Implementação de esquemas de criptografia e de assinatura sob o modelo de criptografia de chave pública sem certificado

Renato da Silva Ramalho

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
MESTRE EM CIÊNCIAS

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Routo Terada

O presente trabalho foi realizado com o apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq - Brasil

– São Paulo, março de 2008 –

À minha família: pais, irmãs, esposa e filho...

Agradecimentos

À Deus, pela vida, pela saúde e por ter me dado os maravilhosos pais que tenho.

Aos meus pais, por mostrarem o caminho correto a ser seguido, ensinarem valores e por sempre me apoiar no caminho da vida. Às irmãs pela convivência e pelo carinho sempre recebidos. À esposa pelo apoio, dedicação e compreensão nos períodos mais turbulentos desta trajetória. Ao meu filho, nascido entre o exame de Qualificação e esta Defesa, pelos momentos felizes e únicos.

Ao meu orientador, Prof. Dr. Routo Terada, por acreditar no meu trabalho, pelos grandes ensinamentos e pelas dicas dadas durante a execução deste trabalho.

Aos Professores Marco Dimas Gubitoso e Roberto Hirata Jr pelas sugestões dadas no exame de qualificação.

A todos amigos, professores e demais pessoas envolvidas direta ou indiretamente na elaboração desta dissertação.

Resumo

Gerenciar certificados digitais em uma infra-estrutura de chaves públicas (ICP) é uma tarefa complexa. O modelo de criptografia de chave pública baseada em identidades (IBE - *Identity-Based Encryption*) pretende realizar essa tarefa de modo menos custoso no que se refere ao controle de certificados digitais fazendo uso de identidades pessoais como chave pública. No entanto, nesses sistemas há a necessidade de uma entidade capaz de gerar as chaves secretas e manter a custódia das chaves. Existem aplicações onde a custódia de chaves é indesejada.

O conceito de Criptografia de Chave Pública sem Certificado, ou *Certificateless Public Key Cryptography* (CL-PKC), surgiu para eliminar a custódia de chaves e, ao mesmo tempo, permanecer com as características importantes dos sistemas baseados em identidades pessoais, como a não necessidade de uma infra-estrutura de chaves públicas e a eliminação de certificados digitais, facilitando o gerenciamento do sistema. Desse modo, a chave pública de um usuário não mais será uma identidade pessoal, mas será gerada a partir de uma informação secreta deste usuário.

Este trabalho visa implementar esquemas de criptografia e de assinatura no modelo de criptografia de chaves públicas sem certificado.

Abstract

To manage digital certificates in a Public Key Infrastructure (PKI) is a complex task. The Identity-Based Encryption(IBE) model intends to accomplish that task in a less expensive way concerning the control of digital certificates making use of personal identities as public key. However, in those systems there is the need of an entity capable of generating the secret keys and to maintain the custody of the keys. There are applications where the custody of keys is not desired.

The concept of Certificateless Public Key Cryptography (CL-PKC), appeared to eliminate the custody of keys and, at the same time, to preserve the important characteristics of the Identity-Based cryptography, as the public key infrastructure and digital certificates are not required, facilitating the administration of the system. This way, a user's public key will not be a personal identity any more, but will be generated starting from a secret information of this user.

This work implements cryptography and signature schemes in the Certificateless Public Key Cryptography model.

Índice

CAPÍTULO 1

Introdução

Proteger os sistemas de comunicação é uma necessidade histórica. O desenvolvimento de novos dispositivos de comunicação e o sucesso da internet (e demais redes) faz com que essa necessidade de segurança na comunicação se intensifique ainda mais. Um dos objetivos da segurança de dados é o sigilo das informações trafegadas nas redes de comunicação. A criptografia provê os mecanismos necessários para a segurança nas comunicações.

Criptografia é uma palavra de origem grega, formada pela união de *kryptós* (escondido) e *grápho* (grafia), isto é, escrita escondida, ou escrita secreta. Os primeiros protocolos criptográficos utilizavam uma única informação secreta (chave secreta) para cifrar e decifrar informações. Esse tipo de protocolo é chamado de protocolo de *criptografia de chave secreta*, ou *criptografia simétrica*. Somente quem conhece a chave secreta tem acesso às informações protegidas por ela. Esse é o princípio da *confidencialidade*.

No entanto, a criptografia simétrica não provê solução para os outros dois princípios da segurança da informação descritos abaixo:

Autenticidade. Tanto o rementente quanto o destinatário das mensagens devem ter a certeza de que não existe um intruso na comunicação, isto é, tanto o remetente tem certeza que o destinatário será o único capaz de ler a informação criptografada (isto é chamado de *autenticação de destino*), quanto o destinatário tem certeza de que o remetente é quem realmente diz ser (isto é chamado de *autenticação de origem*).

Integridade. A informação trafega íntegra entre remetente e destinatário, isto é, não foi adulterada intensionalmente (tentativa de fraude), nem involuntariamente (erro na transmissão ou no armazenamento dos dados).

Na criptografia simétrica entre mais de dois participantes, não é possível garantir autenticidade na comunicação, pois todo participante que conhece determinada chave pode ser o autor da mensagem, e, todo participante que conhece a chave pode decifrar a mensagem.

Para resolver isso, para cada par de participantes, deveria haver uma chave secreta. Mas como combinar as chaves entre os participantes de modo que seja utilizada uma chave com cada um dos demais participantes? Esse é o problema de *troca de chaves* da criptografia simétrica.

A *criptografia de chave assimétrica* ou *criptografia de chave pública* fornece confiabilidade, autenticidade, integridade e combinação segura de chave entre os participantes. Esse modelo de criptografia envolve um par de chaves: uma usada para cifrar, que deve ser tornada pública (chamada de *chave pública*) e outra usada para decifrar, que deve ser mantida em segredo(chamada de *chave privada*).

1.1 Infraestrutura de chave pública

O modelo de chave pública surgiu em (DIFFIE; HELLMAN, 1976). Nesse modelo, são utilizadas duas chaves (a pública e a privada) para as operações de criptografia e decriptografia. No entanto, como não há vínculo entre o par de chaves e a respectiva identificação do usuário, é possível realizar um ataque a esse protocolo — chamado *homem-no-meio* (*man-in-the-middle*) — onde um intruso intercepta as mensagens entre os dois participantes legítimos, passando-se pelo remetente perante o destinatário e ainda passando-se pelo destinatário perante o remetente. Dessa forma, o intruso é capaz de decifrar todas as mensagens trocadas.

Para resolver esse problema, uma modificação foi realizada: a inclusão de um novo valor secreto, conhecido apenas por uma terceira parte, chamada de autoridade de confiança.

Com isso, novos algoritmos baseados no protocolo de Diffie-Hellman surgiram, os algoritmos de criptografia de chave pública. Com o intuito de vincular unicamente a chave pública de uma entidade à sua respectiva identificação, foi sugerido por (KOHNFELDER, 1978) a criação de uma mensagem assinada por uma autoridade de confiança. Essa mensagem (*certificado digital*) contém a representação da identidade, a chave pública correspondente e um identificador temporal (período de validade).

A necessidade de certificados digitais para prover segurança contra o ataque *homem-no-meio* requer o gerenciamento para o uso (e para a revogação) dos certificados digitais. A infra-estrutura responsável por esse gerenciamento é a infra-estrutura de chaves públicas (ICP) — ou *Public Key Infrastructure (PKI)*.

Manter ICPs é uma tarefa custosa. Dificuldades relacionadas ao gerenciamento dos certificados digitais podem ser vistas em (BERBECARU; LIOY; MARIAN, 2001; GUTMAN, 2002; ELLISON; SCHNEIER, 2000). Podemos exemplificar essas dificuldades, citando o problema da revogação de certificados, onde em um espaço de tempo entre o comprometimento de um par de chaves e a efetiva revogação do certificado correspondente, o sistema pode ser comprometido.

Interessado em simplificar o gerenciamento de emails, Shamir propôs em (SHAMIR, 1984) um novo esquema em que não era necessário o uso de certificados digitais (portanto elimina a necessidade de um diretório de chaves públicas, ou seja, elimina o custoso gerenciamento de uma infra-estrutura de chaves públicas) e que pudesse utilizar uma string qualquer arbitrária (um identificador pessoal de um usuário — ID), como por exemplo seu endereço de email, número de CPF ou número de telefone como chave pública. Desse modo, o vínculo entre chave pública e respectivo dono é automático. Com o modelo de Shamir surge um novo conceito em criptografia assimétrica: o modelo de criptografia baseada em identidades, ou *Identity-Based Encryption (IBE).*

O esquema de (BONEH; FRANKLIN, 2001) foi o primeiro protocolo de criptografia (e decriptografia) demonstrado seguro e eficiente computacionalmente e utiliza emparelhamentos bilineares sobre grupos baseados em curvas elípticas, tratados inicialmente em (SAKAI; OHGISHI; KASAHARA, 2000).

Nos modelos baseados em identidades de Shamir e Boneh-Franklin existe uma autoridade (chamada *PKG, Private Key Generator*) responsável por gerenciar o sistema e gerar as chaves privadas de cada usuário. Desse modo, o PKG conhece todas as chaves privadas do sistema, conferindo a característica de *key escrow* ("custódia de chaves", quando é possível recuperar as chaves de usuários em caso de extravio ou outras necessidades) ao mesmo.

No entanto, algumas aplicações devem ser protegidas dessa característica inerente aos sistemas IBE, pois é necessário ter certeza de que apenas o remetente e o destinatário conhecem certa mensagem sigilosa. A partir daí, novos esquemas, incluindo outros esquemas de IBE, como em (CHEN; ZHANG; KIM, 2003), estendendo ou alterando o esquema de Boneh-Franklin, foram propostos para eliminar o *key escrow.*

O trabalho de (AL-RIYAMI; PATERSON, 2003), derivado do esquema de Bonef-Franklin, introduziu o modelo de criptografia de chave pública sem certificado (ou *Certificateless Public Key Cryptography, CL-PKC*). Neste modelo, da mesma forma que em IBE, não é necessário o uso de certificados para vincular chaves públicas com suas respectivas identificações. Uma vantagem nesse modelo é a eliminação da custódia de chaves, ou seja, a autoridade de confiança não é capaz de decifrar as mensagens cifradas por seus usuários. Por outro lado, com CL-PKC, a chave pública perde a característica de ser simplesmente uma identidade pessoal do usuário, como em IBE.

Outra diferença entre IBE e CL-PKC é o poder da chave secreta da autoridade de confiança (a *chave-mestra.* No caso de IBE, o comprometimento da chave-mestra compromete todo o sistema, fato que não ocorre em CL-PKC.

1.2 Motivação

A infra-estrutura de chaves públicas tradicionais possui um alto custo de gerenciamento. Ao mesmo tempo, a criptografia baseada em identidades elimina a exigência de ICPs, mas enfraquece alguns aspectos de segurança, com a custódia de chaves, a não irretratabilidade (já que o PKG pode forjar assinaturas) e o grau de criticidade da chave-mestra. Por estes motivos, o modelo CL-PKC torna-se atraente, pois dispensa a necessidade de uma infra-estrutura de chaves públicas e melhora algumas características de segurança no modelo IBE, e, portanto, deve ser melhor estudado e testado através de implementações.

1.3 Objetivo

Este trabalho tem por objetivo implementar o esquema de CL-PKC descrito em (GOYA, 2006), já que este esquema apresenta algumas vantagens em relação a outros esquemas sob o mesmo modelo.

1.4 Organização do trabalho

Este trabalho está estruturado da seguinte forma:

- o capítulo 2 apresenta alguns fundamentos matemáticos e conceitos básicos ligados à criptografia de chaves públicas;
- o capítulo 3 descreve os esquemas escolhidos para implementação, bem como os motivos pelos quais foram escolhidos;
- o capítulo 4 descreve os procedimentos adotados na implementação proposta, tais como a escolha de parâmetros, ferramentas e bibliotecas;
- o capítulo 5 conclui este trabalho relatando os resultados obtidos e sugerindo trabalhos futuros;
- no apêndice A descrevemos alguns conceitos relevantes em álgebra.
- e, por fim, no apêndice B fornecemos o código-fonte desenvolvido.

CAPÍTULO 2

Conceitos

Neste capítulo são apresentados fundamentos matemáticos e conceitos básicos em criptografia necessários ao entendimento do trabalho.

2.1 Fundamentos Matemáticos

Nesta seção são abordados alguns fundamentos matemáticos importantes relacionados ao trabalho. Conceitos de álgebra, como grupo e corpo encontram-se no Apêndice A.

2.1.1 Curvas Elípticas

Os conceitos desta seção podem ser vistos em (GOYA, 2006), (BARRETO, 1999), (KOBLITZ, 1994) e (TERADA, 2000).

Tanto os esquemas baseados em identidades como os esquemas de criptografia sem certificado são baseados, principalmente, em curvas elípticas.

A aplicação das curvas elípticas em criptografia foi sugerida, inicialmente, por (KOBLITZ, 1987) e (MILLER, 1985), com o intuito de prover maior segurança e permitir o uso de chaves criptográficas de tamanho menor, em relação às chaves utilizadas em outros sistemas criptográficos, conforme mostra a tabela 2.1, cujos dados estão em (BARRETO, 1999) e (CERTICOM, 2004).

Considere um corpo K, os elementos $a, b, c, d, e \in K$, e a equação:

$$y^2 + axy + by = x^3 + cx^2 + dx + e \tag{2.1}$$

Definimos uma *curva elíptica* sobre K, denotada $E(K)$, como o conjunto dos pares $(x, y) \in K \times K$ que satisfazem a equação 2.1, mais um ponto $\mathfrak{O}$, chamado de ponto no

Modelo de criptografia		
Simétrico	Elíptico	RSA
80	163	1024
128	256	3072
192	384	7680
256	512	15360

Tabela 2.1: Equivalência entre tamanhos de chaves criptográficas.

infinito.

O número de pontos de uma curva elíptica E é chamado *ordem da curva* e é denotado por $\#E$. Para corpos finitos $GF(q)$, o *Teorema de Hasse* diz que a ordem da curva satisfaz:

$$q + 1 - 2\sqrt{q} \leq \#E \leq q + 1 + 2\sqrt{q}$$

A equação 2.1 pode ser simplificada por meio de mudanças de coordenadas. Se a característica de K for 2 (isto é, para $GF(2^m)$), a equação é reduzida a uma das duas formas:

$$y^2 + xy = (x^3 + ax^2 + b) \quad (2.2)$$
$$y^2 + cy = (x^3 + ax + b) \quad (2.3)$$

Para corpos de característica 3 (isto é, para $GF(3^m)$), a equação é escrita como

$$y^2 = (x^3 + ax^2 + bx + c) \quad (2.4)$$

Para $p > 3$, $GF(p^m)$), a equação 2.1 pode ser reescrita como

$$y^2 = (x^3 + ax + b) \quad (2.5)$$

Para que os pontos da curva elíptica formem um grupo, precisamos definir uma operação binária sobre eles, e especificar os elementos identidade e inverso.

Como a curva elíptica é simétrica em relação ao eixo x, o elemento inverso de um ponto $P = (x, y)$ é o ponto simétrico $-P = (x, -y)$. O elemento identidade é o ponto no infinito, $\mathfrak{O}$. Em curvas com discriminante diferente de zero, a operação binária do grupo é a soma de dois pontos P e Q, definida da seguinte maneira:

- $P + \mathfrak{O} = \mathfrak{O} + P = P, \forall P$;
- Se $P = -Q$, então $P + Q = \mathfrak{O}$;

- Se $P \neq Q$, traça-se uma reta secante à curva, pelos pontos P e Q. Pode-se provar que essa reta sempre intercepta a curva num terceiro ponto, R; então define-se $P+Q = -R$;
- Se $P = Q$, traça-se uma reta tangente à curva pelo ponto P. Pode-se provar que essa reta sempre intercepta a curva num segundo ponto, R; então define-se $P + Q = P + P = 2P = -R$;

Para aplicações criptográficas, por questões práticas e de eficiência, são interessantes as curvas sobre $GF(2^m)$ ou sobre $GF(p)$. Nas seções a seguir, mostramos como realizar a operação de soma de pontos sobre as curvas de interesse.

Soma de Pontos sobre $GF(2^m)$

Sejam $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ dois pontos na curva elíptica definida pela equação 2.3: $y^2 + cy = (x^3 + ax + b)$, com $c \neq 0 \ mod \ 2^m$.

Então $-P = (x_1, y_1 + c)$. Se P e Q não coincidem com o ponto no infinito, e P não é o inverso de Q, a soma $P + Q = (x_3, y_3)$ é calculada como:

$$x_3 = \begin{cases} \left(\frac{y_1+y_2}{x_1+x_2}\right)^2 + x_1 + x_2 & \text{se} \quad P \neq Q \\ \frac{x_1^4+a^2}{c^2} & \text{se} \quad P = Q \end{cases}$$

$$y_3 = \begin{cases} \left(\frac{y_1+y_2}{x_1+x_2}\right)(x_1 + x_3) + y_1 + c & \text{se} \quad P \neq Q \\ \left(\frac{x_1^2+a}{c}\right)(x_1 + x_3) + y_1 + c & \text{se} \quad P = Q \end{cases}$$

Quando $P = Q$, a soma, $2P$, é chamada de *duplicação de P*. Para obter uma multiplicação escalar λP pode-se utilizar duplicações sucessivas e/ou outras somas de interesse.

Para evitar as divisões por c, em geral, adota-se $c = 1$. Com $c = 1$, existem exatamente três classes isomorfas de curvas elípticas sobre $GF(2^m)$, com m ímpar. Uma curva representativa de cada uma dessas três classes é:

$$\begin{aligned} E_1 &: y^2 + y = x^3 \\ E_2 &: y^2 + y = x^3 + x \\ E_3 &: y^2 + y = x^3 + x + 1 \end{aligned}$$

Soma de Pontos em Curva Elíptica sobre $GF(p)$

Sejam $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ dois pontos na curva elíptica definida pela equação $y^2 = (x^3 + ax + b)$, com $4a^3 + 27b^2 \neq 0 \ mod\ p$. Então $-P = (x_1, -y_1)$. Se P e Q não coincidem com o ponto no infinito, e P não é o inverso de Q, a soma $P + Q = (x_3, y_3)$ é calculada como:

$x_3 = t^2 - x_1 - x_2$

$y_3 = t(x_1 - x_3) - y_1$, onde

$$t = \begin{cases} \left(\frac{y_2 - y_1}{x_2 - x_1}\right) & \text{se} \quad P \neq Q \\ \left(\frac{3x_1^2 + a}{2y_1}\right) & \text{se} \quad P = Q \end{cases}$$

Curvas supersingulares e curvas anômalas

Uma curva elíptica E sobre um corpo finito $GF(q)$, de característica p, é chamada *supersingular* se, e somente se, $p \mid t$, onde $\#E = q + 1 - t$.

Quando E é definida sobre $GF(p)$, com $p > 3$, é possível provar que E é supersingular se, e somente se, $t^2 \in \{0, p, 2p. 3p, 4p\}$, onde $\#E = p + 1 - t$.

Uma curva elíptica E sobre um corpo finito $GF(q)$ é chamada *anômala* se, e somente se, $\#E = q$.

Nota-se pelas definições que poucas curvas de um corpo finito são supersingulares ou anômalas.

Mais informações sobre curvas anômalas e supersingulares podem ser obtidas em (WIN; PRENEEL, 1998).

2.1.2 Emparelhamento (*pairing*)

Seja $\mathbb{G}_1$ um grupo aditivo e $\mathbb{G}_2$ um grupo multiplicativo, ambos grupos cíclicos de ordem prima q.

Um mapeamento $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ é um *mapeamento bilinear* se $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$, para todo $P, Q \in \mathbb{G}_1$ e $a, b \in \mathbb{Z}_q$.

Pela propriedade de bilinearidade também valem as igualdades: $\hat{e}(aP, Q) = \hat{e}(P, aQ) = \hat{e}(P, Q)^a$.

Um mapeamento $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ é, segundo (BONEH; FRANKLIN, 2001), um *mapeamento bilinear admissível* se satisfaz as propriedades:

1. Bilinearidade: $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$, para todo $P, Q \in \mathbb{G}_1$ e $a, b \in \mathbb{Z}_q$;
2. Não-degeneração: o mapeamento não leva todos os pares de $\mathbb{G}_1 \times \mathbb{G}_1$ para a identidade de $\mathbb{G}_2$. Como os grupos possuem ordem prima, é equivalente dizer: para um gerador P de $\mathbb{G}_1$, $\hat{e}(P, P) \neq 1_{\mathbb{G}_2}$, isto é, $\hat{e}(P, P)$ é um gerador de $\mathbb{G}_2$;
3. Eficiência computacional: existe um algoritmo de complexidade de tempo polinomial que calcula $\hat{e}(P, Q)$, para todo $P, Q \in \mathbb{G}_1$.

De modo simplificado, um *emparelhamento bilinear*, ou simplesmente *emparelhamento* é um mapeamento bilinear admissível.

2.2 Conceitos em criptografia

Nesta seção apresentamos os conceitos de criptografia assimétrica tradicional e de criptografia baseada em identidades. Informações mais abrangentes sobre conceitos básicos em criptografia podem ser consultadas na vasta bibliografia referente a estes assuntos, com destaque para (MENEZES; VANSTONE; OORSCHOT, 1996), (SCHNEIER, 1994), (STINSON, 1995), (STALLINGS, 2002) e (TERADA, 2000).

2.2.1 Criptografia assimétrica tradicional

O conceito de criptografia de chave pública surgiu em (DIFFIE; HELLMAN, 1976). Segundo (RIVEST, 1990), um criptossistema de chave pública possui os seguintes algoritmos:

Algoritmo de geração de chaves. Este algoritmo recebe como entrada um parâmetro de segurança k, e fornece na saída, um par de chaves: E (chave pública) e D (chave privada);

Algoritmo de criptografia. Este algoritmo recebe como entrada uma chave pública E e uma mensagem M, e devolve na saída um texto cifrado C, denotado por $E(M)$;

Algoritmo de decriptografia. Este algoritmo recebe como entrada uma chave secreta D e um texto cifrado C, e devolve na saída uma mensagem $M = D(C)$.

Os algoritmos acima possuem as seguintes propriedades:

1. Para toda mensagem $M, D(E(M)) = M$;
2. Para toda mensagem $M, E(D(M)) = M$;
3. Os algoritmos de geração de chaves, de criptografia e decriptografia são executados em tempo polinomial no tamanho de suas entradas;
4. O algoritmo de geração de chaves é um algoritmo probabilístico (a saída depende de um valor aleatório); os algoritmos de criptografia e decriptografia são determinísticos (a mesma saída é obtida para entradas iguais);
5. Dada uma chave pública E, mas não a chave secreta D, um adversário não consegue decriptografar um texto cifrado C em tempo polinomial.

Um exemplo clássico é o RSA (RIVEST; SHAMIR; ADLEMAN, 1978).

2.2.2 Criptografia assimétrica - IBE

Interessado em simplificar o gerenciamento de emails, Shamir propôs em (SHAMIR, 1984) um novo esquema em que não era necessário o uso de certificados digitais e que utiliza uma string qualquer arbitrária (o identificador pessoal de um usuário — ID), como por exemplo seu endereço de email, número de CPF ou número de telefone como chave pública. Desse modo, o vínculo entre chave pública e respectivo dono é automático. Surge, então, o modelo de criptografia baseada em identidades, ou *Identity-Based Encryption (IBE)*.

O modelo de Shamir é composto por 4 algoritmos (no caso de ser feita criptografia e assinatura o modelo é composto por 6 algoritmos):

SETUP. Este algoritmo é executado pela Entidade Raiz (Autoridade de Confiança) do sistema, chamada de PKG (*Public Key Generator*) para gerar os parâmetros globais públicos do sistema params e uma chave-mestra, que deve ser mantida em segredo;

EXTRACT. Este algoritmo recebe como parâmetro a chave-mestra e o identificador de um usuário e retorna na saída a chave-privada correspondente;

ENCRYPT/SIGN. Este algoritmo recebe como parâmetro uma chave pública, uma mensagem a ser criptografada e os parâmetros do sistema params e retorna na saída um texto cifrado C;

DECRYPT/VERIFY. Este algoritmo recebe os parâmetros do sistema params, um texto cifrado C e uma chave privada e retorna a mensagem $M = D(C)$.

2.2.3 Criptografia assimétrica - CL-PKC

O modelo de criptografia assimétrica sem certificado, objeto do nosso estudo neste trabalho, será exposto no capítulo seguinte, juntamente com a descrição dos esquemas que pretendemos estudar detalhadamente para implementação.

2.3 Resumo

Neste capítulo foram apresentados alguns fundamentos matemáticos e conceitos em criptografia assimétrica.

CAPÍTULO 3

Esquemas em CL-PKC

Neste capítulo conceituamos sistemas sob o modelo de criptografia de chave pública sem certificado. Em seguida, descrevemos os esquemas alvo de implementação, além da motivação pela escolha de tais esquemas.

3.1 Nomenclatura

Inicialmente é necessário especificar a nomenclatura utilizada. A sigla CL-PKC (*Certificateless Public Key Cryptography*) denota qualquer criptossistema de chave pública sem certificado, como protocolos para criptografia e decriptografia, esquemas de assinatura, protocolo de troca de chaves, criptografia autenticada, entre outros.

A sigla CL-PKE (de *Certificateless Public Key Encryption*) é utilizada com o significado de esquema (ou protocolo) que especifica procedimentos de criptografia e decriptografia, no modelo de chave pública sem certificado. Já a sigla CL-PKS (de *Certificateless Public Key Signature*) é utilizada com o significado de esquema (ou protocolo) que especifica procedimentos para assinatura.

3.2 Definição de CL-PKE

Um esquema CL-PKE (*Certificateless Public Key Encryption*) é um protocolo de criptografia e decriptografia sob o modelo de criptografia de chave pública sem certificado, que envolve uma entidade confiável, denotada por KGC (*Key Generating Centre*, centro de geração de chaves), responsável por emitir chaves secretas parciais associadas aos destinatários de mensagens criptografadas.

Um CL-PKE é composto por cinco algoritmos((CHENG; COMLEY, 2005) e (AL-RIYAMI; PATERSON, 2003)), conforme a tabela 3.1.

Algoritmo	Entrada	Saída
inicializa	parâmetro k	*chave-mestra* e *params*
extrai	*params*, ID_A e a *chave-mestra*	d_A ($SecEsq$)
publica	*params*	t_A ($SecDir$) e N_A (chave pública)
cript	ID_A, N_A e $m \in \mathcal{M}$	$C \in \mathcal{C}$
decript	$C \in \mathcal{C}$, d_A e t_A	$m \in \mathcal{M}$, ou o sinal $\perp$

Tabela 3.1: Algoritmos para um esquema CL-PKE.

Descrição dos algoritmos da tabela 3.1:

inicializa. Algoritmo executado por KGC, que recebe um parâmetro de segurança k e devolve os parâmetros do sistema params e uma chave-mestra. Os parâmetros do sistema são públicos e incluem descrições do espaço de mensagens $\mathcal{M}$ e do espaço de textos cifrados $\mathcal{C}$. A chave-mestra é mantida em segredo por KGC.

extrai. Algoritmo que recebe params, chave-mestra e um identificador da entidade A, denotado por $ID_A \in \{0,1\}^*$ e devolve uma chave secreta parcial d_A, denotada por $SecEsq$. Este algoritmo também é executado por KGC.

publica. Algoritmo que recebe params e devolve uma informação secreta t_A, denotada por $SecDir$. Publica a chave pública N_A associada à entidade A e dependente de t_A. Algoritmo executado pela entidade A.

cript. Algoritmo que recebe params, um identificador ID_A e a chave pública N_A da entidade A, uma mensagem $m \in \mathcal{M}$ e devolve um texto cifrado $C \in \mathcal{C}$.

decript. Algoritmo que recebe params, $C \in \mathcal{C}$ e as chaves secretas d_A e t_A ($SecEsq$ e $SecDir$) e devolve uma mensagem $m \in \mathcal{M}$.

Os três valores secretos estão relacionados com os valores públicos, criando um vínculo automático entre cada chave de criptografia e o destinatário correspondente. A chave secreta parcial $SecEsq$ é calculada em função da chave-mestra de KGC e da identidade ID_A da entidade A. A chave secreta parcial $SecDir$, conhecida apenas pela respectiva entidade A, é usada para gerar a chave pública N_A.

Esse esquema não utiliza certificado digital, pois a chave de criptografia é composta pela identidade ID_A e pela chave pública N_A. Sem conhecer as chaves $SecEsq$ e $SecDir$ de A, não há como decriptografar um texto cifrado com ID_A, N_A. $SecEsq$ tem a garantia de KGC, por ser função da chave-mestra, e $SecDir$ autentica A, pois apenas a entidade A a conhece.

3.3 Definição de CL-PKS

Um esquema CL-PKS (*Certificateless Public Key Signature*) é um protocolo de assinatura sob o modelo de criptografia de chave pública sem certificado, que envolve uma entidade confiável, denotada por KGC (*Key Generating Centre*, centro de geração de chaves), responsável por emitir as chaves secretas parciais associadas aos usuários que assinam mensagens.

Um esquema CL-PKS é composto pelos seguintes algoritmos ((AL-RIYAMI; PATERSON, 2003) e (HUANG et al., 2005)):

inicializa. Algoritmo que recebe um parâmetro de segurança k e retorna uma lista de parâmetros do sistema chamada params e uma chave-mestra. Algoritmo executado por KGC.

gera-parcial. Algoritmo que recebe uma identidade ID_A, params e chave-mestra e retorna uma chave secreta parcial D_A. Algoritmo também executado por KGC.

gera-info-secreta. Algoritmo que gera uma informação secreta t_A para um usuário de identidade ID_A.

gera-secreta. Algoritmo que recebe uma identidade ID_A, uma chave secreta parcial D_A, uma informação secreta t_A e params, e retorna uma chave secreta (D_A, t_A), usada para assinar mensagens.

publica. Algoritmo que recebe uma identidade ID_A, uma informação secreta t_A e params, e retorna uma chave pública N_A para verificação de assinaturas.

assina. Algoritmo que recebe uma identidade ID_A, uma chave secreta (D_A, t_A), params, uma mensagem M, e retorna uma assinatura σ.

verifica. Algoritmo que recebe uma identidade ID_A, a chave pública N_A e params, uma mensagem M e uma assinatura σ e retorna um bit b; $b = 1$ significa assinatura válida, enquanto $b = 0$ significa assinatura inválida.

3.4 Esquema CL-PKE escolhido

Nesta seção descrevemos o esquema de criptografia e decriptografia baseada no modelo de criptografia de chave pública sem certificado de (GOYA, 2006), composto pelos seguintes algoritmos:

inicializa. Dado um parâmetro de segurança k, inteiros n e k_0, com $0 < k_0 < n$ e k_0 polinomial em n:

1. Gerar dois grupos cíclicos $\mathbb{G}_1$ e $\mathbb{G}_2$ de ordem prima $q > 2^k$ e um emparelhamento bilinear $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Escolher aleatoriamente um gerador $P \in \mathbb{G}_1^*$.
2. Escolher aleatoriamente $s \in \mathbb{Z}_q^*$ e calcular $P_{pub} := sP$.
3. Escolher três funções de *hash*
   $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$
   $H_2 : \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \to \{0,1\}^n$
   $H_3 : \{0,1\}^{n-k_0} \times \{0,1\}^{k_0} \to \mathbb{Z}_q^*$
4. Definir:
   $\mathcal{M} = \{0,1\}^{n-k_0}$, como espaço de mensagens;
   $\mathcal{C} = \mathbb{G}_1^* \times \{0,1\}^n$, como espaço de textos cifrados;
   s, como chave-mestra do sistema;
   params $:= \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, k_0, P, P_{pub}, H_1, H_2, H_3 \rangle$, como parâmetros do sistema.

extrai. Dados um identificador $ID_A \in \{0,1\}^*$, params e a chave-mestra s:

1. Calcular $Q_A := H_1(ID_A)$.
2. Devolver a chave secreta parcial $d_A := sQ_A$.

publica. Dado params, uma entidade A seleciona ao acaso uma informação secreta $t_A \in \mathbb{Z}_q^*$ e calcula sua chave pública $N_A := t_A P$. A guarda t_A em sigilo e publica N_A.

cript. Dados um texto $m \in \mathcal{M}$, uma identidade ID_A, params e a chave pública N_A:

1. Escolher aleatoriamente $\sigma \in \{0,1\}^{k_0}$
2. Calcular
   $r := H_3(m, \sigma)$
   $Q_A := H_1(ID_A)$
   $g^r := \hat{e}(P_{pub}, Q_A)^r$
   $f := rN_A$
3. Devolver o texto cifrado $C = \langle rP, (m \parallel \sigma) \oplus H_2(rP, g^r, f) \rangle$.

decript. Dados $C = \langle U, V \rangle \in \mathcal{C}$ e os valores secretos d_A e t_A:

1. Calcular
   $g' := \hat{e}(U, d_A)$
   $f' := t_A U$
   $(m \parallel \sigma) := V \oplus H_2(U, g', f')$
2. Desmembrar $(m \parallel \sigma)$ e calcular $r := H_3(m, \sigma)$
3. Se $U := rP$, devolver a mensagem m, senão devolver $\perp$.

3.5 Esquema CL-PKS escolhido

Nesta seção descrevemos o esquema de assinatura baseado no modelo de criptografia de chave pública sem certificado de (GOYA, 2006), composto pelos seguintes algoritmos:

inicializa. Dado um parâmetro de segurança k, o KGC:

1. Gera dois grupos cíclicos G_1, G_2, $G_1 \neq G_2$, e G_T de ordem prima $p > 2^k$ e um emparelhamento bilinear $e : G_1 \times G_2 \rightarrow G_T$.
   Escolhe aleatoriamente dois geradores $P \in G_1^*, Q \in G_2^*$, tais que $P = \psi(Q)$, onde $\psi()$ é um homomorfismo de G_2^* em G_1^*, eficientemente computável (isto é, de complexidade de tempo polinomial em k).
2. Calcula $g := e(P, Q) \in G_T$.
3. Escolhe aleatoriamente a chave-mestra $s \in Z_p^*$ e calcula $Q_{pub} := sQ$.
4. Escolhe funções de *hash*
   $H_1 : \{0,1\}^* \rightarrow Z_p^*$
   $H_2 : \{0,1\}^* \times \{0,1\}^* \times G_T \times G_T \rightarrow Z_p^*$.
5. Define:
   $\mathcal{M} = \{0,1\}^*$, como espaço de mensagens;
   $\mathcal{S} = G_1 \times Z_p^*$, como espaço de assinatura;
   s, como `chave-mestra` do sistema;
   `params` $:= \langle p, G_1, G_2, G_T, e, \psi, P, Q, Q_{pub}, g, H_1, H_2 \rangle$, como parâmetros do sistema.

gera-parcial. Dada uma identidade $ID_A \in \{0,1\}^*$, `params` e a `chave-mestra` s, o KGC:

1. Calcula $D_A := \frac{1}{H_1(ID_A)+s} P \in G_1^*$.
2. Entrega a chave secreta parcial D_A para a entidade cuja identidade é ID_A, por meio de um canal seguro.

gera-info-secreta. Dado `params`, a entidade A escolhe aletoriamente uma informação secreta $t_A \in Z_p^*$.

gera-secreta. Dados ID_A, D_A, t_A e `params`, a entidade A define e mantém em sigilo sua chave secreta de assinatura, formada pelo par $(D_A, t_A) \in G_1^* \times Z_p^*$.

publica. Dados ID_A, t_A e `params`, a entidade A calcula sua chave pública $N_A := g^{t_A} \in G_T$. A publica N_A.

assina. Dados `params`, uma mensagem $M \in \mathcal{M} = \{0,1\}^*$, uma identidade ID_A, a chave pública $N_A = g^{t_A}$ e a chave secreta de assinatura de A, formada pelo par (D_A, t_A), A assina M da seguinte forma:

1. Escolhe aleatoriamente $x \in Z_p^*$.
2. Calcula
$$\begin{cases} r := g^x \in G_T \\ h := H_2(M, ID_A, N_A, r) \in Z_p^* \\ S := (x + ht_A)D_A \in G_1 \end{cases}$$
3. A assinatura sobre M é $\sigma := (S, h) \in G_1 \times Z_p^*$.

verifica. Dados **params**, uma mensagem M, a assinatura $\sigma = (S, h)$, a identidade ID_A e a chave pública N_A, o algoritmo gera 1, aceitando σ como autêntica, se e somente se:

$$h = H_2(M, ID_A, N_A, r')$$
onde
$$r' := e[S, H_1(ID_A)Q + Q_{pub}](N_A)^{-h}$$

caso contrário, responde 0.

3.6 Motivação para escolha dos esquemas CL-PKE e CL-PKS

Apresentamos nesta seção dados e tabelas extraídos de (GOYA, 2006) que indicam a eficiência dos esquemas escolhidos em relação à performance e ao uso de espaço.

Os esquemas escolhidos são baseados em propriedades de emparelhamentos bilineares sobre grupos elípticos, sendo assim, a comparação dos esquemas se baseia na quantidade de cada uma das operações de maior custo computacional utilizadas. O cálculo de emparelhamentos são as operações de maior custo computacional, seguido, nessa ordem, por multiplicação escalar, multiexponenciação (multiplicação de pontos da curva elíptica), exponenciação de um ponto e cálculo de uma função de *hash*. As operações de soma e multiplicação de pontos são muito mais rápidas que emparelhamentos, exponenciações e multiplicação escalar e por isso são, em geral, desconsideradas nas análises.

Nas comparações apresentadas nesta seção, são contabilizadas as quantidades de cada tipo de operação utilizada pelos referidos esquemas. A tabela 3.2 lista as operações utilizadas nas comparações, por ordem de complexidade.

Com base na tabela 3.3, que apresenta, em ordem cronológica de publicação, alguns esquemas CL-PKE com suas respectivas quantidades de operações necessárias, verifica-se que o esquema CL-PKE de (GOYA, 2006) possui vantagens de performance em relação aos demais, pois requer apenas um cálculo de emparelhamento e menor quantidade de *hash*.

Em relação à escolha do esquema CL-PKS, é possível verificar na tabela 3.4, que o esquema CL-PKS de (GOYA, 2006) deve ser o mais rápido tanto para assinar mensagens quanto para verificar assinaturas.

E	Emparelhamento bilinear
X	eXponenciação nos grupos multiplicativos
M	Multiplicação escalar nos grupos aditivos
m	multiplicação de pontos nos grupos multiplicativos
S	Soma de pontos nos grupos aditivos
H	cálculo de *Hash*

Tabela 3.2: Operações sobre grupos bilineares, em ordem de complexidade (e legenda)

Esquema CL-PKE	Criptografia				Decriptografia			
	E	X	M	H	E	X	M	H
(AL-RIYAMI; PATERSON, 2003)	3	1	1	4	1	0	1	3
(AL-RIYAMI; PATERSON, 2005)	1	1	2	5	1	0	2	4
(CHENG; COMLEY, 2005)	1	1	2	4	1	0	2	3
(GOYA, 2006)	1	1	2	3	1	0	2	2

Tabela 3.3: Quantidade de operações sobre grupos nos esquemas CL-PKE

Sobre o uso de espaço, podemos verificar as tabelas 3.5 e 3.6. Na tabela 3.5, que lista os tamanhos de chave pública, de texto legível e de texto cifrado requeridos pelos esquemas, verifica-se que o esquema de (GOYA, 2006), potencialmente, produz cifras menores que os demais. Já pela tabela 3.6, que lista o espaço de mensagens assinadas, verifica-se que o esquema de (GOYA, 2006) gera assinaturas de menor tamanho que o esquema de (ZHANG et al., 2006) e, dependendo do valor de n (o comprimento da mensagem assinada relativamente ao parâmetro de segurança k) também gera assinaturas menores que as geradas pelo esquema de (HUANG et al., 2005).

Esquema CL-PKS	Assinatura						Verificação					
	E	X	M	m	S	H	E	X	M	m	S	H
(AL-RIYAMI; PATERSON, 2003)	1	0	3	0	1	1	4	1	0	1	0	1
(HUANG et al., 2005)	2	0	2	0	1	1	4	1	0	1	0	2
(ZHANG et al., 2006)	0	0	3	0	2	2	4	0	0	2	0	3
(GOYA, 2006)	0	1	1	0	0	1	1	1	1	1	1	2

Tabela 3.4: Quantidade de operações sobre grupos nos esquemas CL-PKS

Esquema CL-PKE	Tamanhos (em bits)		
	chave pública	mensagem legível	mensagem cifrada
(AL-RIYAMI; PATERSON, 2003)	$2g$	n	$g + 2n$
(AL-RIYAMI; PATERSON, 2005)	g	n	$g + 2n$
(CHENG; COMLEY, 2005)	g	n	$g + 2n$
(GOYA, 2006)	g	$m - k_0$	$g + n\imath$

Tabela 3.5: Tamanhos requeridos na representação de elementos nos esquemas CL-PKE (em que g é o tamanho em bits para representar um ponto de $\mathbb{G}_1$)

Esquema CL-PKS	Espaço de Assinatura	Espaço de Chave Pública
(AL-RIYAMI; PATERSON, 2003)	$\mathbb{G}_1 \times \{0,1\}^n$	$\mathbb{G}_1$
(HUANG et al., 2005)	$\mathbb{G}_1 \times \{0,1\}^n$	$\mathbb{G}_1$
(ZHANG et al., 2006)	$\mathbb{G}_1 \times \mathbb{G}_1$	$\mathbb{G}_1$
(GOYA, 2006)	$G_1 \times \mathbb{Z}_p^*$	G_T

Tabela 3.6: Espaços de assinatura e de chave pública dos esquemas CL-PKS

3.7 Resumo

Neste capítulo foram apresentadas as definições de sistemas CL-PKE e CL-PKS e comparações de desempenho entre esquemas CL-PKC (tanto CL-PKE quanto CL-PKS) envolvendo aspectos de desempenho e de uso de espaço.

CAPÍTULO 4

Implementação dos esquemas em CL-PKC

Neste capítulo descrevemos os passos adotados na implementação dos esquemas escolhidos sob o modelo de criptografia de chave pública sem certificado, desde a escolha das curvas, emparelhamentos e demais parâmetros necessários até rotinas de programação, estrutura de dados, formato de entrada e saída de dados e equipamentos utilizados.

4.1 Hardware/Sistema Operacional

Na implementação, foi utilizado um micro-computador AMD Duron 1.8Ghz, com 768MB de memória RAM, com dual-boot Windows XP e Linux Debian 4.0.

4.2 Ambiente de desenvolvimento

A implementação foi desenvolvida na linguagem de programação C/C++, utilizando a biblioteca MIRACL. MIRACL (Multiprecision Integer and Rational Arithmetic C/C++ Library) é uma biblioteca com suporte para as primitivas criptográficas, incluindo criptografia baseada em curvas elípticas e emparelhamentos e é amplamente utilizada no meio acadêmico, além de ser livre para uso sem fins lucrativos: educacional ou qualquer uso não-comercial ((SCOTT, 2003)).

O compilador utilizado no windows foi o Microsoft Visual C++ 6.0, onde testamos a implementação. No Linux foi utilizado o gcc/g++, de modo que é possível executar as implementações e a biblioteca MIRACL em ambos os sistemas.

4.3 Escolha das curvas elípticas

A escolha de curvas elípticas está relacionada, entre outros fatores, ao grau de imersão da curva. Um alto grau de imersão é importante para que tenhamos um elevado nível de segurança. No entanto, com grau de imersão pequeno é possível alcançar um desempenho melhor. Buscamos, portanto, curvas cujo grau de imersão é grande o suficiente para manter um nível adequado de segurança, mas pequeno o bastante para permitir que certas operações sejam efetuadas eficientemente.

As curvas elípticas ordinárias, em geral, possuem grau de imersão enorme ((BALASUBRAMANIAN; KOBLITZ, 1998) e (MIYAJI; NAKABAYASHI; TAKANO, 2001)) e as curvas supersingulares possuem grau de imersão menor ou igual a 6 ((MENEZES; VANSTONE; OKAMOTO, 1991)).

4.3.1 Curvas MNT e BN

(MIYAJI; NAKABAYASHI; TAKANO, 2001) mostraram como gerar curvas ordinárias com grau de imersão pré-determinado. Tais curvas são chamadas de curvas MNT e tem o grau de imersão k pertencente ao conjunto 3, 4, 6. O método usado para construir essas curvas é o método de Multiplicação Complexa(*Complex Multiplication - CM*). Mais informações sobre CM podem ser encontradas em (BLAKE I.; SEROUSSI, 1999) e (IEEE/P1363, 2004).

Outros algoritmos para a construção de curvas foram propostos, como por exemplo o algoritmo Dupont-Enge-Morain ((DUPONT; ENGE; MORAIN, 2002)) e o algoritmo de Galbraith, chamado de algoritmo "folclórico"((BARRETO, 2003)).

Em (BARRETO; NAEHRIG, 2005) foi mostrado como construir curvas com grau de imersão k=12 também utilizando o método de Multiplicação Complexa. Além disso, os autores mostraram como otimizar os cálculos sobre F_{q^k} utilizando cálculos em F_{q^2} ao invés de $F_{q^{12}}$, ganhando em desempenho.

Escolhemos para implementação, portanto, curvas das famílias MNT, com grau de imersão k=6 e, da família BN, onde k=12. Esses tipos de curvas permitem algumas otimizações, como será visto na seção seguinte.

4.4 Cálculo do emparelhamento e algumas otimizações

Nesta seção, mostraremos como calcular o emparelhamento de Tate. Partiremos do algoritmo básico e então apresentaremos algumas otimizações. Umas das otimizações para cálculo de emparelhamento, como veremos, depende da família de curvas utilizadas. Nesses casos, é possível realizar um truncamento no loop principal do emparelhamento de Tate. Tal emparelhamento recebe o nome de emparelhamento Ate.

4.4.1 Algoritmo básico: Miller

Define-se o emparelhamento de Tate ($\hat{e} : G_1 \times G_2 \rightarrow G_3$) entre pontos linearmente independentes P e Q da curva $E(F_q^k)$, sendo k o grau de imersão da curva considerada. O primeiro argumento deve ser de ordem r. Na prática, G_1 é um subgrupo do grupo aditivo de pontos de uma curva elíptica sobre F_q, ou seja, P é definido sobre a curva $E(F_q)$. G_2 é um subgrupo do grupo aditivo de pontos de uma curva elíptica sobre F_q^k, isto é, está definido em $E(F_q^k)$. E G_3 é um subgrupo do grupo multiplicativo de um corpo finito, isto é, o emparelhamento $\hat{e}(P, Q)$ assume valores no corpo finito estendido F_q^k.

Mais formalmente, temos:

$\hat{e} : E(F_q)[r] \times E(F_q^k) \rightarrow F_q^k$, onde q é um primo longo e r é a ordem da curva.

Para implementações práticas, q é da ordem de 2^{160}.

Para o cálculo do emparelhamento de Tate, utilizamos o algoritmo de Miller((BARRETO et al., 2002)), descrito abaixo:

Entrada: $P \in E(F_q), Q \in E(F_q^k)$.
Saída: $\hat{e}(P, Q)$.
1. T := P, f := 1
2. for i := $\lfloor lg(r) \rfloor - 1$ downto 0 do
    3. $f := f^2 \cdot l_{T,T}(Q)/v_{2T}(Q)$
    4. T := 2T
    5. if $r_i = 1$ then
        6. $f := f \cdot l_{T,P}(Q)/v_{T+P}(Q)$
        7. T := T + P
8. $f := f^{(p^k-1)/r}$

Onde $l_{A,B}(Q)$, chamada função de linha, é definida por:

$l_{A,B}(Q) = (y_q - y_j) - \lambda_j(x_q - x_j)$, e $v_{A+B}(Q)$ é definida por:

$v_{A+B}(Q) = (x_q - x_{j+1})$, onde A tem coordenadas (x_j, y_j), A+B tem coordenadas (x_{j+1}, y_{j+1}) e o ponto Q tem coordenadas (x_q, y_q).

4.4.2 Otimização 1: Eliminação do Denominador

Uma importante otimização, chamada de *Eliminação de Denominadores* é possível para curvas supersingulares e curvas ordinárias com grau de imersão par. Como escolhemos curvas MNT e BN para implementação, ambas com grau de imersão par, podemos nos beneficiar dessa otimização. Essa otimização acelera em cerca de 125 vezes a velocidade do cálculo do emparelhamento ((BARRETO et al., 2002), (BARRETO; LYNN; SCOTT, 2003) e (BARRETO,

2003)).

O novo algoritmo, é descrito abaixo:

Entrada: $P \in E(F_q), Q \in E(F_q^k)$.
Saída: $\hat{e}(P,Q)$.
1. T := P, f := 1
2. for i := $\lfloor lg(r) \rfloor - 1$ downto 0 do
    3. $f := f^2 \cdot l_{T,T}(Q)$
    4. T := 2T
    5. if $r_i = 1$ then
        6. $f := f \cdot l_{T,P}(Q)$
        7. T := T + P
8. $f := f^{(p^k-1)/r}$

4.4.3 Otimização 2: Emparelhamento Ate

Outro emparelhamento, derivado de otimizações sobre o emparelhamento de Tate, foi proposto em (HESS; SMART; VERCAUTEREN, 2006). Tal emparelhamento, chamado de emparelhamento Ate (pronuncia-se *eight*), é implementado utilizando o twist da curva escolhida. Calcula-se o emparelhamento percorrendo os bits de $s = t - 1$ ao invés de percorrer os bits de r no loop, ou seja, o loop é percorrido menos vezes e, portanto, com melhor desempenho.

O pseudo-código do emparelhamento Ate segue abaixo:

Entrada: $P \in E'(F_q^k), Q \in E(F_q)$.
Saída: $\hat{e}(P,Q)$.
1. T := P, f := 1, s := t - 1
2. for i := $\lfloor lg(s) \rfloor - 1$ downto 0 do
    3. $f := f^2 \cdot l_{T,T}(Q)$
    4. T := 2T
    5. if $s_i = 1$ then
        6. $f := f \cdot l_{T,P}(Q)$
        7. T := T + P
8. $f := f^{(p^k-1)/r}$

O Twist de uma curva definida por $E : y^2 = x^3 + ax + b$ é a curva $E' : y2 = x3 + a'x + b'$ com $a' = v^2a$ e $b' = v^3b$ para algum não-resíduo quadrático $v \in F_q$. As ordens dos grupos de pontos racionais dessas curvas satisfazem a relação $\#E(F_q) + \#E'(F_q) = 2q + 2$ ((BLAKE I.; SEROUSSI, 1999)).

4.4.4 Otimizações dependentes do esquema

Além das otimizações na forma como se calculam os emparelhamentos, pode-se ganhar desempenho na maneira como se implementa um esquema em si. Em (SCOTT, 2007) e (SCOTT, 2005) estão descritas otmizações relacionadas ao produto de emparelhamentos, uso de coordenadas projetivas ao invés de coordenadas afins e ainda sobre o uso de pré-computação. Tais otimizações são descritas à seguir.

Em muitos sistemas criptográficos baseados em emparelhamentos, é necessário calcular $\hat{e}(P, Q)$ com P fixo (por exemplo, se P for o ponto base da curva) ou usado repetidamente (por exemplo, uma chave pública). Nesses casos, a multiplicação escalar do algoritmo de Miller pode ser executado apenas uma vez para pré-calcular os coeficientes das equações de linha. O aumento de desempenho resultante desta técnica é mais proeminente em característica p > 3, e pode atingir cerca de 40% ((BARRETO, 2003)).

Em esquemas onde é necessário calcular um produto de emparelhamentos, como, por exemplo, $\hat{e}(P, Q) \times \hat{e}(R, S)$ alterar o algoritmo de emparelhamento de forma a calcular diretamente o produto dos dois, ao invés de calcular separadamente.

Esquemas que utilizam muitas operações de adição e duplicação de ponto sobre as curvas elípticas podem se beneficiar do uso de coordenadas projetivas ao invés de coordenadas afins, pois adição e duplicação de ponto sobre $E(F_p)$ requerem inversão modular mod p, que é uma operação custosa((IEEE/P1363, 2004)).

4.5 Funções de Hash

Em (MENEZES; VANSTONE; OORSCHOT, 1996) e (BARRETO, 2003), uma função de hash de tamanho n é definida formalmente como uma função $H : \{0, 1\}^* \rightarrow 0, 1^n$ que satisfaz as seguintes propriedades:

- *Resistência ao cálculo de primeira pré-imagem*: Dada uma cadeia binária h de comprimento n, é computacionalmente intratável encontrar uma mensagem M tal que H(M) = h.
- *Resistência ao cálculo de segunda pré-imagem*: Dada uma cadeia binária h de comprimento n e uma mensagem M tal que H(M) = h, é computacionalmente intratável encontrar outra mensagem M_2 tal que $H(M_2) = h$.
- *Resistência a colisões*: É computacionalmente intratável encontrar duas mensagens M e M_2 tais que $H(M) = H(M_2)$, independentemente do valor de H(M).

Informalmente, uma função de hash é uma função computacionalmente unidirecional, resistente a colisões, que mapeia (isto é, calcula de forma rápida, segura e única) cadeias

binárias de tamanho arbitrário em cadeias binárias de tamanho fixo n, adequadamente curtas, chamadas de resumo. Como exemplos de famílias de funções de hash para aplicações criptográficas podemos citar: SHA-1 e SHA-2, definidos em (NIST/FIPS186-2, 2000).

A seguir, comentamos as funções de Hash usadas para a implementação dos esquemas propostos. Adotamos a seguinte nomenclatura: as 3 funções de Hash do esquema CL-PKE-Proposto foram chamadas de H1, H2 e H3 respectivamente. As 2 funções de Hash do esquema CL-PKS-Proposto foram chamadas de H1sign e H2sign respectivamente.

4.5.1 Hash H1

A função Hash H1 é definida da seguinte forma:

$H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$

Em outras palavras, H1 recebe de entrada uma cadeia de bits de tamanho arbitrário e responde com um ponto da curva elíptica.

Para implementar essa função, utilizamos uma outra função chamada de *String_to_Big*, que calcula o hash (com uma função de hash tradicional, no caso foi utilizado a função de hash sha256) da string de tamanho arbitrário recebida de entrada utilizando esses bits para gerar um número inteiro de tamanho escolhido.

A função H1 executa então um loop até conseguir um ponto na curva.

Segue abaixo um pseudo-código para melhor entendimento. O código-fonte encontra-se anexado a este trabalho.

H1(IDentificador ID) {

1. Gera um inteiro X com os bits do hash do identificador ID;
2. Se $y^2 \equiv x^3 + ax + b(mod q)$ devolva o ponto (x, y)
   2.1. senão X := X+1 e volta ao passo 2.

}

4.5.2 Hash H2

A função Hash H2 é definida da seguinte forma:

$H_2 : \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \{0,1\}^n$

Para implementá-la, utilizamos uma outra função chamada de *Big_to_String*, que faz cálculos modulares para obter os dígitos menos significativos de um número para gerar a cadeia de tamanho n requerida.

Além disso, como a biblioteca MIRACL implementa a estrutura de dados ECn (que

armazena um ponto da curva) e a estrutura ZZnK (usada para armazenar a estrutura F_{q^K}) como estruturas compostas de estruturas *Big*(usadas para armazenar inteiros de tamanho arbitrário), seguimos a estratégia de desmembrar as respectivas estruturas *Big* e realizar a soma das mesmas, resultando em um único inteiro *Big*, que é fornecido para a função auxiliar *Big_ to_ String*.

Segue abaixo um pseudo-código da função de Hash H2.

H2(Ponto P, Ponto Q, Ponto P2) {

1. Inicia uma variável auxiliar inteira Big AUX:=0;
2. Extrai as estruturas Big do Ponto P;
3. Adiciona as estruturas Big do passo 2 à variável AUX;
4. Extrai as estruturas Big do Ponto Q;
5. Adiciona as estruturas Big do passo 4 à variável AUX;
6. Extrai as estruturas Big do Ponto P2;
7. Adiciona as estruturas Big do passo 6 à variável AUX;
8. Retorna Big_to_String(AUX);

}

4.5.3 Hash H3

A função Hash H3 é definida da seguinte forma:

$H_3 : \{0,1\}^{n-k_0} \times \{0,1\}^{k_0} \rightarrow \mathbb{Z}_q^*$

Ou seja, H3 tem como entrada duas cadeias de bits, sendo uma de tamanho $n - k_0$ e outra de tamanho k_0 e responde com um inteiro menor que q.

Para esta função, realizamos uma concatenação das duas cadeias de bits de entrada e então novamente utilizamos a função auxiliar *String_ to_ big*, que retorna um número inteiro de tamanho escolhido, neste caso, com valor menor que q.

4.5.4 Hash H1sign

A função Hash $H1sign$ é definida da seguinte forma:

$H_1 : \{0,1\}^* \rightarrow Z_p^*$

Isto é, $H1sign$ recebe de entrada uma cadeia de bits de tamanho arbitrário e responde com um inteiro menor que p.

Esta função nada mais é que executar apenas a função auxiliar *String_ to_ big*.

4.5.5 Hash H2sign

A função Hash *H2sign* é definida da seguinte forma:

$H_2 : \{0,1\}^* \times \{0,1\}^* \times G_T \times G_T \rightarrow Z_p^*$.

Ou seja, *H2sign* recebe duas cadeias de bits de tamanho arbitrário e dois pontos em F_{q^k} e retorna um inteiro menor que p.

Mais uma vez, faremos uso das funções auxiliares e das facilidades das estruturas da biblioteca, como mostrado no pseudo-código abaixo.

```
H2sign(Mensagem M, Identificador ID, Ponto Q, Ponto Q2) {
    1. Calcula AUX := String_to_Big(Mensagem M);
    2. Calcula AUX := AUX + String_to_Big(Identificador ID);
    3. Extrai as estruturas Big do Ponto Q;
    4. Adiciona as estruturas Big do passo 3 à variável AUX;
    5. Extrai as estruturas Big do Ponto Q2;
    6. Adiciona as estruturas Big do passo 5 à variável AUX;
    7. Retorna AUX;
}
```

4.6 Benefícios do uso da biblioteca MIRACL

Aqui citamos algumas vantagens proporcionadas pela biblioteca MIRACL(e seus exemplos - incluindo de emparelhamentos - fornecidos pelo autor da Biblioteca Michael Scott) para abstrair o background matemático necessário para implementação de esquemas de criptografia, principalmente esquemas sobre curvas elípticas e/ou baseados em emparelhamentos, como: aritmética (soma de pontos, duplicação de pontos, multiplicação de um escalar por um ponto entre outras) sobre curvas elípticas; aritmética sobre inteiros de tamanho arbitrário(incluindo inverso de um inteiro mod q); geração de números aleatórios(pseudo-aleatórios) de tamanho arbitrário; estruturas de dados para cálculo e armazenamento de pontos sobre curvas elípticas; estruturas de dados para cálculo em F_{q^k}, incluindo inverso sobre F_{q^k}, que é necessário para implementar o esquema de assinatura proposto; manipulação da base utilizada para entrada e saída de dados, permitindo escolher a base 16, que facilita a entrada e saída de dados em hexadecimal de forma transparente; além de mecanismos (algoritmos) internos otimizados, como multiplicação de Karatsuba e inúmeros outros.

4.7 Exatidão dos cálculos

Aqui ressaltamos que para validar a exatidão dos cálculos (que utilizam valores com grande quantidade de bits e que, portanto, inviabilizam o "cálculo manual"), como cálculo do emparelhamento e de inversos, por exemplo, partimos para a verificação das propriedades dos cálculos.

Para validar o cálculo do emparelhamento, verificamos as propriedades da bilinearidade, fazendo os dois testes a seguir:

1. $\forall x$ inteiro, $0 < \mathrm{x} < \mathrm{r}$, onde r é a ordem da curva, $\hat{e}(x \cdot P, Q) = \hat{e}(P, x \cdot Q) = \hat{e}(P, Q)^x$;

2. $\hat{e}(P, Q)^r = 1$, onde r é a ordem da curva, pois $\hat{e}(P, Q)^r = \hat{e}(r \cdot P, Q) = \hat{e}(\mathfrak{O}, Q) = 1$

Para validar o cálculo do inverso de um elemento α em F_{q^k}, denotado por α^{-1} realizamos o cáculo:

$\alpha \cdot \alpha^{-1} \equiv 1$.

4.8 Estrutura do programa desenvolvido

Para uma execução mais intuitiva, o programa foi desenvolvido como uma "calculadora", no sentido de oferecer um menu para escolha da curva elíptica (MNT ou BN), escolha do emparelhamento (Tate ou Ate) e escolha do esquema (criptografia ou assinatura) para cálculo.

A entrada de dados dos parâmetros da curva, no caso de curvas MNT é feito através de um arquivo texto. O arquivo de entrada com os parâmetros da curva podem ser calculados com um utilitário incluído na MIRACL, chamado cm.exe. Para curvas BN, apenas um parâmetro é introduzido no código-fonte diretamente, pois os demais parâmetros são em função deste.

A saída de dados é feita diretamente da saída padrão em hexadecimal (base 16). Facilidade esta oferecida pela MIRACL.

4.9 Alteração de parâmetros

Pelo modo como está organizado o programa, facilmente se pode alterar os parâmetros utilizados, fornecendo novas curvas para cálculo. Além disso, como já possuímos vários tipos de funções de Hash implementadas, além de dois emparelhamentos, outros esquemas baseados em emparelhamentos também podem ser implementados de forma mais rápida.

4.10 Resumo

Neste capítulo descrevemos o processo que seguimos na implementação e demos dicas sobre possíveis alterações que podem ser feita no nosso código para execução de novos testes ou implementação de novos esquemas.

CAPÍTULO 5

Conclusões

Neste capítulo apresentamos os resultados e as contribuições do nosso trabalho, além dos possíveis trabalhos futuros relacionados.

5.1 Resultados e contribuições

Com a implementação dos esquemas propostos, foi possível um melhor entendimento das dificuldades em implementar de fato sistemas criptográficos baseados em emparelhamentos. Galbraith ((GALBRAITH; PATERSON; SMART, 2006)) afirma que muitos pesquisadores da área tendem a supor que encontrar parâmetros com determinadas características para esquemas com emparelhamentos seja algo fácil, e, portanto, tratam o assunto como uma "caixa-preta"que resolve emparelhamentos com as características desejadas. Vimos com o trabalho, que a escolha dos parâmetros certos permite determinar melhoras no desempenho dos esquemas.

Além de um estudo sobre desempenho de diferentes curvas e emparelhamentos - onde o melhor desempenho conseguido foi com curvas BN e o emparelhamento Ate -, contribuímos com a implementação inédita dos esquemas descritos neste trabalho.

5.2 Trabalhos Futuros

Como trabalhos futuros, podemos sugerir:

1. Estudo sobre outras famílias de curvas, como as citadas no capítulo 4, para verificar também a diferença de desempenho para novas famílias de curvas;
2. Adaptação dos nossos código para a implementação de outros esquemas criptográficos,

para comparar o desempenho não de diferentes parâmetros, mas de diferentes esquemas de criptografia e assinatura sem certificado;

3. Estudo da viabilidade para transformar o esquema de criptografia proposto em um protocolo de trocas de chaves seguro, substituindo a operação de "OU-exclusivo"por um esquema de criptografia simétrico.

5.3 Resumo

Neste capítulo apresentamos nossas contribuições e resultados alcançados. Apresentamos também algumas sugestões para futuros trabalhos relacionados a esta pesquisa.

Conceitos em Álgebra

Neste apêndice são descritos conceitos da Álgebra: grupo e corpo.

A.1 Grupo

Um *grupo* é um conjunto não vazio $\mathbb{G}$, dotado de uma operação binária $\circ : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$, que satisfaz as seguintes propriedades (KOBLITZ, 1994):

- Elemento Identidade: $\exists n \in \mathbb{G} : \forall a \in \mathbb{G} : a \circ n = n \circ a = a$;
- Elemento Inverso: $\forall a \in \mathbb{G} : \exists \bar{a} \in \mathbb{G} : a \circ \bar{a} = n$;
- Associatividade: $\forall a, b, c \in \mathbb{G} : (a \circ b) \circ c = a \circ (b \circ c)$;
- Fechamento: $\forall a, b \in \mathbb{G}, a \circ b \in \mathbb{G}$.

Um *grupo abeliano* (também chamado *grupo comutativo*) é um grupo com a propriedade adicional:

- Comutatividade: $\forall a, b \in \mathbb{G} : a \circ b = b \circ a$.

Quando o símbolo $\circ$ é substituído pelo sinal de adição "+", dizemos que o grupo é um *grupo aditivo*. Nesse caso, o elemento neutro (identidade) é escrito como 0 e o inverso de a é escrito como $-a$. Define-se *multiplicação por escalar*, para um inteiro positivo λ e $a \in \mathbb{G}$, como sendo a operação $\lambda a = a + a + \cdots + a$, onde a soma consiste de λ termos.

Quando o símbolo $\circ$ é substituído pelo sinal de multiplicação ".", dizemos que o grupo é um *grupo multiplicativo*. Nesse caso, o elemento identidade é escrito como 1 e o inverso de

a é escrito como a^{-1}. Define-se *potenciação* para um inteiro positivo λ e $a \in \mathbb{G}$, como sendo a operação $a^{\lambda} = a.a. \ldots .a$, com λ fatores.

Quando o número de elementos de $\mathbb{G}$ é finito, esse número é chamado *ordem* de $\mathbb{G}$.

Um grupo aditivo é dito *grupo cíclico* se existe um elemento $P \in \mathbb{G}$ tal que qualquer elemento $Q \in \mathbb{G}$ pode ser escrito como múltiplo escalar de P, isto é, $Q = \lambda P$, para algum inteiro λ. Um elemento P com tal propriedade é chamado *gerador* de $\mathbb{G}$.

A.2 Corpo

Um *corpo* é uma estrutura algébrica que consiste de um conjunto K e duas operações binárias (KOBLITZ, 1994):

- (Adição) $+ : K \times K \rightarrow K$
- (Multiplicação) $\cdot : K \times K \rightarrow K$

satisfazendo as seguintes propriedades:

- $(K, +)$ é um grupo abeliano, com identidade aditiva denotada por 0.
- $(K^*, \cdot)$, onde $K^* = K \backslash \{0\}$, é um grupo abeliano, com identidade multiplicativa denotada por 1.
- $0 \neq 1$.
- Distributividade: $x \cdot (y + z) = x \cdot y + x \cdot z, \forall x, y, z \in K$

A *característica de um corpo* K, denotada por $char(K)$, é o número de vezes que a identidade multiplicativa de K deve ser adicionada a si mesma para que o resultado seja igual à identidade aditiva. Se o resultado nunca for a identidade aditiva, define-se $char(K) = 0$.

Dizemos que existe um *isomorfismo* entre dois corpos quando há uma função bijetora entre eles que mapeia os elementos identidade e os inversos de um corpo no outro.

Alguns resultados úteis (KOBLITZ, 1994):

- A característica de um corpo é sempre um número primo ou é zero.
- Se $char(K) = 0$, então K contém um subconjunto isomorfo a $\mathbb{Q}$.
- Se $char(K) = p$, para algum primo p, então K contém um subconjunto isomorfo a $\mathbb{Z}_p$.

- O número de elementos de um corpo finito é sempre na forma p^m, onde p é a característica do corpo e m, um inteiro positivo.
- Para cada primo p e cada inteiro m, existe um único corpo finito (a menos de isomorfismo) com p^m elementos, denotado por $GF(p^m)$ ou por $\mathbb{F}_{p^m}$.
- Vale observar que $GF(p)$ é isomorfo a $\mathbb{Z}_p$. O corpo $GF(p^m)$ consiste dos polinômios com coeficientes em $\mathbb{Z}_p$ e grau menor que m, módulo um polinômio redutível de grau m. Para aplicações em criptografia (e computacionais em geral), os corpos $GF(2^m)$ e $GF(p)$ são os de maior interesse.

Código-fonte

Neste apêndice fornecemos o código-fonte desenvolvido.

```
/*
   Implementação de esquemas de criptografia e assinatura sob
   o modelo de criptografia de chaves públicas sem certificado.

   Compilar:
   cl /O2 /GX /DZZNS=8 principal.cpp zzn12.cpp zzn6a.cpp ecn2.cpp zzn2.cpp big.cpp zzn
*/

#include <iostream> #include <fstream> #include <string.h> #include
"ecn.h" #include <ctime> #include "ecn2.h" #include "zzn12.h"
#include <time.h>

using namespace std;

Miracl precision(8,0);


#define HASH_LEN 32 #define N 160   // Multiplo de 8. Representa o
numero de bits retornado por H2 #define K0 32


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

```
void set_frobenius_constant(ZZn12 &X) {
    ZZn12 x;
    Big p=get_modulus();
    x.seti((ZZn6)1);
    X=pow(x,p);
}

ZZn12 line(ECn& A,ECn& C,ZZn& slope,ZZn2& Qx,ZZn2& Qy) {
     ZZn12 w;
     ZZn6 nn,dd;
     ZZn x,y,z,t;

     dd.set1(Qy);
     extract(A,x,y);
     nn.set(slope*x-y,-slope*Qx);
     w.set(nn,dd);
     return w;
}


ZZn12 g(ECn& A,ECn& B,ZZn2& Qx,ZZn2& Qy) {
    ZZn  lam;
    ZZn6 u;
    ZZn12 r;
    big ptr;
    ECn P=A;

    ptr=A.add(B);
    if (ptr==NULL)    return (ZZn12)1;
    else lam=ptr;

    if (A.iszero())   return (ZZn12)1;
    r=line(P,A,lam,Qx,Qy);
    return r;
}

BOOL tate(ECn& P,ZZn2& Qx,ZZn2& Qy,Big &x,ZZn12 &X,ZZn6& res) {
    ECn A;
    int i,nb;
    Big n,p,q,t,rho;
    ZZn12 w,r,a,b,c,rp;

    p=36*pow(x,4)-36*pow(x,3)+24*x*x-6*x+1;
```

```
    t=6*x*x+1;
    q=p+1-t;

    A=P;
    n=q-1;
    nb=bits(n);
    r=1;
    for (i=nb-2;i>=0;i--)
    {
        r*=r;
        r*=g(A,A,Qx,Qy);

        if (bit(n,i))
            r*=g(A,P,Qx,Qy);
    }
    if (A != -P || r.iszero()) return FALSE;
    w=r;
    r.conj();
    r/=w;
    w=r;
    r.powq(X); r.powq(X);
    r*=w;    // r^[(p^6-1)*(p^2+1)]
    if (x>0) a=pow(r,6*x-5);
    else     a=inverse(pow(r,5-6*x));
    b=a; b.powq(X);
    b*=a;
    rp=r; rp.powq(X);
    a*=b;
    w=r; w*=w; w*=w;
    a*=w;
    c=rp*r; w=c; w*=w; w*=w; w*=w; w*=c;
    a*=w; w=(rp*rp);
    rp.powq(X);
    w*=(b*rp);
    c=pow(w,x);
    r=w*pow(c,6*x);  //    r=pow(w,6*x*x+1);
    rp.powq(X); a*=rp;
    r*=a;
    res= real(r);
    return TRUE;
}

BOOL ecap(ECn& P,ECn2& Q,Big& x,ZZn12 &X,ZZn6& r) {
```

```
    BOOL Ok;
    ECn PP=P;
    ZZn2 Qx,Qy;

    normalise(PP);
    Q.get(Qx,Qy);

    Ok=tate(PP,Qx,Qy,x,X,r);

    if (Ok) return TRUE;
    return FALSE;
}


Big String_to_Big(char *string) {
    Big h,p;
    char s[HASH_LEN];
    int i,j;
    sha256 sh;

    shs256_init(&sh);
    for (i=0;;i++)
    {
        if (string[i]==0) break;
        shs256_process(&sh,string[i]);
    }
    shs256_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;
}

void concatena(char *string1, char *string2, char *result) { //
Concatena duas strings
    int i, len1, len2;
```

```
    len1 = strlen(string1);
    len2 = strlen(string2);

    for(i=0; i<len1; i++) result[i] = string1[i];
    for(i=0; i<len2; i++) result[i+len1] = string1[i];
    result[i+len1] = '\0';

    cout << "Str1= " << strlen(string1) << endl;
    cout << "Str2= " << strlen(string2) << endl;
    cout << "Str1+Str2= " << strlen(result) << endl;
}


Big H3(char *string1, char *string2) { // Une duas strings e retorna
um big
    Big aux;
    char temp[(N/8)+1];

    concatena(string1, string2, temp);
    aux = String_to_Big(temp);
    return aux;
}

Big H1_sign(char *ID) {
    Big aux=String_to_Big(ID);
    return aux;
}

ECn2 Big_to_ECn2(Big a) {
    Big aux = a;
    ECn2 S;
    ZZn2 X;

    forever
    {
        aux+=1;
        X.set((ZZn)0,(ZZn)aux);
        if (!S.set(X)) continue;
        break;
    }
    return S;
}
```

```
ECn2 H1(char *ID) {
    int i;
    ECn2 S;
    ZZn2 X;

    Big x0=String_to_Big(ID);

    forever
    {
        x0+=1;
        X.set((ZZn)0,(ZZn)x0);
        if (!S.set(X)) continue;
        break;
    }
    return S;
}

// Gera uma string com os bits finais de um Big
void Big_to_String(Big a, int tam, char *result) {
    int i, temp;
    int i256 = 256;
    Big aux = a;

    for(i=0; i<tam; i++)
    {
        temp = aux%i256;
        result[i] = (char)temp;
        aux = aux/256;
    }
    result[i] = '\0';
}

void H2(ECn& P,ZZn6& Q,ECn& P2, char *result) {
    Big x, y, aux;
    ZZn2 Qx, Qy, Qz;

    Q.get(Qx,Qy, Qz);
    Qx.get(x, y);
    aux = x+y;
    Qy.get(x, y);
    aux += x+y;
    Qz.get(x, y);
```

```
    aux += x+y;
    P.get(x, y);
    aux += x+y;
    P2.get(x, y);
    aux += x+y;

    Big_to_String(aux, (int)N/8, result);
}

Big H2_sign(char *string1, char *string2, ZZn6& Q1, ZZn6& Q2) {
    Big x, y, aux;
    ZZn2 Qx, Qy, Qz;

    aux = String_to_Big(string1) + String_to_Big(string2);

    Q1.get(Qx,Qy,Qz);
    Qx.get(x, y);
    aux += x+y;
    Qy.get(x, y);
    aux += x+y;
    Qz.get(x, y);
    aux += x+y;

    Q2.get(Qx,Qy,Qz);
    Qx.get(x, y);
    aux += x+y;
    Qy.get(x, y);
    aux += x+y;
    Qz.get(x, y);
    aux += x+y;

    return aux;
}

void XOR(char *string1, char *string2, char *result) { // Faz o Xor
entre duas string e retorna na terceira
    int i;
    for(i=0; i<(N/8); i++) result[i] = string1[i] ^ string2[i];
}

int main() {
    miracl *mip = mirsys(100, 0);
    ECn P, Ppub, Na, f, f_, U;
```

```
ECn2 Qa, da;
ZZn6 gr, g_, res;
ZZn12 X;
Big aux, ta, r, big_sigma;
int i, int_aux;
char sigma[(K0/8)+1];
char M[((N-K0)/8)+1] = "MensagemSecreta";
char V[(N/8)+1];
char temp_H2[(N/8)+1];

Big s,p,q,x,cf,t;
time_t seed;

mip->IOBASE=16;
x= "600000000000219B";  // BN.CPP
//x= "-6000000000001F2D"; // BN.CPP

p=36*pow(x,4)-36*pow(x,3)+24*x*x-6*x+1;
t=6*x*x+1;
q=p+1-t;
cf=p-1+t;
modulo(p);
set_frobenius_constant(X);

cout << "Iniciando... " << endl;

time(&seed);
irand((long)seed);

ecurve((Big)0,(Big)3,p,MR_AFFINE);

mip->IOBASE=16;
mip->TWIST=TRUE;   // Usando twist


// Inicializa

// Ponto P aleatoriamente
aux=rand(q);
while (!P.set(aux,aux)) aux+=1;
cout << "P= " << P << endl;

// Calculando Ppub = s*P, onde s é a chave-mestra
```

```
s=rand(q);
cout << "Chave Mestra= " << s << endl;
Ppub=s*P;
cout << "Ppub= " << Ppub << endl;

// Extrai
Qa = H1("Identificador_de_A");
cout << "Qa= " << Qa << endl;
da = s*Qa;
cout << "da= " << da << endl;

// Publica
ta=rand(q);
cout << "ta= " << ta << endl;
Na=ta*P;
cout << "Na= " << Na << endl;

// Cript
aux=rand(q);
Big_to_String(aux, (int)K0/8, sigma);
cout << "Mensagem= " << M << endl;
int_aux = strlen(M);
cout << "Tam_M= " << int_aux << endl;
cout << "Sigma= " << sigma << endl;
//concatena(M, sigma, V);
cout << "Tam_M= " << strlen(M) << endl;
r = H3(M, sigma);
cout << "r= " << r << endl;
// Qa ja esta calculado
if (!ecap(Ppub,Qa,x,X,res)) cout << "Erro no emparelhamento..." << endl;
gr = pow1(res,r);
cout << "gr= " << gr << endl;
f = r*Na;
cout << "f= " << f << endl;
U = r*P;
cout << "U= " << U << endl;
H2(U, gr, f, temp_H2);
cout << "Temp_H2= " << temp_H2 << endl;
concatena(M, sigma, V);
XOR(V, temp_H2, V);
cout << "V= " << V << endl;

// Decript
```

```
if (!ecap(U,da,x,X,g_)) cout << "Erro no emparelhamento..." << endl;
cout << "gr= " << gr << endl;
cout << "g_= " << g_ << endl;
f_ = ta*U;
cout << "f= " << f << endl;
cout << "f_= " << f_ << endl;
H2(U, g_, f_, temp_H2);
cout << "Temp_H2= " << temp_H2 << endl;
XOR(V, temp_H2, temp_H2);
cout << "Decript= " << temp_H2 << endl;
int_aux = strlen(M);
cout << "Tam_M= " << int_aux << endl;
// Separando M||sigma
for(i=0; i<int_aux; i++)
{
    M[i] = temp_H2[i];
}
M[i] = '\0';
cout << "M_final= " << M << endl;
int_aux = strlen(sigma);
for(i=0; i<int_aux; i++)
{
    sigma[i+int_aux] = temp_H2[i+strlen(M)];
}
sigma[i] = '\0';
cout << "Sigma_final= " << sigma << endl;
r = H3(M, sigma);
cout << "r_final= " << r << endl;
f = r*P;
cout << "U= " << U << endl;
cout << "U_final= " << f << endl;

// Inicializa... Assinatura
ECn2 Q, Qpub;
ECn Da, S;
ZZn6 g, NA, NA_H, NA_H_minus, R, R_;
Big h, h2;

// Ponto P aleatoriamente
aux=rand(q);
while (!P.set(aux,aux)) aux+=1;
cout << "P= " << P << endl;
```

```
// Ponto Q aleatoriamente
aux=rand(q);
Q = Big_to_ECn2(aux);
cout << "Q= " << Q << endl;

// Calculando g
if (!ecap(P,Q,x,X,g)) cout << "Erro no emparelhamento..." << endl;
cout << "g= " << g << endl;

// Calculando Qpub = s*Q, onde s é a chave-mestra
s=rand(q);
cout << "Chave Mestra= " << s << endl;
Qpub=s*Q;
cout << "Qpub= " << Qpub << endl;

// Parcial
ta = H1_sign("Identificador_de_A")+s; // ta eh usado para economizar um Big
ta = ta%q;
cout << "Aux= " << ta << endl;
aux = inverse(ta, q);
cout << "Aux_inverso= " << aux << endl;
ta *= aux;
ta = ta%q;
cout << "Aux * Aux_inverso= " << ta << endl;
Da = aux*P;

// Info-secreta
ta=rand(q);

// Secreta:
    // Nada para calcular
    // Entidade A mantem em sigilo sua chave secreta

// Publica
NA = powl(g, ta);

// Assina
aux=rand(q); // Aux usado como Parametro x
cout << "x= " << aux << endl;
R = powl(g, aux);
cout << "R= " << R << endl;
h = H2_sign("Mensagem_para_Assinar", "Identificador_de_A", NA, R);
cout << "h= " << h << endl;
```

```
    // Maneira passo a passo de calcular S com operacao modular
    h2 = h*ta; // Usando h2 pra nao declarar outro Big auxiliar
    cout << "h*ta= " << h2 << endl;
    aux+=h2;
    aux = aux%q;
    S = aux*Da;
    cout << "S= " << S << endl;

    // Verifica
    NA_H = powl(NA, h);
    NA_H_minus = inverse(NA_H);

    if (!ecap(S,(H1_sign("Identificador_de_A")*Q)+Qpub,x,X,R_))
        cout << "Erro no emparelhamento..." << endl;

    cout << "Emparelhamento= " << R_ << endl;

    R_ = R_ * NA_H_minus;
    cout << "R_= " << R_ << endl;
    h2 = H2_sign("Mensagem_para_Assinar", "Identificador_de_A", NA, R_);
    cout << "h2= " << h2 << endl;
    h2 = H2_sign("Mensagem_para_Assinar", "Identificador_de_A", NA, R_);
    cout << "h2= " << h2 << endl;

    if (h==h2) cout << "Assinatura Correta" << endl;
        else cout << "Assinatura Incorreta" << endl;

    return 0;
}
```

Referências Bibliográficas

AL-RIYAMI, S. S.; PATERSON, K. G. Certificateless public key cryptography. In: *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security.* Taipei, Taiwan: Springer, 2003. (Lecture Notes in Computer Science, v. 2894). ISBN 3-540-20592-6. Disponível em: <http://eprint.iacr.org/2003/126>.

AL-RIYAMI, S. S.; PATERSON, K. G. Cbe from cl-pke: A generic construction and efficient schemes. In: *Public Key Cryptography - PKC 2005.* Les Diablerets, Switzerland: Springer, 2005. (Lecture Notes in Computer Science, v. 3386), p. 398–415. ISBN 3-540-24454-9.

BALASUBRAMANIAN, R.; KOBLITZ, N. The improbability that an elliptic curve has subexponential discrete log problem under the menezes-okamoto-vanstone algorithm. *Journal of Cryptology*, v. 11, n. 2, p. 141145, 1998.

BARRETO, P. *Curvas Elípticas e Criptografia - Conceitos e Algoritmos.* 1999. Paulo Barreto's Crypto Page. Disponível em: <http://paginas.terra.com.br/informatica/paulobarreto>.

BARRETO, P.; NAEHRIG, M. Pairing-friendly elliptic curves of prime order. In: *Selected Areas in Cryptography, 12th International Workshop, SAC 2005.* Kingston, ON, Canada: Springer, 2005, (Lecture Notes in Computer Science, v. 3897). p. 319–331.

BARRETO, P. S. L. M. *Criptografia robusta e marcas d'água frágeis: construção e análise de algoritmos para localizar alterações em imagens digitais.* Tese (Doutorado) — Escola Politénica, Universidade de São Paulo, 2003. Disponível em: <http://paginas.terra.com.br/informatica/paulobarreto/Tese.pdf>.

BARRETO, P. S. L. M. et al. Efficient algorithms for pairing-based cryptosystems. In: *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology.* London, UK: Springer-Verlag, 2002. p. 354–368. ISBN 3-540-44050-X.

BARRETO, P. S. L. M.; LYNN, B.; SCOTT, M. On the selection of pairing-friendly groups. In: *Selected Areas in Cryptography — SAC'03*. [S.l.]: Springer-Verlag, 2003, (LNCS).

BERBECARU, D.; LIOY, A.; MARIAN, M. On the complexity of public-key certificate validation. In: *ISC '01: Proceedings of the 4th International Conference on Information Security*. London, UK: Springer-Verlag, 2001. p. 183–203. ISBN 3-540-42662-0.

BLAKE I.; SEROUSSI, G. S. N. P. *Elliptic Curves in Cryptography*. London, UK: Cambridge University Press, 1999.

BONEH, D.; FRANKLIN, M. K. Identity-based encryption from the weil pairing. In: *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 2001. p. 213–229. ISBN 3-540-42456-3. Disponível em: <http://eprint.iacr.org/2001/090>.

CERTICOM. *An Elliptic Curve Cryptography (ECC) Primer*. 2004. The Certicom *"Catch the Curve"* White Paper Series. Disponível em: <http://www.certicom.com/download/aid-317/WP-ECCprimer.pdf>. Acesso em: julho de 2006.

CHEN, X.; ZHANG, F.; KIM, K. A new id-based group signature scheme from bilinear pairings. 2003.

CHENG, Z.; COMLEY, R. *Efficient Certificateless Public Key Encryption*. 2005. Cryptology ePrint Archive, Report 2005/012. Disponível em: <http://eprint.iacr.org/>.

DIFFIE, W.; HELLMAN, M. E. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22, n. 6, p. 644–654, 1976.

DUPONT, R.; ENGE, A.; MORAIN, F. *Building curves with arbitrary small MOV degree over finite prime fields*. 2002. Cryptology ePrint Archive, Report 2002/094. Disponível em: <http://eprint.iacr.org/>. Acesso em: janeiro de 2008.

ELLISON, C.; SCHNEIER, B. Ten risks of PKI: What you're not being told about public-key infrastructure. *Computer Security Journal*, v. 16, n. 1, p. 1–7, 2000. ISSN 0277-0865.

GALBRAITH, S.; PATERSON, K.; SMART, N. *Pairings for Cryptographers*. 2006. Cryptology ePrint Archive, Report 2006/165. Disponível em: <http://eprint.iacr.org/>. Acesso em: maio de 2006.

GOYA, D. *Proposta de Esquemas de Criptografia e de Assinatura sob Modelo de Criptografia de Chave Pública sem Certificado*. Dissertação (Mestrado) — Instituto de Matemática e estatística, 2006.

GUTMAN, P. Pki: It's not dead, just resting. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 35, n. 8, p. 41–49, 2002. ISSN 0018-9162.

HESS, F.; SMART, N.; VERCAUTEREN, F. *The eta pairing revisited.* 2006. Cryptology ePrint Archive, Report 2006/110. Disponível em: <http://eprint.iacr.org/>. Acesso em: março de 2006.

HUANG, X. et al. On the security of certificateless signature schemes from asiacrypt 2003. In: DESMEDT, Y. et al. (Ed.). *CANS.* Xiamen, China: Springer, 2005. (Lecture Notes in Computer Science, v. 3810), p. 13–25. ISBN 3-540-30849-0.

IEEE/P1363. *Standard Specifications For Public Key Cryptography - Amendment 1: Additional Techniques.* [S.l.], 2004. `http://grouper.ieee.org/groups/1363/P1363a`.

KOBLITZ, N. Elliptic curve cryptosytems. *Mathematics of Computation*, v. 48, n. 177, p. 203–209, 1987.

KOBLITZ, N. *A course in number theory and cryptography, 2.ed.* New York, NY, USA: Springer-Verlag, 1994.

KOHNFELDER, L. *A Method for Certification.* Cambridge, MA, 1978.

MENEZES, A.; VANSTONE, S.; OKAMOTO, T. Reducing elliptic curve logarithms to logarithms in a finite field. In: *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing.* New York, NY, USA: ACM Press, 1991. p. 80–89. ISBN 0-89791-397-3.

MENEZES, A. J.; VANSTONE, S. A.; OORSCHOT, P. C. V. *Handbook of Applied Cryptography.* Boca Raton, FL, USA: CRC Press, Inc., 1996. Disponível em: <http://www.cacr.math.uwaterloo.ca/hac/>.

MILLER, V. S. Use of elliptic curves in cryptography. In: WILLIAMS, H. C. (Ed.). *CRYPTO 85.* Santa Barbara, California, USA: Springer, 1985. (Lecture Notes in Computer Science, v. 218), p. 417–426.

MIYAJI; NAKABAYASHI; TAKANO. New explicit conditions of elliptic curve traces for FR-reduction. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, 2001. Disponível em: <citeseer.ist.psu.edu/miyaji01new.html>.

NIST/FIPS186-2. *Digital Signature Standard (DSS), FIPS PUB 186-2.* 2000. 72 pages p. Disponível em: <http://csrc.nist.gov/publications/fips/fips186-2/fips186-2.pdf>.

RIVEST, R. L. *Cryptography - Chapter 13 of Handbook of Theoretical Computer Science.* Elsevier ed. J. Van Leeuwen, 1990. 717–755 p. Disponível em: <http://theory.lcs.mit.edu/~rivest/publications.html>.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. M. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, v. 21, n. 2, p. 120–126, 1978.

SAKAI, R.; OHGISHI, K.; KASAHARA, M. Cryptosystems based on pairing. In: *Symposium on Cryptography and Information Security (SCIS2000).* Okinawa, Japan: Inst. of Electronics, Information and Communication Engineers, 2000. p. 26–28.

SCHNEIER, B. *Applied Cryptography: Protocols, Algorithms and Source Code in C.* [S.l.]: John Wiley & Sons, 1994.

SCOTT, M. *MIRACL—A Multiprecision Integer and Rational Arithmetic C/C++ Library.* [S.l.], 2003. Available at http://indigo.ie/ mscott.

SCOTT, M. Computing the tate pairing. In: *Cryptography track - RSA-2005.* San Francisco, USA: Springer-Verlag, 2005. (Lecture Notes in Computer Science, v. 3376), p. 293–304.

SCOTT, M. Implementing cryptographic pairings. In: *Pairing-Based Cryptography 2007.* Tokyo, Japan: Springer-Verlag, 2007. (Lecture Notes in Computer Science, v. 4575), p. 177–196.

SHAMIR, A. Identity-based cryptosystems and signature schemes. In: *Proceedings of CRYPTO 84 on Advances in cryptology.* New York, NY, USA: Springer-Verlag New York, Inc., 1984. p. 47–53. ISBN 0-387-15658-5.

STALLINGS, W. *Cryptography and Network Security: Principles and Practice.* USA: Pearson Education, 2002. ISBN 0130914290.

STINSON, D. R. *Cryptography: Theory and Practice.* Boca Raton, Florida: CRC Press, 1995.

TERADA, R. *Segurança de Dados - Criptografia em Redes de Computador.* São Paulo, SP: Editora Edgard Blücher, 2000.

WIN, E. D.; PRENEEL, B. *Elliptic Curve Publik Key Cryptossystems - an Introduction.* 1998. 131-141 p. LCNS vol.1528. Disponível em: <ftp://ftp.esat.kuleuven.ac.be/pub/cosic/dewin/coursetext.ps.gz>. Acesso em: julho de 2004.

ZHANG, Z. et al. Certificateless public key signature: Security model and efficient construction. In: *4th. International Conference on Applied Cryptography and Network Security.* Singapore: Springer, 2006. (Lecture Notes in Computer Science, v. 3989).