em itálico.

São definidas também as seguintes macros:

B.9.2.1. IMPAR(x)

Retorna 0 se o número **x** não for ímpar. É feito analisando se o ultimo bit do número é 1.

B.9.2.2. PAR(x)

Retorna 0 se o número x não for par. Essa macro inverte o valor da macro IMPAR(x).

B.9.2.3. ShowMessageWelcome()

Mensagem exibida quando se liga o dispositivo.

B.9.2.4. ShowMessageLowBattery()

Mensagem exibida indicando que a bateria está ficando fraca para operar o sistema adequadamente.

B.9.2.5. ShowMessageTryAgain(x)

O parâmetro **x** aponta para uma mensagem que diz que algo digitado pelo usuário é inválido. Na linha seguinte pede-se para o usuário tentar novamente.

B.9.3. UserMainMenu.c

Essa função exibe uma mensagem para indicar ao usuário que ele está no menu principal e aguarda que alguma tecla seja pressionada.

Em **User.h** foram definidas as teclas que serão reconhecidas por esta função:

```
#define MENU_SEQUENCIAL 3
#define MENU_DESAFIO 1
#define MENU_ASSINATURA 2
#define MENU_NOVO_PIN 4
#define MENU_NOVA_CHAVE 5
#define MENU_DESLIGAR 0
```

Pressionando uma das teclas definidas acima, o respectivo programa é acionado. Caso alguma tecla inválida seja pressionada, a função é terminada, porém **main.c** chama a função novamente e o *token* aguarda novamente que alguma tecla seja pressionada.

Nesta função, tanto as teclas **ENTER**, **CANCELA** e '1' têm a mesma função de executar o programa para gerar OTP. Isso foi feito, pois é a opção mais freqüentemente utilizada.

B.9.4. UserLanguage.c

Este arquivo foi herdado dos primeiros protótipos e não é utilizado no protótipo final.

Originalmente eram rotinas responsáveis por selecionar o idioma do dispositivo, mas perderam o sentido quando o *display* adotado foi reduzido para uma versão de 7 segmentos. Elas foram deixadas pois, como citado nas conclusões, é algo que pode ser utilizado para um sistema que necessite de um sistema disponível para diferentes idiomas.

B.9.4.1. void NavigateLanguageMenu(void)

Funcionamento equivalente a **NavigateMainMenu()**, a única diferença fica por conta da tecla Cancela que aborta a seleção de idiomas.

B.9.4.2. void ShowLanguageMenu(unsigned char CurrentMenu)

Funcionamento equivalente a **ShowMainMenu()**, a principal diferença é que essa é a única função que não utiliza **UserMessages.h** e nem **ShowMessage()**, ambos descritos anteriormente.

Nesse caso não foi considerado necessário utilizar esse recurso, já que essas mensagens são fixas e cada uma está em um idioma diferente.

B.9.5. UserCommonProgs.c

Fora do menu principal, o usuário sempre tem que digitar algum tipo de dado no dispositivo, ou então confirmar alguma operação. As rotinas desse arquivo dão esse suporte aos programas *novo desafio*, *assinatura de transação*, *mudança de chave* e *mudança de PIN*.

B.9.5.1. unsigned char ShowMessage(unsigned char msg)

Exibe a mensagem apontada por **msg**. A mensagem a ser exibida é lida da ROM de mensagens e exibida ao usuário. Todas as funções do tipo *ShowMessage...* baseiam-se nessa função.

Apesar de haverem várias mensagens definidas, o último protótipo utilizou apenas as mensagens que não misturavam letras maiúsculas com minúsculas para evitar erros de leitura e compreensão por parte do usuário.

B.9.5.2. unsigned char ShowMessageWaitKey(unsigned char msg)

Exibe a mensagem apontada por **msg**. A mensagem fica sendo deslocada à esquerda pelo *display* a cada 0,5 segundos até o usuário pressionar alguma tecla. Retorna a tecla pressionada pelo usuário.

Essa função, apesar de implementada, não foi utilizada no último protótipo. Seu uso era é mostrar mensagens longas ao usuário (exemplo: mensagem com 16 caracteres em um *display* de 8 caracteres). Ela não foi utilizada pois o seu uso poderia confundir um usuário do sistema.

B.9.5.3. void ShowMessageToUser(unsigned char msg)

Exibe a mensagem apontada por **msg**. Aguarda o usuário pressionar alguma tecla.

B.9.5.4. unsigned char ShowFastMessage(unsigned char msg)

Exibe, por 1 segundo, a mensagem apontada por **msg**. Retorna a tecla pressionada pelo usuário, caso ele pressione alguma.

B.9.5.5. unsigned char VerifyValidKey(void)

Verifica se já foi gravada alguma chave na EEPROM testando se na posição VALID_AUTKEY_POSITION da memória está gravado '0x55'. Em caso negativo é exibida mensagem de erro ao usuário e o programa, solicitando uma Nova Chave (**RunProgNewKey()**), é executado imediatamente.

B.9.5.6. unsigned char CreateUserField(unsigned char Block, unsigned char FieldSize)

Rotina responsável por desenhar no *display* o campo que o usuário deve preencher. Os campos são limitados a 5 dígitos e sempre são desenhados da forma mais centralizada possível (campos de tamanho ímpar ficam levemente deslocados à esquerda).

O campo é desenhado de acordo com a figura a seguir, no qual x indica o bloco atual, se for necessário escrever o número do bloco atual.

```
Campo de tamanho 1 sem bloco atual: |  =1=   |
Campo de tamanho 3 sem bloco atual: | =123=  |
Campo de tamanho 5 sem bloco atual: |=12345= |
Campo de tamanho 2 sem bloco atual: |  =12=  |
Campo de tamanho 4 sem bloco atual: | =1234= |
Campo de tamanho 1 com bloco atual: |x  =1=  |
Campo de tamanho 3 com bloco atual: |x =123= |
Campo de tamanho 5 com bloco atual: |x=12345=|
Campo de tamanho 2 com bloco atual: |x  =12= |
Campo de tamanho 4 com bloco atual: |x =1234=|
```

Apesar das possibilidades da função, são desenhados campos apenas de tamanho 3 e 4 no *token*.

B.9.5.7. unsigned char DefautUserInterface()

Essa rotina possui os seguintes parâmetros:

```
      unsigned char Mensagem
      unsigned char FirstMsgL2
      unsigned char DefaultMsg
      unsigned char Size
      unsigned char ErrorMsg
      unsigned char *UserInput
      unsigned char Opt
```

Essa função é chamada sempre que o usuário deve entrar com algum dado no dispositivo.

O parâmetro **Opt** deve ser interpretado em 2 etapas analisado os 4 bits menos significativos e depois os 4 mais significativos:

7	6	5	4	3-0
ExibeBloco	*AceitaIncompleto*	*MostrarMascara*	*VariosBlocos*	*OptVal*

Do bit 3 ao bit 0 é armazenada a informação do dígito ou caractere que deve ser mostrado na primeira posição do *display*. Isso é usado para indicar qual bloco está sendo lido atualmente (varia de 0 a 9) ou para indicar ao usuário que o PIN está sendo lido (Letra 'P' (1100) para leitura de PIN, letra 'A' (1101) para leitura do Antigo PIN, letra 'E' (1110) para Entrada do novo PIN e letra 'C' (1111) para Confirmação do novo PIN)

O bit 4 *VariosBlocos* é utilizado quando se deseja que os 4 bits menos significativos tenham alguma função.

O bit 5 *MostrarMascara* demanda o uso da mascara '*' ao invés de se mostrar os números entrados pelo usuário. Esse bit é "setado" apenas pela função **RunProgGetUserPIN()** (leitura de PIN).

O bit 6 *AceitaIncompleto*, diz que a função deve aceitar uma *string* incompleta.

O bit 7 *ExibeBloco* é utilizado para que seja mostrado o valor dos 4 bits menos significativos.

Depois de verificar **Opt**, verifica-se se alguma mensagem inicial deve ser exibida e verifica se o usuário pressionou Cancela, finalizando a função.

Logo a seguir, o programa exibe o bloco atual por 1 segundo e então desenha o campo que deve ser preenchido pelo usuário (**CreateUserField()**) e captura a entrada do usuário com o uso da rotina **GetString()**.

Quando o usuário digita, há 4 opções:

1. Pressionou CANCELA com o campo em branco. Nesse caso a rotina termina e retorna CANCELOU_SEM_DIGITAR.
2. Pressionou ENTRA com o campo em branco. Se for permitida uma *string* incompleta, a rotina termina e retorna ENTER_SEM_DIGITAR, senão a rotina simplesmente redesenha a tela e continua pedindo entrada do usuário.
3. Usuário preencheu parte do campo e pressionou CANCELA. Nesse caso a rotina simplesmente redesenha a tela e continua pedindo entrada do usuário.

4. Usuário preencheu parte do campo e pressionou ENTRA. Se for permitida uma *string* incompleta, a rotina termina e retorna TERMINOU_ANTES, senão a rotina exibe a mensagem de erro apontada por **ErrorMsg** com o auxílio da macro **ShowMessageTryAgain()**, redesenha a tela e continua pedindo entrada do usuário.
5. Usuário preencheu todo o campo. Nesse caso a rotina termina e retorna TERMINOU.

A entrada do usuário fica armazenada no parâmetro ***UserInput**.

B.9.5.8. unsigned char GetInBlocks()

Parâmetros de entrada:

```
unsigned char UserData[]
unsigned char UserDataSize
unsigned char Blocks
unsigned char BlockSize
unsigned char Opt
```

Essa rotina divide a entrada do usuário em **Blocks** blocos de tamanho **BlockSize** cada um. O campo **Opt** determina se os blocos devem ser totalmente preenchidos.

Cada bloco é preenchido por chamadas à rotina **DefautUserInterface()**. Dependendo da resposta da rotina, o bloco anterior é solicitado novamente se o usuário pressionar CANCELA com a tela limpa, ou o mesmo bloco é solicitado se o usuário já havia digitado algo na tela.

Depois que cada um dos blocos são lidos, eles são concatenados e armazenados em **UserData[]**. O campo **UserDataSize** determina o tamanho total de **UserData[]**.

B.9.6. UserNewKey.c

Esse arquivo contém o programa responsável por trocar a chave do usuário.

B.9.6.1. unsigned char RunProgNewKey(void)

Por ser uma operação pouco usual, o programa também solicita o PIN do usuário. A seguir, o programa pede cada um dos 8 blocos da chave do usuário. Sempre que o

usuário pressiona cancela com o campo em branco, o programa retorna para o bloco anterior. Assim se, por exemplo, o usuário estiver no bloco 4, será necessário pressionar cancela 4 vezes para abortar a entrada de uma nova chave.

A chave é lida através da rotina **UserGetInBlocks()**.

Em seguida, a chave é convertida do formato decimal para o formato hexadecimal e o resultado é gravado na memória a partir da posição AUTKEY_POSITION e é gravado '0x55' na posição VALID_AUTKEY_POSITION da EEPROM para indicar que a chave foi gravada adequadamente.

B.9.7. UserPIN.c

B.9.7.1. void RunProgNewPIN(void)

Essa rotina pede o PIN atual através de **EnterYourPIN()** e se ele estiver correto, chama a rotina **GetNewPIN()**. Ela é abortada quando o usuário desiste ou quando o usuário termina a operação.

B.9.7.2. unsigned char RunProgGetUserPIN(unsigned char *UserPIN, unsigned char MessageNumber)

Essa rotina chama a rotina para **DefautUserInterface()** para ler o PIN que o usuário digitar. O PIN lido fica armazenado em ***UserPIN**. Essa função é chamada em diferentes circunstâncias e em cada uma destas, o valor **MessageNumber** é diferente. Ele pode representar: Entrada do PIN atual, PIN antigo, PIN novo e confirmação do PIN novo.

B.9.8. User*Token*.c

Esse arquivo contém os programas que chamam as rotinas principais de operação do dispositivo.

Qualquer um dos programas desse arquivo verifica se a chave do usuário é válida antes de iniciar qualquer processamento.

B.9.8.1. void RunProgNewTicket(void)

Lê a entrada do usuário com auxílio de **UserGetInBlocks()** e chama a rotina de geração de OTP **GeraOTP()**. Uma barra de pregresso é exibida enquanto o processamento esta sendo realizado.

Se o desafio for inválido, é exibida uma mensagem alertando ao usuário.

Após o processamento, a OTP é exibida até que o usuário pressione qualquer tecla.

B.9.8.2. void RunProgSignTransaction(void)

Lê a entrada do usuário com auxílio de **UserGetInBlocks()** e chama a rotina de autenticação de transação **AssinaTransacao()**. Uma barra de pregresso é exibida enquanto o processamento esta sendo realizado.

Após o processamento, o MAC reduzido é exibido até que o usuário pressione qualquer tecla.

B.9.8.3. void Show*Token*Results(unsigned char *Numbers, unsigned char *Letters)

Exibe o resultado do processamento de um dos programas acima. O parâmetro **Numbers** mostra o resultado em formato numérico e o parâmetro **Letters** mostra o resultado em formato alfabético.

B.10. PIN

Contém funções para se trabalhar com o PIN de tamanho arbitrário. O tamanho é definido através do uso da constante **PIN_SIZE**.

O PIN fica armazenado na RAM do microcontrolador e, portanto, caso a bateria seja removida ele é imediatamente apagado.

Além do PIN, existe um contador armazenado na EEPROM de quantas vezes foi cometido algum erro na entrada do PIN. A cada 3 erros o dispositivo é bloqueado. Se forem cometidos **MAX_ERRORS** erros, a EEPROM é limpa, retornando o sistema ao seu estado inicial. **MAX_ERRORS** tem valor padrão em 6.

B.10.1. PINChangePIN.c

Esse arquivo contém as rotinas que realizam mudança no PIN do usuário.

B.10.1.1. void PINFirstRun(void)

Essa rotina é executada apenas quando o dispositivo estiver em seu estado inicial.

Essa rotina solicita um novo PIN com o uso da rotina **GetNewPIN()**. Enquanto não for digitado um novo PIN, a rotina não é terminada.

B.10.1.2. unsigned char GetNewPIN(void)

Essa rotina solicita um PIN novo ao usuário, confirma o PIN e verifica se ambos são iguais. Em caso afirmativo o PIN é gravado na EEPROM.

B.10.2. PINEnterPIN.c

B.10.2.1. unsigned char EnterYourPIN(unsigned char MessageNumber)

Essa rotina lê o PIN atual do usuário e verifica se ele está correto. Caso o PIN esteja correto, o contador de erros é reiniciado.

Caso ele esteja incorreto é exibida uma mensagem de erro e a rotina **LockSystem()** é executada.

O retorno indica se o usuário digitou corretamente o PIN.

B.10.3. PINLockSystem.c

Contém a rotina responsável pelo modo de bloqueio automático do sistema.

B.10.3.1. void LockSystem(void)

Essa rotina tem 3 comportamentos possíveis:

- Reinicializar o sistema, pois o número máximo de erros do PIN foi atingido

- Executar o modo de bloqueio automático do sistema, pois foram cometidos mais de 3 erros seguidos
- Retornar caso nenhuma das condições anteriores tenha sido satisfeita.

Quando a rotina é executada, ela verifica se foram cometidos **MAX_ERRORS** erros. Em caso positivo a EEPROM do sistema é zerada e o sistema é reiniciado. Desta forma o sistema passa a se comportar como se nunca tivesse sido utilizado antes.

Se ainda não foram cometidos **MAX_ERRORS** erros, o sistema verifica se **não** foram cometidos 3 erros consecutivos. Em caso positivo a rotina é terminada. Em caso negativo, o sistema torna-se inacessível por 5 minutos exibindo uma mensagem que o sistema foi bloqueado. Uma contagem regressiva também é mostrada indicando quanto tempo falta para que o sistema seja liberado.

Enquanto o sistema estiver bloqueado, essa rotina desabilita o sistema de economia de energia (desligamento automático).

B.11. Rotina Principal

Rotina responsável por chamar todas as funções para preparar o sistema para uso e exibir o menu principal.

B.11.1. Main.c

B.11.1.1. main()

Essa rotina está dividida em três fases: Inicialização, entrada de PIN e uso normal.

A etapa de inicialização do microcontrolador é executada na seguinte ordem:

- Inicializa as variáveis globais de tempo
- Inicializa periféricos do microcontrolador, teclado e *display*
- Verifica se foi pressionada a tecla ENTRA. As demais são ignoradas retornando o sistema ao estado de *stand-by*.
- Exibe mensagem de bem-vindo
- Verifica a tensão de alimentação

- Verifica se é a primeira vez que o usuário utiliza o sistema
- Verifica se o sistema está bloqueado
- Programa o sistema para se desligar em 10 segundos

Uma vez iniciado, chama-se a rotina para entrada de PIN, **EnterYourPIN()**, até que o usuário acerte o PIN.

Sempre que o sistema volta do modo de economia de energia ou do modo de bloqueio automático, todas as rotinas acima são executadas novamente.

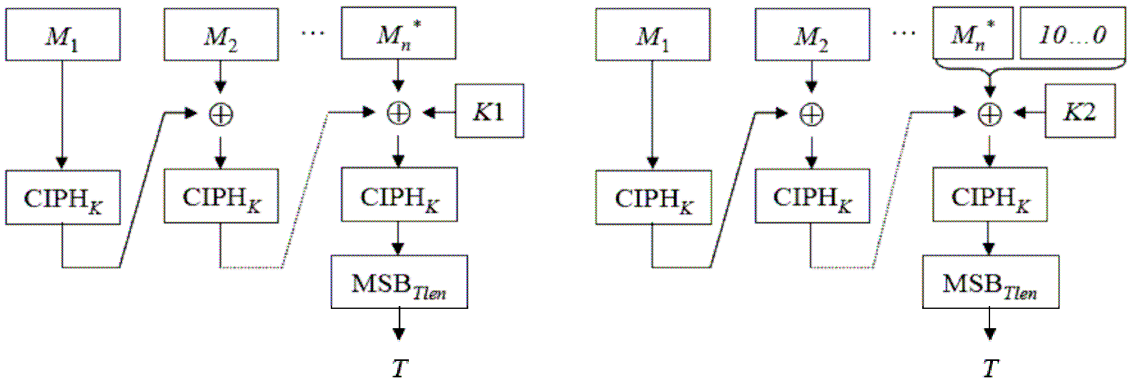
Após a entrada do PIN, o sistema passa exibir o menu "ESCOLHA" para o usuário através da rotina **MainMenu()**. No fim da execução de um programa, o menu "ESCOLHA" é exibido novamente.

B.12. Núcleo

Rotinas do Núcleo envolvem as rotinas referentes aos algoritmos de criptografia utilizados no *token*.

B.12.1. CMAC

Essa seção descreve, passo a passo, a implementação utilizada do CMAC. Maiores detalhes do algoritmo devem ser encontrados em (NIST, 2005). O algoritmo pode ser melhor entendido pelo esquema a seguir retirado de (NIST, 2005):



A figura mostra a mensagem M dividida em blocos numerados de 1 até n. Cada um dos blocos é criptografado com a chave K e então é feito um XOR entre o resultado obtido (vetor V na descrição do algoritmo a seguir) e o bloco atual.

No último bloco existem 2 possibilidades. Quando o bloco é completo, é feito um XOR com uma sub-chave K1. Essa sub-chave K1 é calculada nos passos 4a e 4b do algoritmo CMAC a seguir. Se o bloco necessita de uma padding, a chave K2 é calculada no passo 4c do algoritmo CMAC a seguir. A chave K2 é a chave K1 com a função Shift aplicada novamente.

Uma vez calculada a sub-chave que será utilizada, o resultado é criptografado novamente resultando no MAC T.

B.12.1.1. CMAC

Entradas:

- K – vetor com a Chave AES
- M – Vetor com a mensagem que terá o seu MAC calculado
- m – Tamanho da mensagem

Saídas:

- T – Vetor com o MAC da mensagem M

Variáveis utilizadas:

- MAXBS – Tamanho Máximo de um bloco (128 bits ou 16 bytes)
- L – Vetor de bytes com tamanho MAXBS. Utilizado para armazenar a sub-chave que será utilizada na hora de calcular o *padding*
- V – Vetor de bytes com tamanho MAXBS. Utilizado para guardar os valores intermediários do cálculo
- KS – Armazena o escalonamento das sub-chaves AES

Execução do algoritmo:

1. Preparar o escalonamento das chaves do AES e guardar em KS
2. Inicializar com zeros o vetor V
3. Cálculo do MAC para blocos intermediários:
    a. Determinar o tamanho do bloco que será calculado
    b. Fazer o XOR entre V e a mensagem M em blocos de 128 bits
    c. Ir para o passo 4 se estivermos no último bloco, isto é, bloco menor ou igual a 128 bits
    d. Criptografar V

    e. Posicionar para o próximo bloco de 128 bits
    f. Voltar para o passo 3.a.
4. Cálculo do Padding:
    a. Criptografar o vetor 0 com a chave e guardar o resultado em L
    b. Executar a função Shift em L
    c. Verificar o último bloco precisa de padding
        i. Em caso afirmativo executar Shift em L novamente
        ii. Completar V com o padding
    d. Fazer o XOR entre V e L
    e. Criptografar V
5. Limpar o vetor L
6. Limpar o escalonamento KS
7. Copiar V para T
8. Limpar o vetor L

B.12.1.2. Shift

Entrada:

- L – Vetor que será deslocado

Saída:

- L – Vetor deslocado de uma posição à direita

Variáveis utilizadas:

- c – Armazena o valor do bit mais significativo

Execução do Algoritmo:

1. Determina o valor do bit mais significativo
2. Desloca o vetor à direita
3. Se o bit mais significativo for diferente de 0 então realizar um XOR o número 87h $(10000111)_b$.

B.12.2. AES

Essa seção descreve, passo a passo, a implementação utilizada do AES. Maiores detalhes do algoritmo e o funcionamento do escalonamento de chaves podem ser encontrados no FIPS 197 (FIPS197, 2001).

O algoritmo AES-128 trata as chaves e a mensagem a ser cifrada como uma matriz de tamanho 4 x 4, no qual cada célula da matriz armazena um byte diferente. A

partir de uma chave principal são geradas sub-chaves que são utilizadas em cada um dos *rounds* do AES. Cada *round* é composto de 4 passos (BARRETO, 2005):

- AddRoundKey – Aplicação da chave
- ShiftRow – Difusão entre as colunas de dados
- SubBytes – Camada de não-linearidade
- MixColumn – Difusão dentro de cada coluna

B.12.2.1. Passos do AES

Abaixo estão descritos os 4 passos do AES, para os quais as entradas K (Sub-chave) e M (Estado atual do algoritmo) e a saída R (Resultado do processamento) são matrizes 4x4:

- R = AddRoundKey(K, M) – XOR entre a chave do round atual com M:

$$\text{Para cada elemento } i,j \text{ de } M \text{ faça: } R_{i,j} = K_{i,j} \text{ xor } M_{i,j}$$

- R = ShiftRow(M) – Rotação de 1 para a esquerda com a 1ª linha, 2 para a 2ª Linha e 3 para a 3ª linha:

$$\text{Para cada elemento } i,j \text{ de } M \text{ faça: } R_{i,j} = M_{i,(j+i \text{ and } 3)}$$

- R = SubBytes(M) – Substituição de cada elemento o conteúdo pela S-Box do algoritmo:

$$\text{Para cada elemento } i,j \text{ de } M \text{ faça: } R_{i,j} = S[M_{i,j}]$$

- R = MixColunms(M) – Resultado da multiplicação de cada uma das colunas pela matriz $\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$.

Multiplicação de um número 'm' por 02 é feita como definido em (FIPS197, 2001), seção 4.2.1, isto é, um deslocamento à esquerda. Em caso de *overflow* no deslocamento é feito um XOR com o hexa 11B.

```
Se (m << 1) tem overflow
      c = (m << 1) XOR 11B
senão
      c = (m << 1)
```

Multiplicação de um número 'm' por 03 é feita com uma multiplicação por 02 e uma posterior soma (XOR) com m.

```
d =  MultiplicacaoPor02(m) XOR m
```

Multiplicação de A por 01: elemento neutro

```
e = m
```

Multiplicação de Coluna por matriz:

```
Para cada elemento i,j de M faça:
      c = MultiplicacaoPor02(M_{i,j})
      d = MultiplicacaoPor03(M_{(i+1)&3,j})
      e = MultiplicacaoPor01(M_{(i+2)&3,j})
      f = MultiplicacaoPor01(M_{(i+3)&3,j})
      Ri,j = c XOR d XOR e XOR f
```

B.12.2.2. Implementação Algoritmo de Criptografia

Na implementação utilizada, é declarada uma estrutura de dados chamada KS que armazena 13 matrizes 4x4 numeradas de 0 a 12. A posição A (11) e a posição B (12) são utilizadas para armazenar temporariamente resultados do processamento dos *rounds* do AES:

- KS[0] – Armazena a chave
- KS[1] a KS[10] – Armazenam as sub-chaves de cada rodada
- KS[11] e KS[12] – Armazena temporariamente

Entradas:

- KS – Ver descrição acima
- In – Vetor de 16 bits com a mensagem aberta

Saída:

- Out – Vetor de 16 bits com a mensagem cifrada

Execução do Algoritmo:

1. KS[A] = AddRoundKey(KS[0], Mensagem)
2. Para cada um dos *rounds* faça:
    - KS[B] = SubBytes(ShiftRow(KS[A]))
    - KS[A] = AddRoundKey(MixColumns(KS[B]))
3. out = SubBytes(AddRoundKey(ShiftRow(KS[A])))

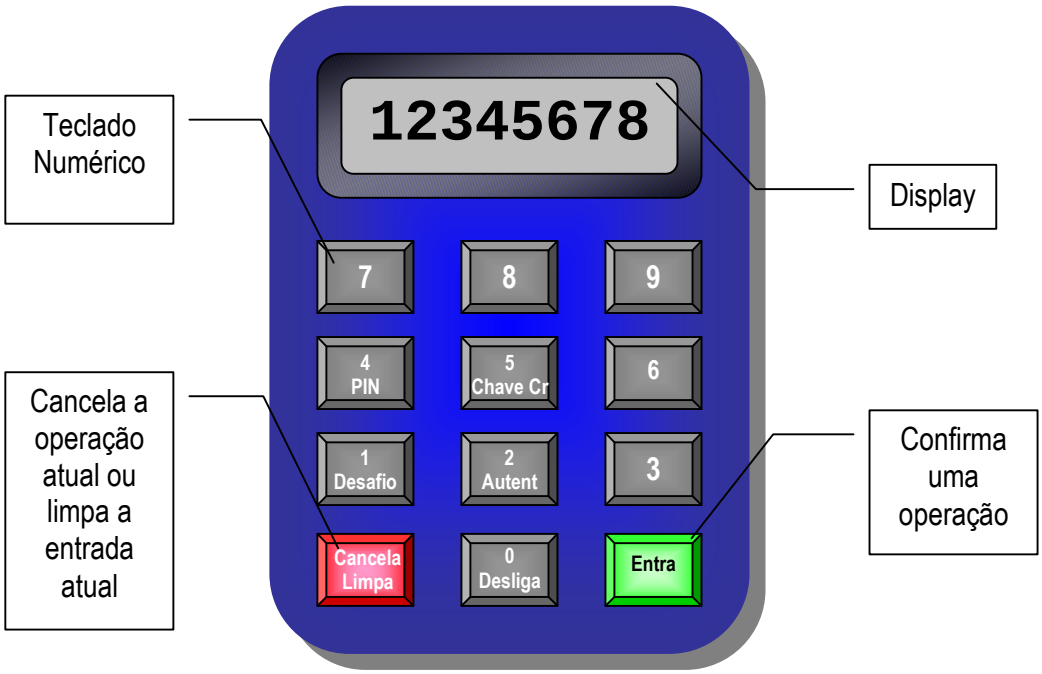
APÊNDICE C - MANUAL DO USUÁRIO

O manual do usuário é apresentado como se o dispositivo fosse adotado por uma instituição financeira. O funcionamento do dispositivo é explicado no nível de operador do sistema.

As principais funções apresentadas são:

- Primeiro uso
- Geração de OTP
- Autenticação de transações
- Entrada de nova chave de criptografia
- Escolha de um PIN

C.1. Teclas de navegação



Legenda

📱➪💻 Indica que o procedimento a seguir deve ser realizado na página aberta no computador.

☜📱 Indica que o procedimento a seguir deve voltar a ser realizado no aparelho

☠ Indica que algum problema ocorreu

C.1.1. No protótipo

No protótipo utilizado, tem-se o seguinte mapa de teclas:

On	MRC	M-	M+
OFF	Raiz	%	/
7	8	9	X
4	5	6	-
1	2	3	+
0	.	=	

As teclas marcadas em cinza não têm nenhuma função. As teclas coloridas seguem as mesmas funções do desenho anterior:

- 1 - Desafio
- 2 - Autenticação
- 4 - Mudança de PIN
- 5 - Mudança da chave de criptografia
- - - Cancela
- + - Entra

C.2. Usando pela primeira vez

Esse é o aparelho que torna as transações feitas via internet mais seguras.

Ao ligar o aparelho, a seguinte mensagem é exibida:

```
LArC
```

Logo a seguir o aparelho indica que nunca foi utilizado através da mensagem:

```
1o USO
```

Por ser um dispositivo novo, é preciso escolher um PIN de 4 dígitos para o aparelho:

```
E ⌋ ____ ⌊
```

A letra 'E' indica que deve ser escolhido um novo PIN. Para evitar erros de digitação, o novo PIN é solicitado novamente:

```
C ⌋ ____ ⌊
```

A letra 'C' indica o pedido de confirmação do PIN. Qualquer dúvida sobre a etapa de definição do PIN pode ser resolvida na seção C.7. PIN.

Após definir um PIN é exibido o seguinte campo:

```
1 ⌋ ____ ⌊
```

Nesta etapa é necessário definir uma nova Chave de Criptografia. Para isso siga os procedimentos em C.8. Nova Chave.

☠ Atenção! Caso seja a primeira vez que o aparelho seja usado e as telas acima não aparecerem, remova a bateria e coloque-a novamente para que o aparelho seja devidamente reiniciado.

☠ Se a Chave de Criptografia não for definida, sempre que se tentar utilizar o aparelho para se calcular um desafio ou uma autenticação, uma mensagem de "Erro" é exibida e os procedimentos em C.8. Nova Chave são solicitados novamente.

Se o aparelho for desligado sem uma nova Chave de Criptografia, ao ligá-lo novamente ele se comportará como se tivesse sido ligado pela primeira vez, solicitando inclusive o PIN novamente.

C.3. Ligando o aparelho

Para ligar o aparelho pressione a tecla ENTRA. Ao ligá-lo é exibida o campo abaixo:

```
P ⌋ ____ ⌊
```

A letra 'P' no começo indica que o aparelho está esperando pela entrada do PIN definido anteriormente.

Após digitar o PIN veja a seção C.4. Menu de escolha.

Caso o PIN seja digitado incorretamente, a seguinte mensagem será exibida:

```
Erro
```

O PIN é solicitado novamente após a mensagem.

☠ Para proteção contra acessos não autorizados o aparelho é imediatamente bloqueado após 3 tentativas inválidas. Veja mais detalhes em C.7. PIN.

C.4. Menu de escolha

Ao ligar o aparelho é pedido um Desafio. Caso essa não seja a opção desejada, pressione CANCELA e o menu de escolha é exibido.

Quando o menu principal é carregado, a seguinte mensagem aparece na tela:

```
ESCOLHA
```

Ela indica que deve ser escolhida uma das 4 opções disponíveis através do teclado numérico do aparelho:

1. Desafio – Para entrar com um novo Desafio, deve-se pressionar a tecla 'Desafio'. Veja mais detalhes em C.5. Desafio.
2. Autenticar – Para realizar uma autenticação digital, deve-se pressionar a tecla 'Autenticação'. Veja mais detalhes na seção A.6 "Autenticação Eletrônica".
3. PIN – Para definir um novo PIN, deve-se pressionar a tecla 'PIN'. Veja mais detalhes em C.7. PIN.
4. Chave de Criptografia – Para trocar a Chave de Criptografia, deve-se pressionar a tecla 'Chave Cr'. Veja mais detalhes em C.8 "Nova Chave".

Ao pressionar CANCELA ou ENTRA a qualquer instante, a opção DESAFIO é automaticamente selecionada.

C.5. Desafio

O desafio é um número gerado apenas pela página do banco e serve para tornar a entrada não a página mais segura. Esse número muda sempre, pois mesmo que alguma pessoa capture esse número, ela jamais poderá acessar a página do banco em outra ocasião, pois o desafio sempre será diferente a cada novo acesso garantindo maior segurança às pessoas que desejam utilizar algum serviço bancário.

☠ Caso a Chave de Criptografia atual não seja válida, é exibida a seguinte mensagem assim que a tecla Desafio é pressionada:

Erro

Pressione qualquer tecla e siga os procedimentos descritos em C.8. Nova Chave.

Para entrar com um desafio no aparelho, pressione a tecla DESAFIO no menu de escolhas. O desafio é sempre solicitado em 3 blocos de 3 dígitos cada. Nesse modo é exibido o número do bloco atual e o campo onde deve ser digitado o desafio:

1 ⌋ ___ ⌊

Entre com o bloco e pressione ENTRA para ir para o próximo bloco. Se for cometido um erro de digitação, pressione CANCELA e a entrada atual é apagada. Se for pressionado CANCELA e o bloco atual estiver em branco, o bloco anterior é solicitado novamente. Se o bloco atual for o 1º e ele estiver em branco quando é pressionado CANCELA, a operação atual é cancelada e o Menu Principal exemplo exibido novamente.

Enquanto o sistema estiver calculando a resposta do desafio (pouco menos de 4 segundos), é exibida uma barra de progresso de 8 etapas:

```
--------
```

Após o processamento, a resposta do desafio é exibida como no exemplo abaixo:

```
⌋ 987654 ⌊
```

📱➪💻 Volte ao computador e entre com esse número na página do banco.

☟📱 Se pressionada qualquer tecla o menu principal será carregado novamente.

☠ Caso o Desafio tenha sido digitado incorretamente ou a página seja falsa é exibida a mensagem:

```
Erro
```

Neste caso pressione qualquer tecla para voltar ao Menu de Escolha. Antes de prosseguir, tenha certeza de que a página acessada é a página verdadeira.

Veja o exemplo da entrada do desafio 503 041 980:

Passo	Dispositivo	Observação
1	`1 ⌋ 503 ⌊`	Entrada do bloco 1
2	`2 ⌋ 041 ⌊`	Entrada do bloco 2
3	`3 ⌋ 980 ⌊`	Entrada do bloco 3
4	`--------`	Aparelho calculando (4s)
5a	`⌋ 104015 ⌊`	📱➪💻 Resposta para a página
5b	`Erro`	☠ Página falsa ou erro de digitação

C.6. Autenticação eletrônica

Cada transação possui um único código de autenticação e para uma mesma transação pessoas diferentes também geram códigos de autenticação diferentes. Com isso garante-se que você tenha certeza da transação que ele está realizando e a página do banco tem a certeza de quem a gerou.

☠ Caso a Chave de Criptografia atual não seja válida, é exibida a seguinte mensagem assim que a tecla AUTENTICAÇÃO é pressionada:

```
Erro
```

Pressione qualquer tecla e siga os procedimentos descritos em C.8. Nova Chave.

Sempre que for necessário realizar uma autenticação eletrônica de uma transação, pressione a tecla AUTENTICAÇÃO no menu de escolha. A seguinte tela é exibida indicando a seleção dessa opção:

```
1 ⌋ ____ ⌊
```

O aparelho lê uma transação de até 32 dígitos em 8 blocos de 4 dígitos. Entre com o bloco e pressione ENTRA para ir para o próximo bloco. Se eventualmente houver um bloco de tamanho menor simplesmente pressione ENTRA antes de terminar o bloco. Se todos os blocos já tiverem sido digitados e o aparelho solicitar um novo bloco, pressione ENTRA no bloco vazio.

Se for cometido um erro de digitação, pressione CANCELA e a entrada atual é apagada. Se for pressionado CANCELA e o bloco atual estiver em branco, o bloco anterior é solicitado novamente. Se o bloco atual for o 1º e ele estiver em branco quando é pressionado CANCELA a operação atual é cancelada e o menu de escolha é exibido novamente.

☠ Caso a Chave de Criptografia atual não seja válida, é exibida a seguinte mensagem assim que a tecla AUTENTICAÇÃO é pressionada:

```
Erro
```

Pressione qualquer tecla e siga os procedimentos descritos em C.8. Nova Chave.

Enquanto o sistema estiver processando (pouco menos de 4 segundos), é exibida a mensagem abaixo:

```
--------
```

Após o processamento, a autenticação é exibida no *display* como no exemplo abaixo:

```
⌋ 123456 ⌊
```

📟➪💻 Volte ao computador e entre com esse número na página do banco.

↳📟 Depois se pressionada qualquer tecla o menu principal será carregado novamente.

Veja o exemplo para a autenticação da seguinte transação:

- Transação: 1 (transferência)
- Agencia: 095
- Conta: 1980
- Valor: R$ 341,20

Caracterizador da transação: 1 095 1980 341

Passo	Dispositivo	Observação
1	`1 ⌋ 1 ⌊`	Entrada do bloco 1
2	`2 ⌋ 095 ⌊`	Entrada do bloco 2
3	`3 ⌋ 1980 ⌊`	Entrada do bloco 3
4	`4 ⌋ 341 ⌊`	Entrada do bloco 4
5	`5 ⌋ ⌊`	Finaliza (bloco 5 em branco)
6ª	`⌋ 365412 ⌊`	📟➪💻 Resposta para a página
6b	`Erro`	☠ Página falsa ou erro de digitação

C.7. PIN

O PIN é uma senha curta de 4 dígitos utilizada para evitar que uma pessoa não autorizada tente utilizar o aparelho. Recomenda-se sempre que essa senha seja alterada com freqüência e que não seja anotada em qualquer lugar.

Sempre que necessário, é possível substituir o PIN atual. Para isso, pressione a tecla PIN no menu de escolha. Logo a seguir o seguinte campo é exibido:

```
A ⌋ ____ ⌊
```

A letra 'A' indica que PIN antigo deve ser digitado.

☠ Para proteção contra acessos não autorizados o aparelho é imediatamente bloqueado após 3 tentativas inválidas e a seguinte mensagem é exibida:

FALHA

O aparelho é liberado apenas após 5 minutos de espera. Enquanto o aparelho estiver bloqueado, exibe-se uma contagem regressiva até que o aparelho seja liberado:

Err 4 59

Após a liberação, se houver novamente mais 3 tentativas inválidas, a memória do aparelho é imediatamente apagada juntamente com o PIN e a Chave de Criptografia e o aparelho se comportará como se fosse novo. Esse procedimento garante que mesmo que o aparelho seja extraviado por algum motivo, ninguém tenha acesso às suas informações.

Após digitar o PIN antigo será solicitado um novo PIN:

E ⌋ ____ ⌊

A letra 'E' indica que deve ser escolhido um novo PIN. Para evitar erros de digitação, o novo PIN é solicitado novamente:

C ⌋ ____ ⌊

A letra 'C' indica o pedido de confirmação do PIN.

Se for cometido um erro de digitação, pressione CANCELA e a entrada atual será apagada. Se for pressionado CANCELA sem que haja algo digitado, o processo será cancelado.

Se o PIN antigo for digitado incorretamente, ou se o novo PIN não coincidir com a confirmação, a seguinte mensagem será exibida:

Erro

☠ Caso o PIN seja esquecido será necessário entrar em contato com o suporte para que seja enviada uma nova Chave de Criptografia.

C.8. Nova Chave

A Chave de Criptografia é o que garante que as transações serão realizadas de forma segura. As duas únicas pessoas que possuem a senha são você e a página. Nunca a revele para ninguém. Uma vez que ela é colocada no aparelho, a senha pode ser imediatamente destruída para que ninguém mais tenha acesso a ela.

Sempre que necessário, é possível substituir a Chave de Criptografia atual. Para isso pressione a tecla Senha Cr no menu de escolha. Como essa é uma operação pouco usual, O PIN é solicitado novamente:

```
P ⌋ ____ ⌊
```

A letra 'P' no começo indica que o aparelho está esperando pela entrada do PIN definido anteriormente.

☠ Para proteção contra acessos não autorizados o aparelho é imediatamente bloqueado após 3 tentativas inválidas. Veja mais detalhes em C.7. PIN.

Após entrar com o PIN, a nova chave de criptografia deve ser digitada em 8 blocos de 4 dígitos cada. O seguinte campo é exibido:

```
1 ⌋ ____ ⌊
```

O número '1' indica que o aparelho está aguardando o primeiro bloco da Chave de Criptografia. Entre com o bloco e pressione ENTRA para ir para o próximo bloco. Se for cometido um erro de digitação, pressione CANCELA e a entrada atual será apagada. Se for pressionado CANCELA e o bloco atual estiver em branco, o bloco anterior será solicitado novamente. Se o bloco atual for o 1º e ele estiver em branco quando é pressionado CANCELA a operação atual será cancelada e o menu de escolha será exibido novamente.

Se o Bloco atual for inválido, a seguinte tela será exibida:

```
Erro
```

Aguarde um instante ou pressione qualquer tecla e o Bloco será solicitado novamente.

Veja o exemplo para entrar com a seguinte chave de criptografia 1456 3654 3698 1478 7412 9632 4123 6321:

Passo	Dispositivo	Observação
1	**1 ⌋ 1456 ⌊**	Entrada do bloco 1
2	**2 ⌋ 3654 ⌊**	Entrada do bloco 2
3	**3 ⌋ 3698 ⌊**	Entrada do bloco 3
4	**4 ⌋ 1478 ⌊**	Entrada do bloco 4
5	**5 ⌋ 7412 ⌊**	Entrada do bloco 5
6	**6 ⌋ 9632 ⌊**	Entrada do bloco 6
7	**7 ⌋ 4123 ⌊**	Entrada do bloco 7
8	**8 ⌋ 6321 ⌊**	Entrada do bloco 8
9	**--------**	Depois de concluído o menu principal é carregado

☠ Jamais troque a Chave de Criptografia do aparelho a menos que a página tenha feito explicitamente essa solicitação. **Essa solicitação jamais é feita via Internet.**

C.9. Funções extras

C.9.1. Desligando

O aparelho pode ser desligado ao pressionar a tecla OFF no menu de escolha. Após 1 minuto de inatividade, o aparelho também se desliga automaticamente.

O Desligamento automático não é ativado enquanto o sistema estiver bloqueado após três tentativas inválidas de entrada do PIN. Ver mais sobre sistema bloqueado em C.7. PIN.

C.9.2. Bateria

O aparelho utiliza baterias tipo "botão" de 3V modelo CR2032.

Quando a alimentação do sistema torna-se fraca, o aparelho exibe a seguinte mensagem:

```
Bat Fr
```

Nesse caso substitua a bateria assim que possível. A Chave de Criptografia e o PIN são apagados sempre que a bateria é removida do aparelho.

Normalmente uma bateria nova é o suficiente para utilizar o aparelho por 1 ano supondo-se que o aparelho seja utilizado diariamente por 5 minutos.

C.9.3. Entrada de dados

Sempre que for necessário digitar algum dado no aparelho um campo semelhante ao da figura abaixo é exibido:

```
⌋ ____ ⌊
```

- Para entrar dados no sistema utilize o teclado numérico e pressione ENTRA para confirmar a entrada atual.
- Caso seja cometido um erro de digitação, pressione CANCELA e a entrada atual será apagada.
- Caso seja uma entrada em blocos, para retornar para o bloco anterior pressione CANCELA com o campo em branco.
- Para cancelar a operação de uma entrada comum apague o campo e pressione CANCELA.
- Para cancelar a operação de uma entrada em blocos volte para o primeiro bloco, apague-o e pressione CANCELA.

A qualquer instante em uma mensagem é possível pressionar qualquer tecla para pular a mensagem.

APÊNDICE D - DIAGRAMAS DE CASOS DE USO

Este apêndice apresenta os diagramas de casos de uso para o dispositivo de autenticação.

Os casos de uso apresentados podem ter dois atores diferentes. O primeiro ator é o usuário que utiliza o sistema. O segundo ator seria um relógio interno do sistema que pode influenciar no funcionamento do sistema.

D.1. Caso de uso: Ligar *token*

Usuário liga o *token*, ou o *token* foi reiniciado.

Seqüência de eventos

1. *Token* sai do estado de "*stand-by*"
2. Inicializa registradores internos
3. Inicializa LCD
4. Incializa Teclado
5. Verifica se foi pressiona a tecla "ENTRA" (em caso de erro segue caminho alternativo)
6. Exibe mensagem de boas-vindas
7. Verifica a bateria do sistema
8. Verifica se não é a primeira execução do sistema (em caso de erro segue caminho alternativo)
9. Executa caso de uso "Verifica Bloqueio do Sistema"
10. Sistema programa-se para autodesligar em 10 segundos
11. *Token* executa caso de uso "Entra PIN" e verifica se retornou com sucesso (em caso de erro segue caminho alternativo)

Evento iniciador

Os seguintes eventos podem iniciar o caso de uso:

- Usuário pressiona qualquer tecla do *token*
- *Token* foi reiniciado por algum motivo: Bateria foi trocada ou usuário errou PIN mais de 3 vezes

Pré-condições

Token está desligado ou *token* reiniciado.

Caminhos alternativos

- Tecla "ENTRA" não foi pressionada: *Token* volta para o estado de *stand-by*
- Primeira execução: *Token* executa os casos de uso "Escolha de novo PIN" e "Escolha de nova chave de criptografia" ao finalizar os casos de uso o fluxo prossegue normalmente
- Erro em entra PIN: Executa caso de uso "Entra PIN" novamente

Ator

Usuário.

D.2. Caso de uso: Desligamento Automático

Token é desligado automaticamente depois de algum tempo sem uso.

Seqüência de eventos

1. *Token* desliga todas as interrupções
2. *Token* prepara-se para ligar quando qualquer tecla for pressionada
3. *Token* desliga os periféricos (LCD e contadores)
4. *Token* coloca as portas no modo de menor consumo de energia

5. *Token* muda para o estado de *stand-by*, reduzindo consumo de bateria.

Evento iniciador

Contador de tempo chega a 0.

Pré-condições

Algoritmo de criptografia não está sendo executado.

Pós-condições

Token se reinicia se qualquer tecla for pressionada.

Caminhos alternativos

Nenhum.

Ator

Relógio.

D.3. Caso de uso: Verifica Bloqueio do Sistema

Token verifica se ele deve se bloquear devido ao excesso de erros do PIN.

Seqüência de eventos

1. Verifica se o número máximo de erros de PIN não foi atingido (em caso de erro, segue caminho alternativo)
2. Verifica se não ocorreram 3 erros desde o último bloqueio do sistema (em caso de erro, segue caminho alternativo)
3. Finaliza caso de uso

Evento iniciador

Os seguintes eventos podem iniciar o caso de uso:

- Bloqueio é verificado pelo caso de uso "Ligar *token*"
- Bloqueio é verificado pelo caso de uso "Entra PIN"

Pré-condições

Nenhuma.

Caminhos alternativos

Número máximo de erros atingido:

1. Exibe mensagem avisando que houve muitos erros
2. Limpa a memória do sistema (Apaga todo o conteúdo da memória)
3. Reinicia o *token* (Executa o caso de uso "Ligar *token*")

Ocorreram 3 erros, isto é, sistema bloqueado por 5 minutos:

1. Exibe mensagem de bloqueio com um contador regressivo
2. Exibe na tela um contador de 5 minutos
3. A cada segundo esse contador é subtraído de 1
4. O sistema é reiniciado quando o contador chegar a zero

Atores

Usuário e relógio.

D.4. Caso de uso: Tela de Escolha

A partir da tela de escolha o usuário pode executar qualquer uma das funções do *token*.

Seqüência de eventos

- *Token* exibe a mensagem "ESCOLHA" e aguarda o usuário pressionar uma tecla do *token*. A seguir estão as teclas funcionais nessa etapa:
  - Tecla 0: Executa o caso de uso "Desligar *token*"
  - Tecla 1: Executa o caso de uso "Geração de OTP"
  - Tecla 2: Executa o caso de uso "Autenticação"
  - Tecla 4: Executa o caso de uso "Escolha de novo PIN"
  - Tecla 5: Executa o caso de uso "Escolha de nova chave de criptografia"
  - Tecla "ENTRA": Executa o caso de uso "Geração de OTP"
  - Tecla "LIMPA": Executa o caso de uso "Geração de OTP"

Evento iniciador

Os seguintes eventos podem iniciar o caso de uso:

- Caso de uso "Ligar *Token*" é finalizado com sucesso
- Caso de uso "Geração de OTP" é finalizado
- Caso de uso "Autenticação" é finalizado
- Caso de uso "Escolha de novo PIN" é finalizado
- Caso de uso "Escolha de nova chave de criptografia" é finalizado

Pré-condições

Nenhuma.

Caminhos alternativos

Nenhuma.

Ator

Usuário.

D.5. Caso de uso: Entra PIN

Token solicita ao usuário o seu PIN antes de permitir que alguma função crítica seja executada.

Seqüência de eventos

1. Solicita PIN ao usuário
2. Usuário digita PIN de 4 dígitos e pressiona a tecla "ENTRA"
3. Verifica se o PIN está correto (em caso de erro segue caminho alternativo)
4. *Token* zera o contador de erros e retorna que a operação foi realizada com sucesso

Evento iniciador

Os seguintes eventos podem iniciar o caso de uso:

- Usuário ligou o *token*
- Usuário deseja trocar a chave de criptografia
- Usuário deseja trocar o PIN

Pré-condições

Nenhuma.

Caminhos alternativos

Usuário digita o PIN incorreto:

1. Adiciona 1 ao contador de erros
2. Exibe mensagem de erro

3. Executa caso de uso "Verifica Bloqueio do Sistema"
4. Retorna Erro

Usuário digita PIN com menos de 4 dígitos:

1. Exibe mensagem de erro
2. Reinicia este caso de uso

Usuário utiliza a tecla LIMPA/CANCELA ao invés da tecla ENTRA:

1. Usuário pressiona Cancela com a tela sem nada digitado:
    a. Operação é cancelada avisando que o usuário desistiu
2. Usuário pressiona Cancela com a tela com parte de desafio digitado:
    a. O sistema limpa a tela

Ator

Usuário.

D.6. Caso de uso: Escolha de novo PIN

Usuário deseja trocar o seu PIN.

Seqüência de eventos

1. Executa o caso de uso "Entra PIN" (em caso de erro segue caminho alternativo)
2. Solicita novo PIN
3. Usuário digita o novo PIN de 4 dígitos e pressiona ENTRA (em caso de erro, segue caminho alternativo)
4. Solicita confirmação de PIN
5. Usuário confirma o PIN de 4 dígitos e pressiona ENTRA (em caso de erro, segue caminho alternativo)

6. Verifica se ambos os PIN digitados estão iguais (em caso de erro segue caminho alternativo)
7. Atualiza o PIN atual e finaliza com sucesso

Evento iniciador

Usuário se encontra na tela "ESCOLHA" e pressiona a tecla 4 do *token*.

Pré-condições

Nenhuma.

Caminhos alternativos

Se o caso de uso "Entra PIN" retorna Erro, o caso de uso atual é reiniciado.

Se ambos os PIN estão diferentes, o caso de uso é reiniciado.

Se usuário digita PIN com menos de 4 dígitos:

1. Exibe mensagem de erro
2. Solicita o PIN novamente

Se o usuário utiliza a tecla LIMPA/CANCELA ao invés da tecla ENTRA:

1. Usuário pressiona Cancela com a tela sem nada digitado:
    a. Operação é cancelada avisando que o usuário desistiu
2. Usuário pressiona Cancela com a tela com parte de desafio digitado:
    a. O sistema limpa a tela

Ator

Usuário.

D.7. Caso de uso: Escolha de nova chave de criptografia

Usuário precisa trocar a sua chave de criptografia de 32 dígitos.

Seqüência de eventos

1. Executa o caso de uso "Entra PIN"
2. *Token* verifica se é a primeira execução do sistema. Em caso afirmativo, o caso de uso "Escolha de novo PIN" é executado
3. *Token* solicita o primeiro grupo de dígitos da chave
4. Usuário digita os 4 primeiros dígitos do desafio e pressiona "ENTRA" (em caso de erro segue caminho alternativo)
5. *Token* solicita o próximo grupo de dígitos da chave
6. Usuário digita os 4 primeiros dígitos do desafio e pressiona "ENTRA" (em caso de erro segue caminho alternativo)
7. Passos 4 e 5 são repetidos até que seja digitado o oitavo grupo de dígitos
8. *Token* converte a entrada do usuário (*string*) em um número de 128 bits (16 bytes). Para isso o *token* aumenta a velocidade do *clock*.
9. *Token* guarda o valor convertido na memória
10. *Token* marca na memória que já foi escolhida uma nova chave de criptografia

Evento Iniciador

Existem duas possibilidades:

- Ainda não foi digitada uma chave de criptografia, isto é, primeiro uso do dispositivo
- Usuário se encontra na tela "ESCOLHA" e pressiona a tecla 5 do *token*

Pré-condições

Caso de uso "Ligar *token*" foi executado.

Caminhos alternativos

1. Se usuário digita bloco com menos de 4 dígitos:

   a. Exibe mensagem de erro
   b. Solicita o bloco novamente
2. Usuário pressiona LIMPA/CANCELA com a tela sem nada digitado
   a. Se o grupo de dígitos atual for o primeiro, o sistema cancela a execução do sistema reportando desistência do usuário. Nos demais casos o sistema retorna para o grupo de dígitos imediatamente anterior
3. Usuário pressiona LIMPA/CANCELA com a tela com parte de desafio digitado
   a. O sistema limpa a tela

Ator

Usuário.

D.8. Caso de uso: Geração de OTP

Usuário digita o desafio e o *token* processa a resposta do desafio (OTP).

Seqüência de eventos

1. *Token* solicita o primeiro grupo de dígitos do desafio
2. Usuário digita os 3 primeiros dígitos do desafio e pressiona "ENTRA"
3. *Token* solicita o segundo grupo de dígitos do desafio
4. Usuário digita os 3 dígitos seguintes do desafio e pressiona "ENTRA"
5. *Token* solicita o terceiro grupo de dígitos do desafio
6. Usuário digita os 3 últimos dígitos do desafio e pressiona "ENTRA"
7. *Token* processa o desafio
8. *Token* retorna a OTP para o usuário

Evento Iniciador

Usuário se encontra na tela "ESCOLHA" e pressiona a tecla 1 do *token*.

Pré-condições

Caso de uso "Escolha de nova chave de criptografia" já foi terminado com sucesso.

Caminhos alternativos

1. Se usuário digita bloco com menos de 3 dígitos:
    a. Exibe mensagem de erro
    b. Solicita o bloco novamente
2. Usuário pressiona LIMPA/CANCELA com a tela sem nada digitado
    a. Se o grupo de dígitos atual for o primeiro, o sistema cancela a execução do sistema reportando desistência do usuário. Nos demais casos o sistema retorna para o grupo de dígitos imediatamente anterior
3. Usuário pressiona LIMPA/CANCELA com a tela com parte de desafio digitado
    a. O sistema limpa a tela

Ator

Usuário.

D.9. Caso de uso: Autenticação

Usuário deseja gerar o código de autenticação para alguma transação.

Seqüência de eventos

1. *Token* solicita o primeiro grupo de dígitos do desafio
2. Usuário digita os primeiros dígitos do desafio e pressiona "ENTRA"
3. *Token* solicita o próximo grupo de dígitos do desafio
4. Usuário digita os dígitos seguintes do desafio e pressiona "ENTRA"

5. Os passos 3 e 4 são repetidos até o usuário digitar o 8º bloco, ou pressionar a tecla "ENTRA" duas vezes seguidas. Cada bloco tem no máximo 4 dígitos e no mínimo 1.
6. Calcula a autenticação da transação
7. Exibe o valor de autenticação de transação para o usuário

Evento Iniciador

Usuário se encontra na tela "ESCOLHA" e pressiona a tecla 2 do *token*.

Pré-condições

Caso de uso "Escolha de nova chave de criptografia" já foi terminado com sucesso.

Caminhos alternativos

1. Usuário pressiona LIMPA/CANCELA com a tela sem nada digitado
    a. Se o grupo de dígitos atual for o primeiro, o sistema cancela a execução do sistema reportando um erro. Nos demais casos o sistema retorna para o grupo de dígitos imediatamente anterior
2. Usuário pressiona LIMPA/CANCELA com a tela com algum dígito já na tela:
    a. O sistema limpa a tela

Ator

Usuário.

D.10. Caso de Uso: Desligar o *token*

Usuário deseja desligar o *token*.

Seqüência de eventos

Segue a mesma seqüência de eventos do caso de uso "Desligamento automático".

Evento Iniciador

Usuário se encontra na tela "ESCOLHA" e pressiona a tecla 0 do *token*.

Pré-condições

Nenhuma.

Caminhos alternativos

Nenhuma.

Ator

Usuário.