UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

José Manuel Ramírez Núñez

IBE Aplicado à Criptografia Temporal

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação.

Adriana Elissa Notoya
Orientadora

Florianópolis, julho de 2004

IBE Aplicado à Criptografia Temporal

José Manuel Ramírez Núñez

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do grau de Bacharel em Ciência da Computação e aprovada em sua forma final pela comissão julgadora.

Adriana Elissa Notoya
Orientador

Prof. Dr. Ricardo Felipe Custódio
Co-orientador

Fernando Carlos Pereira

Júlio da Silva Dias

Aos meus pais, Mirtha e José.

Agradecimentos

A Deus, aos meus pais, a minha namorada, a Ale, a Kevin, às minhas irmãs, ao Prof. Custódio, a meu colega Leonardo Neves, a Notoya, a Pereira, a Dias, a Harrison, a Everson, a todos meus amigos, meus colegas e meus familiares. Todos me ajudaram de alguma maneira. Muchas Gracias!

Sumário

Símbolos e abreviações

AC: Autoridade Certificadora;

ACT: Autoridade Certificadora Temporal;

Alice, Beto e Carlos: Nomes ficticios de pessoas ou entidades, onde Alice e Beto representam as entidades que desejam estabelecer um canal de comunicação seguro, e Carlos é um intruso, ou o criptanalista que deseja ter acesso a informação não autorizada para ele;

Chave: Conjunto ou sequência de *bits*, usado para cifrar e decifrar uma mensagem;

Cifrar: Operação de transformar um texto legível em texto cifrado, correspondente ao termo *cifragem*;

Criptanálise: Arte e ciência que estuda métodos para analisar textos cifrados, com o objetivo de decifra-los. Um criptanalise consiste em tentar recuperar um texto cifrado sem previamente ter conhecimento da chave, estudando as vulnerabilidades de um sistema criptografico;

Criptografia tradicional: todo tipo de criptografia que não seja IBE;

Decifrar: Operação inversa a cifrar. Visa recuperar um texto legível a partir de um texto cifrado, através do uso da chave apropriada. Correspondente ao termo *decifragem*;

IBE: Criptografia Baseada em Identidades;

ICPT: Infra-estrutura de Chaves Públicas Temporais.

PKG: Gerador de chaves particulares. Tambem conhecido como autoridade de confiança TA, ou terceira parte TP;

Texto Cifrado, Texto Ilegível: Resultado do processo de cifragem aplicado a um texto legível, com a finalidade de torná-lo ilegível a pessoas ou entidades não autorizadas;

Texto Limpo, Texto Claro, Texto Plano, Texto Legível, m: Sinônimos de texto legível;

Texto Original: Mensagem ou informação legível para qualquer pessoa ou entidade, e cuja privacidade se deseja salvaguardar;

Abstract

Digital documents are more and more important in daily trades. Because of the advantages offered by the internet, the companies, public and private institutions began to chance the traditional documents for the digital ones. However, it is important to keep the same requirement that the traditional documents attended, like the authenticity, integrity, not-rejection and secrecy. The secrecy of temporal eletronic documents, a fundamental requirement in auctions and wills, usually consists in keeping the secret document, or its decipher key in custody of a trustful entity. This results in hight storage and processing costs, besides of the need of a great security structure by this entity. In this paper it is showed a solution that uses the Identity-Based Encryption to create the decipher keys only in the moment of the secrecy break, by the use of traditional assimetric cryptography to keep a safe channel between the entities that are participating of the communication. This approach requires lass efforts by the reliable entity responsible by the secrecy, simplifying the use of the temporal cryptography system.

Resumo

Documentos digitais são cada vez mais importantes nas transações do dia a dia. Com as facilidades que a Internet oferece, as empresas e instituições públicas e privadas passaram a trocar o papel impresso pelo documento digital. Entretanto, é importante manter os mesmos requisitos dos documentos em papéis para os documentos eletrônicos, tais como: a autenticidade, a integridade, o não-repúdio e o sigilo. O sigilo temporal de documentos eletrônicos, requisito fundamental em aplicações como leilões, testamentos e licitações públicas, normalmente consiste em manter o documento sigiloso, ou a sua chave de deciframento, sob a custódia de uma terceira entidade confiável. Isto implica em altos custos de armazenamento e processamento por parte desta entidade, além da robusta infra-estrutura de segurança por trás dela. Neste trabalho é apresentada uma solução que utiliza a criptografia baseada em identidade para gerar as chaves de deciframento somente no momento da quebra do sigilo, usando criptografia assimétrica tradicional para manter um canal seguro entre as entidades participantes da comunicação. Esta abordagem poupa esforços à entidade de confiança responsável pelo sigilo, simplificando o uso do complicado sistema de criptografia temporal.

Capítulo 1

Introdução

Documentos digitais são cada vez mais importantes nas transações do dia a dia. Com as facilidades que a Internet oferece, as empresas e instituições públicas e privadas passaram a trocar o papel impresso pelo documento digital. Entretanto, é importante manter os mesmos requisitos dos documentos em papéis para os documentos eletrônicos, tais como: a autenticidade, a integridade, o não-repúdio e o sigilo [Custódio et al, 2003]. Os três primeiros requisitos podem ser alcançados com o uso de técnicas da assinatura digital. Já o sigilo, embora seja um requisito essencial em muitas aplicações, ainda carece de soluções apropriadas, principalmente relacionados ao sigilo temporal.

O sigilo temporal visa garantir a confidencialidade de um documento por um determinado período de tempo. Aplicações como leilões, licitações públicas e testamentos são alguns exemplos em que este requisito é crítico.

Uma das formas de alcançar o sigilo temporal é através da submissão do documento a uma terceira parte confiável responsável pela custódia do documento até o momento da quebra do sigilo. Esta técnica é *"perigosa e de alto custo, pois é necessário confiar na terceira parte tanto do ponto de vista de sua honestidade quanto da sua capacidade de manter íntegro e inacessível o documento até a solicitação por alguma entidade autorizada"*, [Custódio et al, 2003].

Outra forma é através de técnicas de criptografia simétrica e assimétrica. Nesta abordagem, o interessado em manter o documento sigiloso cifra o documento e mantém a chave sob sigilo até o momento de liberação do documento. Porém, se a chave for perdida, o documento não poderá ser recuperado. Uma variante deste esquema é a cifragem dos documentos com chaves públicas cujas respectivas chaves privadas de deciframento não sejam geradas senão até o momento marcado para a liberação do documento.

A criptografia baseada em identidades (IBE) possui como principal característica a facilidade no gerenciamento de chaves criptográficas, permitindo cifrar documentos com identificadores como sendo chaves públicas. Um exemplo de identificador pode ser um endereço de correio eletrônico, ou mesmo uma data e hora. Estas chaves públicas podem não possuir ainda as suas respectivas chaves privadas no momento do seu uso, chaves privadas que somente serão geradas no momento em que forem solicitadas. Estas características fazem com que a criptografia baseada em identidades possa ser aplicada à criptografia temporal, onde poderemos cifrar os documentos com a chave pública definida por uma data e hora no futuro, momento que uma vez atingido, permitirá a geração e liberação de uma chave privada correspondente à data e hora especificada no documento.

1.1- Objetivos

1.1.1- Objetivo Geral

O objetivo principal desta monografia é a apresentação de um protótipo de criptografia temporal utilizando criptografia baseada em identidades.

1.1.2- Objetivos Específicos

- Realizar uma revisão bibliográfica sobre criptografia baseada em identidades;
- Estudar esquemas de criptografia temporal;
- Estudar ferramentas de criptografia simétrica e assimétrica;
- Implementar um protótipo de criptografia temporal sob o esquema da criptografia baseada em identidades.

1.2- Organização

Este trabalho apresenta os conceitos de segurança computacional no capítulo 2, descrevendo os conceitos mais relevantes, como criptografia, tipos de criptografia, assinatura digital e autoridade certificadora.

O capítulo 3 descreve em forma básica a criptografia baseada em identidade, algumas aplicacões, suas vantagens e desvantagens, finalizando com um exemplo de implementação.

O capítulo 4 apresenta a criptografia temporal, descrevendo os seus conceitos e dando uma breve descrição das abordagens propostas, dando ênfase ao projeto Autoridade Certificadora Temporal do LabSEC.

No capítulo 5 é apresentado um protótipo de criptografia temporal baseada em IBE, e que é assunto principal deste trabalho.

No capítulo 6 encontram-se as conclusões, considerações da implantação do IBE dentro da ACT, e umas sugestões para trabalhos futuros.

Capítulo 2

Conceitos Fundamentais

Este capítulo tem como objetivo dar uma visão geral dos conceitos necessários ao entendimento deste trabalho. Veremos os conceitos básicos da criptografia, seus tipos e aplicações, assim como alguns exemplos das técnicas mais utilizadas.

2.1- Criptografia: Histórico e definição

Do grego *Kryptos* (ocultar) e *grafos* (escrever). Significa, literalmente, "escrita oculta". O seu uso data do antigo Egito (onde os faraós mandavam ocultar informações sobre os seus tesouros) e foi fundamental nos capítulos mais delicados da história da humanidade. Usada na antiga Roma por Júlio César (50 a.C.), foi sempre arma de militares, diplomáticos e espiões, e é defesa das comunicacões e nos dados que circulam na grande rede que une o mundo, a Internet.

A proteção da informação é feita mudando a sua forma, por meio do processo chamado cifragem (ou transformação criptográfica), na qual um texto legível é tornado ilegível. Analogicamente, chama-se decifragem ao processo inverso, onde o texto ilegível, ou cifrado, é tornado legível, como na sua forma original.

A criptografia é a arte e ciência de cifrar e decifrar mensagens de maneira a estabelecer um canal de comunicação sigiloso entre as pessoas envolvidas na troca de informação [Pino C.,G., 2002].

2.2- Tipos de Criptografia

A criptografia está classificada em duas categorias básicas, diferenciadas pelo tipo de chaves utilizadas: a Criptografia Simétrica, que ultiliza o conceito de chave secreta, e a Criptografia Assimétrica, que se baseia no conceito de chave pública. Logo vêm as diferentes implementações e aplicações destes esquemas, como a Criptografia Baseada em Identidades (baseada em criptografia assimétrica), e a criptografia temporal (aplicação da criptografia num determinado espaço de tempo).

2.2.1- Criptografia Simétrica

A criptografia simétrica ou de chave secreta, foi o primeiro tipo de criptografia criado e funciona compartilhando uma mesma chave *secreta* em ambos os extremos da comunicação. Ela transforma um

texto legível em uma mensagem cifrada, ou ilegível, e através da definição de um canal seguro, ela deve ser transmitida ao receptor da mensagem, que deve usar esta mesma chave secreta para posteriormente decifrar a mensagem, tornando-a novamente um texto legível [Schneier, B., 1996].

A principal vantagem deste tipo de criptografia é a sua performance, pois é muito veloz e segura.

Já a sua principal desvantagem radica na gerência das chaves, que devem ser transmitidas por canais seguros por serem vulneráveis a roubo. Outra desvantagem é a quantidade das chaves necessárias para a comunicação, que cresce significativamente com cada novo usuário. Por exemplo, para *n* usuários são necessárias *(n*(n-1))/2* combinações de chaves.

O esquema pode ser descrito como segue [Pino C., G., 1996]:

Sejam:

k	:	a chave secreta que será usada por Alice e Beto.
C_k, D_k	:	algoritmos de cifragem e decifragem, respectivamente, onde $D_k( ) = 1/C_k( )$
m	:	texto original
N	:	texto cifrado

As seguintes relações são válidas:

$$C_k(m) = N$$
$$D_k(N) = m$$

A figura 1 apresenta o esquema do sigilo da criptografia simétrica.

Figura 1. *Criptografia simétrica: uma mesma chave, compartilhada por Beto e Alice, é utilizada tanto para cifrar quanto para decifrar um documento.*

2.2.1.1- Exemplo de criptografia simétrica: AES (Padrão Avançado de Cifragem).

Em 2 de Outubro de 2000, o NIST (National Institute of Standards and Technology) anunciou um novo padrão de uma chave secreta de cifragem, escolhido entre 15 candidatos [Pierre Loidreau, 2002].

[Pierre Loidreau, 2002] diz que *"o novo padrão pretendia substituir o velho algoritmo DES, cujo tamanho das chaves está se tornando muito pequeno. O Rijndael - um nome comprimido, originário dos seus inventores Rijmen e Daemen - foi escolhido para se tornar o futuro AES. Este sistema de cifragem é dito ser um "bloco" de cifragem. À medida que as mensagens são cifradas em blocos inteiros, com unidades de 128-bits"*.

É apresentado o esquema genérico do AES na figura 2.

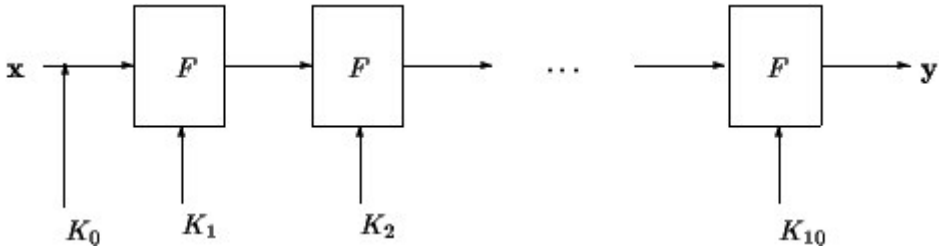

Figura 2. *Esquema de criptografia AES.*

O AES opera da seguinte maneira: primeiro uma chave secreta K_0 é ajustada bit a bit à mensagem *x*. Depois, semelhantemente a todos os blocos de cifragem, a função *F* é interada, utilizando sub-chaves geradas a partir de uma rotina de expansão, inicializada pela chave K_0,[Pierre Loidreau, 2002].

2.2.2- Criptografia Assimétrica

A Criptografia Assimétrica, também conhecida como criptografia de chaves públicas, diferentemente da criptografia simétrica que utiliza uma mesma chave tanto para cifrar uma mensagem quanto para decifrá-la, utiliza duas chaves, uma para cifrar e outra para decifrar a mensagem.

Cada usuário gera seu próprio par de chaves e publica uma delas, a *chave pública,* e mantém a outra sob seu poder, a *chave privada*.

Para que um emissor possa enviar um documento sigiloso para um receptor, o primeiro deve cifrar este documento com a chave pública do destinatário, que somente com a sua chave privada poderá decifrar a mensagem e obter o texto original.

Embora sua desvantagem resida na sua lentidão e no tamanho das chaves, se comparada com a criptografia simétrica, ela é muito vantajosa no gerenciamento das chaves, pois além de diminuir para *n*2* o número de chaves necessárias para *n* usuários, ela garante mais segurança por não estar tão exposta a roubo das chaves [Schneier, B., 1996].

A criptografia assimétrica funciona como segue:

Sejam:

C_k, D_k	:	algoritmos de cifragem e decifragem, respectivamente
P_b	:	chave pública do usuário Beto
S_b	:	chave privada do usuário Beto
K	:	universo de todas a chaves possíveis

m : texto original
N : texto cifrado

As seguintes relações são válidas:

$m = D_{Sb}(N) = D_{Sb}[C_{Pb}(m)];$
$N = C_{Pb}(m) = C_{Pb}[D_{Sb}(N)].$

Veja o esquema do sigilo da criptografia assimétrica na figura 3.

Figura 3. *Criptografia assimétrica: chaves diferentes para cifrar e decifrar mensagens. Alice procura no seu diretório de chaves públicas a chave de Beto, cifra o documento com esta chave e envia o documento sigiloso para Beto, que decifra a mensagem usando a sua chave privada.*

Um exemplo de criptografia assimétrica muito utilizada é o RSA, que será resumido a seguir.

2.2.2.1- Exemplo de criptografia assimétrica: RSA

RSA é o mais popular algoritmo de chave assimétrica e de fato um padrão mundial.
É um algoritmo de chave pública baseado na exponenciação aritmética modular.
O nome RSA advém do nome (sobrenome) de seus três inventores: Ron Rivest, Adi Shamir e Leonard Adleman. A segurança do RSA é baseada na dificuldade de fatorar números muito grandes, [Grandi, A., 2002]. As chaves públicas e privadas são funções de dois números primos muito grandes (200 dígitos ou mais). O RSA tem resistido durante muitos anos a extensivos ataques. Com o aumento do poder computacional, pode-se aumentar a segurança do RSA com o aumento do tamanho das chaves. O funcionamento e os fundamentos matemáticos são explicados por [Quevedo, 2002].

2.2.3- Alguns requisitos de segurança garantidos com a criptografia simétrica e assimétrica

Sigilo: É garantido pela criptografia simétrica e assimétrica, e trata de preservar os dados confidenciais entre o emissor e o receptor de uma mesagem, tornando-a ilegível a intrusos;

Autenticidade: Garantida somente pela criptografia assimétrica, trata de identificar os participantes de uma comunicação, diferenciado-os entre si;

Integridade: Garantida pela criptografia simétrica e assimétrica, ela trata de provar que uma mensagem que chega ao destino é exatamente a mesma que partiu da origem;

2.2.4- Criptografia unidirecional (Hash)

As funções hash (também chamadas de criptografia unidirecional, criptografia de mão única, funçãs resumo ou message digest) são essenciais para a criptografia. A função hash é uma função que toma como entrada um documento de tamanho variável e produz uma saída tamanho fixo (o valor de hash). Se o valor de hash de duas mensagens são iguais, é improvável que as duas mensagens não sejam idênticas. A função de hash deve ser fácil de computar e impossível de ser revertida. Em uma boa função de hash também deve ser difícil de acontecer uma colisão, ou seja, para duas entradas diferentes o valor de hash deve ser o mesmo.As funções de hash mais utilizadas são o MD5 e o Secure Hash Algorithm 1 (SHA-1). MD5 foi projetado por Ron Rivest (co-inventor do RSA). SHA-1 é baseado no MD5 e foi projetado pelo National Institute of Standards and Technology (NIST) e pelo National Security Agency (NSA) para uso com o DSS (Digital Signature Standard). MD5 produz 128 bits de hash, enquanto SHA-1 produz 160 bits de hash. O SHA-1 é uma função hash mais segura que o MD5.

Veja o esquema do resumo hash na figura 4.

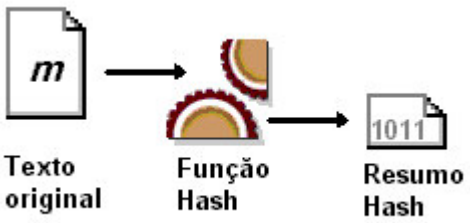

Figura 4. *Esquema de um resumo hash.*

2.3- Assinatura Digital

A assinatura digital é um conjunto de dados associados a uma mensagem, que permite verificar a identidade do assinante e a integridade da mensagem. O procedimento para assinar digitalmente um documento consiste em gerar um resumo hash do documento, cifrar este resumo com a chave privada do assinante, gerando assim a “assinatura digital”, a qual deve ser anexada ao documento como mostra a figura 5.

O destinatário do documento pode verificar a identidade do assinante com os seguintes procedimentos: ele gera um resumo hash do documento (excluindo a assinatura), decifra a assinatura com a chave pública do suposto assinante, e confere se o resumo hash e a assinatura decifrada são iguais. Em caso afirmativo, foi constatada a auntenticidade da origem, com a assinatura correspondente ao remetente, e também foi verificada a integridade do documento.

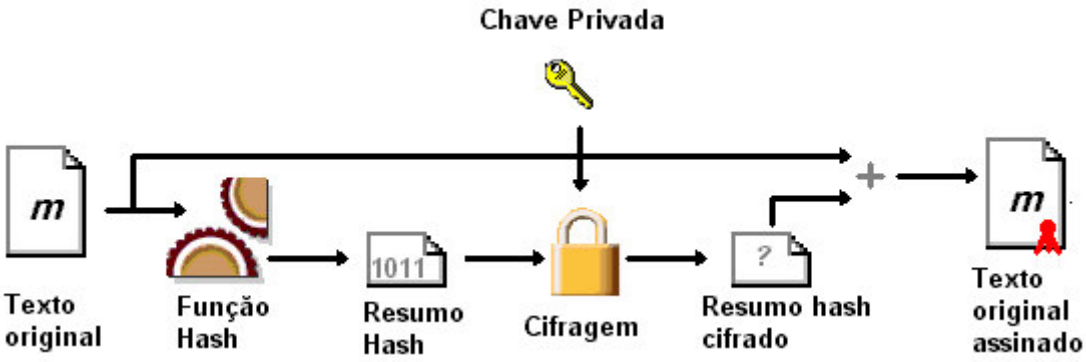

Figura 5. *Assinatura digital: com a chave privada do emissor é cifrado um resumo hash do documento, que logo é anexado ao documento.*

2.4- Autoridade Certificadora e Certificados Digitais

Uma Autoridade Certificadora, chamada também de autoridade de confiança, ou AC, é uma entidade de confiança do emissor e do receptor de uma comunicação. A confiança nesta "terceira entidade" permite que qualquer um dos dois participantes da comunicação confie nos documentos emitidos e assinados por ela, mais especificamente nos "certificados digitais".

A autoridade certificadora, além da emissão de certificados digitais também é responsável pelo agendamento da data de expiração de um certificado, assim como também é o seu dever providenciar uma publicação dos certificados revogados.

Os certificados digitais são documentos que vinculam uma chave pública com o seu dono, e são assinados por uma autoridade certificadora.

Os certificados digitais mais utilizados são do padrão X.509, e contêm informações tais como: nome da entidade, chave pública da entidade, nome da autoridade certificadora, assinatura da autoridade certificadora, período de validade, número de série, nome do domínio da entidade e algoritmo utilizado na assinatura. Veja um exemplo na figura 6.

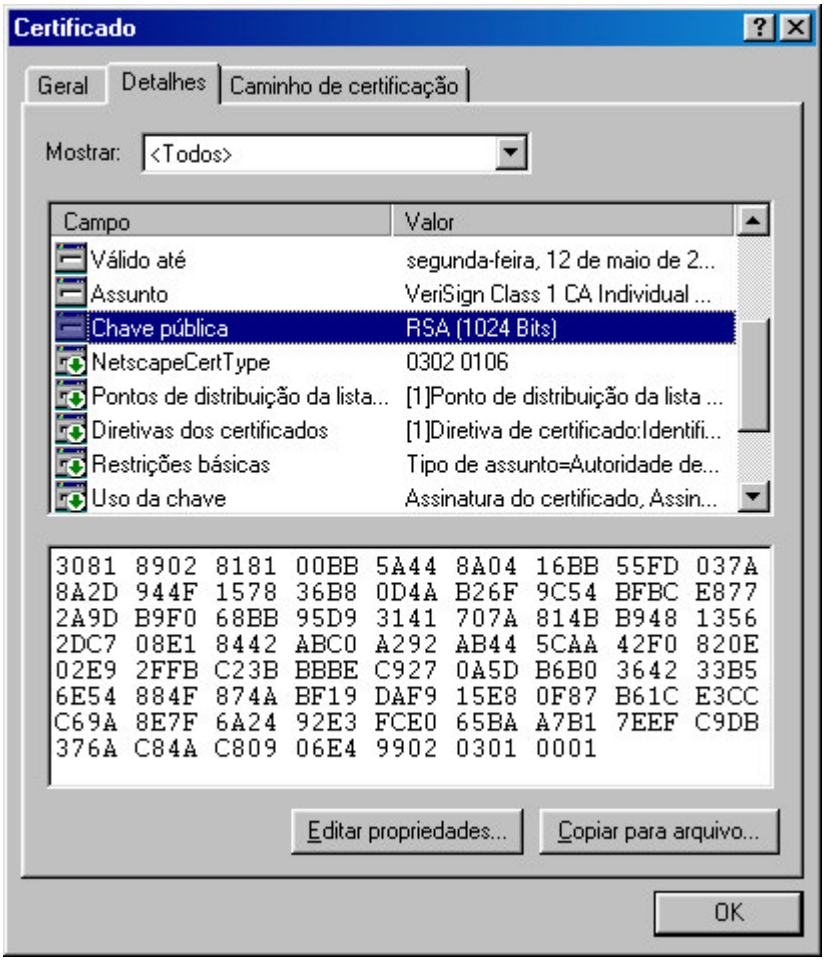

Figura 6. *Certificado digital: neste exemplo podemos ver a chave pública, a validade, entre outras informações.*

Capítulo 3

Criptografia baseada em identidades - IBE

Baseada na criptografia assimétrica tradicional, este novo conceito de criptografia foi proposto por [Shamir, A., 1984] e consiste em utilizar um *string* que identifique o usuário de forma única, como por exemplo, seu endereço de correio eletrônico, seu endereço IP, ou o seu CPF, ao invés de utilizar chaves públicas aleatórias associadas a um usuário.

A idéia deste conceito de criptografia é simplificar a infra-estrutura de chaves públicas, sendo que com isso a comunicação segura entre duas entidades, Alice e Beto, possa ocorrer de uma maneira mais simples. Por exemplo, Beto não precisaria ter que obter um par de chaves criptográficas para que Alice envie um documento sigiloso para ele, e nem mesmo ter conhecimentos sobre chaves criptográficas. Basta que Alice cifre a mensagem ou documento com algum identificador de Beto, e logo após, enviá-lo a ele. Beto, porém, só a partir deste momento poderia se preocupar em obter uma chave privada para poder decifrar esta mensagem. Basta que ele compareça até uma PKG (Gerador de chaves privadas) para provar a sua identidade e obter assim a chave privada correspondente ao identificador com que foi cifrada a mensagem.

"O papel da PKG pode ser desempenhado por qualquer entidade que possua um par de chaves criptográficas. Um exemplo poderia ser uma matriz de um banco que administre as chaves das filiais. Esta entidade recebe como parâmetro de entrada a chave pública de um usuário e devolve a chave particular através de um canal seguro, sempre que comprovar a identidade do solicitante", [Benits Jt. 2003].

A PKG seria então a autoridade de confiança dos usuários Alice e Beto, pois ela conhece as chaves particulares deles. Eles podem recuperar essas chaves, mas também devem confiar plenamente na idoneidade da PKG.

Uma vez estabelecida, a PKG gera um par de chaves e publica a sua chave pública, a partir dali Alice pode cifrar um documento para Beto, usando como chave pública dele um identificador, associando essa chave à chave pública da PKG. Alice então está pronta para enviar o documento para Beto, quem então pode solicitar a sua chave privada para a PKG, sempre que comprove sua identidade. A PKG julga a identidade de Beto e, caso a confirme, gera então a chave particular dele para logo em seguida enviá-la por meio de um canal seguro. Beto está pronto para decifrar o documento. A figura 7 apresenta o esquema IBE.

3.1- Uma noção dos fundamentos matemáticos do IBE

Os sistemas criptográficos convencionais geralmente se baseiam na dificuldade computacional

do Problema do Logaritmo Discreto (PLD). [Boneh, D. & Franklin, M., 2001], baseados nas propriedades das curvas elípticas, conseguiram uma solução satisfatoria para a criptografia baseada em identidades, e que consiste no Problema do Logaritmo Discreto em Curvas Elípticas (PLD-CE).

Dados dois pontos *R, P,* de uma curva elíptica definida sobre um corpo finito, achar um inteiro *s* tal que:

$$R = sP$$

Onde o par (R_{TA}, s) representa o par de chaves pública/privada de uma PKG. Apesar de os valores R e P serem públicos, o valor de s não pode ser calculado eficientemente, pois está protegido pelo PLD-CE.

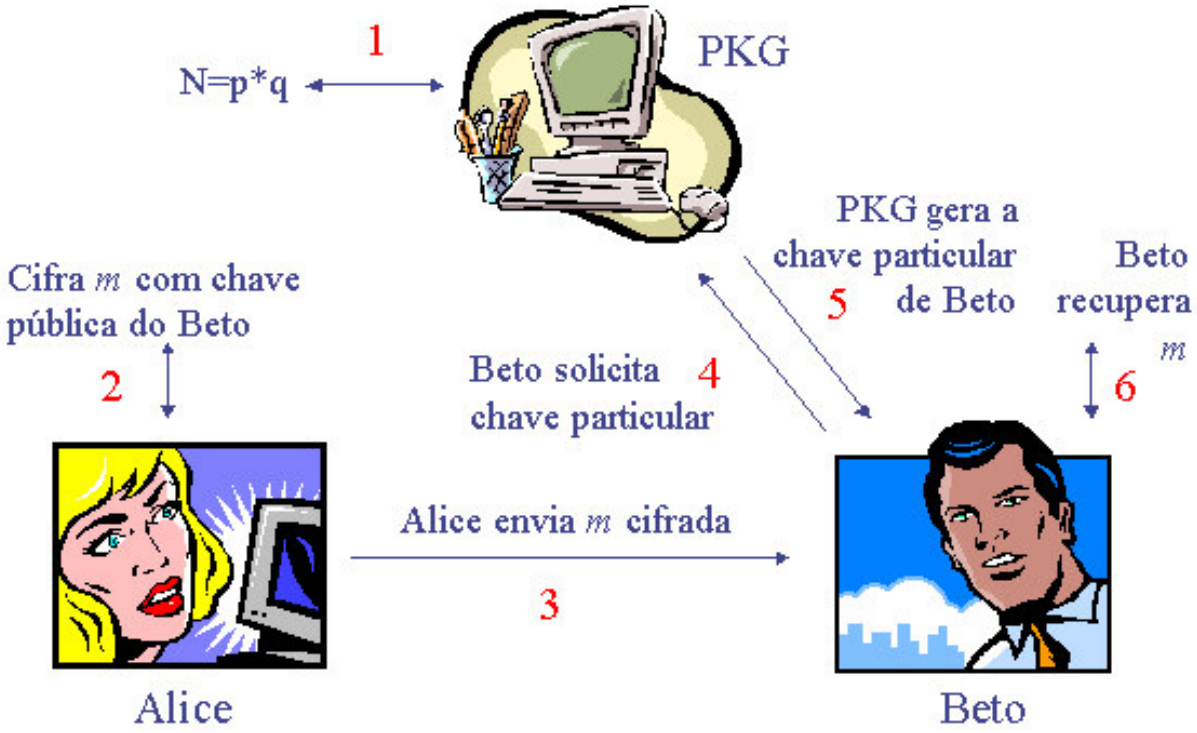

Figura 7. *Criptografia baseada em identidades: 1- A PKG gera um par de chaves e publica a sua chave pública. 2- Com ela, Alice cifra um documento para Beto, usando como chave pública dele um identificador. 3- Alice envia o documento para Beto. 4- Beto solicita a sua chave particular, comprovando sua identidade. 5- PKG gera a chave de Beto e a envia a ele por um canal seguro. 6- Beto decifra o documento.*

3.1.1- Par de chaves Pública/Particular baseadas em identidades (Q_{ID}, S_{ID})

Sejam *o par de chaves* (Q_{ID}, S_{ID}), suponha que exista uma autoridade de confiança PKG com um par de chaves padrão, (R_{TA}, s), gerada pela equação R = sP, de modo que valem as seguintes relações:

$$S_{ID} = sQ_{ID}$$
$$Q_{ID} = H_1(ID)$$

Onde *ID* é o identificador (p. ex. um endereço eletrônico: alice@com.br) e H_1 é uma função de espalhamento, definida por [Benits Jr., W., 2003], que também observa: *"Note-se que apesar dos valores*

R e P serem públicos, o valor de s não pode ser calculado eficientemente, pois está protegido pelo PLD-CE. Da mesma forma, mesmo se Beto possuir um par de chaves válido (Q_{beto},S_{beto}), ele não consegue recuperar a chave particular s da autoridade de confiança. O leitor mais atento deverá ter notado que nesse tipo de sistema, diferentemente da criptografia assimétrica tradicional, a autoridade de confiança tem conhecimento da chave particular de todos os seus usuários. Tal fato é chamado de custódia de chaves (key escrow)".

3.2- Exemplo de implementação IBE

Veremos a seguir o modelo de criptografia baseada em identidades proposto por Boneh & Franklin [Boneh, D. & Franklin, M., 2001], que esta dividido em 4 etapas:

Sejam:

(Q_{beto}, S_{beto})	:	o par de chaves baseadas em identidades de Beto;
R_{TA}	:	a chave pública padrão da autoridade de confiança que gerou a chave particular de Beto;
m	:	a mensagem que Alice deseja enviar para Beto.
$H_1(),H_3(), e_t()$	:	duas funções de espalhamento e uma de emparelhamento respectivamente, como descritas em [Benits Jr., 2003].

3.2.1- Fase de Configuração (setup)

- seleção dos parâmetros q, G_1, G_2, P e e_t;
- escolha da chave particular s, tal que $s \in Z^*_q$;
- cálculo da chave pública R_{TA};
- escolha das funções de espalhamento H_1 e H_3.

Os parâmetros do sistema são os valores públicos $(q,G_1,G_2, e_t, P,R_{TA},H_1,H_3)$, onde q é um número primo da ordem 2^{160}, G_1 e G_2 são grupos onde o PLD-CE é difícil.

A chave particular *s*, também chamada de *chave mestra*, é um parâmetro conhecido apenas pela autoridade de confiança.

3.2.2- Fase de Extração (extract)

- cálculo de Q_{ID}, para um dado ID.
- cálculo da chave particular S_{ID} baseada em identidades.

Aqui o usuário ingressa a sua chave pública *(ID)*, com a qual é obtido o Q_{ID} (de $Q_{ID} = H_1(ID)$)
Logo, é aplicada a equação $S_{ID} = s\, Q_{ID}$, onde é obtida a chave particular S_{ID}

3.2.3- Fase de Cifragem (encrypt)

- cálculo de (U, V).

Alice escolhe um elemento aleatório *r* e calcula o par:

$$\{U = rP,\ V = m \oplus H_3(e_t(rQ_{beto},\ R_{TA}))\}$$

Então envia o texto cifrado *(U, V)* para Beto. Note que os valores *P*, Q_{beto} e R_{TA} são valores públicos, assim como a função H_3. O único valor secreto usado na cifragem é o aleatório *r*.

[Benits Jr., W., 2003] observe que *"Boneh & Franklin, em seu artigo, não destacam a importância na escolha de r.*

É importante que o elemento aleatório r escolhido por Alice seja diferente para cada mensagem a ser cifrada (números com tal característica são conhecidos na literatura específica como "NONCE" (Number used ONCE.); caso contrário, haverá uma falha de segurança".

3.2.4- Fase de Decifragem (decrypt)

- recuperação do legível m.

Beto, recebendo *(U, V)* de Alice, faz o seguinte cálculo para recuperar o legível *m*:

$$m = V \oplus H_3(e_t(S_{beto},\ U))$$

Vemos que Beto utiliza sua chave particular baseada em identidades (S_{beto}) para recuperar *m*.

3.3- Demonstração

[Benits Jr., W., 2003] demonstra que Beto consegue recuperar m.

$V \oplus H_3(e_t(S_{beto},\ U)) = V \oplus H_3(e_t(S_{beto},\ rP))$, *pois* $U = rP$
$= V \oplus H_3(e_t(sQ_{beto},\ rP))$, *pois* $S_{beto} = sQ_{beto}$
$= V \oplus H_3(e_t(Q_{beto},\ P)rs)$, *por bilinearidade*
$= V \oplus H_3(e_t(rQ_{beto},\ sP))$, *por bilinearidade*
$= V \oplus H_3(e_t(rQ_{beto}, R_{TA}))$, *pois* $R_{TA} = sP$
$= m$
$V = m \oplus H_3(e_t(rQ_{beto}, R_{TA}))$."

Os algoritmos configuração e extração são executados pela autoridade de confiança *PKG* e os algoritmos criptografia e decifragem pelos participantes da comunicação.

3.4- Vantagens e desvantagens do IBE

- ***Vantagens***

- Não é necessário um diretório de chaves públicas, o que economiza armazenamento e gerência destas chaves.

- Qualquer entidade que possua um par de chaves criptográficas pode fazer o papel de PKG, possibilitando a uma rede de lojas ter como autoridade de confiança a sua matriz.
- O PKG tem conhecimento de todas as chaves particulares dos usuários, o que possibilita recuperar chaves privadas perdidas.
- Alice pode enviar mensagens cifradas para Beto, mesmo se ele ainda não obteve seu par de chaves da PKG, possibilitando a Beto ser beneficiário dos serviços de criptografia mesmo sem ter conhecimento prévio dos conceitos de chaves criptográficas.
- Não é necessário Alice obter o certificado da chave pública de Beto, pois a própria chave pública de Beto diz repeito a ele.
- Tamanho menor da chave para um mesmo nível de segurança, se comparado à criptografia assimétrica tradicional. Exemplo: 160 bits em IBE contra 1024 bits em RSA.

• ***Desvantagens***

- Grande dependência da chave mestra do PKG. A decorrência de algum problema com ela pode comprometer todo o sistema .
- O PKG tem conhecimento de todas as chaves particulares dos usuários, deixando-a mais exposta a ataques.
- Dificuldade da implementação. O trabalho sobre curvas elípticas exige uso de aritmética de alta precisão nas funções de emparelhamento.
- Processo de criptografia mais lento se comparado à criptografia assimétrica tradicinal, decorrente das complexas operações matemáticas.

3.5- Exemplo de implementação IBE (Boneh & Franklin).

MIRACL (Multiprecision Integer and Rational Arithmetic C/C++ Library): Biblioteca de aritmética de precisão em linguagem C/C++ que inclui uma implementação de criptografia baseada em identidade de [Boneh & Franklin, 2001]. MIRACL foi desenvolvida por Michael Scott (mscott@indigo.ie) e foi recomendado para este trabalho por Keith Harrison (key_harrison@hp.com).

No pacote do MIRACL está disponível uma outra implementação de IBE, criada por [Cocks, C., 2002], porém, a implementação de Boneh & Franklin será motivo deste estudo.

A implementação IBE Boneh & Franklin no MIRACL consta de quatro programas:

- Configuração (ibe_set);
- Extração (ibe_ext);
- Cifragem (ibe_enc), e;
- Decifragem (ibe_dec).

Estes programas não são compilados ao instalar o MIRACL, e por isso, precisam ser compilados separadamente, como descrito mais abaixo. Outras implementações IBE também podem ser encontradas na biblioteca MIRACL.

3.5.1- Instalação do MIRACL

O ambiente de trabalho escolhido para o teste foi Red Hat Linux 9.0, com compiladores gcc/g++.

Os passos para a instalação são os seguintes:

1- Obtemos o arquivo miracl.zip do endereço *http://indigo.ie/~mscott/#download*
2- Procedemos a descompactá-lo e rodar o arquivo de compilação com os seguintes comandos:

```
# unzip -j -aa -q -L miracl.zip
# bash linux
```

O primeiro comando descompacta os arquivos em uma pasta ignorando a estrutura de sub-diretórios e o segundo comando roda os comandos de compilação como especificado no arquivo "linux" no diretório corrente.

Agora compilaremos os quatro programas IBE, digitando os seguintes comandos:

```
# g++ I. ibe_set.cpp zzn2.cpp big.cpp monty.cpp elliptic.cpp miracl.a –o ibe_set
# g++ I. ibe_ext.cpp big.cpp monty.cpp elliptic.cpp miracl.a –o ibe_ext
# g++ I. ibe_enc.cpp zzn2.cpp big.cpp monty.cpp elliptic.cpp miracl.a –o ibe_enc
# g++ I. ibe_dec.cpp zzn2.cpp big.cpp monty.cpp elliptic.cpp miracl.a –o ibe_dec
```

3.5.2- Executando o IBE

Digitamos o seguinte comando:

 # ./ibe_set

Fase de configuração na figura 8.

Este comando inicializa os parâmetros a partir de um número randômico. É solicitado entrar com um valor semente de até 9 digitos.

Depois do programa ter rodado, ele produz o arquivo ***common.ibe*** que será usado para cifrar, decifrar e gerar chaves privadas. O arquivo contém: *<Tamanho do módulo primo em bits> <Primo p> <Primo q> <Ponto P – coordenada x> <Ponto P - coordenada y> <Ponto Ppub - coordenada x> <Ponto Ppub - coordenada y> <Raiz cúbica da unidade em Fp2 – componente x> <Raiz cúbica da unidade em Fp2 - componente y>*

O programa produz também o arquivo ***master.ibe***, que contém a chave mestra *s* que será usada na geração das chaves particulares dos usuários.

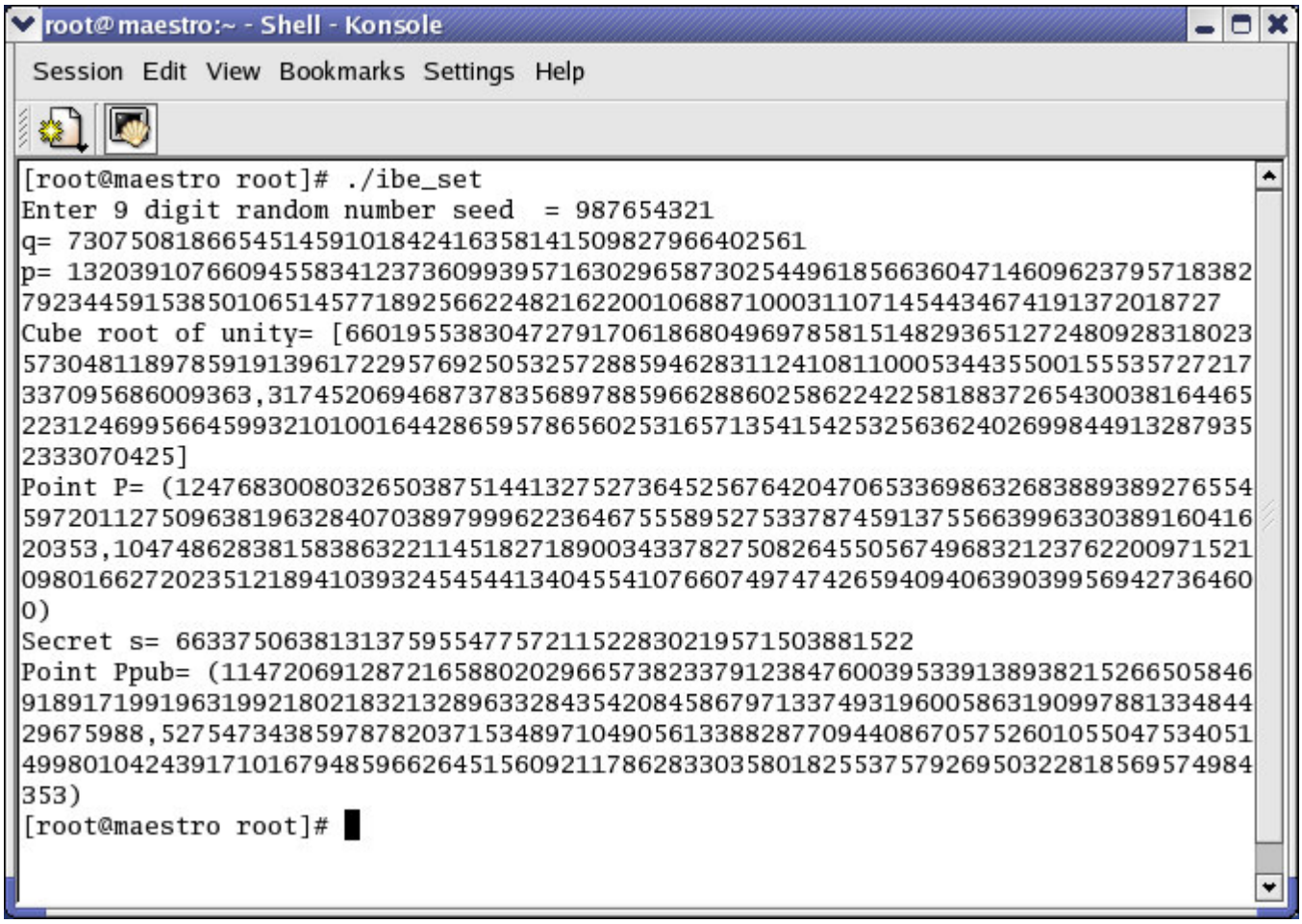

Figura 8. *Programa ibe_set: set-up dos parâmetros da PKG.*

Passamos então à tentativa de recuperação de uma chave particular, através do comando:

 # ./ibe_ext

Fase de recuperação da chave privada na figura 9.

Neste caso, o programa supõe que o identificador (a chave pública) seja um endereço de e-mail, mas o programa aceita qualquer string, o que possibilitará que mais adiante trabalhemos com tempo.

Depois de executado o programa, é produzido o arquivo ***private.ibe*** que contém a chave privada.

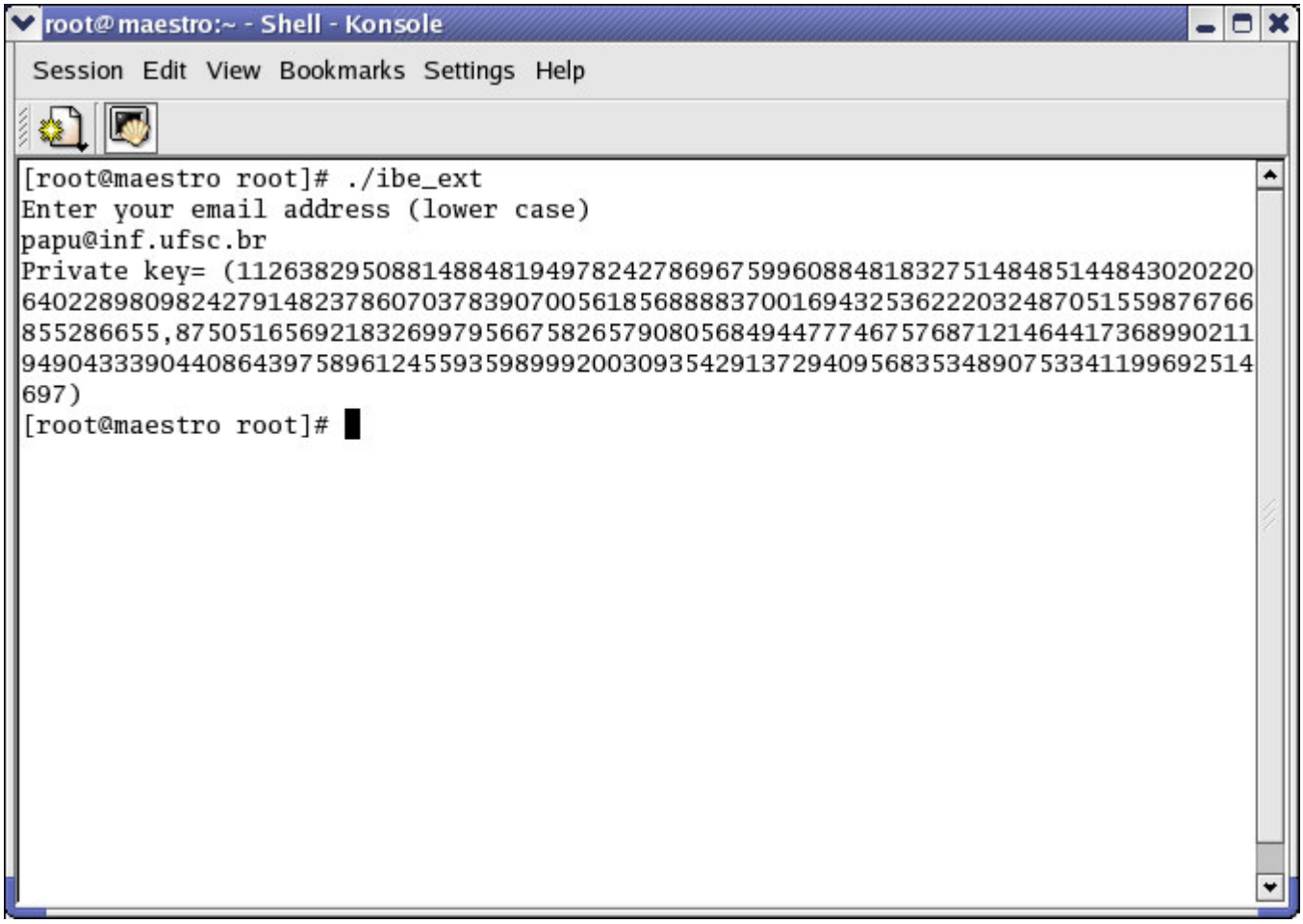

Figura 9. *Programa ibe_ext: geração do arquivo private.key contendo a chave particular.*

Procedemos a cifrar um documento. Note que este passo poderia ser efetuado antes do passo anterior, obedecendo à característica do IBE de não ser necessário o usuário possuir uma chave particular antes de ser o beneficiário de um documento cifrado. Para cifrar com IBE, digitamos o seguinte comando:

 # ./ibe_enc

Fase de cifragem na figura 10.

Aqui é gerada uma chave aleatória de sessão para cifrar o arquivo com criptografia simétrica AES.
A chave de sessão é cifrada com IBE e guardada no arquivo ***<filename>.key***.
O documento cifrado é guardado com o nome ***<filename>.ibe***.

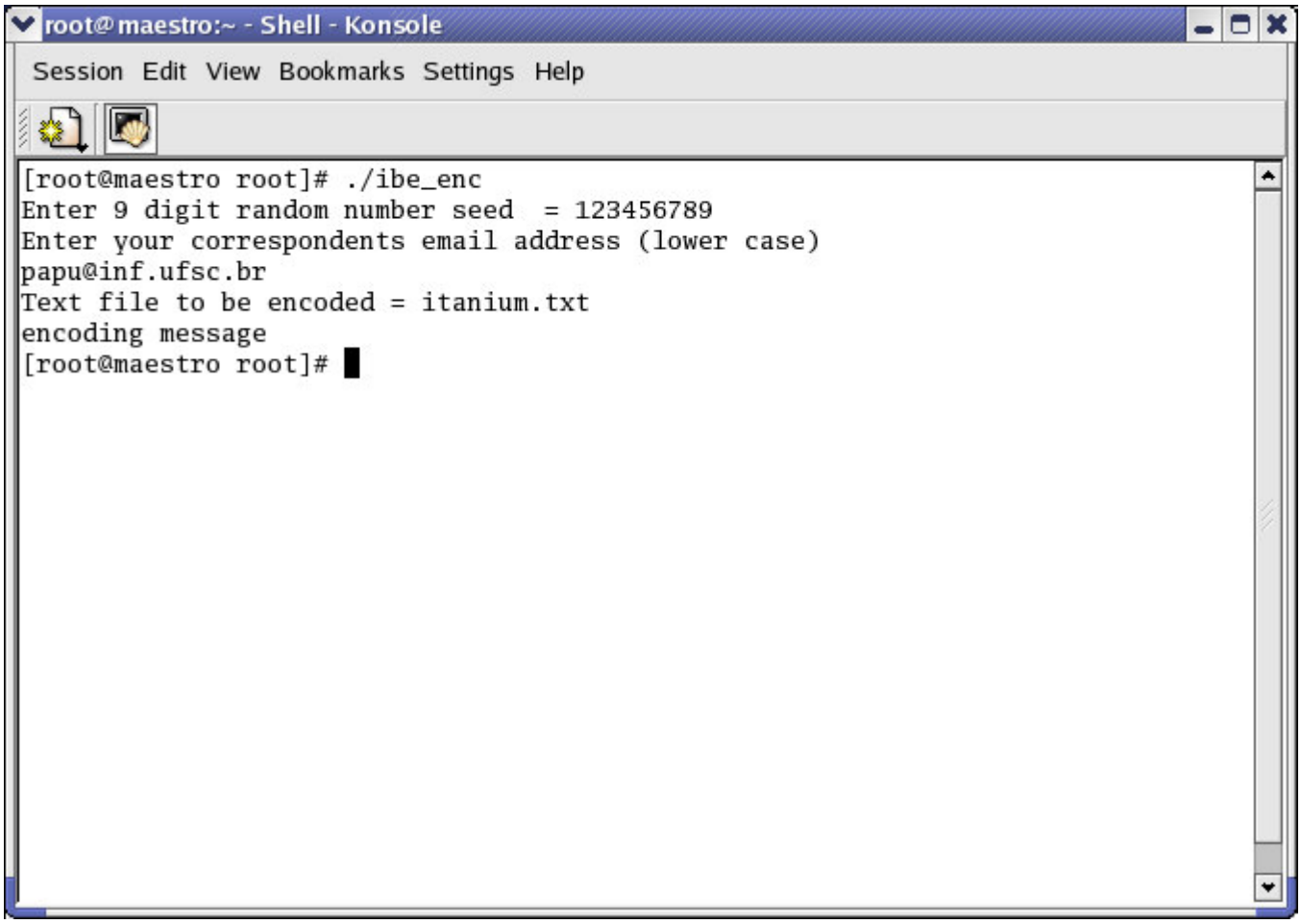

Figura 10. *Programa ibe_enc: O usuario digita uma semente para gerar o randômico r e entra com o nome do arquivo a cifrar.*

Finalmente podemos proceder ao processo de decifragem. O pré-requisito para este passo é antes ter adquirido a chave particular com o passo de extração. A decifragem é iniciada por meio do seguinte comando:

 # ./ibe_dec

Fase de decifragem na figura 11.

Procura a chave de sessão IBE no arquivo ***<filename>.key***
Decifra essa chave com o arquivo ***private.ibe*** (gerado por ibe_ext)
Decifra o documento ***<filename>.ibe*** com a chave de sessão do AES
Imprime o texto original na tela.

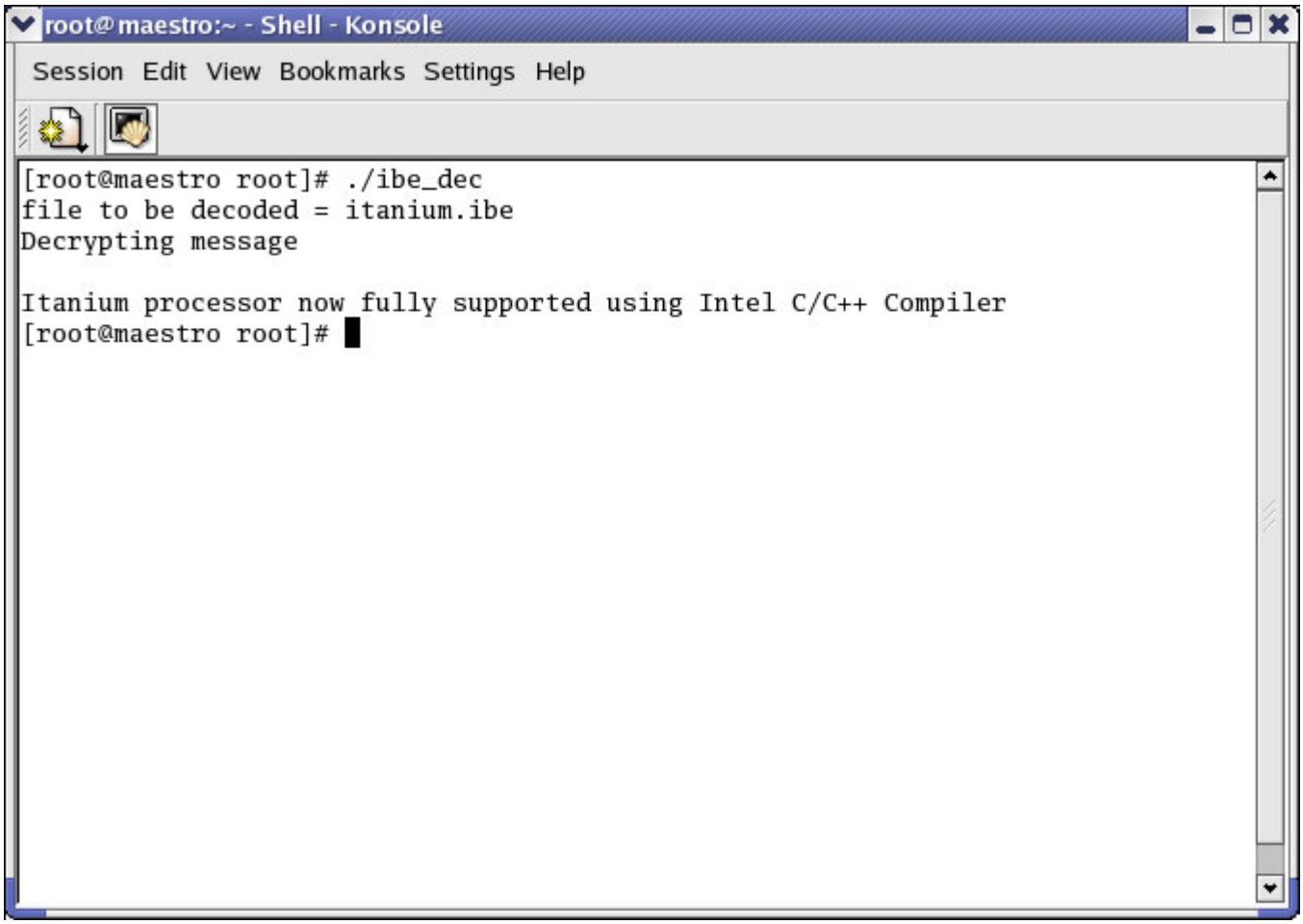

Figura 11. *Programa ibe_dec: Decifra um documento imprimindo o texto original na tela.*

Capítulo 4

Criptografia temporal

Como discutimos anteriormente, o sigilo temporal de documentos é muito importante em aplicações do dia a dia. Salvaguardar informações é de interesse de instituições públicas e privadas, num tempo em que a Internet tornou-se grande aliada pelas suas facilidades de comunicação e também a grande vilã pelas suas vulnerabilidades em segurança.

A guarda temporal de documentos sigilosos pode consistir em tornar o mesmo inacessível (custódia do documento por parte de uma terceira entidade *TP*), ou somente torná-lo ilegível com criptografia. Esta última alternativa é a mais conveniente.

May [May, 1993] foi o primeiro a discutir a criptografia temporal no contexto de liberação de documentos eletrônicos, apresentando uma solução que incluía uma terceira entidade TP como responsável pela guarda do documento para sua posterior liberação (Veja figura 12). A outra solução apresentada consistia em manter em sigilo somente a chave criptográfica de decifragem do documento. A segunda proposta foi a mais utilizada, e o esquema pode ser apreciado na figura 13.

Figura 12. *A primeira proposta consite em manter o documento sob a custódia de uma terceira entidade TP, que o torna inacessível até a data previamente combinada. Esta abordagem é perigosa e de alto custo, pois exige maior robustez e confiança na TP*

A solução apresentada na seguinte página, na figura 13, é a mais conveniente pelo fato de que ela ganha em desempenho, escala e economia de recursos [Custóio et al, 2003].

4.1- Abordagens propostas para criptografia temporal

A primeira abordagem, proposta por Rivest [Rivest et al., 1996], consite em implementar um algoritmo que seja executado em *n* etapas sequenciais, não paralelizáveis, de modo que o resultado desta execução seja a chave criptográfica de deciframento correspondente à chave pública utilizada para gerar o documento. Esta

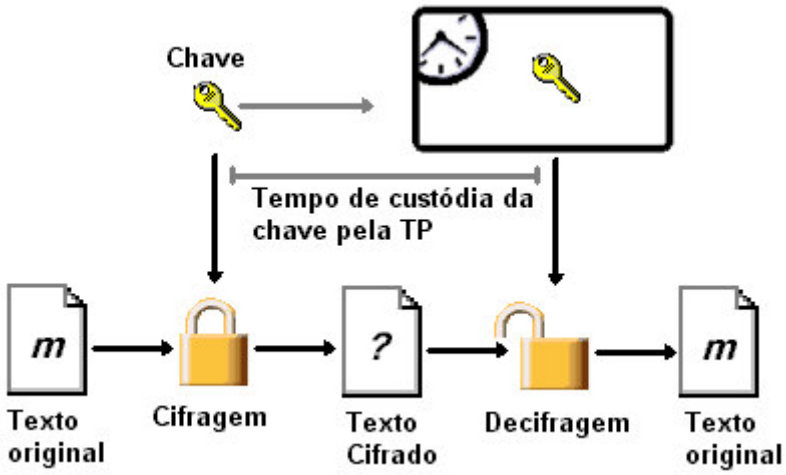

Figura 13. *A proposta mais utilizada e mais conveniente consiste em manter o documento protegido com criptografia, liberando a sua distribuição, e somente protegendo a sua chave de deciframento.*

abordagem requer muito poder de processamento e memória disponível. Outra abordagem consiste em cifrar o documento com criptografia simétrica, cifrando a chave utilizada neste processo com a chave pública de uma TP. Assim, ela poderá decifrar a chave no momento oportuno. Esta abordagem é muito utilizada pela suas facilidades na implementação.

[Mont et al., 2003] propôs uma outra abordagem utilizando criptografia baseada em identidades para a liberação das chaves de deciframento. Estas chaves somente serão produzidas na data e hora especificada para sua liberação, o que se torna uma grande vantagem. Na proposta feita por [Mont et al., 2003], é incluída uma comparação de criptografia tradicional (RSA) da segunda abordagem acima resumida, e a criptografia baseada em identidade (IBE) para trabalhar com tempo, que é mostrada a seguir:

4.1.1- Criptografia temporal RSA, principais passos (Veja figura 14):

Descrição dos principais passos da Criptografia temporal RSA:

1- Alice define a Data/Hora da liberação do documento.
2- O programa cliente de Alice gera uma chave simétrica Ks.
3- O programa cliente de Alice cifra o documento com Ks.
4- O programa cliente de Alice cifra Ks com a chave pública da TP, e cria um meta-documento contendo Ks cifrada e a Data/Hora cifrada também com a chave pública da TP.
5- Alice envia o documento a Beto.
6- Beto recebe o meta-documento e o seu programa cliente lhe informa sobre a data da liberação da chave de deciframento. Beto solicita a chave à TP.
7- A TP interpreta o meta-documento, decifra a Data/Hora para validar o pedido, ou não.
8- TP envia a chave de deciframento para Beto, ou uma mensagem de erro no caso de não ter atingido a Data/Hora alvo.
9- O programa cliente de Beto decifra o documento, caso possua a chave, ou notifica a Beto a impossibilidade do procedimento.

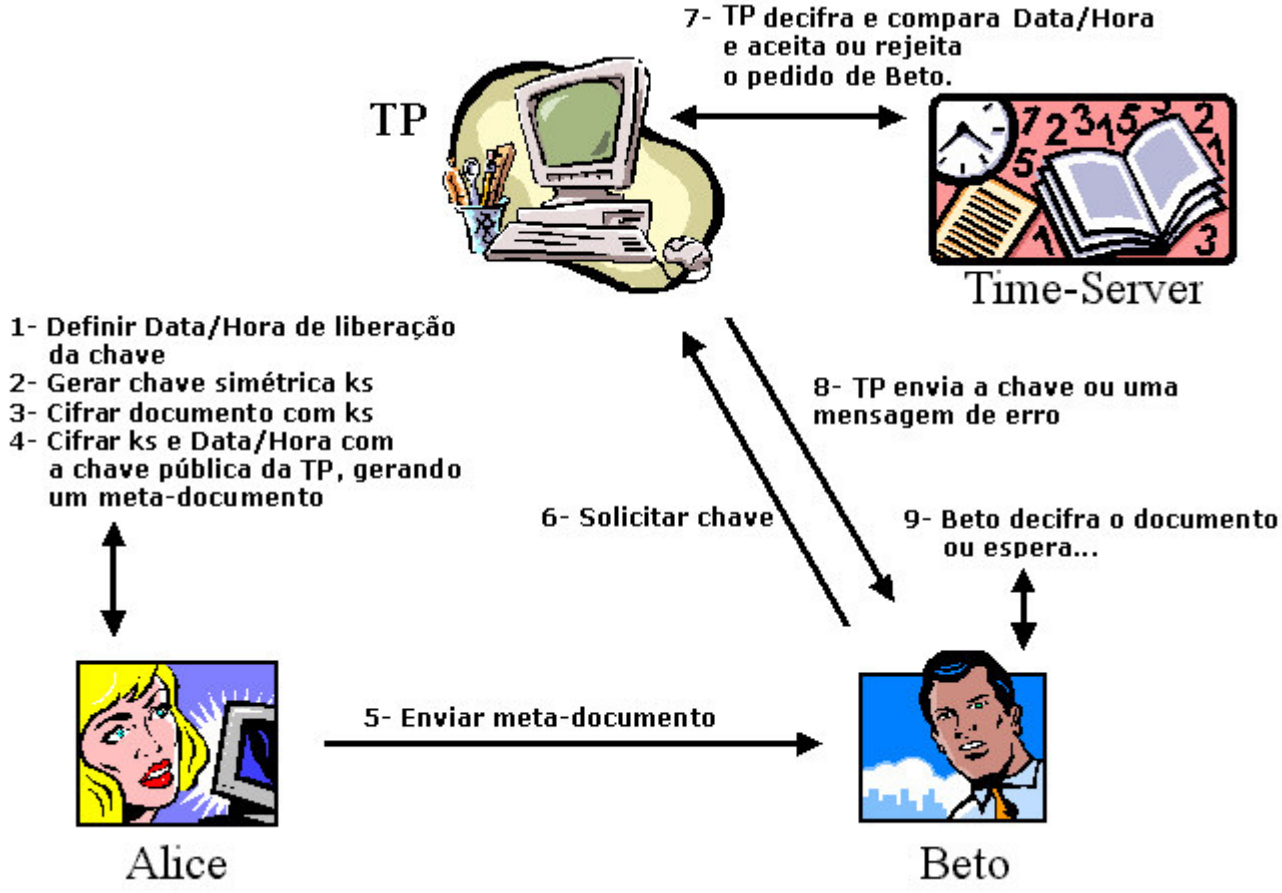

Figura 14. *Criptografia tradicional (RSA) aplicada à criptografia temporal.*

4.1.2- Criptografia temporal IBE:

Descrição dos principais passos na figura 15.

Descrição da criptografia temporal IBE:

1- Alice define a Data/Hora da liberação do documento.
2- O programa cliente de Alice gera uma chave simétrica Ks.
3- O programa cliente de Alice cifra o documento com Ks.
4- O programa cliente de Alice cifra Ks com a chave pública IBE, no formato Data/Hora, e cria um meta-documento contendo Ks cifrada e a Data/Hora legível.
5- Alice envia o documento a Beto.
6- Beto recebe o meta-documento e o seu programa cliente lhe informa sobre a data da liberação da chave de deciframento. Beto solicita a chave à TP.
7- A TP interpreta o meta-documento, lê a Data/Hora para validar o pedido, ou não.
8- TP envia a chave de deciframento para Beto, ou uma mensagem de erro no caso de não ter atingido a Data/Hora alvo.
9- O programa cliente de Beto decifra o documento, caso possua a chave, ou notifica a Beto a impossibilidade do procedimento.

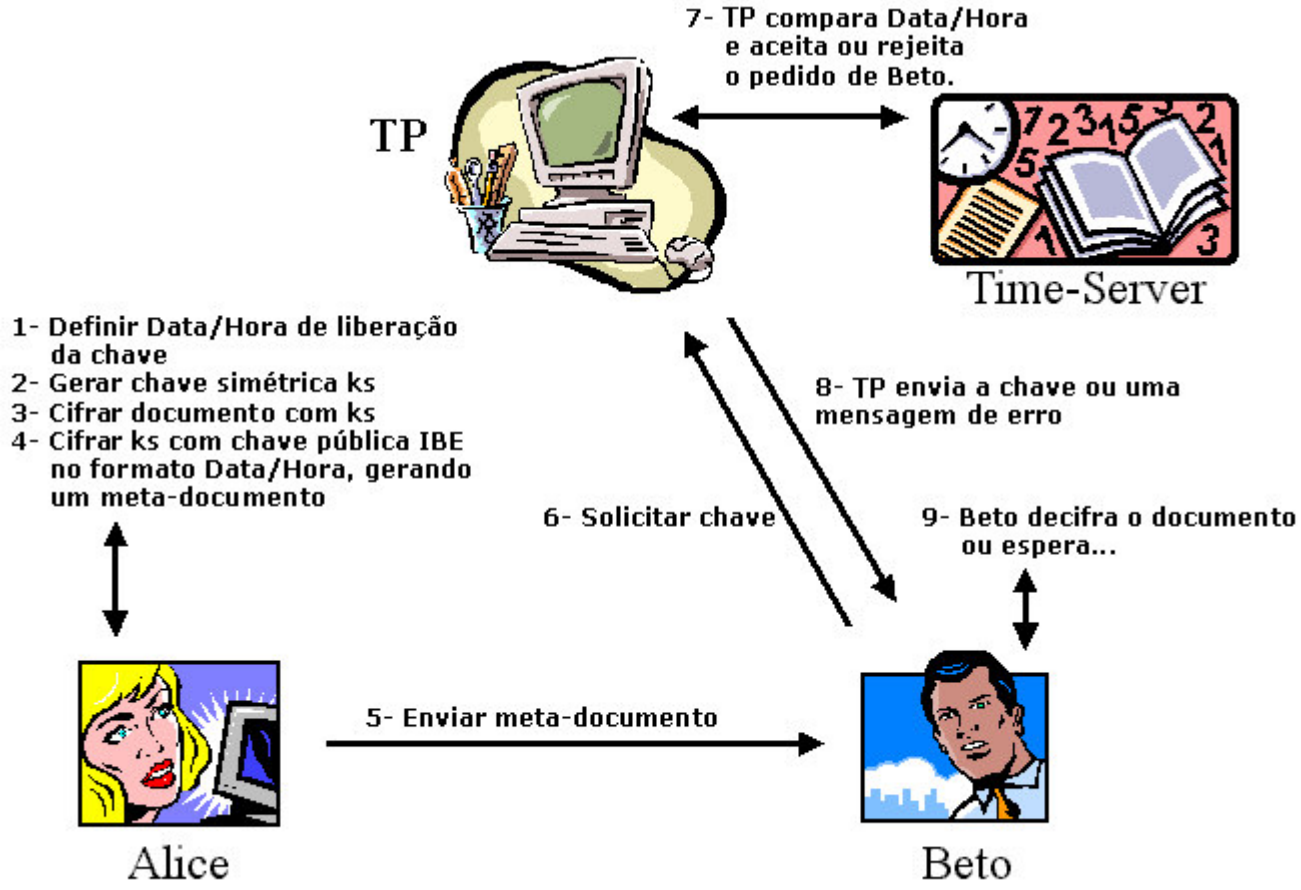

Figura 15. *Criptografia baseada em identidades, IBE, aplicada à criptografia temporal.*

4.1.3- Considerações da criptografia temporal RSA e IBE

Ambas as opções trabalham muito bem no seu propósito, porém, o IBE poupa esforços por parte da TP no que se refere a processamento de meta-documento e deciframento de meta-dados. Mais especificamente, com IBE não precisamos cifrar a Data/Hora da liberação da chave no momento em que ciframos o documento, como ocorre no RSA, evitando com isso que a TP precise decifrar essa Data/Hora para poder compará-la com o seu relógio interno. No IBE a Data/Hora pode perfeitamente estar legível, dado que ela é a chave pública com a qual o documento foi cifrado e é somente a chave privada correspondente a essa Data/Hora que conseguirá abrir o documento. Isto também evita tentativas de fraudes, como por exemplo, tentar mudar a Data/Hora legíveis no meta-documento IBE.

Note que as diferenças fundamentais estão nos passos 4 e 7, nos quais o IBE não precisa cifrar a Data/Hora da liberação da chave, e nem decifrá-la, respectivamente, diminuindo o tráfego na rede por tempo de serviço e diminuindo a demanda do processador da TP.

4.2- Autoridade Certificadora Temporal – ACT

A ACT, assim como uma autoridade certificadora convencional, é responsável pela geração de pares de chaves criptográficas assimétricas, mas com o propósito de permitir a implementação de criptografia

temporal com estas chaves. Ela também está encarregada da emissão de certificados digitais temporais das chaves criadas por ela.

Para permitir o sigilo temporal de documentos, a ACT mantém sob sua custódia e em absoluto sigilo a chave privada que ela criou, publicando o par correspondente. A chave privada somente será publicada na data que o solicitante do par estabelecer no momento da solicitação, e somente nos termos definidos pelo solicitante.

Uma vez obtida a chave pública, o solicitante das chaves, também chamado de *cliente*, poderá cifrar documentos com esta chave e emití-los aos chamados *usuários* da ACT. Estes usuários, por sua vez, deverão aguardar até o momento definido para a liberação das chaves privadas correspondentes, momento no qual deverão solicitar à ACT estas chaves.

O cliente também poderá fazer outras solicitações à ACT, tais como a prorrogação da data de liberação da chave privada, ou até mesmo a destruição da mesma, o que leva a considerar destruído também o documento sigiloso. O esquema do funcionamento da criptografia temporal aplicado pela ACT está representado na figura 16.

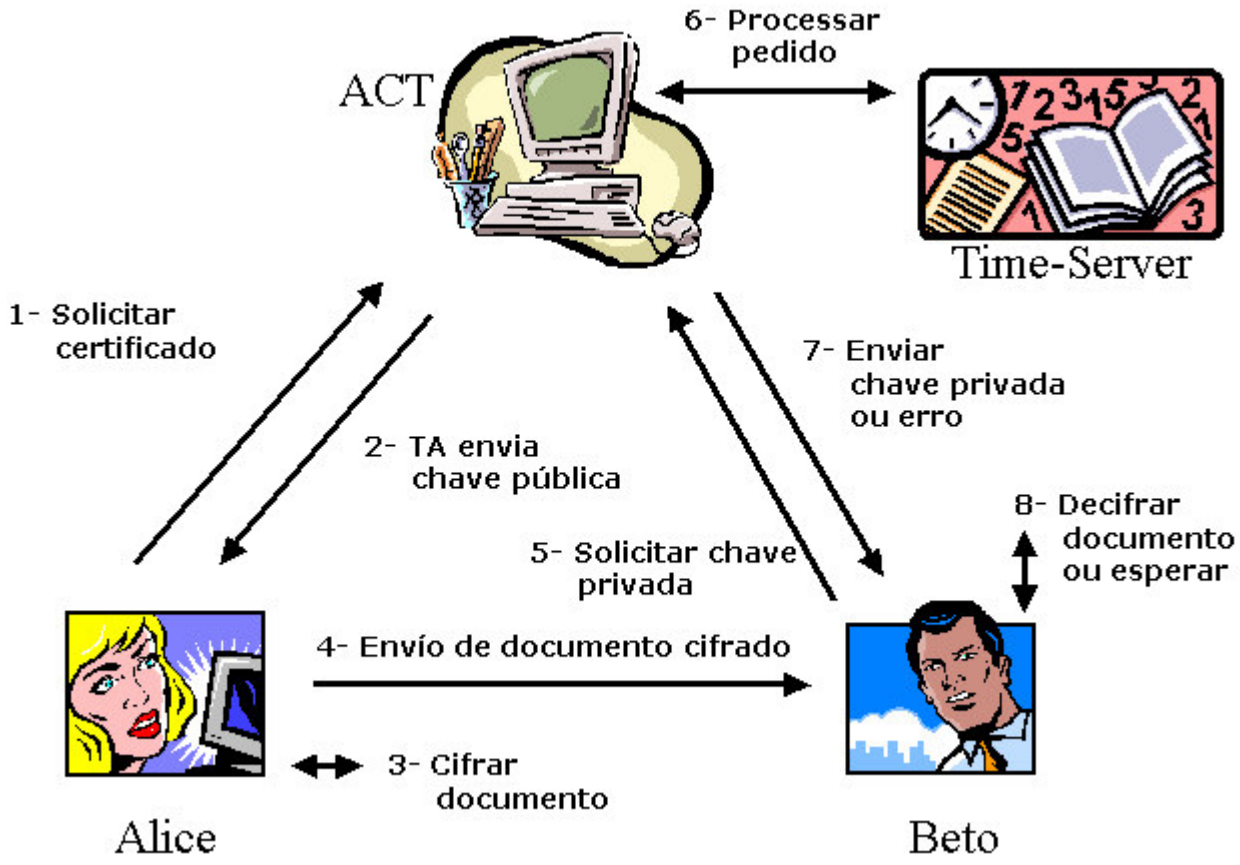

Figura 16. *Criptografia temporal, projeto ACT: Alice solicita um certificado de chave pública temporal, informando a data da liberação da chave privada, o propósito ao que se destina o certificado, assim como a forma da liberação da chave privada. Com a chave pública temporal, Alice cifra o documento e o envia a Beto, que, por sua vez, solicita a chave no momento previsto por Alice. Uma vez aprovada a liberação da chave privada pela ACT, ela é publicada pela ACT e recuperada por Beto, que a utiliza para liberar o documento.*

1- O cliente da ACT, Alice, solicita um certificado de chave pública temporal, informando a data da liberação da chave privada, o propósito ao que se destina o certificado, assim como a forma da liberação da chave privada;
2- A ACT atende à requisição do cliente, gerando um par de chaves criptográficas e publicando somente a chave pública. A ACT cuida do sigilo da chave privada correspondente;
3- Com a chave pública, Alice cifra o documento;
4- Alice já pode enviar o documento a Beto;
5- Beto solicita a chave privada no momento previsto por Alice;
6- A ACT estuda a solicitação do usuário Beto;
7- Uma vez aprovada a liberação da chave privada pela ACT, ela é entregue ao detentor do documento sigiloso;
8- Beto pode agora liberar o documento.

A ACT faz parte de uma infra-estrutura de chaves públicas (ICPT) para o sigilo temporal de documentos, que tenta garantir alguns requisitos mínimos de segurança, citados a seguir [Custodio et al., 2003]:

I- Não pode ser possível determinar o conteúdo do documento sigiloso antes da data de sua abertura.

II- A chave de deciframento que permite o acesso ao conteúdo do documento não deve ser conhecida antes da data combinada para sua liberação.

III- Deve ser possível destruir o documento sigiloso, sem que o seu conteúdo seja revelado.

IV- Deve ser possível a auditoria das atividades realizadas pelas entidades envolvidas, bem como a auditoria dos recursos utilizados.

V- Deve ser possível controlar o acesso ao conteúdo do documento. A chave de deciframento só pode ser liberada para a entidade que tem a posse do documento sigiloso;

VI- Deve ser possível provar a autenticidade e o conteúdo do documento. O documento, após ser revelado, deve ser autêntico e seu conteúdo deve corresponder ao anteriormente alegado pelo autor.

VII- Não se pode negar conhecer o conteúdo do documento após a liberação da chave de deciframento;

VIII- Deve ser possível registrar quem ou o conjunto de pessoas que testemunharam a revelação do conteúdo do documento.

Para garantir estes requisitos, a ACT interage com outros módulos componentes da ICPT que dão suporte ao gerenciamento de clientes, controle estrito de tempo, controles antifraude e até o suporte a compatibilidade, como o uso de criptografia assimétrica tradicional RSA, emitindo certificados X.509, padrão a nível mundial, usado na maioria dos aplicativos de clientes de e-mail e browsers.

Capítulo 5

O protótipo de criptografia temporal IBE

O protótipo de criptografia temporal IBE consiste na implementação de criptografia temporal baseada em IBE (na manipulação de chaves de tempo) e complementada com criptografia tradicional para segurança do canal de comunicação.

Esta implementação tem como objetivo garantir a fácil manipulação das chaves públicas de tempo assim como estabelecer um canal seguro entre as entidades envolvidas na comunicação. Em outras palavras, deve ser fácil para Alice cifrar um documento para Beto de maneira a somente ele poder ter acesso ao conteúdo do mesmo, na data e hora especificada por Alice. Esta facilidade é garantida graças ao IBE devido ao fato de não ser necessário a criação de um par de chaves criptográficas para tornar sigiloso um documento através do tempo. Basta cifrar este documento com uma chave pública indicando o tempo da sua abertura, no formato data/hora, e garantir o controle de liberação de chaves sob uma estrita vigilância do tempo. O esquema é apresentado na figura 17.

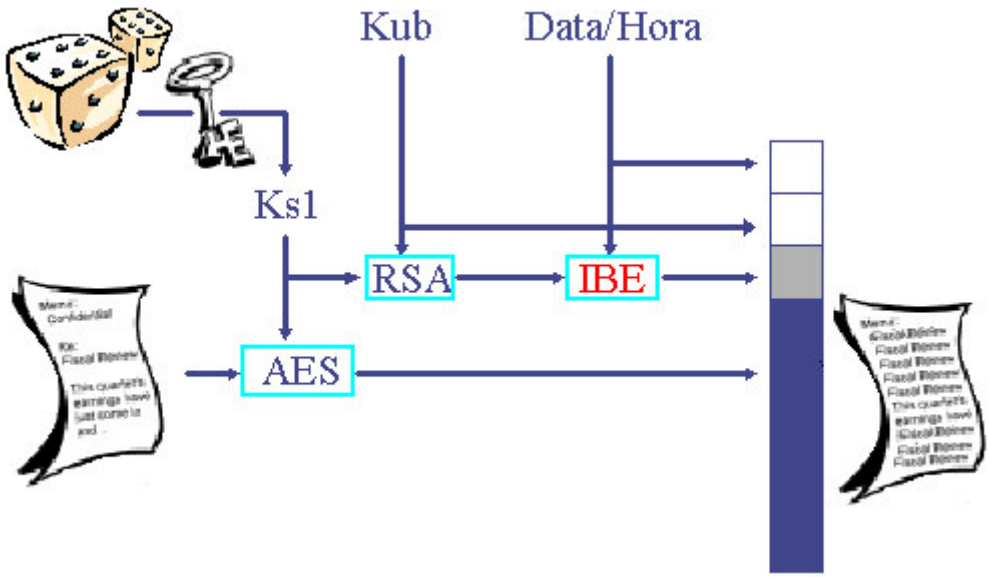

Figura 17. *Fluxograma de dados do esquema do projeto: O documento é cifrado com uma chave aleatória Ks1 usando criptografia simétrica AES. Ks1 é cifrada com a chave pública de Beto usando criptografia assimétrica RSA (chave no formato padrão PEM), o resultado é novamente cifrado, mas desta vez com a chave pública IBE, usando uma data/hora. A chave cifrada é concatenada ao documento cifrado, juntamente com a chave pública RSA do destinatário e a chave pública IBE, informando a data/hora da quebra do sigilo. O meta-documento resultante contém informações sobre a data da quebra do sigilo e o destinatário do documento.*

Foi criado um programa gerenciador chamado ***ibe-act***, o qual gerencia três dos quatro passos da implementação de Boneh & Franklin (Cifragem, Decifragem e Extração). O programa ***ibe-act*** esta encarregado de chamar os sub-programas IBE através de uma lista de opções. Programas que foram modificados para dar suporte ao canal de comunicação seguro RSA, conforme figura 17.

Vamos supor agora que Alice queira cifrar um documento (carta.txt) para Beto, mas ela só quer que ele o abra no dia 20 de abril de 2005 às 15h30. Para isso, deve acontecer o seguinte:

1- Alice abre o programa ***ibe-act*** e manda cifrar o documento entrando com os seguintes parâmetros: Semente geradora de *r* (número de até 9 dígitos), documento a cifrar *(carta.txt)*, destinatário (chave pública de Beto: KuB, no formato PEM) e data e hora da publicação da chave de deciframento (no formato, i.e.: 200504201530).

2- O programa cliente gera uma chave aleatória (*Ks1*) para cifrar o documento com criptografia simétrica (AES) e cifra esta chave com a chave pública do Beto (RSA).

3- A chave *Ks1* cifrada acima é novamente cifrada, mas desta vez com uma data/hora (como chave pública do IBE), logo é concatenada ao documento cifrado junto com a data/hora especificada para a quebra do sigilo, mais a chave pública do destinatário Como resultado final temos um documento extendido contendo informações sobre a data da quebra do sigilo e o destinatário do documento.

Alice já pode entregar o documento para o seu destinatário, no caso Beto, quem não poderá obter a chave de deciframento da PKG senão até a data especificada por Alice. Não adianta Beto mudar a data no documento, visto que a chave devolvida pela PKG não corresponderá à chave pública IBE com que foi cifrado o documento. Beto só pode esperar.

Supondo que um intruso, ou um criptanalista (Carlos) obtenha a chave com a qual Alice cifrou o documento para Beto, não adianta Carlos ter uma cópia do documento enviado por Alice, pois uma vez liberada a chave para abrir o documento, somente Beto pode chegar até o texto original graças a sua chave privada RSA.

Beto somente solicita à PKG a chave correspondente à data/hora especificada no documento. A PKG produz a chave correspondente só se a data/hora solicitada já foi atingida, e, o mais importante, só quando a chave é solicitada. Esta implementação evita congestionamentos no tráfego de dados assim como processamentos e armazenamentos desnecessários, uma vez que todos os dados transmitidos e processados são mínimos e atendem aos critérios de segurança para os quais são destinados.

5.1- Detalhes da implementação

Para a implementação o ambiente escolhido foi Mandrake 8.2, com a biblioteca OpenSSL, e compiladores gcc/g++.

A PKG está compreendida pelos programas IBE_SET (o mesmo que o do MIRACL), e IBE_EXT_ACT (modificado para controle de tempo).

O servidor onde deve rodar a PKG deve possuir um controle de tempo seguro (protocolo NTPv4), sincronizado com o relógio atômico do Observatório Nacional.

A PKG está encarregada de gerar e publicar os valores *N=p*q* (por meio de IBE_SET), assim como da liberação ou não das chaves de deciframento (por meio do programa IBE_EXT_ACT).

A própria IBE (ibe_enc.cpp do MIRACL) já implementa a cifragem de documentos via criptografia simétrica (AES), por meio de uma chave aleatória. Este procedimento será aproveitado para a implementação do esquema proposto. Por esse motivo, o anexo A apresenta uma pequena variante com relação ao esquema da figura 17. A diferença consiste em que depois da cifragem de *Ks1* com criptografia assimétrica RSA, é criado um meta-documento temporario que novamente será cifrado com AES, mas utilizando uma segunda chave de sessão (*ks2*). Somente esta segunda chave é cifrada com IBE. Mas, por motivo de desempenho o esquema proposto na figura 17 é o mais conveniente, já que não apresenta redundância na cifragem com AES.

5.1.1- Ferramentas utilizadas

Bibliotecas OpenSSL.
Bibliotecas e programas do MIRACL.
Bibliotecas da linguagem C/C++
Protocolo NTPv4 de controle de tempo seguro.

5.1.1.1- Funções específicas OpenSSL

Biblioteca de erro openssl <openssl/err.h>
Com a seguinte função podemos debugar erros nas funções RSA.
ERR_error_string(ERR_get_error(), NULL);

Biblioteca RSA <openssl/rsa.h>
Com esta função validamos, ou não, uma chave RSA.
Usada em ibe_enc_act.cpp, ibe_dec_act.cpp
RSA_check_key(key_)

A função a seguir cifra uma mensagem com uma chave pública.
Usada em ibe_enc_act.cpp
RSA_public_encrypt(len_, plain_, *cipher_, key_, RSA_PKCS1_PADDING)

Esta função decifra uma mensagem com uma chave particular.
Usada em ibe_dec_act.cpp
RSA_private_decrypt(len_, cipher_, *plain_, key_, RSA_PKCS1_PADDING)

Gerador de números aleatórios <openssl/rand.h>
Usada em ibe_enc_act.cpp
irand(seed);
for (i=0;i<HASH_LEN;i++) key[i]=(char)**brand()**

Biblioteca para chaves padrão PEM <openssl/pem.h>
Esta função lê uma chave pública em formato .pem
Usada em ibe_enc_act.cpp
PEM_read_RSAPublicKey(fp_,NULL,NULL,NULL)

Esta função lê uma chave pública em formato .pem
Usada em ibe_dec_act.cpp
PEM_read_RSAPrivateKey(fp_,NULL,NULL,NULL)

5.1.1.2- Funções específicas MIRACL

Programa IBE já implementando criptografia simétrica AES. É preciso a instalação da biblioteca para dar suporte às implementações das curvas elípticas do IBE. Assim como a aritmética de precisão.

Foram modificados os seguintes arquivos:
-ibe_enc.cpp
-ibe_dec.cpp
-ibe_ext.cpp

Os detalhes das moficações estão disponíveis no anexo que acompanha esta monografia.
Os arquivos originais estão disponíveis em http://indigo.ie/~mscott/

5.1.1.3- Funções específicas das bibliotecas C/C++

Biblioteca de tempo <time.h>
Consulta do tempo:
time_ptr = localtime(&time_now)
time(&time_now)

Formatando o tempo para exibição na tela:
strftime(string_time,64,"%Y%m%d%H%M",time_ptr)

Biblioteca de sistema <cstdlib>
Chamadas a sistema:
system("clear")

Biblioteca de sistema <unistd.h>
Manipulando arquivos externos:
unlink(ifname)

5.2- Exemplo de uso

5.2.1- Configuração da PKG:

Primeiramente devemos configurar a PKG, com exatamente o mesmo procedimento do capítulo anterior, gerando e publicando $N=p*q$ através do programa IBE_SET.

O programa IBE_EXT_ACT, que gera as chaves privadas, só fica aguardando ser chamado.
A configuração de IBE_SET é mostrada na figura 18:

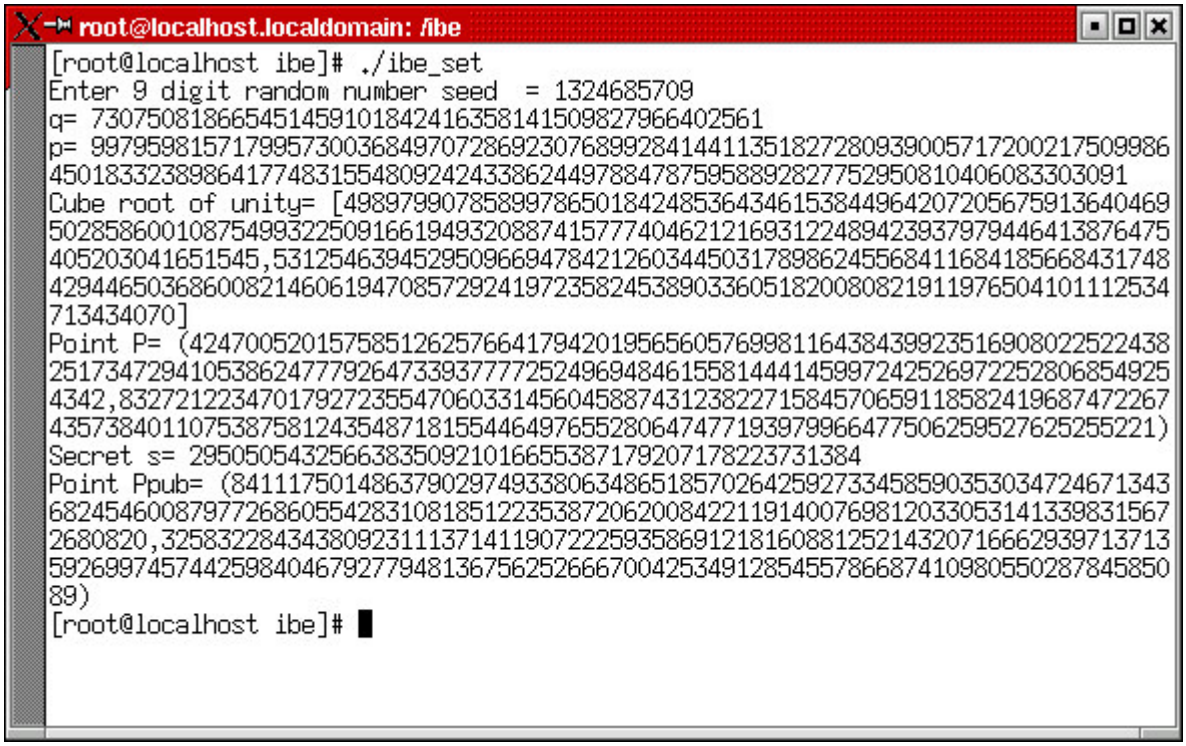

Figura 18. *Programa de configuração da PKG do IBE.*

5.2.2- Cifrando e decifrando arquivos

Agora que possuímos o arquivo "common.ibe" (gerado por IBE_SET) podemos proceder ao uso dos programas de manipulação de arquivos. Vamos supor que queiramos cifrar o arquivo "miracl-ibe.zip" para ser aberto numa data futura. No exemplo, o dia 08 de maio de 2004, às 14h00. Chamamos então o programa IBE-ACT (O resultado aparece na figura 19):

 # ./ibe-act

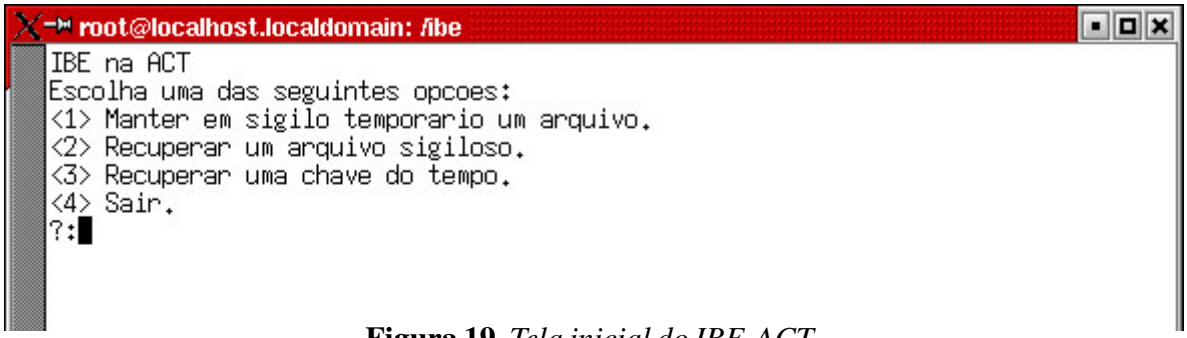

Figura 19. *Tela inicial do IBE-ACT.*

Escolhemos então a primeira opção, na qual é chamado o programa de cifragrem IBE_ENC_ACT, como se vê na figura 20.

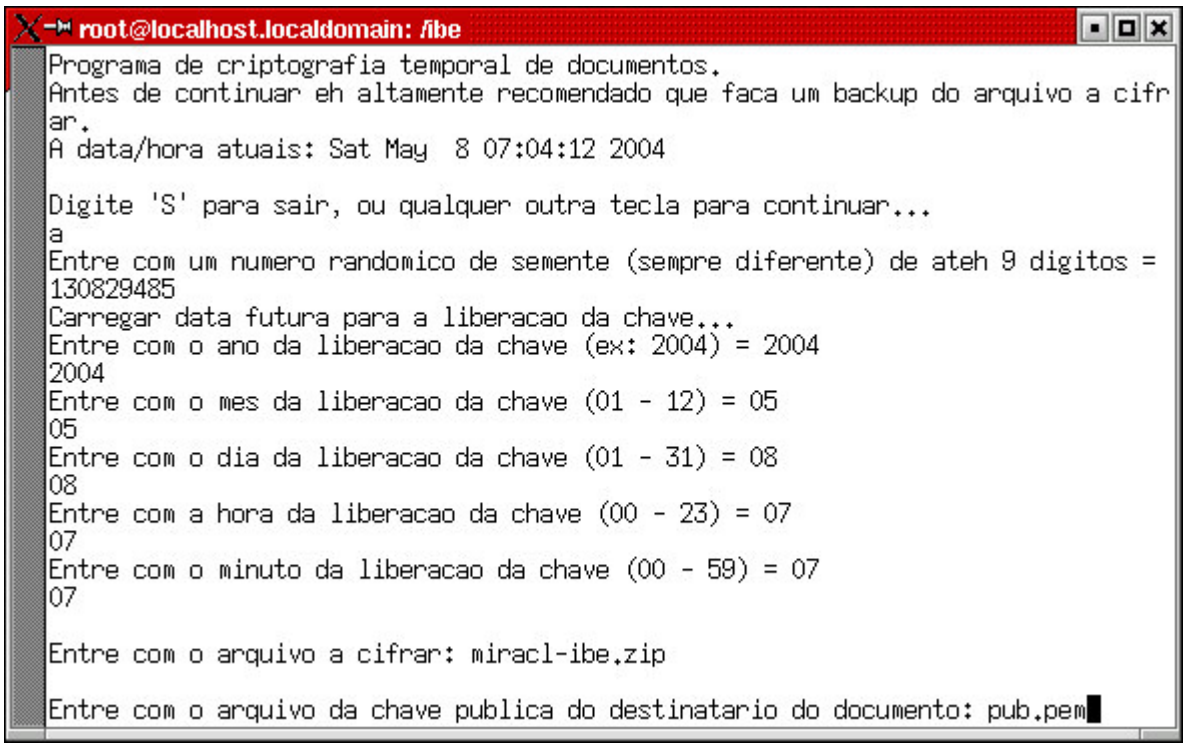

Figura 20. *Tela da opção 1, onde ciframos um documento (miracl-ibe.zip) para às 7h07 do dia 8 de maio de 2004.*

A data/hora atual é exibida somente para servir de referência.

É solicitado ao usuário digitar um número randômico para servir de semente na geração de *r*. Logo, são inseridos os dados do momento da liberação da chave. Este processo está dividido em entrada de ano, de mês, de dia, de hora e de minuto para o simples efeito de controle da validade dos dados e evitar confusão ao usuário. Note que o sistema permite entrada de dados de tempo do tipo inexistente, como por exemplo, 30 de fevereiro de 2005, mas isso não representa problema por que a liberação da chave poderá ser realizada no primeiro dia válido após a data questionada, no caso, a chave seria liberada no dia 01 de março de 2005.

Continuando com o programa em execução, por fim é solicitado o nome do arquivo a cifrar, digitamos "miracl-ibe.zip" e logo é solicitado que ingressemos o nome do arquivo da chave pública do destinatário (no formato "pem"), no nosso exemplo foi "pub.pem".
Uma vez acabada a etapa de cifragem do documento, é gerado o meta documento "miracl-ibe.act", que é o nosso arquivo cifrado.

Voltando à tela inicial saímos do programa para verificar o arquivo gerado. Veja na figura 21.

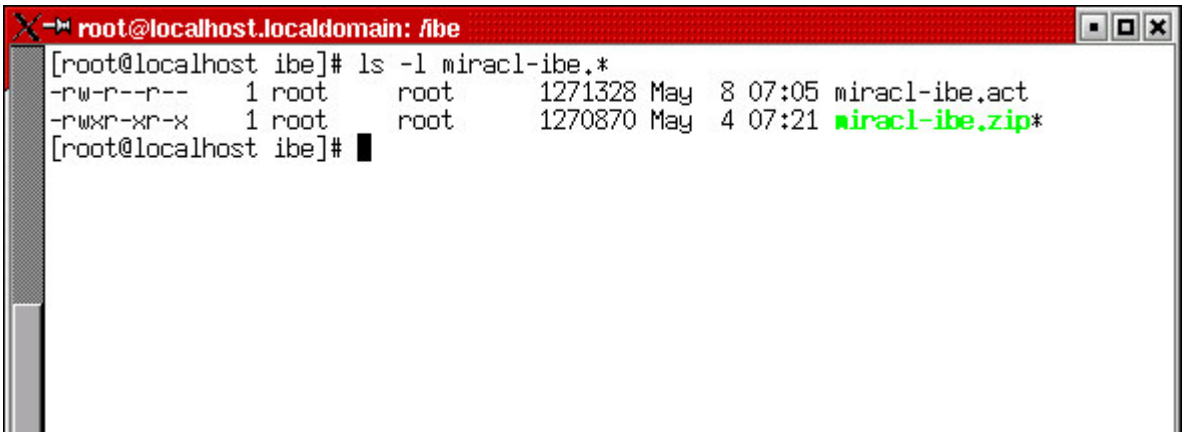

Figura 21. *Lista dos documentos do teste (miracl-ibe*).*

Note que o meta documento ***miracl-ibe.act*** ocupa mais espaço em disco devido às informações referentes à recuperação do arquivo original.

Vamos proceder à tentativa de recuperação do documento, supondo que a condição data/hora não tenha sido cumprida. Chamando de novo o nosso programa IBE-ACT, desta vez ingressamos com a opção "recuperar uma chave do tempo", pois precisaremos da chave privada IBE, correspondente à data/hora, que virá num arquivo ***".pri"***, e em nosso exemplo seria ***miracl-ibe.pri***. Veja na figura 22.

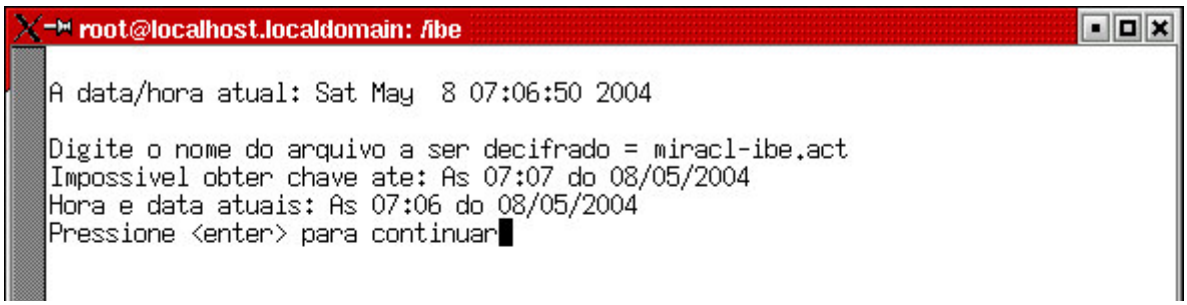

Figura 22. *Mensagem de erro no programa IBE_EXT_ACT.*

O programa lê o cabeçalho do arquivo *.act* e verifica se a data/hora contida nele é menor ou igual à impressa na tela (horário do servidor). Caso afirmativo, retorna satisfatoriamente um arquivo *.pri*, caso contrário, exibe uma mensagem contendo a data/hora em que o usuário deve solicitar a chave.

Veja na figura 23 um outro exemplo no qual o usuário consegue a chave devido ao tempo alvo ter sido atingido:

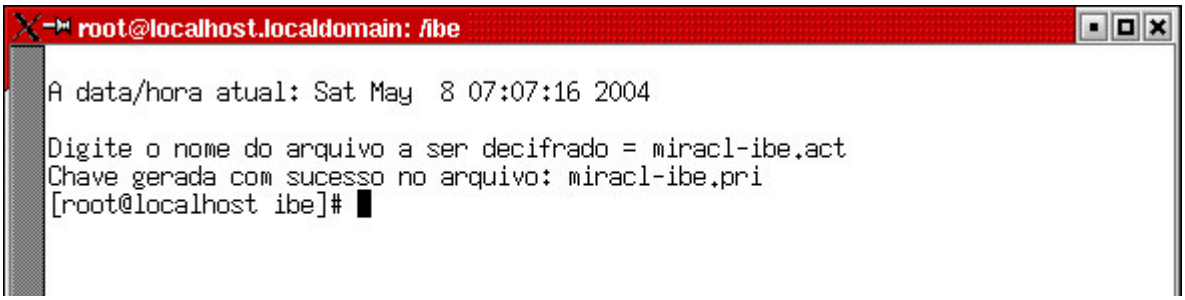

Figura 23. *Sucesso na recuperação da chave particular IBE.*

Continuando com o nosso exemplo, uma vez obtida a chave junto à TA, devemos prosseguir à abertura do documento. Antes conferimos se possuímos os arquivos necessários: miracl-ibe.act e miracl-ibe-ibe.pri, além de, é claro, a nossa chave particular RSA (no exemplo, sec.pem). Rodamos então o IBE-ACT e vamos à segunda opção.

O programa IBE_DEC_ACT aparece na tela. Veja figura 24.

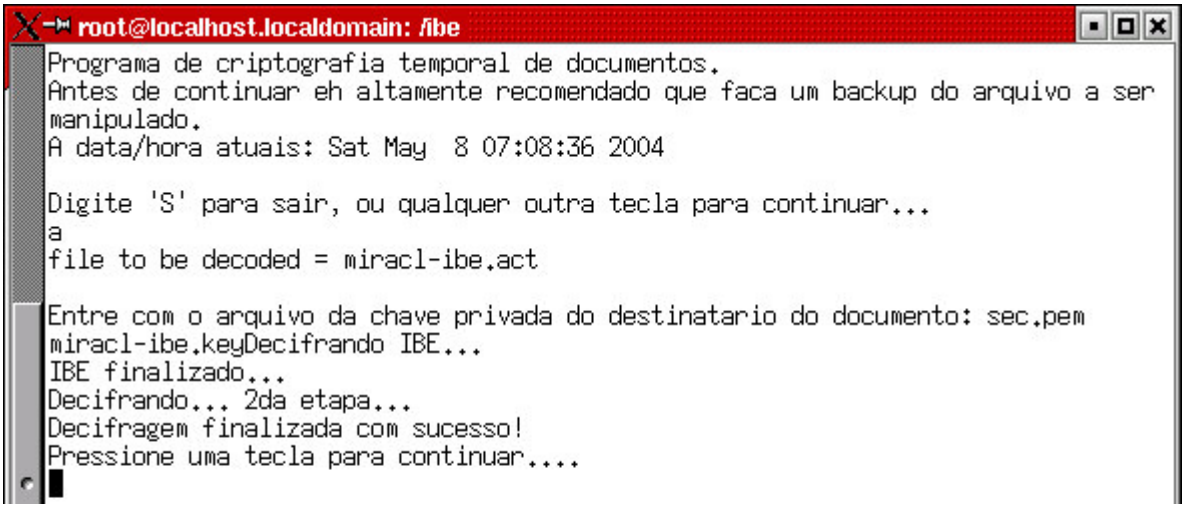

Figura 24. *Decifragem finalizada com sucesso!*

Ingressamos o nome do arquivo a ser decifrado e o arquivo da nossa chave particular RSA.

O programa processa o meta documento *.act* e devolve o arquivo original com a extensão *.ptx* (plaintext). O arquivo *miracl-ibe.ptx* é exatamente igual ao arquivo original *miracl-ibe.zip*, e foi nomeado assim para fins didáticos. Vejamos na figura 25 os arquivos gerados.

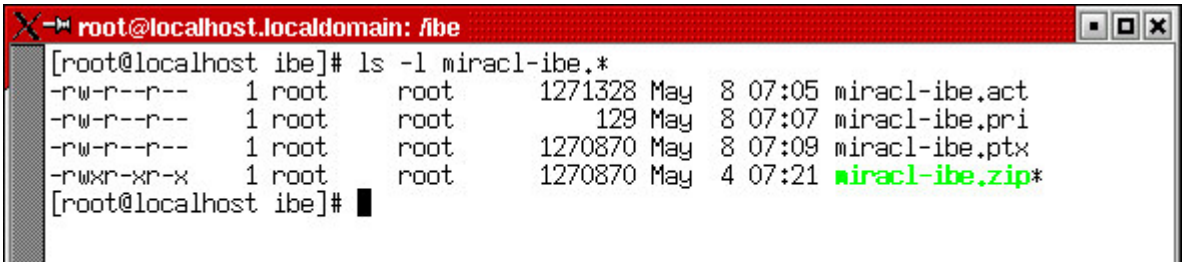

Figura 25. *Os 3 arquivos gerados no processo.*

Note que o arquivo ***.ptx*** ocupa exatamente o mesmo espaço em disco que o seu original ***.zip***.

Nos testes realizados o sistema reconheceu corretamente o arquivo com a extensão ***.ptx*** como cópia fiel do documento original, sendo lido corretamente e não apresentando nenhum erro no processo de descompactação.

Capítulo 6

Conclusões

A criptografia baseada em identidade facilita a manipulação de chaves públicas, permitindo controlar o tempo em que as chaves privadas correspondentes serão criadas, viabilizando a sua aplicação em sistemas criptográficos temporais. Podemos usar strings em qualquer formato nas chaves públicas para um melhor controle da geração das chaves privadas em momentos específicos. Para garantir um canal seguro de comunicação entre duas entidades foi utilizada, com sucesso, a criptografia assimétrica tradicional (RSA), com ferramentas OpenSSL, utilizando chaves criptográficas no padrão PEM. Tudo isso combinado à eficiência da criptografia simétrica (AES), a qual garante melhor performance nas transformações criptográficas de grande quantidade de dados.

Embora a implementação deste trabalho não mantenha um controle sobre o destino das chaves privadas liberadas, elas somente terão utilidade para o destinatário do documento sigiloso. Cifrar inicialmente o documento com a chave pública do destinatário é um procedimento opcional, conforme no protótipo, que vista agregar segurança ao documento, pois impede que entidades alheias à comunicação que tenham acesso à chave privada temporal consigam ler o documento sigiloso. O resultado foi um sistema que realmente protege documentos através do tempo, sempre dentro de um canal seguro e cumprindo com os propósitos impostos sobre o sigilo.

Em comparação com o modelo da ACT, o IBE não oferece o suporte necessário a alguns requisitos de segurança, como por exemplo, a destruição de documentos sigilosos, pois não seria possível a destruição de uma chave privada, visto que o seu respectivo par público poderia ser usado para cifrar um documento para vários usuários distintos. Outro ponto fraco é o controle da liberação das chaves privadas, já que não existe um controle explícito no gerenciamento das chaves. Também a auditoria seria menos abrangente, pois o gerador de chaves privadas PKG do IBE não conhece as entidades envolvidas na comunicação. O esquema de segurança do IBE é mais fraco que o da ACT, pois o comprometimento da chave mestra da PKG do IBE pode significar na liberação ou perda de todas as chaves privadas temporais de todos os usuários, o que não aconteceria na ACT, visto que um comprometimento na sua infra-estrutura a afetaria apenas parcialmente.

A adoção do IBE dentro da Autoridade Certificadora Temporal acarretaria na perda da autonomia e da facilidade do gerenciamento de chaves, ideais do IBE, visando garantir os requisitos de segurança. Por exemplo, para poder destruir um documento sigiloso, poderíamos extender a chave pública IBE com o hash do documento. A chave pública não seria mais então somente a Data/Hora da liberação da chave e sim a Data/Hora mais o resumo hash do documento. Isto possibilitaria a solicitação da exclusão da chave privada, visto que seria mantido um registro dos documentos destruídos. Para isto, é necessário manter um registro das atividades realizadas pelo cifrador de documentos. Assim, se chegar uma solicitação de liberação de uma chave privada de um documento registrado como destruído, esta seria negada. Tudo isto somente seria possível

com a ajuda da ICPT, que tornaria válida uma solicitação assim, com a autenticação dos clientes que solicitem este serviço. A PKG teria que conhecer as entidades envolvidas na comunicação.

O principal problema encontrado durante este trabalho foi a implementação IBE usada no protótipo, já que a sua arquitetura pouco flexível não permitiu o seu uso como biblioteca. Houve necessidade de alterar diretamente o fluxo original do programa para tornar possível o trabalho.

Na implementação do canal seguro entre as duas entidades (confidencialidade) não utilizamos IBE ao invés de RSA (poderíamos usar também o IBE pensando na idéia de libertar o sistema totalmente de toda a infra-estrutura de chaves públicas, como dita o ideal do esquema IBE) pelo fato de que ainda há desvantagens apresentadas pelo IBE, como sendo a grande dependência da chave mestra da PKG, e o conhecimento por parte dela de todas as chaves dos usuários. O esquema de segurança físico por trás dela deve ser muito forte, além da confiança que seria depositada toda numa única fonte. Isto dificulta esse tipo de implementação.

Finalmente, uma sugestão para trabalho futuro seria implementar uma biblioteca IBE para permitir criar aplicações mais otimizadas e flexíveis baseadas nesta tecnologia. Os recursos disponíveis até o fim deste trabalho carecem de flexibilidade.

Bibliografia

[Benits Jr., W., 2003] Sistemas Criptográficos Baseados em Identidades Pessoais. Master's thesis, Universidade de São Paulo, 2003.

[Boneh, D. & Franklin, M., 2001] Identity-based encryption. Disponível em <http://crypto.stanford.edu/ibe/ >

[CPLUSPLUS, 2004] The C++ Resources Network. Disponível em <http://www.cplusplus.com>

[Custódio et al., 2003] Custodio R. F., da Silva Dias J., Pereira F. C. e Notoya A. C., 2003. Uma infra-estrutura para o sigilo temporal de documentos eletrônicos. Universidade Federal de Santa Catarina, 2003.

[Custódio, R. F., 2000] CUSTÓDIO, R. F. Notas de aula. Disponível em <http://www.inf.ufsc.br/~custodio/cursos/INE5386/Transparencias/Apostila.zip>

[Grandi, A., 2002] Andrea Grandi, PGP. Mettere i propri dati al sicuro. Disponível em <http://pdf.apogeonline.com/ebook/2002/90025/pdf/PGP.pdf >

[Liberty, J., 1999] LIBERTY, J., C++ para principiantes. Indianapolis - USA, Pretince Hall - 1999.

[May, 2003] Timed-release crypto. Disponível em <http://cypherpunks.venona.com/date/1993/02/msg00129.html>

[Mont et al., 2003] Mont, M. C., Harrison, K., and Sadler, M., 2003. The HP Time Vault Service: exploited ibe for timed release of confidencial information. In *In Proceedings of the twelfth international conference on World Wide Web*, pages 160-169. ACM.

[NTP.org, 2003] NTP: The Network Time Protocol. Disponível em <http://www.ntp.org>

[Observatorio Nacional, 2004] Hora legal Brasileira. Disponível em <http://pcdsh01.on.br>

[OpenSSL, 2004]The open project. Disponível em <http://www.openssl.org>.

[Patroklos G. Argyroudis, 2004] Identity-based encryption resources. Disponível em <http://ntrg.cs.tcd.ie/~argp/ibe.html>

[Pierre Loidreau, 2002] System administrator: Introducción a la criptografía. Disponível em <http://www.linuxfocus.org/Castellano/May2002/article243.shtml>

[Pino C.,G., 2002] Pino Caballero Gil, Introducción a la Criptografia. Madrid. RA-MA - 2002

[Quevedo, 2002] El rincón de Quevedo. Disponível em <http://www.rinconquevedo.tk/>

[Scott, M., 2000] Shamus Software Ltd - MIRACL. Disponível em <http://indigo.ie/~mscott/>

[Schneier, B., 1996] Schneier, B., *Applied Cryptography*, 2º edición, Ed. Wiley, 1996

[Tkotz 1999] TKOTZ, V. Página na internet sobre criptologia. 1999. Disponível em <http:www.numaboa.com/informatica/criptologia>.

[Wilkich, 2004] The Miracl Library. Disponível em <http://www.auburn.edu/~wilkich/>

Anexo A

Código Fonte:
ibe_ext_act.cpp
ibe_enc_act.cpp
ibe_dec_act.cpp
ibe-act.cpp

```
//ibe_ext_act.cpp modificado

#include <iostream>
#include <fstream>
#include <cstring>
#include "elliptic.h"
#include "monty.h"

//Begin ibe-act

#include <time.h>

//End ibe-act


using namespace std;

// Using SHA-1 as basic hash algorithm

#define HASH_LEN 20

//
// Hash function
//

Big H1(char *string)
{ // Hash a zero-terminated string to a number < modulus
    Big h,p;
    char s[HASH_LEN];
    int i,j;
    sha sh;

    shs_init(&sh);

    for (i=0;;i++)
    {
        if (string[i]==0) break;
        shs_process(&sh,string[i]);
    }
    shs_hash(&sh,s);

    p=get_modulus();
    h=1; j=0; i=1;
    forever
    {
        h*=256;
        if (j==HASH_LEN)  {h+=i++; j=0;}
        else          h+=s[j++];
        if (h>=p) break;
    }
    h%=p;
    return h;
}

//
// Given y, get x=(y^2-1)^(1/3) mod p (from curve equation)
//
```

```
Big getx(Big y)
{
    Big p=get_modulus();
    Big t=modmult(y+1,y-1,p);   // avoids overflow
    return pow(t,(2*p-1)/3,p);
}

//
// MapToPoint
//

ECn map_to_point(char *ID)
{
    ECn Q;
    Big x0,y0=H1(ID);

    x0=getx(y0);

    Q.set(x0,y0);

    return Q;
}

/**************Begin IBE-ACT*****************/
void strip(char *name)
{ /* strip off filename extension */
    int i;
    for (i=0;name[i]!='\0';i++)
    {
        if (name[i]!='.') continue;
        name[i]='\0';
        break;
    }
}


bool controle(int i, char *arg){
  int size,valor;
  //size = strlen(arg);
  valor = atoi(arg);
  int temp1,temp2;

  if (i==1){
    temp1 = 2003;

  //  if (size!=4){
    //  printf("ano invalido!\n");
    //  return FALSE;
   // }
    if (valor<=temp1){
      printf("ano invalido!\n");
      return FALSE;
    }

  }
```

```
  if (i==2){
    temp1 = 1;
    temp2 = 12;

   // if (size!=2){
     // printf("mes invalido!\n");
    //  return FALSE;
   // }
    if (valor<temp1 || valor>temp2){
      printf("mes invalido!\n");
      return FALSE;
    }
  }

  if (i==3){
    temp1=1;
    temp2=31;

   // if (size!=2){
   //   printf("dia invalido!\n");
   //   return FALSE;
   // }
    if (valor<temp1 || valor>temp2){
      printf("dia invalido!\n");
      return FALSE;
    }
  }

  if (i==4){
    temp1=0;
    temp2=23;

   // if (size!=2){
   //   printf("hora invalida!\n");
   //   return FALSE;
   // }
    if (valor<temp1 || valor>temp2){
      printf("hora invalida!\n");
      return FALSE;
    }
  }

  if (i==5){
    temp1=0;
    temp2=59;

   // if (size!=2){
   //   printf("minuto invalido!\n");
   //   return FALSE;
   // }
    if (valor<temp1 || valor>temp2){
      printf("minuto invalido!\n");
      return FALSE;
    }
  }
  return TRUE;
}
```

```
/**************End IBE-ACT*****************/

int main()
{
    miracl *mip=mirsys(18,0);     // thread-safe ready.  (36,0) for 1024 bit p
    ifstream common("common.ibe");
    ifstream master("master.ibe");
    ifstream plaintext;
    ofstream private_key;//("private.ibe");
    char ch,ifname[100];
    ECn Qid,Did;
    Big p,q,cof,s,x,y;
    int bits;
    bool ok;
    common >> bits;
    mip->IOBASE=16;
    common >> p >> q;
    master >> s;
    mip->IOBASE=10;

    ecurve(0,1,p,MR_PROJECTIVE);
    cof=(p+1)/q;

/**************Begin IBE-ACT*****************/
    char id[20],id_[20],ano[10],mes[10],dia[10],hora[10],minuto[10];
    int valor_,i;
    bool libera;
   time_t time_now;
   time_t time_doc;
   struct tm *time_ptr;
   struct tm *time_ptr_;
   time_ptr = localtime(&time_now);
   time_ptr_ = (struct tm *)malloc(sizeof(struct tm));

   //time_t rawtime;
   time(&time_now);

   system("clear");
   printf("\nA data e hora atuais: %s \n", ctime(&time_now));



    cout << "Digite o nome do arquivo a ser decifrado = " ;
    cin >> ifname;

    strip(ifname);
    strcat(ifname,".act");
    plaintext.open(ifname,ios::in);
    if (!plaintext)
    {
        cout << "Impossivel abrir " << ifname << "\n";
        return 0;
    }

     for(i=0;i<12;i++)
    {
```

```
      plaintext.get(ch);
      if (plaintext.eof()) break;
      id_[i] = ch;
    }

    ok=FALSE;

    for (i=0;i<4;i++)ano[i]=id_[i];
    ok=controle(1,ano);
    if(ok==FALSE) return 0;
    valor_ = atoi(ano);
    time_ptr_->tm_year = valor_ - 1900;


    for (i=0;i<2;i++)mes[i]=id_[i+4];
    ok=controle(2,mes);
    if(ok==FALSE) return 0;
    valor_ = atoi(mes);
    time_ptr_->tm_mon = valor_ - 1;



    for (i=0;i<2;i++)dia[i]=id_[i+6];
    ok=controle(3,dia);
    if(ok==FALSE) return 0;
    valor_ = atoi(dia);
    time_ptr_->tm_mday = valor_;



    for (i=0;i<2;i++)hora[i]=id_[i+8];
    ok=controle(4,hora);
    if(ok==FALSE) return 0;
    valor_ = atoi(hora);
    time_ptr_->tm_hour = valor_;



    for (i=0;i<2;i++)minuto[i]=id_[i+10];
    ok=controle(5,minuto);
    if(ok==FALSE) return 0;
    valor_ = atoi(minuto);
    time_ptr_->tm_min = valor_;


    char string_time[64],string_time_[64];
    char string_time_p[64],string_time_p_[64];
    strftime(string_time_,64,"%Y%m%d%H%M",time_ptr_);
    strftime(string_time,64,"%Y%m%d%H%M",time_ptr);


//rotina de comparacao de tempos
    libera = FALSE;
    i=0;
    while(libera == FALSE){
      if (string_time_[i] > string_time[i]){
      strftime(string_time_p,64,"As %H:%M do %d/%m/%Y",time_ptr_);//data e hora
```

```
do documento
      strftime(string_time_p_,64,"As %H:%M do %d/%m/%Y",time_ptr);//data e hora
atuais
      printf("Impossivel obter chave ate: %s \n", string_time_p);
      printf("Hora e data atuais: %s \n", string_time_p_);
      return 0;
      }
      if (string_time_[i] < string_time[i]) libera = TRUE;//se data/hora do
documento for menor, libera
      if (string_time_[i] == string_time[i]) i++;//se for igual, passa ao
seguinte i
      if (i>11) libera = TRUE;//se forem todos os i's iguais, libera.
    }



//o seguinte codigo comentado eh uma tentativa sem exito de comparacao mais
explicita dos tempos
/*
    //if (time_ptr > time_ptr_) cout << "tempo atual maior!" << endl;
    //    strptime(string_time,,"%Y %m %d %H %M",time_ptr_);
    //   time(&time_doc);
    // time_doc = mktime(time_ptr_);

    //    if(string_time_ > string_time){
    // printf("nao libera\n");
    //  }else{
    //      if(string_time_ == string_time)printf("iguais!\n");
    //    printf("sim, libera\n");
    //   }
    //  time(&time_doc);
    //printf("A data/hora atual2: %s", ctime(&time_doc));

    //double i_;
    // i_=difftime(time_doc,time_now);

    // cout << i_ << endl;
*/

  //codigo apagado do arquivo ibe_ext.cpp original
    //    i = strlen(id_);
    //    cout << "id_ size: " << i << endl;

    //    cout << "Enter your email address (lower case)" << endl;
    //    cin.get();
    //    cin.getline(id,20);


    strcpy(id,id_);

    strip(ifname);
    strcat(ifname,".pri");
    private_key.open(ifname,ios::binary|ios::out);



/**************End IBE-ACT****************/
```

```
// EXTRACT

    Qid=map_to_point(id);
    Did=s*Qid;

    Did.get(x,y);
    mip->IOBASE=16;
    private_key << y << endl;


/*************Begin IBE-ACT*****************/

    cout << "Chave gerada com sucesso no arquivo: " << ifname << endl;
    cout << "Pressione <enter> para encerrar...";
    getchar();

/**************End IBE-ACT*****************/



    return 0;
}
```

```
//ibe_enc_act.cpp

#include <iostream>
#include <fstream>
#include <cstring>
#include "elliptic.h"
#include "monty.h"
#include "ebrick.h"
#include "zzn2.h"

//Begin ibe-act
#include <time.h> //controle de tempo
#include <openssl/err.h> //biblioteca de erro openssl
#include <openssl/rsa.h> //biblioteca RSA
#include <openssl/rand.h> //Gerador de numeros aleatorios
#include <openssl/pem.h> //biblioteca para chaves padrao PEM
#include <openssl/objects.h> //objetos openssl
#include <cstdlib> //chamadas a sistema
#include <unistd.h> //manipulacao de arquivos externos (unlink...)
//End ibe-act

using namespace std;

#define PBITS 512
#define QBITS 160

// Using SHA-1 as basic hash algorithm

#define HASH_LEN 20
```

```
//
// Define one or the other of these
//
// Which is faster depends on the I/M ratio - See imratio.c
// Roughly if I/M ratio > 16 use PROJECTIVE, otherwise use AFFINE
//

// #define AFFINE
#define PROJECTIVE

// define SIMPLE if using fixed group order q = 2^159+2^17+1
#define SIMPLE

//
// Tate Pairing Code
//
// Extract ECn point in internal ZZn format
//

void extract(ECn& A,ZZn& x,ZZn& y)
{
    x=(A.get_point())->X;
    y=(A.get_point())->Y;
}

void extract(ECn& A,ZZn& x,ZZn& y,ZZn& z)
{
    big t;
    x=(A.get_point())->X;
    y=(A.get_point())->Y;
    t=(A.get_point())->Z;
    if (A.get_status()!=MR_EPOINT_GENERAL) z=1;
    else                                   z=t;
}

//
// Line from A to destination C. Let A=(x,y)
// Line Y-slope.X-c=0, through A, so intercept c=y-slope.x
// Line Y-slope.X-y+slope.x = (Y-y)-slope.(X-x) = 0
// Now evaluate at Q -> return (Qy-y)-slope.(Qx-x)
//

ZZn2 line(ECn& A,ECn& C,ZZn& slope,ZZn2& Qx,ZZn& Qy)
{
    ZZn2 n=Qx;
    ZZn x,y,z,t,w=Qy;
#ifdef AFFINE
    extract(A,x,y);
    n-=x; n*=slope;            // 2 ZZn muls
    w-=y; n-=w;
#endif
#ifdef PROJECTIVE
    extract(A,x,y,z);
    x*=z; t=z; z*=z; z*=t;
    n*=z; n-=x;                // 9 ZZn muls
    w*=z; w-=y;
    extract(C,x,y,z);
```

```
    w*=z; n*=slope; n-=w;
#endif
    return n;
}

//
// Vertical line through point A
//

ZZn2 vertical(ECn& A,ZZn2& Qx)
{
    ZZn2 n=Qx;
    ZZn x,y,z;
#ifdef AFFINE
    extract(A,x,y);
    n-=x;
#endif
#ifdef PROJECTIVE
    extract(A,x,y,z);
    z*=z;
    n*=z; n-=x;                // 3 ZZn muls
#endif
    return n;
}

//
// Add A=A+B  (or A=A+A) or
// Sub A=A-B
// Bump up num and denom
//
// AFFINE doubling     - 12 ZZn muls, plus 1 inversion
// AFFINE adding       - 11 ZZn muls, plus 1 inversion
//
// PROJECTIVE doubling - 26 ZZn muls
// PROJECTIVE adding   - 34 ZZn muls
//

#define ADD 1
#define SUB 2

void g(ECn& A,ECn& B,ZZn2& Qx,ZZn& Qy,ZZn2& num,ZZn2& denom,int as,BOOL first)
{
    ZZn  lam,mQy;
    ZZn2 d,u;
    big ptr;
    ECn P=A;

    if (as==ADD)
    { // Evaluate line from A, and then evaluate vertical through destination
        ptr=A.add(B);
        if (ptr==NULL)    { num=0; return; }
        else lam=ptr;

        if (A.iszero())   { u=vertical(P,Qx); d=1; }
        else
        {
            u=line(P,A,lam,Qx,Qy);
```

```
            d=vertical(A,Qx);
        }
    }
    else // as==SUB
    { // Evaluate Vertical at A, and then line from A to destination
      // (Note swap num and denom, Qy=-Qy, process lines "backwards")
        u=vertical(A,Qx);
        ptr=A.sub(B);
        if (ptr==NULL)    { num=0; return; }
        else lam=ptr;

        if (A.iszero())   { d=u;  }
        else
        {
            mQy=-Qy;
            d=line(P,A,lam,Qx,mQy);
        }
    }

    if (first) {num= u; denom= d; }
    else       {num*=u; denom*=d; }   // 6 ZZn muls
}

//
// Tate Pairing - note ha -> numerator, had -> denominator
// Trying to minimize number of modular inversions
//
// Special optimized and deterministic version of Tate Pairing algorithm
// Requires that the point (0,1) is on the curve AND
// P and Q(x,y) linearly independent, that is P!=r.Q for any r, and odd order q.
// If P & Q are linearly dependent it might fail, but this will be detected.
//
// Sketch of proof:-
// Consider S=(0,1) - a point of order 3 - as the random contribution in the
// "normal" Tate algorithm i.e. Q <- (Q+S)-(S). Note that the coordinates of P
// are both in Fp. Under these circumstances the contribution of S to the
// numerator and denominator of the cumulative multiplicative total will always
// be in Fp. All such contributions will be "wiped out" by the final raising
// to the power of (p-1). Therefore S can be ommitted altogether, and all
// calculations involving it removed.
//
// Furthermore the calculation will not fail. The numerator and denominator
// will never evaluate to 0, as a consequence of linear independence. The point
// Q cannot be both on the curve and on a line joining two point in the
// calculation of r.P
//
// P & Q(x,y) are both points of order q.
// Note that P is a point on the curve over Fp, Q(x,y) a point on the
// quadratic extension field Fp^2
//

BOOL fast_tate_pairing(ECn& P,ZZn2& Qx,ZZn& Qy,Big& q,ZZn2& res)
{
    int i;
    Big p;
    ECn A;
    ZZn2 ha,had;
```

```
#ifndef SIMPLE
    ECn P2,t[11];
    ZZn2 hc,hcd,z2n,z2d,zn[11],zd[11];
    Big q3;
#endif

    ha=had=1;

#ifdef SIMPLE

// q.P = 2^17*(2^142.P +P) + P

    A=P;    // reset A
    for (i=0;i<142;i++)
    {
        ha*=ha; had*=had;
        g(A,A,Qx,Qy,ha,had,ADD,FALSE);      // 16 ZZn muls + 1 inverse
        if (ha==0 || had==0) return FALSE;
    }                                       // 30 ZZn muls (Projective)
    g(A,P,Qx,Qy,ha,had,ADD,FALSE);          // 11 ZZn muls + 1 inverse
    if (ha==0 || had==0) return FALSE;
    for (i=0;i<17;i++)                      // 34 ZZn muls (Projective)
    {
        ha*=ha; had*=had;
        g(A,A,Qx,Qy,ha,had,ADD,FALSE);      // 16 ZZn muls + 1 inverse
        if (ha==0 || had==0) return FALSE;
    }                                       // 30 ZZn muls (Projective)
    g(A,P,Qx,Qy,ha,had,ADD,FALSE);          // 11 ZZn muls + 1 inverse
    if (ha==0 || had==0) return FALSE;
                                            // 34 ZZn muls (Projective)
#else

    q3=q*3;
    zn[0]=zd[0]=1;
    t[0]=P2=A=P;
    g(P2,P2,Qx,Qy,z2n,z2d,ADD,TRUE);     // P2=P+P
//
// Build NAF windowing table
//
    for (i=1;i<11;i++)
    {                                   // 17 ZZn muls + 1 inverse (Affine)
        g(A,P2,Qx,Qy,hc,hcd,ADD,TRUE);  // 40 ZZn muls (Projective)
        t[i]=A;                         // precalculate t[i] = (2i+1).P
        zn[i]=z2n*zn[i-1]*hc;
        zd[i]=z2d*zd[i-1]*hcd;
    }

    A=P;    // reset A

// Left to right method
    nb=bits(q3);
    for (i=nb-2;i>=1;i-=(nbw+nzs))
    {
        n=naf_window(q,q3,i,&nbw,&nzs);  // standard MIRACL NAF windowing

        for (j=0;j<nbw;j++)
```

```
        {
            ha*=ha; had*=had;
            g(A,A,Qx,Qy,ha,had,ADD,FALSE);      // 16 ZZn muls + 1 inverse
        }                                       // 30 ZZn muls (Projective)
        if (n>0)
        {
            ha*= zn[n/2]; had*=zd[n/2];
            g(A,t[n/2],Qx,Qy,ha,had,ADD,FALSE); // 17 ZZn muls + 1 inverse
        }                                       // 40 ZZn muls (Projective)
        if (n<0)
        {
            n=(-n);
            ha*=zd[n/2]; had*=zn[n/2];
            g(A,t[n/2],Qx,Qy,ha,had,SUB,FALSE); // 17 ZZn muls + 1 inverse
        }                                       // 40 ZZn muls (Projective)
        for (j=0;j<nzs;j++)
        {
            ha*=ha; had*=had;
            g(A,A,Qx,Qy,ha,had,ADD,FALSE);      // 16 ZZn muls + 1 inversion
        }                                       // 30 ZZn muls (Projective)
        if (ha==0 || had==0) return FALSE;
    }

#endif

    if (!A.iszero()) return FALSE;

    res=(ha/had);

    p=get_modulus();          // get p
    res= pow(res,(p+1)/q);    // raise to power of (p^2-1)/q
    res=conj(res)/res;
    if (res.isunity()) return FALSE;
    return TRUE;
}

//
// ecap(.) function
//

BOOL ecap(ECn& P,ECn& Q,Big& order,ZZn2& cube,ZZn2& res)
{
    ZZn2 Qx;
    ZZn  Qy,iy;
    Big xx,yy;

    Q.get(xx,yy);
    Qx=(ZZn)xx*cube;
    Qy=(ZZn)yy;

    iy=(ZZn)1/(Qy+1);
    Qx=-2*Qx*iy;
    Qy=(Qy-3)*iy;         // Q+=(0,1)

    if (fast_tate_pairing(P,Qx,Qy,order,res)) return TRUE;
    return FALSE;
}
```

```
//
// Hash functions
//

Big H1(char *string)
{ // Hash a zero-terminated string to a number < modulus
    Big h,p;
    char s[HASH_LEN];
    int i,j;
    sha sh;

    shs_init(&sh);

    for (i=0;;i++)
    {
        if (string[i]==0) break;
        shs_process(&sh,string[i]);
    }
    shs_hash(&sh,s);
    p=get_modulus();
    h=1; j=0; i=1;
    forever
    {
        h*=256;
        if (j==HASH_LEN)  {h+=i++; j=0;}
        else         h+=s[j++];
        if (h>=p) break;
    }
    h%=p;
    return h;
}

int H2(ZZn2 x,char *s)
{ // Hash an Fp2 to an n-byte string s[.]. Return n
    sha sh;
    Big a,b;
    int m;

    shs_init(&sh);
    x.get(a,b);
    while (a>0)
    {
        m=a%256;
        shs_process(&sh,m);
        a/=256;
    }
    while (b>0)
    {
        m=b%256;
        shs_process(&sh,m);
        b/=256;
    }
    shs_hash(&sh,s);

    return HASH_LEN;
}
```

```
Big H3(char *x1,char *x2)
{
    sha sh;
    char h[HASH_LEN];
    Big a;
    int i;

    shs_init(&sh);
    for (i=0;i<HASH_LEN;i++)
        shs_process(&sh,x1[i]);
    for (i=0;i<HASH_LEN;i++)
        shs_process(&sh,x2[i]);
    shs_hash(&sh,h);
    a=from_binary(HASH_LEN,h);
    return a;
}

void H4(char *x,char *y)
{ // hashes y=h(x)
    int i;
    sha sh;
    shs_init(&sh);
    for (i=0;i<HASH_LEN;i++)
        shs_process(&sh,x[i]);
    shs_hash(&sh,y);
}

//
// Given y, get x=(y^2-1)^(1/3) mod p (from curve equation)
//

Big getx(Big y)
{
    Big p=get_modulus();
    Big t=modmult(y+1,y-1,p);   // avoids overflow
    return pow(t,(2*p-1)/3,p);
}

//
// MapToPoint
//

ECn map_to_point(char *ID)
{
    ECn Q;
    Big x0,y0=H1(ID);

    x0=getx(y0);

    Q.set(x0,y0);

    return Q;
}

void strip(char *name)
{ /* strip off filename extension */
```

```
    int i;
    for (i=0;name[i]!='\0';i++)
    {
        if (name[i]!='.') continue;
        name[i]='\0';
        break;
    }
}

/***************begin ibe-act**********************/

//controle de validade dos dados de entrada
bool controle(int i, char *arg){
  int size,valor;
  size = strlen(arg);
  valor = atoi(arg);
  int temp1,temp2;
  if (i==1){
    temp1 = 2003;
    cout << arg << endl;
    if (size!=4){
      printf("ano invalido!\n");
      return FALSE;
    }
    if (valor<=temp1){
      printf("ano invalido!\n");
      return FALSE;
    }
  }

  if (i==2){
    temp1 = 1;
    temp2 = 12;
    cout << arg << endl;
    if (size!=2){
      printf("mes invalido!\n");
      return FALSE;
    }

      if (valor<temp1 || valor>temp2){
      printf("mes invalido!\n");
      return FALSE;
    }
  }

  if (i==3){
    temp1=1;
    temp2=31;
    cout << arg << endl;
    if (size!=2){
      printf("dia invalido!\n");
      return FALSE;
    }
    if (valor<temp1 || valor>temp2){
      printf("dia invalido!\n");
      return FALSE;
    }
```

```
  }

  if (i==4){
    temp1=0;
    temp2=23;
    cout << arg << endl;
    if (size!=2){
      printf("hora invalida!\n");
      return FALSE;
    }
    if (valor<temp1 || valor>temp2){
      printf("hora invalida!\n");
      return FALSE;
    }
  }

  if (i==5){
    temp1=0;
    temp2=59;
    cout << arg << endl;
    if (size!=2){
      printf("minuto invalido!\n");
      return FALSE;
    }
    if (valor<temp1 || valor>temp2){
      printf("minuto invalido!\n");
      return FALSE;
    }
  }

  return TRUE;
}


//cifrando primeira chave aleatoria com RSA
int rsa_encrypt(RSA *key_, unsigned char *plain_, int len_, unsigned char
**cipher_)
{
  int clen=0;

  srand(time(NULL));
  if((clen = RSA_public_encrypt(len_, plain_, *cipher_, key_, RSA_PKCS1_PADDING))
== -1) {
    fprintf(stderr, "%s\n", ERR_error_string(ERR_get_error(), NULL));
    exit(EXIT_FAILURE);
  } else
    return clen;
}


//lendo chave publica formato pem
RSA* readpemkeys(char *PUBFILE)
{
  FILE *fp_;
  RSA *key_=NULL;

    if((fp_ = fopen(PUBFILE,"r")) == NULL) {
```

```
      fprintf(stderr,"Error: Public Key file doesn't exists.\n");
      exit(EXIT_FAILURE);
    }
    if((key_ = PEM_read_RSAPublicKey(fp_,NULL,NULL,NULL)) == NULL) {
      fprintf(stderr,"Error: problems while reading Public Key.\n");
      exit(EXIT_FAILURE);
    }
    fclose(fp_);
    return key_;
}

//apagando archivos temporarios...
void apagar(char *_ofname){
    strip(_ofname);
    strcat(_ofname,".ibe");
    unlink(_ofname);
    strip(_ofname);
    strcat(_ofname,".id");
    unlink(_ofname);
    strip(_ofname);
    strcat(_ofname,".key");
    unlink(_ofname);
    strip(_ofname);
    strcat(_ofname,".out");
    unlink(_ofname);
    strip(_ofname);
    strcat(_ofname,".kfp");
    unlink(_ofname);
    strip(_ofname);
    strcat(_ofname,".kfc");
    unlink(_ofname);
    strip(_ofname);
    strcat(_ofname,".mtd");
    unlink(_ofname);
    strip(_ofname);
    strcat(_ofname,".aes");
    unlink(_ofname);
}


/*********************end ibe-atc*********************/

int main()
{
    miracl *mip=mirsys(18,0);   // thread-safe ready. (36,0) for 1024 bit p
    ifstream common("common.ibe");
    ifstream plaintext;
    ofstream key_file,ciphertext;
    ECn U,P,Ppub,Qid,infinity;
    ZZn2 gid,cube,w;
    char key[HASH_LEN],pad[HASH_LEN],rho[HASH_LEN],V[HASH_LEN],W[HASH_LEN];
    char ifname[100],ofname[100],ch,iv[16];
    Big p,q,r,x,y,cof;
    int i,bits;
    long seed;
    aes a;
```

```
/******************begin ibe-act*********************/
  int ch_,size_=0,len_=0,ks_=0;
  ifstream plaintext_,cabecalho,corpo,a_chave;
  ofstream key_file_,ciphertext_,meta_doc_um;
  char
_ofname[100],ofname_[100],id[20],ano[10],mes[10],dia[10],hora[10],minuto[10],dest_name[100],sair;
  RSA *key_=NULL;
  FILE *fpin_=NULL, *fpout_=NULL;
  aes b;
  bool ok;
  unsigned char *cipher_=NULL,*plain_=NULL;

  time_t rawtime;
  time(&rawtime);

  system("clear"); //limpa a tela
  printf("Programa de criptografia temporal de documentos.\n");
  printf("Antes de continuar eh altamente recomendado que faca um backup do
arquivo a cifrar.\n");
  printf("A data e hora atuais: %s\n", ctime(&rawtime));
  printf("Digite 'S' + <enter> para sair, ou qualquer outra tecla + <enter> para
continuar...\n");

  cin >> sair;

  if ((sair == 's')||(sair == 'S')) return 0;

    cout << "Digite um numero randomico de semente (sempre diferente) de ateh 9
digitos = ";
    cin >> seed;
    irand(seed);

    cout << "\nCarregar data futura para a liberacao da chave...\n" << endl;

    do{
    ok=FALSE;
    cout << "Digite o ANO da liberacao da chave (ex: 2004) = ";
    cin >> ano;
    ok = controle(1,ano);
    }while(ok==FALSE);
    strcpy(id,ano);

    do{
    ok=FALSE;
    cout << "Digite o MES da liberacao da chave (01 - 12) = ";
    cin >> mes;
    ok = controle(2,mes);
    }while(ok==FALSE);
    strcat(id,mes);

    do{
    ok=FALSE;
    cout << "Digite o DIA da liberacao da chave (01 - 31) = ";
    cin >> dia;
    ok = controle(3,dia);
    }while(ok==FALSE);
    strcat(id,dia);
```

```
    do{
    ok=FALSE;
    cout << "Digite a HORA da liberacao da chave (00 - 23) = ";
    cin >> hora;
    ok = controle(4,hora);
    }while(ok==FALSE);
    strcat(id,hora);

    do{
    ok=FALSE;
    cout << "Digite o MINUTO da liberacao da chave (00 - 59) = ";
    cin >> minuto;
    ok = controle(5,minuto);
    }while(ok==FALSE);
    strcat(id,minuto);

  /****************end ibe-act******************/


// ENCRYPT

    common >> bits;
    mip->IOBASE=16;
    common >> p >> q;

    cof=(p+1)/q;

    common >> x >> y;
    EBrick B(x,y,(Big)0,(Big)1,p,QBITS);   // precomputation based on fixed P

#ifdef AFFINE
    ecurve(0,1,p,MR_AFFINE);
#endif
#ifdef PROJECTIVE
    ecurve(0,1,p,MR_PROJECTIVE);
#endif

    P.set(x,y);

    common >> x >> y;
    Ppub.set(x,y);

    common >> x >> y;
    cube.set(x,y);

    mip->IOBASE=10;
    Qid=map_to_point(id);

// This can be done before we know the message to encrypt

    if (!ecap(Ppub,Qid,q,cube,gid))      // ** swap argument order
    {                                    // Qid must be second
        cout << "Bad Parameters" << endl;
        exit(0);
    }
```

```
/*************************Begin ibe-act*********************************/

    cout << "\nDigite o nome do arquivo a cifrar: " ;
    cin >> ifname;

    cout << "\nDigite o nome do arquivo da CHAVE PUBLICA do DESTINATARIO do
documento: ";
    cin >> dest_name;

    char ofname___[100];
    ofstream id_file;

    strcpy(ofname___,ifname);
    strip(ofname___);
    strcat(ofname___,".id");
    id_file.open(ofname___);
    id_file << id << endl;
    id_file.close();



    //gerando chave para cifrar documento com AES
      for (i=0;i<HASH_LEN;i++) key[i]=(char)brand();

      strip(ofname___);
      strcat(ofname___,".kfp");

      //guardando chave aleatoria AES gerada acima
      key_file_.open(ofname___);
      key_file_ << key << endl;
      key_file_.close();



   //cifrando com RSA
      //lendo chave publica RSA para cifrar chave
       key_ = readpemkeys(dest_name);
       if(!(fpin_ = fopen(ofname___, "r"))) {
         fprintf(stderr, "Error: Cannot locate input file.\n");
         exit(EXIT_FAILURE);
       }

     strip(ofname___);
     strcat(ofname___,".kfc");


      //guardando chave aleatoria AES cifrada
      fpout_ = fopen(ofname___, "w");

      ks_ = RSA_size(key_);
      plain_ = (unsigned char *)malloc(ks_ * sizeof(unsigned char));
      cipher_ = (unsigned char*)malloc(ks_ * sizeof(unsigned char));

      while(!feof(fpin_)) {
        memset(plain_,'\0',ks_ + 1);
        memset(cipher_, '\0', ks_ + 1);
        len_ = fread(plain_, 1, ks_ - 11, fpin_);
```

```
        size_ = rsa_encrypt(key_, plain_, len_, &cipher_);
        fwrite(cipher_, 1, size_, fpout_);
      }

      fclose(fpout_);
      fclose(fpin_);
      free(cipher_);
      free(plain_);
      RSA_free(key_);


      //cifrando... AES
     for (i=0;i<16;i++) iv[i]=i; // set CFB IV
     aes_init(&a,MR_CFB1,16,key,iv);

     strcpy(ofname,ifname);
     strip(ofname);
     strcat(ofname,".aes");
     plaintext_.open(ifname,ios::in);

   if (!plaintext_)
   {
     cout << "Impossible abrir arquivo: " << ifname << endl;
       return 0;
   }
   cout << "Cifrando arquivo primeira etapa... " << ifname << endl;
   ciphertext_.open(ofname,ios::binary|ios::out);

// now encrypt the plaintext file (texto plano -> texto.aes)

   forever
   { // encrypt input ..
       plaintext_.get(ch);
       if (plaintext_.eof()) break;
       aes_encrypt(&a,&ch);
       ciphertext_ << ch;
   }
   plaintext_.close();
   ciphertext_.close();
   aes_end(&a);


   //criando o primeiro meta-docmento ".mtd":
     //chave publica (em texto plano) + chave aletoria (cifrada RSA) + corpo
(.aes)
   strcpy(_ofname,ofname);
   strip(_ofname);
   strcat(_ofname,".mtd");
   meta_doc_um.open(_ofname,ios::binary|ios::out);

   cabecalho.open(dest_name,ios::in);

   strip(_ofname);
   strcat(_ofname,".kfc");
   a_chave.open(_ofname,ios::in);

   strip(_ofname);
```

```
    strcat(_ofname,".aes");
    corpo.open(_ofname,ios::in);

    //anexando chave publica do destinatario (dest_name)
    forever
    {
        cabecalho.get(ch);
        if (cabecalho.eof()) break;
        meta_doc_um << ch;
    }
    meta_doc_um << "#1" << endl;

    //anexando primeira chave de sessao (aleatoria e cifrada via RSA - .kfc)
    forever
    {
        a_chave.get(ch);
        if (a_chave.eof()) break;
        meta_doc_um << ch;
    }
    meta_doc_um << "#2" << endl;

    //anexando corpo (documento original, cifrado com AES) - .aes
    forever
    {
        corpo.get(ch);
        if (corpo.eof()) break;
        meta_doc_um << ch;
    }
      meta_doc_um.close();
      cabecalho.close();
      corpo.close();
      a_chave.close();

    printf("Primeira etapa concluida com sucesso....\n");
    printf("Precione <enter> para continuar....\n");
    getchar();
    getchar();



/*************************End ibe-act**********************************/



//
// prepare to encrypt file with random session key
//
    //gerando segunda chave de sessao...
      for (i=0;i<HASH_LEN;i++) key[i]=(char)brand();
      for (i=0;i<16;i++) iv[i]=i; // set CFB IV
      aes_init(&a,MR_CFB1,16,key,iv);



   /* set up input file */
    strip(_ofname);
    strcat(_ofname,".mtd");
```

```
    plaintext.open(_ofname);
    if (!plaintext)
    {
        cout << "Unable to open file ofname:" << _ofname << "\n";
        return 0;
    }
    strip(_ofname);
    strcat(_ofname,".ibe");

    cout << "Agora cifrando com IBE..." << _ofname << endl;

    ciphertext.open(_ofname,ios::binary|ios::out);

// now encrypt the plaintext file

    forever
    {
        plaintext.get(ch);
        if (plaintext.eof()) break;
        aes_encrypt(&a,&ch);
        ciphertext << ch;
    }
    ciphertext.close();
    plaintext.close();
    aes_end(&a);

    cout << "IBE... ok!" << endl;

//
// Now IBE encrypt the session key
//

    for (i=0;i<HASH_LEN;i++) rho[i]=(char)brand();

    r=H3(rho,key);

    B.mul(r,x,y);       // U=r*P
    U.set(x,y);

    w=pow(gid,r);
    H2(w,pad);

    for (i=0;i<HASH_LEN;i++)
    {
        V[i]=rho[i]^pad[i];
        pad[i]=0;
    }
    H4(rho,rho);
    for (i=0;i<HASH_LEN;i++)
    {
        W[i]=key[i]^rho[i];
        rho[i]=0;
    }

    strip(ofname);
    strcat(ofname,".key");
```

```
    mip->IOBASE=16;
    key_file.open(ofname);
    U.get(x,y);
    key_file << y << endl;
    x=from_binary(20,V);      // output bit strings in handy Big format
    key_file << x << endl;
    x=from_binary(20,W);
    key_file << x << endl;

    /**********************begin ibe-act**************************/
    key_file.close();

      //Gravando chave....

    //Guardando meta-documento .act (final)
    strip(_ofname);
    strcat(_ofname,".act");
    cout << _ofname << endl;
    meta_doc_um.open(_ofname,ios::binary|ios::out);

    //lendo id (data/hora) para guardar no meta-documento
    strip(_ofname);
    strcat(_ofname,".id");
    cabecalho.open(_ofname,ios::in);
    if (!cabecalho)
    {
        cout << "Unable to open file .id:" << _ofname << "\n";
        return 0;
    }

    //lendo segunda chave de sessao cifrada e guardada pelo IBE para anexar ao
meta-documento
    a_chave.open(ofname,ios::in);
    if (!a_chave)
    {
        cout << "Unable to open file a_chave:" << ofname ;
        return 0;
    }

    //lendo corpo (primeiro meta-documento) a ser anexado ao .act
    strip(_ofname);
    strcat(_ofname,".ibe");
    corpo.open(_ofname,ios::in);
     if (!corpo)
    {
        cout << "Unable to open file corpo:" << _ofname << "\n";
        return 0;
    }

     //anexando cabecalho (.id)
    forever
    {
        cabecalho.get(ch);
        if (cabecalho.eof()) break;
      meta_doc_um << ch;
    }
    meta_doc_um << "#1" << endl;
```

```
    //anexando chave (.key)
    forever
    {
        a_chave.get(ch);
        if (a_chave.eof()) break;
      meta_doc_um << ch;
    }
    meta_doc_um << "#2" << endl;

    //anexando corpo (.ibe)
    forever
    {
        corpo.get(ch);
        if (corpo.eof()) break;
        meta_doc_um  << ch;
    }
      meta_doc_um.close();
      cabecalho.close();
      corpo.close();
      a_chave.close();

    //apagando arquivos temporarios...
    apagar(_ofname);


    printf("Cifragem finalizada!\n");
    printf("O documento soh podera ser aberto as %s:%s do %s/%s/
%s!\n",hora,minuto,dia,mes,ano);
    printf("Pressione uma tecla para encerrar...\n");
    getchar();

    /*****************end ibe-act************************/

    return 0;
}
```

```
//ibe_dec_act.cpp

#include <iostream>
#include <fstream>
#include <cstring>
#include "elliptic.h"
#include "monty.h"
#include "ebrick.h"
#include "zzn2.h"

//Begin ibe-act
#include <time.h> //controle de tempo
#include <openssl/err.h> //biblioteca de erro openssl
#include <openssl/rsa.h> //biblioteca RSA
#include <openssl/rand.h> //Gerador de numeros aleatorios
#include <openssl/pem.h> //biblioteca para chaves padrao PEM
#include <openssl/objects.h> //objetos openssl
#include <cstdlib> //chamadas a sistema
```

```
#include <unistd.h> //manipulacao de arquivos externos (unlink...)
//End ibe-act

using namespace std;

#define PBITS 512
#define QBITS 160

// Using SHA-1 as basic hash algorithm

#define HASH_LEN 20

//
// Define one or the other of these
//
// Which is faster depends on the I/M ratio - See imratio.c
// Roughly if I/M ratio > 16 use PROJECTIVE, otherwise use AFFINE
//

// #define AFFINE
#define PROJECTIVE

// define this if group order q is "simple" = 2^159+2^17+1
#define SIMPLE

//
// Tate Pairing Code
//
// Extract ECn point in internal ZZn format
//

void extract(ECn& A,ZZn& x,ZZn& y)
{
    x=(A.get_point())->X;
    y=(A.get_point())->Y;
}

void extract(ECn& A,ZZn& x,ZZn& y,ZZn& z)
{
    big t;
    x=(A.get_point())->X;
    y=(A.get_point())->Y;
    t=(A.get_point())->Z;
    if (A.get_status()!=MR_EPOINT_GENERAL) z=1;
    else                                   z=t;
}

//
// Line from A to destination C. Let A=(x,y)
// Line Y-slope.X-c=0, through A, so intercept c=y-slope.x
// Line Y-slope.X-y+slope.x = (Y-y)-slope.(X-x) = 0
// Now evaluate at Q -> return (Qy-y)-slope.(Qx-x)
//

ZZn2 line(ECn& A,ECn& C,ZZn& slope,ZZn2& Qx,ZZn& Qy)
{
    ZZn2 n=Qx;
```

```
    ZZn x,y,z,t,w=Qy;
#ifdef AFFINE
    extract(A,x,y);
    n-=x; n*=slope;            // 2 ZZn muls
    w-=y; n-=w;
#endif
#ifdef PROJECTIVE
    extract(A,x,y,z);
    x*=z; t=z; z*=z; z*=t;
    n*=z; n-=x;                // 9 ZZn muls
    w*=z; w-=y;
    extract(C,x,y,z);
    w*=z; n*=slope; n-=w;
#endif
    return n;
}

//
// Vertical line through point A
//

ZZn2 vertical(ECn& A,ZZn2& Qx)
{
    ZZn2 n=Qx;
    ZZn x,y,z;
#ifdef AFFINE
    extract(A,x,y);
    n-=x;
#endif
#ifdef PROJECTIVE
    extract(A,x,y,z);
    z*=z;
    n*=z; n-=x;                // 3 ZZn muls
#endif
    return n;
}

//
// Add A=A+B  (or A=A+A) or
// Sub A=A-B
// Bump up num and denom
//
// AFFINE doubling     - 12 ZZn muls, plus 1 inversion
// AFFINE adding       - 11 ZZn muls, plus 1 inversion
//
// PROJECTIVE doubling - 26 ZZn muls
// PROJECTIVE adding   - 34 ZZn muls
//

#define ADD 1
#define SUB 2

void g(ECn& A,ECn& B,ZZn2& Qx,ZZn& Qy,ZZn2& num,ZZn2& denom,int as,BOOL first)
{
    ZZn  lam,mQy;
    ZZn2 d,u;
    big ptr;
```

```
    ECn P=A;

    if (as==ADD)
    { // Evaluate line from A, and then evaluate vertical through destination
        ptr=A.add(B);
        if (ptr==NULL)    { num=0; return; }
        else lam=ptr;

        if (A.iszero())   { u=vertical(P,Qx); d=1; }
        else
        {
            u=line(P,A,lam,Qx,Qy);
            d=vertical(A,Qx);
        }
    }
    else // as==SUB
    { // Evaluate Vertical at A, and then line from A to destination
      // (Note swap num and denom, Qy=-Qy, process lines "backwards")
        u=vertical(A,Qx);
        ptr=A.sub(B);
        if (ptr==NULL)    { num=0; return; }
        else lam=ptr;

        if (A.iszero())   { d=u;  }
        else
        {
            mQy=-Qy;
            d=line(P,A,lam,Qx,mQy);
        }
    }

    if (first) {num= u; denom= d; }
    else       {num*=u; denom*=d; }   // 6 ZZn muls
}

//
// Tate Pairing - note ha -> numerator, had -> denominator
// Trying to minimize number of modular inversions
//
// Special optimized and deterministic version of Tate Pairing algorithm
// Requires that the point (0,1) is on the curve AND
// P and Q(x,y) linearly independent, that is P!=r.Q for any r, and odd order q.
// If P & Q are linearly dependent it might fail, but this will be detected.
//
// Sketch of proof:-
// Consider S=(0,1) - a point of order 3 - as the random contribution in the
// "normal" Tate algorithm i.e. Q <- (Q+S)-(S). Note that the coordinates of P
// are both in Fp. Under these circumstances the contribution of S to the
// numerator and denominator of the cumulative multiplicative total will always
// be in Fp. All such contributions will be "wiped out" by the final raising
// to the power of (p-1). Therefore S can be ommitted altogether, and all
// calculations involving it removed.
//
// Furthermore the calculation will not fail. The numerator and denominator
// will never evaluate to 0, as a consequence of linear independence. The point
// Q cannot be both on the curve and on a line joining two point in the
// calculation of r.P
```

```
//
// P & Q(x,y) are both points of order q.
// Note that P is a point on the curve over Fp, Q(x,y) a point on the
// quadratic extension field Fp^2
//

BOOL fast_tate_pairing(ECn& P,ZZn2& Qx,ZZn& Qy,Big& q,ZZn2& res)
{
    int i;
    Big p;
    ECn A;
    ZZn2 ha,had;

#ifndef SIMPLE
    Big q3;
    ECn P2,t[11];
    ZZn2 hc,hcd,z2n,z2d,zn[11],zd[11];
#endif

    ha=had=1;

#ifdef SIMPLE

// q.P = 2^17*(2^142.P +P) + P

    A=P;    // reset A
    for (i=0;i<142;i++)
    {
        ha*=ha; had*=had;
        g(A,A,Qx,Qy,ha,had,ADD,FALSE);      // 16 ZZn muls + 1 inverse
        if (ha==0 || had==0) return FALSE;
    }                                       // 30 ZZn muls (Projective)
    g(A,P,Qx,Qy,ha,had,ADD,FALSE);          // 11 ZZn muls + 1 inverse
    if (ha==0 || had==0) return FALSE;
                                            // 34 ZZn muls (Projective)
    for (i=0;i<17;i++)
    {
        ha*=ha; had*=had;
        g(A,A,Qx,Qy,ha,had,ADD,FALSE);      // 16 ZZn muls + 1 inverse
        if (ha==0 || had==0) return FALSE;
    }                                       // 30 ZZn muls (Projective)
    g(A,P,Qx,Qy,ha,had,ADD,FALSE);          // 11 ZZn muls + 1 inverse
    if (ha==0 || had==0) return FALSE;
                                            // 34 ZZn muls (Projective)
#else

    q3=q*3;
    zn[0]=zd[0]=1;
    t[0]=P2=A=P;
    g(P2,P2,Qx,Qy,z2n,z2d,ADD,TRUE);     // P2=P+P
//
// Build NAF windowing table
//
    for (i=1;i<11;i++)
    {                                   // 17 ZZn muls + 1 inverse (Affine)
        g(A,P2,Qx,Qy,hc,hcd,ADD,TRUE);  // 40 ZZn muls (Projective)
        t[i]=A;                         // precalculate t[i] = (2i+1).P
```

```
        zn[i]=z2n*zn[i-1]*hc;
        zd[i]=z2d*zd[i-1]*hcd;
    }
    A=P;    // reset A

/* Left to right method  */
    nb=bits(q3);
    for (i=nb-2;i>=1;i-=(nbw+nzs))
    {
        n=naf_window(q,q3,i,&nbw,&nzs);  // standard MIRACL NAF windowing
        for (j=0;j<nbw;j++)
        {
            ha*=ha; had*=had;
            g(A,A,Qx,Qy,ha,had,ADD,FALSE);      // 16 ZZn muls + 1 inverse
        }                                       // 30 ZZn muls (Projective)
        if (n>0)
        {
            ha*= zn[n/2]; had*=zd[n/2];
            g(A,t[n/2],Qx,Qy,ha,had,ADD,FALSE); // 17 ZZn muls + 1 inverse
        }                                       // 40 ZZn muls (Projective)
        if (n<0)
        {
            n=(-n);
            ha*=zd[n/2]; had*=zn[n/2];
            g(A,t[n/2],Qx,Qy,ha,had,SUB,FALSE); // 17 ZZn muls + 1 inverse
        }                                       // 40 ZZn muls (Projective)
        for (j=0;j<nzs;j++)
        {
            ha*=ha; had*=had;
            g(A,A,Qx,Qy,ha,had,ADD,FALSE);      // 16 ZZn muls + 1 inversion
        }                                       // 30 ZZn muls (Projective)
        if (ha==0 || had==0) return FALSE;
    }

#endif

    if (!A.iszero()) return FALSE;

    res=(ha/had);

    p=get_modulus();          // get p
    res= pow(res,(p+1)/q);    // raise to power of (p^2-1)/q
    res=conj(res)/res;
    if (res.isunity()) return FALSE;

    return TRUE;
}

//
// ecap(.) function
//

BOOL ecap(ECn& P,ECn& Q,Big& order,ZZn2& cube,ZZn2& res)
{
    ZZn2 Qx;
    ZZn  Qy,iy;
    Big xx,yy;
```

```
    BOOL Ok;

    Q.get(xx,yy);
    Qx=(ZZn)xx*cube;
    Qy=(ZZn)yy;

    iy=(ZZn)1/(Qy+1);
    Qx=-2*Qx*iy;
    Qy=(Qy-3)*iy;        // Q+=(0,1)

    Ok=fast_tate_pairing(P,Qx,Qy,order,res);

    if (Ok) return TRUE;
    return FALSE;
}

//
// Given y, get x=(y^2-1)^(1/3) mod p (from curve equation)
//

Big getx(Big y)
{
    Big p=get_modulus();
    Big t=modmult(y+1,y-1,p);   // avoids overflow
    return pow(t,(2*p-1)/3,p);
}

//
// Hash functions
//

int H2(ZZn2 x,char *s)
{ // Hash an Fp2 to an n-byte string s[.]. Return n
    sha sh;
    Big a,b;
    int m;

    shs_init(&sh);
    x.get(a,b);
    while (a>0)
    {
        m=a%256;
        shs_process(&sh,m);
        a/=256;
    }
    while (b>0)
    {
        m=b%256;
        shs_process(&sh,m);
        b/=256;
    }
    shs_hash(&sh,s);

    return HASH_LEN;
}

Big H3(char *x1,char *x2)
```

```
{
    sha sh;
    char h[HASH_LEN];
    Big a;
    int i;

    shs_init(&sh);
    for (i=0;i<HASH_LEN;i++)
        shs_process(&sh,x1[i]);
    for (i=0;i<HASH_LEN;i++)
        shs_process(&sh,x2[i]);
    shs_hash(&sh,h);
    a=from_binary(HASH_LEN,h);
    return a;
}

void H4(char *x,char *y)
{ // hashes y=h(x)
    int i;
    sha sh;
    shs_init(&sh);
    for (i=0;i<HASH_LEN;i++)
        shs_process(&sh,x[i]);
    shs_hash(&sh,y);
}

/********************begin ibe-act******************/
//funcao para cortar a extencao do nome do aquivo, tirada de ibe_enc.cpp
void strip(char *name)
{ /* strip extension off filename */
    int i;
    for (i=0;name[i]!='\0';i++)
    {
        if (name[i]!='.') continue;
        name[i]='\0';
        break;
    }
}

//decifrando chave aleatoria com RSA
int rsa_decrypt(RSA *key_, unsigned char *cipher_, int len_, unsigned char
**plain_)
{
  int plen=0;

  if((plen = RSA_private_decrypt(len_, cipher_, *plain_, key_,
RSA_PKCS1_PADDING)) == -1) {
    fprintf(stderr, "%s\n", ERR_error_string(ERR_get_error(), NULL));
    exit(EXIT_FAILURE);
  } else
    return plen;
}

//lendo chave privada
RSA* readpemkeys(char *SECFILE)
{
  FILE *fp_;
```

```
  RSA *key_=NULL;

    if((fp_ = fopen(SECFILE,"r")) == NULL) {
      fprintf(stderr,"Error: Private Key file doesn't exists.\n");
      exit(EXIT_FAILURE);
    }

    if((key_ = PEM_read_RSAPrivateKey(fp_,NULL,NULL,NULL)) == NULL) {
      fprintf(stderr,"Error: problmes while reading Private Key.\n");
      exit(EXIT_FAILURE);
    }
    fclose(fp_);

    if(RSA_check_key(key_) == -1) {
      fprintf(stderr,"Error: Problems while reading RSA Private Key in '%s'
file.\n",SECFILE);
      exit(EXIT_FAILURE);
    } else if(RSA_check_key(key_) == 0) {
      fprintf(stderr,"Error: Bad RSA Private Key readed in '%s'
file.\n",SECFILE);
      exit(EXIT_FAILURE);
    }
    else
      return key_;
}

//apagar arquivos temporarios...
void apagar(char *ofname){
    strip(ofname);
    strcat(ofname,".ibe");
    unlink(ofname);
    strip(ofname);
    strcat(ofname,".id");
    unlink(ofname);
    strip(ofname);
    strcat(ofname,".key");
    unlink(ofname);
    strip(ofname);
    strcat(ofname,".out");
    unlink(ofname);
    strip(ofname);
    strcat(ofname,".kfo");
    unlink(ofname);
    strip(ofname);
    strcat(ofname,".kfc");
    unlink(ofname);
    strip(ofname);
    strcat(ofname,".pem");
    unlink(ofname);
    strip(ofname);
    strcat(ofname,".aes");
    unlink(ofname);
}

int main()
{
    miracl *mip=mirsys(18,0);    // thread-safe ready.  (36,0) for 1024 bit p
```

```
    ifstream common("common.ibe");
    ifstream private_key;//("private.ibe");
    ifstream key_file,ciphertext;
    ofstream ciphertext__,ciphertext_;
    ECn U,P,rP,Ppub,Qid,Did,infinity;
    char key[HASH_LEN],pad[HASH_LEN],rho[HASH_LEN],V[HASH_LEN],W[HASH_LEN],sair;
    char ifname[100],ch,iv[16];
    ZZn2 gid,cube,w;
    Big s,p,q,r,cof,x,y;
    int i,Bits;
    aes a;

  /*******begin ibe-act*********/
  int ch_,size_=0,len_=0,ks_=0;
  char ofname[100],dest_name[100],ofname___[100];
  RSA *key_=NULL;
  FILE *fpin_=NULL, *fpout_=NULL;
  unsigned char *cipher_=NULL,*plain_=NULL;
  time_t rawtime;
  time(&rawtime);
  ifstream meta_doc_um,key_file__,plaintext_;
  ofstream cabecalho,a_chave,corpo;
  /********end ibe-act**********/


// DECRYPT

    common >> Bits;
    mip->IOBASE=16;
    common >> p >> q;

    cof=(p+1)/q;

    common >> x >> y;
    EBrick B(x,y,(Big)0,(Big)1,p,QBITS);   // precomputation based on fixed P

#ifdef AFFINE
    ecurve(0,1,p,MR_AFFINE);
#endif
#ifdef PROJECTIVE
    ecurve(0,1,p,MR_PROJECTIVE);
#endif

    P.set(x,y);

    common >> x >> y;
    Ppub.set(x,y);

    common >> x >> y;
    cube.set(x,y);


 /***************begin ibe-act***************************/

  system("clear");//limpa tela
  printf("IBE - DECIFRAGEM.\n");
  printf("Programa de criptografia temporal de documentos.\n");
```

```
  printf("Antes de continuar eh altamente recomendado que faca um backup do
arquivo a ser manipulado.\n");
  printf("A data e hora atuais: %s\n", ctime(&rawtime));
  printf("Digite 'S' + <enter> para sair, ou qualquer outra tecla + <enter> para
continuar...\n");

  cin >> sair;

      if ((sair == 's')||(sair == 'S')) return 0;

    cout << "Digite o nome do arquivo a ser decifrado = ";
    cin >> ifname;

    cout << "\nDigite o nome do arquivo da CHAVE PRIVADA do DESTINATARIO do
documento: ";
    cin >> dest_name;

    //preparando meta-documento .act
    meta_doc_um.open(ifname,ios::in);

    strip(ifname);
    strcat(ifname,".id");
    cabecalho.open(ifname,ios::binary|ios::out);

    strip(ifname);
    strcat(ifname,".key");
    a_chave.open(ifname,ios::binary|ios::out);

    strip(ifname);
    strcat(ifname,".ibe");
    corpo.open(ifname,ios::binary|ios::out);

    //estirpando ID (data/hora) do .act
    forever
    {
        meta_doc_um.get(ch);
        if (meta_doc_um.eof()) break;
      if (ch=='#'){
        meta_doc_um.get(ch);
        if (ch=='1'){
             meta_doc_um.get(ch);
          break;
        }else{
          cabecalho << "#";
          }
      }
        cabecalho << ch;
    }

    //estirpando chave cifrada por IBE
    forever
    {
        meta_doc_um.get(ch);
        if (meta_doc_um.eof()) break;
      if (ch=='#'){
        meta_doc_um.get(ch);
        if (ch=='2'){
```

```
            meta_doc_um.get(ch);
          break;
        }else{
          a_chave << "#";
          }
      }
        a_chave << ch;
    }

    //recuperando corpo do .act
    forever
    {
        meta_doc_um.get(ch);
        if (meta_doc_um.eof()) break;
        corpo << ch;
    }
      meta_doc_um.close();
      cabecalho.close();
      corpo.close();
      a_chave.close();


      //preparando chave privada IBE
    strip(ifname);
    strcat(ifname,".pri");
    private_key.open(ifname,ios::in);

  /*****************end ibe-act**************************/

// get private key

    private_key >> y;
    x=getx(y);              // get unique x from y

    Did.set(x,y);

// figure out where input is coming from

    strip(ifname);          // open key file
    strcat(ifname,".key");
    cout << ifname;
    key_file.open(ifname,ios::in);
    if (!key_file)
    {
        cout << "Key file " << ifname << " not found\n";
        return 0;
    }

    strip(ifname);          // open ciphertext
    strcat(ifname,".ibe");
    ciphertext.open(ifname,ios::binary|ios::in);
    if (!ciphertext)
    {
        cout << "Unable to open file " << ifname << "\n";
        return 0;
    }
```

```
    //criando arquivo .out que contera o primeiro meta-documento gerado por
ibe_enc_act
    strip(ifname);
    strcat(ifname,".out");
    ciphertext_.open(ifname,ios::binary|ios::out);
    if (!ciphertext_)
    {
        cout << "Unable to open file " << ifname << "\n";
        return 0;
    }

    mip->IOBASE=16;

// get file info

    key_file >> y;
    x=getx(y);      // get unique x from y

    U.set(x,y);

    key_file >> x;

    to_binary(x,HASH_LEN,V,TRUE);
    key_file >> x;
    to_binary(x,HASH_LEN,W,TRUE);

    mip->IOBASE=10;

// DECRYPT - extract session key


// No need for this as ecap calculates q*U implicitly
//
//    if (q*U!=infinity)
//    {
//        cout << "Ciphertext rejected" << endl;
//        exit(0);
//    }

    //    cout << "Decrypting message" << endl;

    if (!ecap(U,Did,q,cube,w))
    {
        cout << "Ciphertext rejected" << endl;
        exit(0);
    }

    H2(w,pad);
    for (i=0;i<HASH_LEN;i++) rho[i]=V[i]^pad[i];

    H4(rho,pad);
    for (i=0;i<HASH_LEN;i++) key[i]=W[i]^pad[i];

    r=H3(rho,key);

    B.mul(r,x,y);        // r*P
    rP.set(x,y);
```

```
    if (U!=rP)
    {
        cout << "Ciphertext rejected" << endl;
        exit(0);
    }

//
// Use session key to decrypt file
//
    for (i=0;i<16;i++) iv[i]=i;  // Set CFB IV
    aes_init(&a,MR_CFB1,16,key,iv);

//begin IBE-ACT
    cout << "Decifrando IBE..." << endl;
//end IBE-ACT

    forever
    { // decrypt input ..
        ciphertext.get(ch);
        if (ciphertext.eof()) break;
        aes_decrypt(&a,&ch);
        ciphertext_ << ch;
    }
    ciphertext_.close();
    aes_end(&a);




    /************begin ibe-act****************/
       cout << "IBE finalizado..." << endl;

      //preparando arquivos para gerar chave publica do destinatario, chave
aleatoria cifrada com RSA e documento cifrado
    //preparando primeiro meta-documento..
    strip(ifname);
    strcat(ifname,".out");
    meta_doc_um.open(ifname,ios::in);

    strip(ifname);
    strcat(ifname,".aes");
    corpo.open(ifname,ios::binary|ios::out);

    strip(ifname);
    strcat(ifname,".kfc");
    a_chave.open(ifname,ios::binary|ios::out);

    strip(ifname);
    strcat(ifname,".pem");
    cabecalho.open(ifname,ios::binary|ios::out);

    //recuperando chave publica do destinatario .pem
    forever
    {
        meta_doc_um.get(ch);
```

```
        if (meta_doc_um.eof()) break;
      if (ch=='#'){
        meta_doc_um.get(ch);
        if (ch=='1'){
             meta_doc_um.get(ch);
          break;
        }else{
          cabecalho << "#";
          }
      }
        cabecalho << ch;
    }

    //recuperando chave aleatoria cifrada com RSA
    forever
    {
        meta_doc_um.get(ch);
        if (meta_doc_um.eof()) break;
      if (ch=='#'){
        meta_doc_um.get(ch);
        if (ch=='2'){
             meta_doc_um.get(ch);
          break;
        }else{
          a_chave << "#";
          }
      }
        a_chave << ch;
    }

    //recuperando primeiro meta-documento (.mtd do ibe_enc_act)
    forever
    {
        meta_doc_um.get(ch);
        if (meta_doc_um.eof()) break;
      //cout << ch;
        corpo << ch;
    }
      meta_doc_um.close();
      cabecalho.close();
      corpo.close();
      a_chave.close();


       //decifrando chave aleatoria com RSA
       key_ = readpemkeys(dest_name);

      strcpy(ofname___,ifname);
      strip(ofname___);
      strcat(ofname___,".kfc");

       if(!(fpin_ = fopen(ofname___, "r"))) {
         fprintf(stderr, "Erro: nao eh possivel abrir chave aleatoria cifrada
.kfc\n");
         exit(EXIT_FAILURE);
       }
```

```
    strip(ofname___);
    strcat(ofname___,".kfo");

     fpout_ = fopen(ofname___, "w");
     ks_ = RSA_size(key_);
     cipher_ = (unsigned char*)malloc(ks_ * sizeof(unsigned char));
     plain_ = (unsigned char*)malloc(ks_ * sizeof(unsigned char));

     while(!feof(fpin_)) {
       memset(cipher_, '\0', ks_);
       memset(plain_, '\0', ks_);
       if ((len_ = fread(cipher_, 1, ks_, fpin_)) == 0)
      break;
       size_ = rsa_decrypt(key_, cipher_, len_, &plain_);
       fwrite(plain_, 1, size_, fpout_);
     }

     fclose(fpout_);
     fclose(fpin_);
     free(plain_);
     free(cipher_);
     RSA_free(key_);


  //decirfando documento com AES
     key_file__.open(ofname___,ios::in);
     //preparando chave
     i = 0;
     forever
     {
       key_file__.get(ch);
       if (key_file__.eof()) break;
       key[i] = ch;
       i++;
     }
cout << "Decifrando documento..." << endl;

     //iniciando decifragem final
  aes_init(&a,MR_CFB1,16,key,iv);

  strip(ifname);
  strcat(ifname,".aes");
  plaintext_.open(ifname,ios::in);
  if (!plaintext_){
     cout << "Nao eh possivel abrir " << ifname << "\n";
     return 0;
  }

  strcpy(ofname,ifname);
  strip(ofname);
  strcat(ofname,".ptx");
  ciphertext__.open(ofname,ios::binary|ios::out);
  if (!ciphertext__){
      cout << "Nao eh possivel abrir " << ofname << "\n";
      return 0;
  }
```

```
    //concretamente decifrando o texto plano!...
    forever
    { // decrypt input ..
        plaintext_.get(ch);
        if (plaintext_.eof()) break;
        aes_decrypt(&a,&ch);
        ciphertext__ << ch;
    }

    ciphertext__.close();
    plaintext_.close();
    aes_end(&a);


    //apagando arquivos temporarios
    apagar(ofname);

    strip(ofname);
    strcat(ofname,".ptx");

    printf("Decifragem finalizada com sucesso!\n");
    cout << "O arquivo original foi salvo em: " << ofname << endl;
    printf("Renomeie o arquivo com sua extensao original.\n");
    printf("Pressione uma tecla para continuar....\n");
    getchar();
    getchar();
       /**********************End IBE-ACT**********************/
    return 0;
}
```

```
//ibe-act.cpp

#include <cstdlib>
#include <iostream>
#include <stdio.h>

int main(){
char op;

do{
system("clear");
printf("IBE na ACT\n");
printf("Escolha uma das seguintes opcoes:\n");
printf("<1> Manter em sigilo temporario um arquivo.\n");
printf("<2> Recuperar um arquivo sigiloso.\n");
printf("<3> Recuperar uma chave do tempo.\n");
printf("<4> Sair.\n");
printf("?:");

cin >> op;

     if (op=='1'){system("./ibe_enc_act");
     op='5';
     }
```

```
     if (op==’2'){system(“./ibe_dec_act”);
     op=’5';
     }
     if (op==’3'){system(“./ibe_ext_act”);
     op=’5';
     }
     if (op==’4')system(“clear”);

}while((op!=’1')&&(op!=’2')&&(op!=’3')&&(op!=’4'));

return 0;
}
```

Apêndice A

Artigo TCC

IBE Aplicado à Criptografia Temporal

José Manuel Ramírez Núñez

Universidade Federal de Santa Catarina

papu@inf.ufsc.br

Abstract. *The secrecy of temporal eletronic documents, a fundamental requirement in auctions and wills, usually consists in keeping the secret document, or its decipher key in custody of a trustful entity. This results in hight storage and processing costs, besides of the need of a great security structure by this entity. In this paper it is showed a solution that uses the Identity-Based Encryption to create the decipher keys only in the moment of the secrecy break, by the use of traditional assimetric cryptography to keep a safe channel between the entities that are participating of the communication. This approach requires lass efforts by the reliable entity responsible by the secrecy, simplifying the use of the temporal cryptography system.*

Resumo. *O sigilo temporal de documentos eletrônicos, requisito fundamental em aplicações como leilões, testamentos e licitações públicas, normalmente consiste em manter o documento sigiloso, ou a sua chave de deciframento, sob a custódia de uma terceira entidade confiável. Isto implica em altos custos de armazenamento e processamento por parte desta entidade, além da robusta infra-estrutura de segurança por trás dela. Neste trabalho é apresentada uma solução que utiliza a criptografia baseada em identidade para gerar as chaves de deciframento somente no momento da quebra do sigilo, usando criptografia assimétrica tradicional para manter um canal seguro entre as entidades participantes da comunicação. Esta abordagem poupa esforços à entidade de confiança responsável pelo sigilo, simplificando o uso do complicado sistema de criptografia temporal.*

1- Introdução

O sigilo temporal visa garantir a confidencialidade de um documento por um determinado período de tempo. Aplicações como leilões, licitações públicas e testamentos são alguns exemplos em que este requisito é crítico.

Uma das formas de alcançar o sigilo temporal é através da submissão do documento a uma terceira parte confiável responsável pela custódia do documento até o momento da quebra do sigilo. Esta técnica é *"perigosa e de alto custo, pois é necessário confiar na terceira parte tanto do ponto de vista de sua honestidade quanto da sua capacidade de manter íntegro e inacessível o documento até a solicitação por alguma entidade autorizada"*, [Custódio et al, 2003].

Outra forma é através de técnicas de criptografia simétrica e assimétrica. Nesta abordagem, o interessado em manter o documento sigiloso cifra o documento e mantém a chave sob sigilo até o momento de liberação do documento. Porém, se a chave for perdida, o documento não poderá

ser recuperado. Uma variante deste esquema é a cifragem dos documentos com chaves públicas cujas respectivas chaves privadas de deciframento não sejam geradas senão até o momento marcado para a liberação do documento.

A criptografia baseada em identidades (IBE) possui como principal característica a facilidade no gerenciamento de chaves criptográficas, permitindo cifrar documentos com identificadores como sendo chaves públicas. Um exemplo de identificador pode ser um endereço de correio eletrônico, ou mesmo uma data e hora. Estas chaves públicas podem não possuir ainda as suas respectivas chaves privadas no momento do seu uso, chaves privadas que somente serão geradas no momento em que forem solicitadas. Estas características fazem com que a criptografia baseada em identidades possa ser aplicada à criptografia temporal, onde poderemos cifrar os documentos com a chave pública definida por uma data e hora no futuro, momento que uma vez atingido, permitirá a geração e liberação de uma chave privada correspondente à data e hora especificada no documento.

2- Criptografia Temporal

May [May, 1993] foi o primeiro a discutir a criptografia temporal no contexto de liberação de documentos eletrônicos, apresentando uma solução que incluía uma terceira entidade TP como responsável pela guarda do documento para sua posterior liberação. Uma outra solução apresentada consistia em manter em sigilo somente a chave criptográfica de decifragem do documento. A segunda proposta foi a mais utilizada, principalmente por razões de desempenho [Custódio et al, 2003].

A primeira abordagem para a criptografia temporal foi proposta por Rivest [Rivest et al., 1996] e consite em implementar um algoritmo que seja executado em *n* etapas sequenciais, não paralelizáveis, de modo que o resultado desta execução seja a chave criptográfica de deciframento correspondente à chave pública utilizada para gerar o documento. Esta abordagem requer muito poder de processamento e memória disponível. Outra abordagem consiste em cifrar o documento com criptografia simétrica, cifrando a chave utilizada neste processo com a chave pública de uma TP. Assim, ela poderá decifrar a chave no momento oportuno. Esta abordagem é muito utilizada pela suas facilidades na implementação.

Mont [Mont et al., 2003] propôs uma outra abordagem utilizando criptografia baseada em identidades para a liberação das chaves de deciframento. Estas chaves somente serão produzidas na data e hora especificada para sua liberação, o que se torna uma grande vantagem. Na proposta feita por [Mont et al., 2003], é incluída uma comparação de criptografia tradicional (RSA) da segunda abordagem acima resumida, e a criptografia baseada em identidade (IBE) para trabalhar com tempo, e que demonstrava que o IBE é mais eficiente em termos de desempenho. A única diferença da proposta feita por Mont [Mont et al., 2003] com a proposta deste trabalho é a do estabelecimento de um canal de comunicação seguro entre duas entidades por meio de criptografia assimétrica tradicional RSA.

3- O protótipo de Criptografia Temporal Baseada em IBE

O protótipo de criptografia temporal IBE consiste na implementação de criptografia

temporal baseada em IBE (na manipulação de chaves de tempo) e complementada com criptografia tradicional para segurança do canal de comunicação.

Esta implementação tem como objetivo garantir a fácil manipulação das chaves públicas de tempo assim como estabelecer um canal seguro entre as entidades envolvidas na comunicação. Em outras palavras, deve ser fácil para Alice cifrar um documento para Beto de maneira a somente ele poder ter acesso ao conteúdo do mesmo, na data e hora especificada por Alice. Esta facilidade é garantida graças ao IBE devido ao fato de não ser necessário a criação de um par de chaves criptográficas para tornar sigiloso um documento através do tempo. Basta cifrar este documento com uma chave pública indicando o tempo da sua abertura, no formato data/hora, e garantir o controle de liberação de chaves sob uma estrita vigilância do tempo. O esquema é apresentado na figura 1.

O documento é cifrado com uma chave aleatória Ks1 usando criptografia simétrica AES. Ks1 é cifrada com a chave pública de Beto usando criptografia assimétrica RSA (chave no formato padrão PEM), o resultado é novamente cifrado, mas desta vez com a chave pública IBE, usando uma data/hora. A chave cifrada é concatenada ao documento cifrado, juntamente com a chave pública RSA do destinatário e a chave pública IBE, informando a data/hora da quebra do sigilo. O meta-documento resultante contém informações sobre a data da quebra do sigilo e o destinatário do documento.

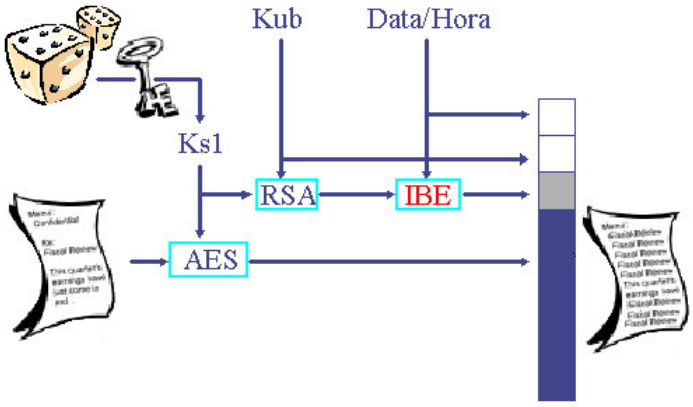

Figura 1. *Fluxograma de dados do esquema do protótipo.*

Foi criado um programa gerenciador chamado ***ibe-act***, o qual gerencia três dos quatro passos da implementação de Boneh & Franklin (Cifragem, Decifragem e Extração). O programa ***ibe-act*** esta encarregado de chamar os sub-programas IBE através de uma lista de opções. Programas que foram modificados para dar suporte ao canal de comunicação seguro RSA, conforme figura 17.

Vamos supor agora que Alice queira cifrar um documento (carta.txt) para Beto, mas ela só quer que ele o abra no dia 20 de abril de 2005 às 15h30. Para isso, deve acontecer o seguinte:

1- Alice abre o programa ***ibe-act*** e manda cifrar o documento entrando com os seguintes parâmetros: Semente geradora de *r* (número de até 9 dígitos), documento a cifrar *(carta.txt)*, destinatário (chave pública de Beto: KuB, no formato PEM) e data e hora da publicação da chave de deciframento (no formato, i.e.: 200504201530).

2- O programa cliente gera uma chave aleatória (*Ks1*) para cifrar o documento com

criptografia simétrica (AES) e cifra esta chave com a chave pública do Beto (RSA).

3- A chave *Ks1* cifrada acima é novamente cifrada, mas desta vez com uma data/hora (como chave pública do IBE), logo é concatenada ao documento cifrado junto com a data/hora especificada para a quebra do sigilo, mais a chave pública do destinatário Como resultado final temos um documento extendido contendo informações sobre a data da quebra do sigilo e o destinatário do documento.

Alice já pode entregar o documento para o seu destinatário, no caso Beto, quem não poderá obter a chave de deciframento da PKG senão até a data especificada por Alice. Não adianta Beto mudar a data no documento, visto que a chave devolvida pela PKG não corresponderá à chave pública IBE com que foi cifrado o documento. Beto só pode esperar.

Supondo que um intruso, ou um criptanalista (Carlos) obtenha a chave com a qual Alice cifrou o documento para Beto, não adianta Carlos ter uma cópia do documento enviado por Alice, pois uma vez liberada a chave para abrir o documento, somente Beto pode chegar até o texto original graças a sua chave privada RSA.

Beto somente solicita à PKG a chave correspondente à data/hora especificada no documento. A PKG produz a chave correspondente só se a data/hora solicitada já foi atingida, e, o mais importante, só quando a chave é solicitada. Esta implementação evita congestionamentos no tráfego de dados assim como processamentos e armazenamentos desnecessários, uma vez que todos os dados transmitidos e processados são mínimos e atendem aos critérios de segurança para os quais são destinados.

Para a implementação o ambiente escolhido foi Mandrake 8.2, com a biblioteca OpenSSL, e compiladores gcc/g++. A implementação IBE utilizada no trabalho é a de Boneh & Franklin [Boneh, D. & Franklin, M., 2001], disponibilizada na biblioteca MIRACL, [Wilkich, 2004].

A PKG está compreendida pelos programas IBE_SET (original do MIRACL), e IBE_EXT_ACT (modificado de ibe_ext.cpp para controle de tempo).

O servidor onde deve rodar a PKG deve possuir um controle de tempo seguro (protocolo NTPv4), sincronizado com o relógio atômico do Observatório Nacional.

A PKG está encarregada de gerar e publicar os valores $N=p*q$ (por meio de IBE_SET), assim como da liberação ou não das chaves de deciframento (por meio do programa IBE_EXT_ACT).

A própria IBE (ibe_enc.cpp do MIRACL) já implementa a cifragem de documentos via criptografia simétrica (AES), por meio de uma chave aleatória. Este procedimento será aproveitado para a implementação do esquema proposto. Os arquivos moficados fora da PKG são ibe_enc.cpp e ibe_dec.cpp para a cifragem e decifragem de documentos respectivamente. O teste de uso do protótipo é apresentado a seguir:

Primeiramente devemos configurar a PKG, gerando e publicando $N=p*q$ através do programa IBE_SET.

O programa IBE_EXT_ACT, que gera as chaves privadas, só fica aguardando ser chamado. A configuração de IBE_SET é mostrada na figura 2:

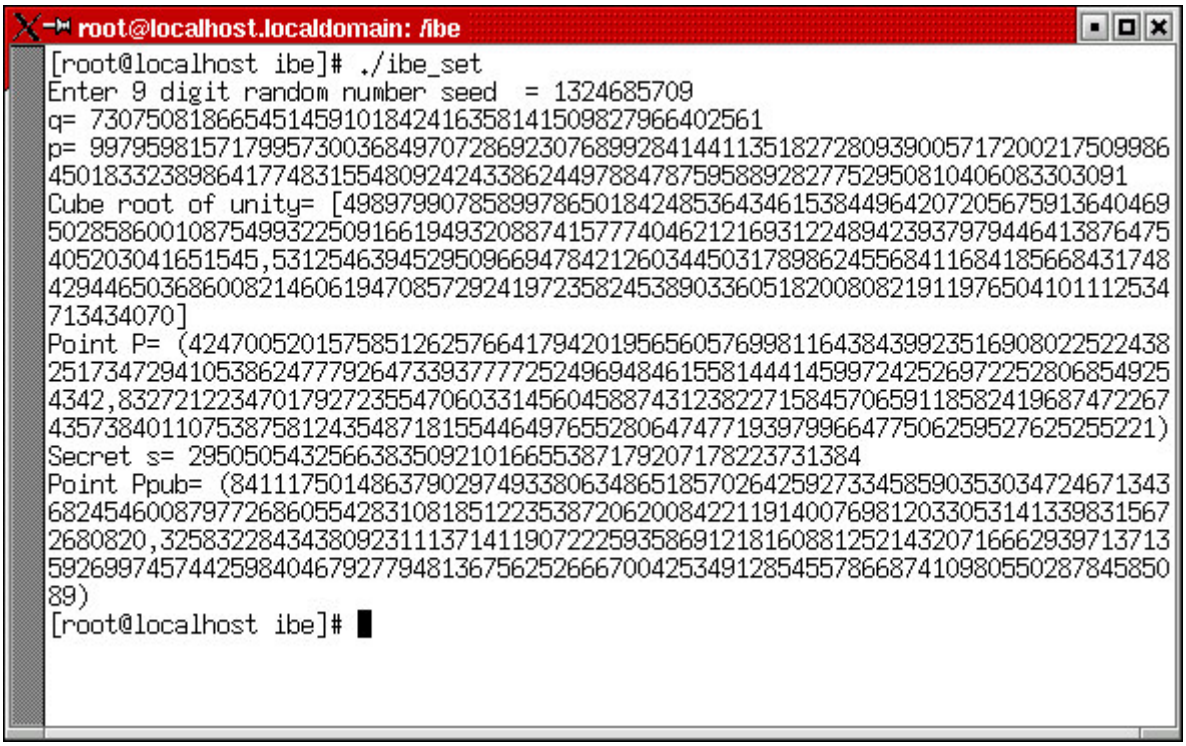

Figura 2. *Programa de configuração da PKG do IBE.*

Agora que possuímos o arquivo "common.ibe" (gerado por IBE_SET) podemos proceder ao uso dos programas de manipulação de arquivos. Vamos supor que queiramos cifrar o arquivo "miracl-ibe.zip" para ser aberto numa data futura. No exemplo, o dia 08 de maio de 2004, às 14h00. Chamamos então o programa IBE-ACT (O resultado aparece na figura 3):

 # ./ibe-act

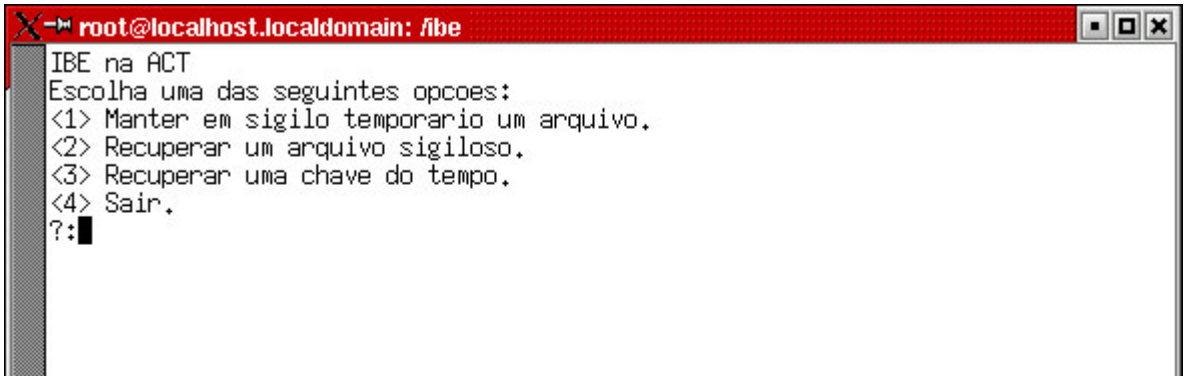

Figura 3. *Tela inicial do IBE-ACT.*

Escolhemos então a primeira opção, na qual é chamado o programa de cifragrem IBE_ENC_ACT, como se vê na figura 4.

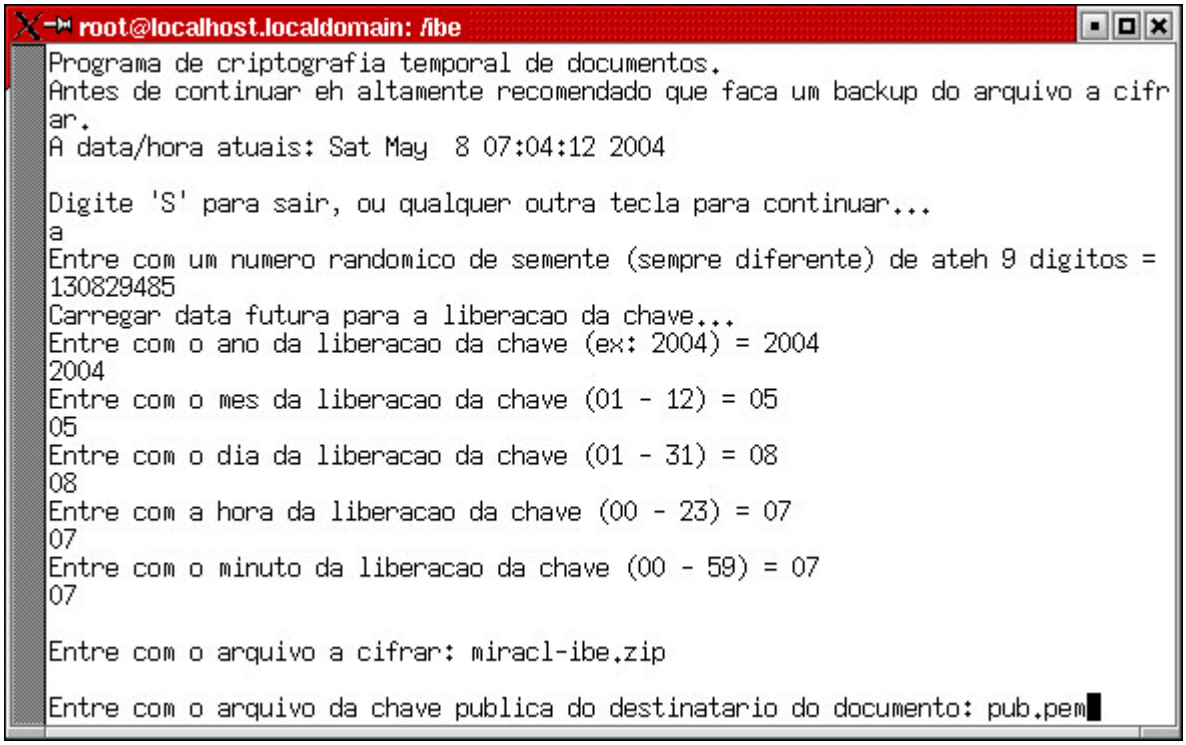

Figura 4. *Tela da opção 1, onde ciframos um documento (miracl-ibe.zip) para às 7h07 do dia 8 de maio de 2004.*

A data/hora atual é exibida somente para servir de referência.

É solicitado ao usuário digitar um número randômico para servir de semente na geração de *r*. Logo, são inseridos os dados do momento da liberação da chave. Este processo está dividido em entrada de ano, de mês, de dia, de hora e de minuto para o simples efeito de controle da validade dos dados e evitar confusão ao usuário. Note que o sistema permite entrada de dados de tempo do tipo inexistente, como por exemplo, 30 de fevereiro de 2005, mas isso não representa problema por que a liberação da chave poderá ser realizada no primeiro dia válido após a data questionada, no caso, a chave seria liberada no dia 01 de março de 2005.

Continuando com o programa em execução, por fim é solicitado o nome do arquivo a cifrar, digitamos "miracl-ibe.zip" e logo é solicitado que ingressemos o nome do arquivo da chave pública do destinatário (no formato "pem"), no nosso exemplo foi "pub.pem".
Uma vez acabada a etapa de cifragem do documento, é gerado o meta documento "miracl-ibe.act", que é o nosso arquivo cifrado.

Voltando à tela inicial saímos do programa para verificar o arquivo gerado. Veja na figura 5.

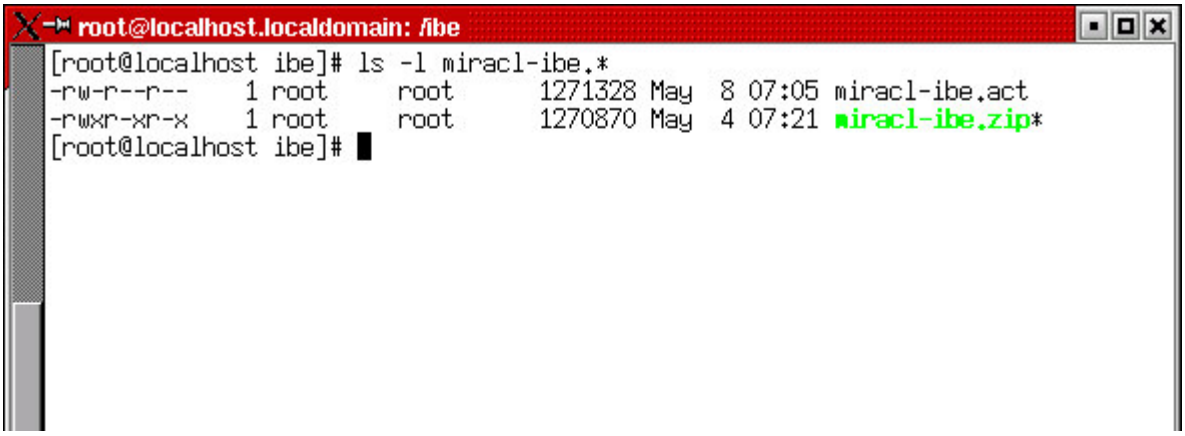

Figura 5. *Lista dos documentos do teste (miracl-ibe*).*

Note que o meta documento ***miracl-ibe.act*** ocupa mais espaço em disco devido às informações referentes à recuperação do arquivo original.

Vamos proceder à tentativa de recuperação do documento, supondo que a condição data/hora não tenha sido cumprida. Chamando de novo o nosso programa IBE-ACT, desta vez ingressamos com a opção "recuperar uma chave do tempo", pois precisaremos da chave privada IBE, correspondente à data/hora, que virá num arquivo ***".pri"***, e em nosso exemplo seria ***miracl-ibe.pri***. Veja na figura 6.

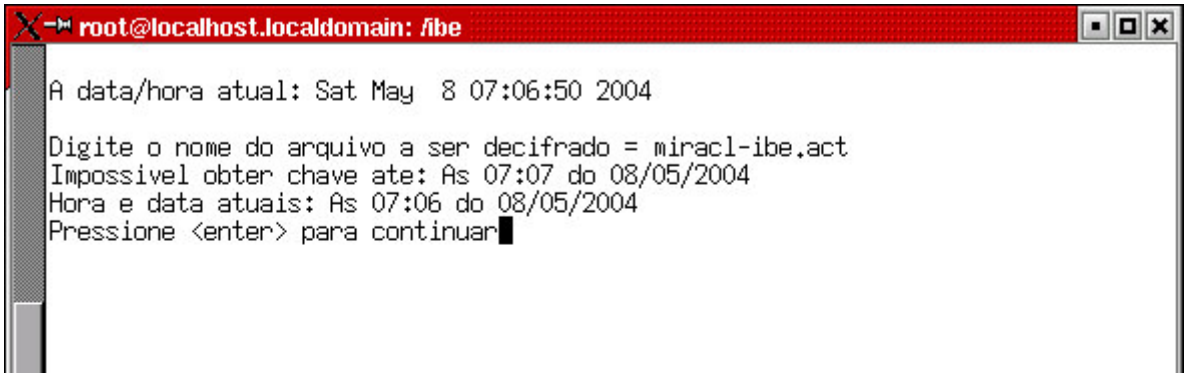

Figura 6. *Mensagem de erro no programa IBE_EXT_ACT.*

O programa lê o cabeçalho do arquivo *.act* e verifica se a data/hora contida nele é menor ou igual à impressa na tela (horário do servidor). Caso afirmativo, retorna satisfatoriamente um arquivo *.pri*, caso contrário, exibe uma mensagem contendo a data/hora em que o usuário deve solicitar a chave.

Veja na figura 7 um outro exemplo no qual o usuário consegue a chave devido ao tempo alvo ter sido atingido:

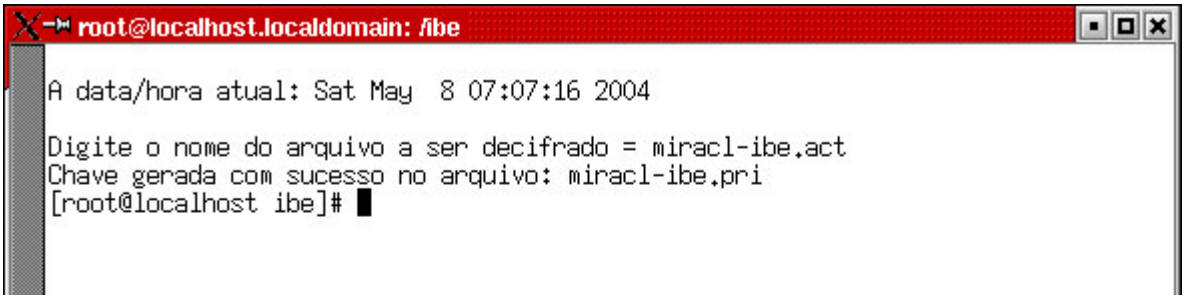

Figura 7. *Sucesso na recuperação da chave particular IBE.*

Continuando com o nosso exemplo, uma vez obtida a chave junto à TA, devemos prosseguir à abertura do documento. Antes conferimos se possuímos os arquivos necessários: miracl-ibe.act e miracl-ibe-ibe.pri, além de, é claro, a nossa chave particular RSA (no exemplo, sec.pem). Rodamos então o IBE-ACT e vamos à segunda opção.

O programa IBE_DEC_ACT aparece na tela. Veja figura 8.

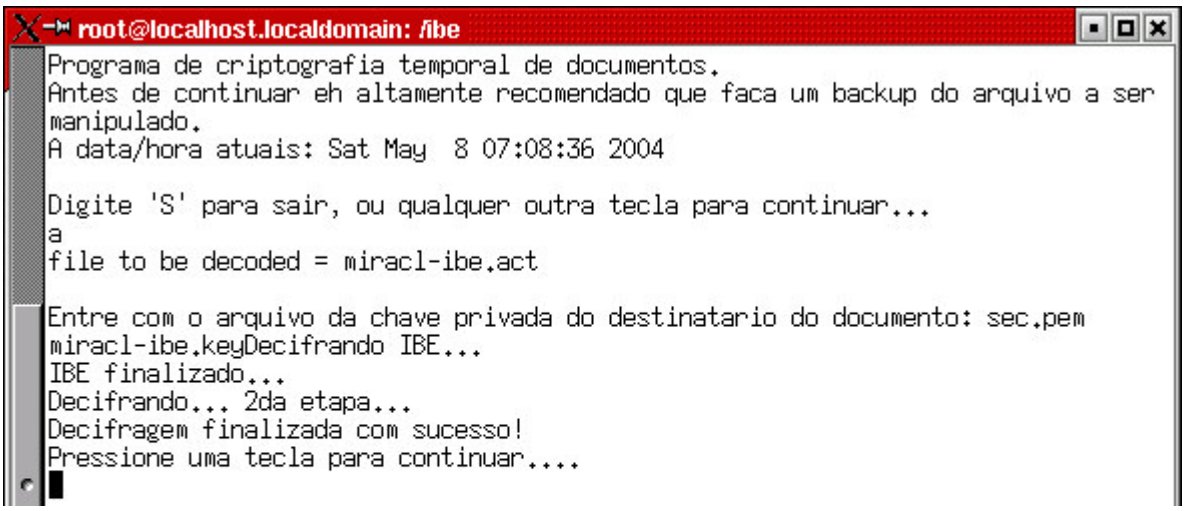

Figura 8. *Decifragem finalizada com sucesso!*

Ingressamos o nome do arquivo a ser decifrado e o arquivo da nossa chave particular RSA.

O programa processa o meta documento *.act* e devolve o arquivo original com a extensão *.ptx* (plaintext). O arquivo *miracl-ibe.ptx* é exatamente igual ao arquivo original *miracl-ibe.zip*, e foi nomeado assim para fins didáticos. Vejamos na figura 9 os arquivos gerados.

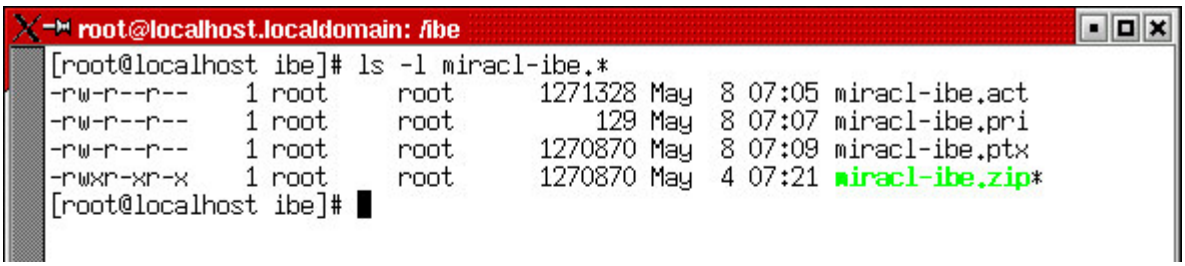

Figura 9. *Os 3 arquivos gerados no processo.*

Note que o arquivo ***.ptx*** ocupa exatamente o mesmo espaço em disco que o seu original ***.zip***.

Nos testes realizados o sistema reconheceu corretamente o arquivo com a extensão ***.ptx*** como cópia fiel do documento original, sendo lido corretamente e não apresentando nenhum erro no processo de descompactação.

4- Conclusões

A criptografia baseada em identidade facilita a manipulação de chaves públicas, permitindo controlar o tempo em que as chaves privadas correspondentes serão criadas, viabilizando a sua aplicação em sistemas criptográficos temporais. Podemos usar strings em qualquer formato nas chaves públicas para um melhor controle da geração das chaves privadas em momentos específicos. Para garantir um canal seguro de comunicação entre duas entidades foi utilizada, com sucesso, a criptografia assimétrica tradicional (RSA), com ferramentas OpenSSL, utilizando chaves criptográficas no padrão PEM. Tudo isso combinado à eficiência da criptografia simétrica (AES), a qual garante melhor performance nas transformações criptográficas de grande quantidade de dados.

Embora a implementação deste trabalho não mantenha um controle sobre o destino das chaves privadas liberadas, elas somente terão utilidade para o destinatário do documento sigiloso. Cifrar inicialmente o documento com a chave pública do destinatário é um procedimento opcional, conforme no protótipo, que vista agregar segurança ao documento, pois impede que entidades alheias à comunicação que tenham acesso à chave privada temporal consigam ler o documento sigiloso. O resultado foi um sistema que realmente protege documentos através do tempo, sempre dentro de um canal seguro e cumprindo com os propósitos impostos sobre o sigilo.

Na implementação do canal seguro entre as duas entidades (confidencialidade) não utilizamos IBE ao invés de RSA (poderíamos usar também o IBE pensando na idéia de libertar o sistema totalmente de toda a infra-estrutura de chaves públicas, como dita o ideal do esquema IBE) pelo fato de que ainda há desvantagens apresentadas pelo IBE, como sendo a grande dependência da chave mestra da PKG, e o conhecimento por parte dela de todas as chaves dos usuários. O esquema de segurança físico por trás dela deve ser muito forte, além da confiança que seria depositada toda numa única fonte. Isto dificulta esse tipo de implementação.

7- Bibliografia

[Benits Jr., W., 2003] Sistemas Criptográficos Baseados em Identidades Pessoais. Master's thesis, Universidade de São Paulo, 2003.

[Boneh, D. & Franklin, M., 2001] Identity-based encryption. Disponível em <http://crypto.stanford.edu/ibe/>

[CPLUSPLUS, 2004] The C++ Resources Network. Disponível em <http://www.cplusplus.com>

[Custódio et al., 2003] Custodio R. F., da Silva Dias J., Pereira F. C. e Notoya A. C., 2003. Uma infra-estrutura para o sigilo temporal de documentos eletrônicos. Universidade Federal de Santa Catarina, 2003.

[Custódio, R. F., 2000] CUSTÓDIO, R. F. Notas de aula. Disponível em <http://www.inf.ufsc.br/~custodio/cursos/INE5386/Transparencias/Apostila.zip>

[Grandi, A., 2002] Andrea Grandi, PGP. Mettere i propri dati al sicuro. Disponível em <http://pdf.apogeonline.com/ebook/2002/90025/pdf/PGP.pdf >

[Liberty, J., 1999] LIBERTY, J., C++ para principiantes. Indianapolis - USA, Pretince Hall - 1999.

[May, 2003] Timed-release crypto. Disponível em <http://cypherpunks.venona.com/date/1993/02/msg00129.html>

[Mont et al., 2003] Mont, M. C., Harrison, K., and Sadler, M., 2003. The HP Time Vault Service: exploited ibe for timed release of confidencial information. In *In Proceedings of the twelfth international conference on World Wide Web*, pages 160-169. ACM.

[NTP.org, 2003] NTP: The Network Time Protocol. Disponível em <http://www.ntp.org>

[Observatorio Nacional, 2004] Hora legal Brasileira. Disponível em <http://pcdsh01.on.br>

[OpenSSL, 2004]The open project. Disponível em <http://www.openssl.org>.

[Patroklos G. Argyroudis, 2004] Identity-based encryption resources. Disponível em <http://ntrg.cs.tcd.ie/~argp/ibe.html>

[Pierre Loidreau, 2002] System administrator: Introducción a la criptografía. Disponível em <http://www.linuxfocus.org/Castellano/May2002/article243.shtml>

[Pino C.,G., 2002] Pino Caballero Gil, Introducción a la Criptografia. Madrid. RA-MA - 2002

[Quevedo, 2002] El rincón de Quevedo. Disponível em <http://www.rinconquevedo.tk/>

[Scott, M., 2000] Shamus Software Ltd - MIRACL. Disponível em <http://indigo.ie/~mscott/>

[Schneier, B., 1996] Schneier, B., *Applied Cryptography*, 2º edición, Ed. Wiley, 1996

[Tkotz 1999] TKOTZ, V. Página na internet sobre criptologia. 1999. Disponível em <http:www.numaboa.com/informatica/criptologia>.

[Wilkich, 2004] The Miracl Library. Disponível em <http://www.auburn.edu/~wilkich/>