PUC-Rio – Departamento de Informática
Cursos: Ciência/Engenharia da Computação
Disciplina: INF1416 – Segurança da Informação
Prof.: Anderson Oliveira da Silva

Trabalho 3 – Cofre Digital (Digital Vault)
(entrega: 18/5/2025, 23:59h – todos os grupos)
(apresentações: 19/5/2025, 20/5/2025, 21/5/2025, 22/5/2025)

Construir um sistema em Java (plataforma JDK SE), chamado Cofre Digital, que utiliza um banco de dados relacional (ex: SQLLite, MySQL), um processo de autenticação multifator e controle de integridade, autenticidade e sigilo para proteger uma pasta de arquivos secretos.

Quando o cofre digital for executado pela primeira vez (processo de partida do sistema), sem nenhum usuário cadastrado no sistema, a aplicação deve fazer o cadastro do usuário administrador da pasta segura, conforme descrito na opção de cadastro. A frase secreta da chave privada do administrador deverá ser mantida em memória para que o sistema tenha acesso a essa chave privada durante a sua execução. Após esse cadastro inicial, o sistema deverá iniciar o processo de autenticação de usuários. Tipicamente, o administrador do sistema deverá ser o primeiro a entrar no sistema para fazer o cadastro dos demais usuários do sistema. Se o sistema for encerrado, a chave secreta do usuário deverá ser apagada da memória.

Quando o cofre digital for executado da segunda vez em diante (novo processo de partida do sistema), a aplicação deverá solicitar a frase secreta da chave privada do administrador, que deve passar pelo processo de validação, da mesma forma como realizado no processo de cadastro. Se a validação da chave privada for negativa, o sistema deve notificar o usuário do ocorrido e deve encerrar a execução do sistema. Se a validação da chave privada for positiva, o sistema deverá manter a frase secreta em memória e iniciar o processo de autenticação de usuários.

Na primeira etapa de autenticação, deve-se solicitar a identificação do usuário (*login name*) no sistema, que deve ser um e-mail válido. O e-mail do usuário deve ser coletado do seu respectivo certificado digital no momento do seu cadastramento no sistema. Se a identificação for inválida, o usuário deve ser apropriadamente avisado e o processo deve permanecer na primeira etapa. Se a identificação for válida e o acesso do usuário estiver bloqueado, o mesmo deve ser apropriadamente avisado e o processo deve permanecer na primeira etapa. Caso contrário, o processo deve seguir para a segunda etapa.

Na segunda etapa, deve-se verificar a senha pessoal do usuário (algo que ele conhece) que é fornecida através de um *teclado virtual numérico sobrecarregado* com cinco botões, cada um com dois números, que são distribuídos aleatoriamente e sem repetição entre todos os botões. As senhas pessoais são sempre formadas por oito, nove ou 10 números. A cada pressionamento de um botão, os números são redistribuídos aleatoriamente entre os cinco botões. Se a verificação da senha for negativa, o usuário deve ser apropriadamente avisado e o processo deve contabilizar um erro de verificação de senha pessoal. Após três erros consecutivos sem que ocorra uma verificação positiva entre os erros, deve-se seguir para a primeira etapa e o acesso do usuário deve ser bloqueado por 2 minutos (outros usuários poderão tentar ter acesso). Se a verificação for positiva, o processo deve seguir para a terceira etapa.

Na terceira e última etapa de autenticação, deve-se verificar o *token do usuário* (algo que ele possui) fornecido para o sistema através do aplicativo Google Authenticator. Esse aplicativo gera uma Time-based One Time Password (TOTP) com base nos RFCs 4226, 4648 e 6238. O RFC 6238 define o uso de um contador de tempo que conta a quantidade de intervalos de 30 segundos (time step in seconds) decorridos desde Janeiro 1, 1970, 00:00:00 GMT. E, o RFC 4226, define o uso do algoritmo HMAC-SHA1 para gerar um HMAC em função do contador de

tempo e de uma chave secreta de 160 bits (20 bytes). Por sua vez, o RFC 4648 define três formas de codificação de valores binários: BASE64, BASE32 e BASE16. A chave secreta de 160bits deverá ser codificada em BASE32 para exportação entre sistemas e decodificada para o cálculo do HMAC-SHA1. O código TOTP é um valor de 6 algarismos produzido a partir do truncamento do hash do HMAC-SHA1, conforme especificado no RFC 4226. O código TOTP é preenchido com zeros à esquerda quando necessário e muda seu valor a cada 30 segundos. Esse código TOTP deve ser fornecido pelo usuário na tela de autenticação do programa Cofre Digital, que, por sua vez, deve trabalhar com uma margem de erro de 30 segundos a menos e 30 segundos a mais. Para isso, deve calcular três valores de token: (i) um valor com a quantidade de intervalos de 30 segundos decorridos até o presente momento; (ii) outro valor com um intervalo de 30 segundos à menos; e (iii) outro valor com um intervalo de 30 segundos a mais. Se a verificação for negativa para os três valores calculados, o usuário deve ser apropriadamente avisado e o processo deve contabilizar um erro de verificação de token, retornando para o início da terceira etapa. Após três erros consecutivos sem que ocorra uma verificação válida do token, deve-se seguir para a primeira etapa e o acesso do usuário deve ser bloqueado por 2 minutos (outros usuários poderão tentar ter acesso). Se a verificação for positiva, o processo deve permitir acesso ao sistema.

Após um processo de autenticação positivo, o sistema deve apresentar uma tela com informações e menus distintos em função do grupo do usuário no sistema. Para organizar a apresentação, a tela é dividida em três partes: cabeçalho, corpo 1 e corpo 2. Para o grupo administrador, o sistema deve apresentar a Tela Principal com as informações do usuário no cabeçalho, o total de acessos do usuário no corpo 1, e o Menu Principal no corpo 2, conforme abaixo:

Cabeçalho
Login: login_name_do_usuário
Grupo: grupo_do_usuário
Nome: nome_do_usuário

Corpo 1
Total de acessos do usuário: total_de _acessos_do_usuario

Corpo 2
Menu Principal:

1 – Cadastrar um novo usuário
2 – Consultar pasta de arquivos secretos do usuário
3 – Sair do Sistema

Quando a opção 1 for selecionada, a Tela de Cadastro deve ser apresentada com o mesmo cabeçalho da Tela Principal, com o total de usuários do sistema no corpo 1 e com o Formulário de Cadastro no corpo 2, conforme abaixo:

Cabeçalho
Login: login_name_do_usuário
Grupo: grupo_do_usuário
Nome: nome_do_usuário

Corpo 1
Total de usuários do sistema: total_de_usuários

Corpo 2
Formulário de Cadastro:

– Caminho do arquivo do certificado digital: <campo com 255 caracteres>
– Caminho do arquivo da chave privada: <campo com 255 caracteres>
– Frase secreta: <campo de 255 caracteres>
– Grupo: <lista de opções: Administrador e Usuário>
– Senha pessoal: <campo com 10 caracteres>
– Confirmação senha pessoal: <campo com 10 caracteres>

<Botão Cadastar> <Botão Voltar de Cadastrar para o Menu Principal>

Os valores entrados nos campos devem ser criticados adequadamente. As senhas pessoais são sempre formadas por oito, nove ou dez números formados por dígitos de 0 a 9. Não podem ser aceitas sequências de números repetidos. Quando o Botão Cadastrar for pressionado, o sistema deve conferir se as senhas digitadas são as mesmas e apresentar uma tela de confirmação com os seguintes campos do certificado digital: Versão, Série, Validade, Tipo de Assinatura, Emissor, Sujeito (Friendly Name) e E-mail. Se os dados forem confirmados, deve-se incluir o usuário no sistema apenas se o login name (e-mail do usuário) for único, notificando o usuário em caso de erro. O nome do usuário e o login name devem ser extraídos do campo de Sujeito do certificado.

A *frase secreta* da chave privada deve ser testada e a chave privada deve ser verificada com a validação da assinatura digital de um array aleatório de 8192 bytes com a chave pública que consta no certificado digital fornecido. Se a verificação for negativa, o usuário deve ser apropriadamente avisado e o cadastrado do usuário não deve ser realizado. Se a assinatura digital for verificada com sucesso, então o sistema deve armazenar a chave privada no seu formato criptografado (binário) e o certificado digital no seu formato PEM (Privacy-Enhanced Mail) na base de dados de forma associada ao UID do usuário na tabela *Chaveiro*. O registro do par chave privada e certificado digital deve receber uma identificação única de KID. O KID deve também ser armazenado no registro do usuário, na tabela *Usuarios*.

A senha pessoal deve ser armazenada no registro do usuário, na tabela *Usuarios* do banco de dados, conforme o requisito para armazenamento de senhas.

O requisito para armazenamento da senha pessoal é o armazenamento padrão do hash calculado pela função bcrypt, que armazena a *versão* do algoritmo usado pelo bcrypt (no caso do trabalho, versão 2y), o *custo* das iterações do bcrypt (**no caso do trabalho, valor 8**), o *SALT* codificado em BASE64 (diferente do RFC4648) e o *HASH* codificado em BASE64 (diferente do RFC4648), totalizando 60 caracteres, conforme mostrado a seguir:

Valor_Armazenado = $version$cost$BASE64(salt)BASE64(hash)

Exemplo de armazenamento da senha “13572468”:

$2y$08$ajDZdV0XFoYiLRE0ZrSx6OmRbhhB5qM.Ap6c.0LqRu2cIY1jpytuO

Onde,
2y = versão do algoritmo usado na função bcrypt.
08 = custo do bcrypt (2^8 iterações).
ajDZdV0XFoYiLRE0ZrSx6O = salto codificado em BASE64.
mRbhhB5qM.Ap6c.0LqRu2cIY1jpytuO = hash codificado em BASE64.

O provider BouncyCastle fornece a classe OpenBSDBCrypt. Essa classe possui o método *generate*, que gera o hash brcrypt versão 2y, e o método *checkPassword*, que compara um hash bcrypt com uma senha em texto plano.

A *chave secreta* usada no algoritmo que produz o código TOTP é formada por 20 bytes aleatórios para cada usuário do sistema e deve ser gerada no momento do cadastro do usuário. Essa chave secreta deve ser codificada em BASE32 (RFC4648) e armazenada no registro do usuário, na tabela *Usuarios*, de forma criptografada com o algoritmo simétrico AES/ECB/PKCS5Padding. A chave AES deve ter 256 bits e deve ser gerada a partir da *senha pessoal do usuário* cadastrada no programa Cofre Digital. O SHA1PRNG deve ser utilizado para fazer a geração da chave simétrica. A classe BASE32 é fornecida no anexo e a classe TOTP deve ser implementada usando, APENAS, as classes de apoio javax.crypto.Mac, javax.crypto.spec.SecretKeySpec, java.util.Date e BASE32. A estrutura da classe TOTP é mostrada no anexo desta especificação.

ATENÇÂO: Não serão aceitas classes implementadas por terceiros para a geração do código TOTP.

A forma codificada em BASE32 também deve ser informada ao usuário para a realização do cadastro no aplicativo Google Authenticator. Opcionalmente, pode-se apresentar um QRCode com a representação da seguinte URI:

otpauth://TYPE/LABEL?PARAMETERS

Ex: otpauth://totp/Cofre%20Digital:admin@inf1416.puc-rio.br?secret=JXXMGK33L7S3H3JOY5GMUXC7G7ASJHTD

Onde,

totp é o tipo do One-Time Password, ou seja, Time-based OTP (TOTP).
Cofre%20Digital é o identificador do programa no qual a TOTP deve ser usada.
admin@inf1416.puc-rio.br é a identificação da conta do usuário.
JXXMGK33L7S3H3JOY5GMUXC7G7ASJHTD é o segredo do código TOTP em BASE32

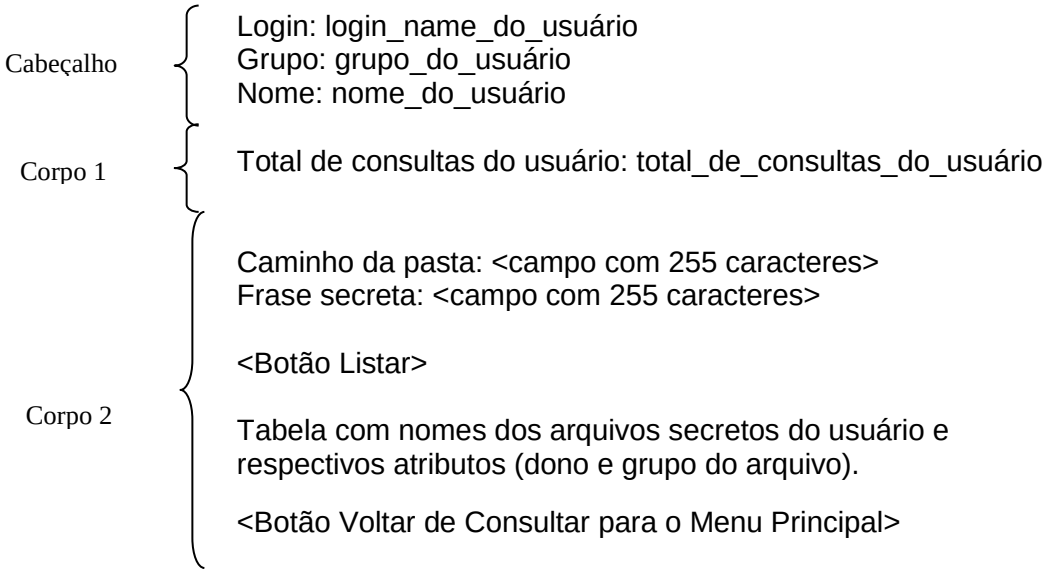
Se o cadastro for efetivado, deve-se retornar à Tela de Cadastro com o formulário vazio. Caso contrário, deve-se retornar à Tela de Cadastro com o formulário preenchido com os dados fornecidos. Quando o Botão Voltar de Cadastrar para o Menu Principal for pressionado, deve-se retornar à Tela Principal.

O arquivo da chave privada é binário e será fornecido em um token (por exemplo, pendrive). O arquivo do certificado digital é ASCII codificado em BASE64, no formato PEM (Privacy-Enhanced Mail) e padrão X.509. Por questão de segurança, o arquivo da chave privada está criptografado com AES/ECB/PKCS5Padding. A chave AES deve ter 256 bits e deve ser gerada a partir de uma FRASE SECRETA do usuário dono da chave privada. O Java oferece classes prontas para gerar a chave simétrica com base em uma FRASE SECRETA (*KeyGenerator* e *SecureRandom*). O PRNG para geração da chave AES é o **SHA1PRNG**.

A chave privada decriptada usa o padrão PKCS8 e o certificado digital usa o padrão X.509, ambos codificados em BASE64. O Java oferece classes prontas para manipular com os dados codificados que estão armazenados nesses arquivos, respectivamente, as classes *PKCS8EncodedKeySpec*, *X509Certificate* e *Base64*. A partir da decodificação dos dados dos arquivos feita por essas classes, o Java também possibilita a restauração das chaves privadas e públicas com as classes *KeyFactory*, *PrivateKey* e *PublicKey*, e do certificado digital com a classe *CertificateFactory*.

O banco de dados é organizado em cinco tabelas: *Usuarios*, *Chaveiro*, *Grupos*, *Mensagens* e *Registros*. A tabela *Usuarios* deve guardar as informações pessoais dos usuários, inclusive o valor armazenado da senha pessoal do usuário, conforme o requisito de armazenamento de senhas (cada usuário deve ter um UID único). O certificado digital e a chave privada do usuário devem ser armazenados na tabela *Chaveiro* (cada par certificado digital e chave privada possui um KID único e está associado a um único UID). A tabela *Grupos* deve armazenar os grupos do sistema (cada grupo possui um GID, número decimal único de identificação do grupo). A tabela *Mensagens* deve armazenar as mensagens da Tabela de Mensagens de Registro (cada mensagem deve ter um MID único). E, a tabela de *Registros* deve armazenar os registros relacionados ao uso do sistema, identificando a data e hora de um registro, relacionando com um usuário quando necessário (cada registro deve ter um RID único e um MID).

Quando a opção 2 for selecionada, a Tela de Consultar Pasta de Arquivos Secretos do Usuário deve ser apresentada com o mesmo cabeçalho e corpo 1 da Tela Principal, e com o total de consultas feitas pelo usuário corrente no corpo 2, conforme abaixo:



Quando o Botão Listar for pressionado, a frase secreta deve ser validade da mesma forma que foi previamente realizada durante o cadastro do usuário. Em caso de validação negativa, o usuário deve ser notificado de forma apropriada em função do erro. Se a validação for positiva, deve-se decriptar o arquivo de índice da pasta fornecida (cifra AES, modo ECB e enchimento PKCS5), chamado index.enc; verificar a integridade e autenticidade do arquivo de índice; e listar o conteúdo do arquivo de índice apresentando APENAS os atributos dos arquivos (nome código, nome, dono e grupo) do usuário ou do grupo do usuário. O envelope digital do arquivo de índice é armazenado no arquivo index.env (protege a semente SHA1PRNG que gera a chave secreta AES) e a assinatura digital do arquivo de índice é armazenada no arquivo index.asd (representação binária da assinatura digital). **O envelope digital e a assinatura digital do índice da pasta são gerados com as respectivas chaves assimétricas do usuário administrador e as classes Cipher e Signature.** O arquivo de índice decriptado possui zero ou mais linhas formatadas da seguinte forma:

NOME_CODIGO_DO_ARQUIVO<SP>NOME_SECRETO_DO_ARQUIVO<SP>DONO_ARQUIVO <SP><GRUPO_ARQUIVO><EOL>

Onde:

NOME_CODIGO_DO_ARQUIVO: caracteres alfanuméricos (nome código do arquivo).
NOME_SECRETO_DO_ARQUIVO: caracteres alfanuméricos (nome original do arquivo).
DONO_ARQUIVO: caracteres alfanuméricos (atributo do arquivo).
GRUPO_ARQUIVO: caracteres alfanuméricos (atributo do arquivo).
<SP> = caractere espaço em branco.
<EOL> = caractere nova linha (\n).

Observação: O arquivo de índice da pasta pertence ao administrador do sistema.

Quando o nome secreto de um arquivo da lista apresentada for selecionado, o sistema deve verificar se o usuário pode ou não acessar o arquivo. A política de controle de acesso é simples: o usuário só pode acessar um arquivo se for o dono do mesmo. Em caso afirmativo, o sistema deve (i) decriptar o arquivo secreto (cifra AES, modo ECB e enchimento PKCS5) selecionado, notificando o usuário sobre eventuais erros de integridade, autenticidade e sigilo; e (ii) gravar os dados decriptados em um novo arquivo com o nome secreto. Caso contrário, o sistema deve notificar o usuário que ele não tem permissão de acesso.

O nome do arquivo criptografado usa o nome código do arquivo e a extensão *.enc*. A assinatura digital, gerada com a classe *Signature* e a chave assimétrica do usuário, é mantida em um arquivo com o nome código e a extensão *.asd* (representação binária da assinatura digital). O envelope

digital do arquivo é mantido em um arquivo com o nome código e a extensão *.env* (protege a semente SHA1PRNG que gera a chave secreta AES). Quando o Botão Voltar de Consultar para o Menu Principal for pressionado, deve-se retornar à Tela Principal.

Quando a opção 3 for selecionada, a Tela de Saída deve ser apresentada com o mesmo cabeçalho da Tela Principal. O corpo 1 deve apresentar o total de acessos do usuário corrente e o corpo 2 deve ser apresentado conforme abaixo:

Cabeçalho	Login: login_name_do_usuário Grupo: grupo_do_usuário Nome: nome_do_usuário
Corpo 1	Total de acessos do usuário: total_de _acessos_do_usuario
Corpo 2	Saída do sistema: Mensagem de saída. <Botão Encerrar Sessão> <Botão Encerrar Sistema> <Botão Voltar de Sair para o Menu Principal>

O sistema deve apresentar a mensagem de saída “Pressione o botão Encerrar Sessão ou o botão Encerrar Sistema para confirmar.” e os três botões. Quando o Botão Encerrar Sessão for pressionado, deve-se encerrar a sessão do usuário e iniciar o processo de autenticação de usuário. Quando o Botão Encerrar Sistema for pressionado, deve-se encerrar a execução do sistema. Se o botão <Voltar de Sair para o Menu Principal> for pressionado, deve-se retornar à Tela Principal.

Para o grupo usuário, o sistema deve funcionar de forma equivalente. Porém, o cabeçalho das telas deve apresentar o grupo como Usuário e o Menu Principal não deve apresentar a opção Cadastrar um Novo Usuário. O corpo 2 deve continuar apresentando a mensagem “Total de acessos do usuário: total_de_acessos_do_usuário”.

Cada uma das operações executadas pelo sistema deve ser registrada em uma tabela de Registros no banco de dados, armazenando, pelo menos, a data e hora do registro, o código da mensagem, quando possível, a identificação do usuário corrente e do arquivo selecionado para decriptação. Não é permitido armazenar o texto das mensagens dos registros nessa tabela. As mensagens e seus respectivos códigos devem ser armazenadas na tabela Mensagens do banco de dados. **Os registros devem ser visualizados em ordem cronológica apenas por um programa de apoio (logView) que deve também ser implementado.**

As mensagens de registro e os respectivos códigos estão listados na Tabela de Mensagens de Registro em anexo.

O **prazo de submissão do projeto** deste trabalho, com todos os arquivos fontes em Java, no sistema de EAD da PUC-Rio, é dia **18/5/2024, 23:59h**. O prazo máximo para submissão é dia **19/5/2024, 11:59h**. Cada integrante do grupo deve fazer uma submissão.

Tabela de Mensagens de Registro	
1001	Sistema iniciado.
1002	Sistema encerrado.
1003	Sessão iniciada para <login_name>.
1004	Sessão encerrada para <login_name>.
1005	Partida do sistema iniciada para cadastro do administrador.
1006	Partida do sistema iniciada para operação normal pelos usuários.
2001	Autenticação etapa 1 iniciada.
2002	Autenticação etapa 1 encerrada.
2003	Login name <login_name> identificado com acesso liberado.
2004	Login name <login_name> identificado com acesso bloqueado.
2005	Login name <login_name> não identificado.
3001	Autenticação etapa 2 iniciada para <login_name>.
3002	Autenticação etapa 2 encerrada para <login_name>.
3003	Senha pessoal verificada positivamente para <login_name>.
3004	Primeiro erro da senha pessoal contabilizado para <login_name>.
3005	Segundo erro da senha pessoal contabilizado para <login_name>.
3006	Terceiro erro da senha pessoal contabilizado para <login_name>.
3007	Acesso do usuario <login_name> bloqueado pela autenticação etapa 2.
4001	Autenticação etapa 3 iniciada para <login_name>.
4002	Autenticação etapa 3 encerrada para <login_name>.
4003	Token verificado positivamente para <login_name>.
4004	Primeiro erro de token contabilizado para <login_name>.
4005	Segundo erro de token contabilizado para <login_name>.
4006	Terceiro erro de token contabilizado para <login_name>.
4007	Acesso do usuario <login_name> bloqueado pela autenticação etapa 3.
5001	Tela principal apresentada para <login_name>.
5002	Opção 1 do menu principal selecionada por <login_name>.
5003	Opção 2 do menu principal selecionada por <login_name>.
5004	Opção 3 do menu principal selecionada por <login_name>.
6001	Tela de cadastro apresentada para <login_name>.
6002	Botão cadastrar pressionado por <login_name>.
6003	Senha pessoal inválida fornecida por <login_name>.
6004	Caminho do certificado digital inválido fornecido por <login_name>.
6005	Chave privada verificada negativamente para <login_name> (caminho inválido).
6006	Chave privada verificada negativamente para <login_name> (frase secreta inválida).
6007	Chave privada verificada negativamente para <login_name> (assinatura digital inválida).
6008	Confirmação de dados aceita por <login_name>.
6009	Confirmação de dados rejeitada por <login_name>.
6010	Botão voltar de cadastro para o menu principal pressionado por <login_name>.
7001	Tela de consulta de arquivos secretos apresentada para <login_name>.
7002	Botão voltar de consulta para o menu principal pressionado por <login_name>.
7003	Botão Listar de consulta pressionado por <login_name>.
7004	Caminho de pasta inválido fornecido por <login_name>.
7005	Arquivo de índice decriptado com sucesso para <login_name>.
7006	Arquivo de índice verificado (integridade e autenticidade) com sucesso para <login_name>.
7007	Falha na decriptação do arquivo de índice para <login_name>.
7008	Falha na verificação (integridade e autenticidade) do arquivo de índice para <login_name>.
7009	Lista de arquivos presentes no índice apresentada para <login_name>.
7010	Arquivo <arq_name> selecionado por <login_name> para decriptação.
7011	Acesso permitido ao arquivo <arq_name> para <login_name>.
7012	Acesso negado ao arquivo <arq_name> para <login_name>.
7013	Arquivo <arq_name> decriptado com sucesso para <login_name>.
7014	Arquivo <arq_name> verificado (integridade e autenticidade) com sucesso para <login_name>.
7015	Falha na decriptação do arquivo <arq_name> para <login_name>.
7016	Falha na verificação (integridade e autenticidade) do arquivo <arq_name> para <login_name>.
8001	Tela de saída apresentada para <login_name>.
8002	Botão encerrar sessão pressionado por <login_name>.
8003	Botão encerrar sistema pressionado por <login_name>.
8004	Botão voltar de sair para o menu principal pressionado por <login_name>.

Estrutura da classe TOTP:

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Date;

public class TOTP {
    private byte [] key = null;
    private long timeStepInSeconds = 30;

    // Construtor da classe. Recebe a chave secreta em BASE32 e o intervalo
    // de tempo a ser adotado (default = 30 segundos). Deve decodificar a
    // chave secreta e armazenar em key. Em caso de erro, gera Exception.
    public TOTP(String base32EncodedSecret, long timeStepInSeconds)
    throws Exception {

    }

    // Recebe o HASH HMAC-SHA1 e determina o código TOTP de 6 algarismos
    // decimais, prefixado com zeros quando necessário.
    private String getTOTPCodeFromHash(byte[] hash) {

    }

    // Recebe o contador e a chave secreta para produzir o hash HMAC-SHA1.
    private byte[] HMAC_SHA1(byte[] counter, byte[] keyByteArray) {

    }

    // Recebe o intervalo de tempo e executa o algoritmo TOTP para produzir
    // o código TOTP. Usa os métodos auxiliares getTOTPCodeFromHash e HMAC_SHA1.
    private String TOTPCode(long timeInterval) {

    }

    // Método que é utilizado para solicitar a geração do código TOTP.
    public String generateCode() {

    }

    // Método que é utilizado para validar um código TOTP (inputTOTP).
    // Deve considerar um atraso ou adiantamento de 30 segundos no
    // relógio da máquina que gerou o código TOTP.
    public boolean validateCode(String inputTOTP) {

    }
}
```

Classe BASE32:

```
import java.io.*;

/**
 * Routines for converting between Strings of Base32-encoded data and arrays
 * of binary data.  This currently supports the Base32 and Base32hex alphabets
 * specified in RFC 4648, sections 6 and 7.
 *
 * @author Brian Wellington
 */

public class Base32 {

      public static class Alphabet {
            private Alphabet() {}

            public static final String BASE32 =
                        "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567=";
            public static final String BASE32HEX =
                        "0123456789ABCDEFGHIJKLMNOPQRSTUV=";
      };
/**
 * Routines for converting between Strings of Base32-encoded data and arrays
 * of binary data.  This currently supports the Base32 and Base32hex alphabets
 * specified in RFC 4648, sections 6 and 7.
 *
 * @author Brian Wellington
 */

      private String alphabet;
      private boolean padding, lowercase;

      /**
       * Creates an object that can be used to do Base32 conversions.
       * @param alphabet Which alphabet should be used
       * @param padding Whether padding should be used
       * @param lowercase Whether lowercase characters should be used.
       * default parameters (The standard Base32 alphabet, no padding,
uppercase)
       */
      public
      Base32(String alphabet, boolean padding, boolean lowercase) {
            this.alphabet = alphabet;
            this.padding = padding;
            this.lowercase = lowercase;
      }

      static private int
      blockLenToPadding(int blocklen) {
            switch (blocklen) {
                  case 1:
                        return 6;
                  case 2:
                        return 4;
                  case 3:
                        return 3;
                  case 4:
                        return 1;
                  case 5:
                        return 0;
                  default:
                        return -1;
            }
      }
```

```
static private int
paddingToBlockLen(int padlen) {
	switch (padlen) {
		case 6:
			return 1;
		case 4:
			return 2;
		case 3:
			return 3;
		case 1:
			return 4;
		case 0:
			return 5;
		default :
			return -1;
	}
}

/**
 * Convert binary data to a Base32-encoded String
 *
 * @param b An array containing binary data
 * @return A String containing the encoded data
 */
public String
toString(byte [] b) {
	ByteArrayOutputStream os = new ByteArrayOutputStream();

	for (int i = 0; i < (b.length + 4) / 5; i++) {
		short s[] = new short[5];
		int t[] = new int[8];

		int blocklen = 5;
		for (int j = 0; j < 5; j++) {
			if ((i * 5 + j) < b.length)
				s[j] = (short) (b[i * 5 + j] & 0xFF);
			else {
				s[j] = 0;
				blocklen--;
			}
		}
		int padlen = blockLenToPadding(blocklen);

		// convert the 5 byte block into 8 characters (values 0-31).

		// upper 5 bits from first byte
		t[0] = (byte) ((s[0] >> 3) & 0x1F);
		// lower 3 bits from 1st byte, upper 2 bits from 2nd.
		t[1] = (byte) (((s[0] & 0x07) << 2) | ((s[1] >> 6) & 0x03));
		// bits 5-1 from 2nd.
		t[2] = (byte) ((s[1] >> 1) & 0x1F);
		// lower 1 bit from 2nd, upper 4 from 3rd
		t[3] = (byte) (((s[1] & 0x01) << 4) | ((s[2] >> 4) & 0x0F));
		// lower 4 from 3rd, upper 1 from 4th.
		t[4] = (byte) (((s[2] & 0x0F) << 1) | ((s[3] >> 7) & 0x01));
		// bits 6-2 from 4th
		t[5] = (byte) ((s[3] >> 2) & 0x1F);
		// lower 2 from 4th, upper 3 from 5th;
		t[6] = (byte) (((s[3] & 0x03) << 3) | ((s[4] >> 5) & 0x07));
		// lower 5 from 5th;
		t[7] = (byte) (s[4] & 0x1F);

		// write out the actual characters.
		for (int j = 0; j < t.length - padlen; j++) {
```

```
                        char c = alphabet.charAt(t[j]);
                        if (lowercase)
                              c = Character.toLowerCase(c);
                        os.write(c);
                  }

                  // write out the padding (if any)
                  if (padding) {
                        for (int j = t.length - padlen; j < t.length; j++)
                              os.write('=');
                  }
            }

            return new String(os.toByteArray());
      }

      /**
       * Convert a Base32-encoded String to binary data
       *
       * @param str A String containing the encoded data
       * @return An array containing the binary data, or null if the string is
invalid
       */
      public byte[]
      fromString(String str) {
            ByteArrayOutputStream bs = new ByteArrayOutputStream();
            byte [] raw = str.getBytes();
            for (int i = 0; i < raw.length; i++)
            {
                  char c = (char) raw[i];
                  if (!Character.isWhitespace(c)) {
                        c = Character.toUpperCase(c);
                        bs.write((byte) c);
                  }
            }

            if (padding) {
                  if (bs.size() % 8 != 0)
                        return null;
            } else {
                  while (bs.size() % 8 != 0)
                        bs.write('=');
            }

            byte [] in = bs.toByteArray();

            bs.reset();
            DataOutputStream ds = new DataOutputStream(bs);

            for (int i = 0; i < in.length / 8; i++) {
                  short[] s = new short[8];
                  int[] t = new int[5];

                  int padlen = 8;
                  for (int j = 0; j < 8; j++) {
                        char c = (char) in[i * 8 + j];
                        if (c == '=')
                              break;
                        s[j] = (short) alphabet.indexOf(in[i * 8 + j]);
                        if (s[j] < 0)
                              return null;
                        padlen--;
                  }
                  int blocklen = paddingToBlockLen(padlen);
                  if (blocklen < 0)
```

```
                        return null;

                  // all 5 bits of 1st, high 3 (of 5) of 2nd
                  t[0] = (s[0] << 3) | s[1] >> 2;
                  // lower 2 of 2nd, all 5 of 3rd, high 1 of 4th
                  t[1] = ((s[1] & 0x03) << 6) | (s[2] << 1) | (s[3] >> 4);
                  // lower 4 of 4th, high 4 of 5th
                  t[2] = ((s[3] & 0x0F) << 4) | ((s[4] >> 1) & 0x0F);
                  // lower 1 of 5th, all 5 of 6th, high 2 of 7th
                  t[3] = (s[4] << 7) | (s[5] << 2) | (s[6] >> 3);
                  // lower 3 of 7th, all of 8th
                  t[4] = ((s[6] & 0x07) << 5) | s[7];

                  try {
                        for (int j = 0; j < blocklen; j++)
                              ds.writeByte((byte) (t[j] & 0xFF));
                  }
                  catch (IOException e) {
                  }
            }

            return bs.toByteArray();
      }

}
```