Dissertação

Implementação da Cifra de Curva Elíptica num PDA

Aluno:

Diogo Miguel Costa Rodrigues

Número 50702, LEEC

Instituto Superior Técnico

26 de Setembro de 2007

Presidente:

Professor Mário Serafim dos Santos Nunes (DEEC)

Orientador:

Professor Rui Gustavo Crespo (DEEC)

Vogal:

Professor Leonel Augusto Pires Seabra de Sousa (DEEC)

Índice

Agradecimentos

Gostava de agradecer a todos os colegas, amigos e familiares que ao longo da concretização deste trabalho sempre me incentivaram e contribuíram para que o pudesse concluir.

De maneira especial, quero também agradecer ao Professor Rui Gustavo Crespo pela orientação deste trabalho, agradecendo também a sua paciência, correcções e ideias para a realização da tese.

1. Introdução

A ideia de usar curvas elípticas na criptologia não é nova e remota ao ano 1985 [Koblitz&Miller, 85]. Apesar da proposta ser antiga, nunca teve uma grande divulgação talvez devido ao facto de envolver elevada complexidade matemática e a sua segurança não ter sido exaustivamente analisada.

A cifra assimétrica, baseada em curvas elípticas, compromete-se a reduzir significativamente o número de bits da chave necessários para a codificação de dados em relação a outras cifras assimétricas como o RSA [Rivest&Shamir&Adleman, 77] mantendo o mesmo grau de segurança. Esta solução revela maior utilidade em dispositivos com menor poder de processamento, como são os casos dos dispositivos móveis.

A dissertação está estruturada de maneira a que sejam dadas todas as noções necessárias para o entendimento da cifra tanto a nível de implementação como a possíveis melhoramentos que possam ser feitos no futuro.

1.1 Segurança

Hoje em dia, a questão da segurança assume um papel determinante na vida de cada indivíduo. Um utilizador de qualquer sistema informático tem que se sentir seguro, de forma transparente, para poder realizar todas as tarefas que lhe são incumbidas .

Nos últimos anos têm surgido várias empresas dedicadas exclusivamente a esta questão, dada a sua relevância e já é possível concluir que, para assegurar a vitalidade de um sistema, seja ele qual for, é necessário garantir o cumprimento de regras que façam com que esse mesmo sistema seja seguro, inviolável e imune a ameaças e ataques.

Face à crescente importância da informática e da valorização da Informação, é necessário que sejam garantidas diversos factores, nomeadamente a confidencialidade (garantir que a informação não está disponível a entidades que não tenham autorização), a integridade (a informação não é alterada por terceiros) e a autenticidade das mensagens (a mensagem é enviada/recebida apenas pelo emissor/receptor).

Actualmente, a expansão do mercado de PDAs e smartPhones (telefone móvel com características semelhantes a um computador) fez com que fosse alargado o âmbito da análise e procura de soluções associadas a questões de segurança e desse facto decorre a principal motivação deste trabalho.

1.3 Noções Matemáticas

A Curva elíptica assenta numa determinada complexidade matemática. Antes de qualquer avanço nesta área, é importante realçar alguns aspectos matemáticos que envolvem o núcleo da implementação. Desde a noção de corpos, como a descrição de como se realizam as principais operações matemáticas, são bases imprescindíveis à boa compreensão do funcionamento da Curva Elíptica.

Serão também referenciados alguns algoritmos conhecidos, como por exemplo a função de dispersão SHA1 [SHA1, 95], que serão usados em alguns protocolos de cifra baseados em Curva Elíptica.

1.3 Curva Elíptica

Uma das formas de proteger a Informação é a sua codificação. A função de codificação (cifra) tem de satisfazer as seguintes condições:

- Ser injectiva, o que possibilita recuperar os dados iniciais (designados por texto plano).
- A função de cifra depende de um parâmetro secreto, designado por chave.
- O resultado da cifra (designado por criptograma ou texto cifrado) tem de ser distinto dos dados a proteger.
- É computacionalmente fácil cifrar e decifrar, conhecendo a chave. A partir do criptograma, sem conhecimento da chave é computacionalmente muito pesado determinar o texto plano.

Ao longo dos tempos, foram surgindo muitos protocolos de cifra, tendo alguns caindo em desuso e outros sido considerados como normas e utilizados todos os dias, tanto na Internet, em sistemas operativos ou em outros programas.

No âmbito de encontrar um novo modelo, mais seguro e mais rápido, surgiu a cifra de curva elíptica. Esta cifra pretende usar chaves de menor dimensão que usado em outras cifras (como por exemplo o RSA) mantendo o mesmo nível de segurança.

A cifra de curva elíptica baseia-se num conjunto de operações aritméticas (soma, subtracção, multiplicação e divisão) sobre uma curva definida pela equação de *Weierstrass*.

Com um algoritmo complexo, pretende-se que a cifra seja rápida e segura, de maneira a que seja usada em dispositivos com menor poder de processamento como os PDA ou *smartphones*.

Existem uma diversidade muito grande em relação ao parâmetros que podem ser usados nas curvas elípticas, e porque podem representar um nível de segurança muito baixo, existem entidades que se responsabilizam pela publicação dos parâmetros considerados seguros, como por exemplo, o governo federal norte-americano que publicou um artigo com os valores que devem ser usados, valores estes que se encontram no Anexo 1. Para além de satisfazerem as condições de segurança, os parâmetros seleccionados permitem implementar de forma mais eficiente as operações de cifra e de decifra.

1.4 Implementação

A constante inovação das tecnologias permite que sejam usadas cada vez mais ferramentas próprias para a construção de programas destinados a sistemas específicos.

Um PDA-Personal Digital Assistant, Assistente Pessoal Digital em português, é um dispositivo móvel normalmente pequeno, que cabe na palma da mão. Era usado para serviços de calendarização mas actualmente a sua área de intervenção é muito mais abrangente. Apesar de possuir um processador menos poderoso (velocidade típica de centenas de MHz, contra os milhares de MHz para um PC) e possuir memória mais reduzida que um computador de secretária, um PDA permite ter os programas básicos como folhas de cálculo, jogos ou mesmo um processador de texto. Normalmente os PDAs vêm acompanhados com diversas tecnologias que permitem a interacção e transmissão de dados com outras pessoas, como por exemplo: Wi-fi, infravermelhos, bluetooth, cabo série, rede GSM ou GPS.

Nos últimos anos surgiu um movimento que se tem mostrado interessado por instalar ambientes de desenvolvimentos em sistemas portáteis como os PDAs. Estes dispositivos possuem um sistema operativo dedicado, como o Windows CE, o qual já foi lançado em várias versões, sendo a última a versão 6.

Visto ser um aparelho fechado, possui características que têm de ser levadas em conta, aquando o desenvolvimento de programas. Apesar de ser possível aumentar memória existe um limite físico, assim como a nível de processamento (talvez o mais importante) que é reduzido. É preciso não exagerar no tipo de operações de um programa para que este não seja impossível de ser executado. Associado também à elevada taxa de processamento está a alimentação do PDA. Como normalmente é para ser usado em operações simples do dia a dia, é preciso ter em conta que quanto mais processamento for necessário, mais energia será necessária fazendo com que as baterias do dispositivo gastem-se mais rapidamente.

Como foi referido no paragrafo anterior, um dos grandes problemas dos PDA's é a sua memória reduzida e o poder de processamento mais fraco face a um computador de secretária. Outro problema é que qualquer meio de segurança terá de ser aplicado em software, devido a que quando se compra um dispositivo, este vêm já com o hardware específico e não maleável. Tempo, memória e Energia são os factores que se devem ter em conta quando se produz um programa para ser usado em PDAs.

Uma solução possível, é implementar um algoritmo de cifra numa API (interface de aplicação). Tem como vantagem de ser maleável podendo ser sempre modificada quando surgirem melhores algoritmos de forma a garantir uma maior segurança.

Neste trabalho, será usado a ferramenta da Microsoft Visual Studio 2005 como compilador, a linguagem de programação C# e o dispositivo móvel HTC p3600 com o sistema operativo Windows CE 5 com a extensão Compact Framework 2.0 (compatível com o PocketPC 2003).

1.5 Estrutura da dissertação

Esta dissertação pretende oferecer um bom ponto de partida para quem quer aprofundar de uma maneira simples e compreensível na área das Curvas Elípticas. O trabalho encontra-se feito para que seja possível qualquer pessoa aproveitá-lo e expandi-lo facilmente. Qualquer nova tecnologia de troca de informação (como por exemplo Wi-fi), novo protocolo de cifra ou a implementação de uma nova base, torna-se relativamente fácil de introduzir porque o código foi estruturado de maneira a não ser um código fechado.

A restante dissertação é estruturada na seguinte forma:

- O capítulo 2 faz uma breve introdução às bases matemáticas necessárias para descrever e representar curvas elípticas.
- Seguidamente, o capítulo 3 aborda alguns dos métodos conhecidos de cifra de curva elíptica.
- O capítulo 4 descreve a arquitectura do projecto e analisa alguns dos problemas que surgiram na implementação da cifra de curva elíptica.
- Por fim, o capítulo 5 desta dissertação, apresenta alguns exemplos práticos que servirão para concluir se de facto a solução apresentada neste trabalho, é ou não, uma solução viável, terminando com uma abordagem de possíveis extensões ao trabalho.

2. Bases Matemáticas

Este capítulo é dedicado exclusivamente à abordagem matemática que envolve os aspectos teóricos da tese. Antes de se aprofundar no tema da cifra de curva elíptica é necessário primeiro obter alguns conceitos que permitem a correcta compreensão dos algoritmos.

O conteúdo será dividido em várias secções, sendo a primeira secção fundamental para a continuação da leitura da dissertação. Nela serão descritas as formulações matemáticas por detrás da curva elíptica, sendo esta secção uma introdução à implementação propriamente dita.

Serão também abordados, em detalhe, as várias variantes da equação que define a curva, dando especial atenção ao formato adoptado na parte prática.

Por fim, neste capítulo será feita uma breve descrição de todos os algoritmos que serão utilizados no projecto.

2.1 Corpos

A criptografia baseada em curva elíptica opera sobre corpos (*fields*) finitos (com um número fixo de elementos) numéricos. Este tipo de corpos são também conhecidos por corpos de Galois e são representados por $GF(p^n)$ ou por F_{p^n}, onde p é um número primo e n um número inteiro. Define-se a ordem do grupo por p^n, e esta só pode ser igual a um número primo ou a uma potência do mesmo. Neste conjunto F é obrigatório que existam duas operações: a de adição (representada por +) e a de multiplicação (representada por .). Estas operações têm que respeitar as seguintes propriedades:

- (F,+) é um grupo abeliano com identidade aditiva a ser representada por O.
- (F \ {0}, .) é um grupo abeliano com identidade multiplicativa = 1.
- Leis distributivas: para todos os elementos a,b,c∈F tem-se $(a+b).c = a.c + b.c$
- Uma operação sobre dois elementos resulta num elemento pertencente ao conjunto F.
- Todos os elementos dos grupos (F,+) e (F\{0},.) possuem um inverso.

Num grupo $(G,1,_G^{-1},\bullet)$ existe sempre um elemento identidade 1 e todos os elementos possuem um inverso tal que $a \bullet a^{-1} = a^{-1} \bullet a = 1$ sendo o operador de grupo • associativo e comutativo.

A subtracção e a divisão são definidas nas operações soma e multiplicação respectivamente.

Existem vários tipos de corpos que foram estudados exaustivamente, havendo dois casos particulares mais interessantes de ponto de vista criptográfico, os chamados corpos finitos, por possuírem um número fixo de elementos igual a p^n. Dentro deste grupo, existem ainda dois tipos de corpos, os "corpos finitos primos" (de ordem prima, representada por GF(p)) e os corpos finitos binários (de característica dois, representado por GF(2^n)), sendo o último mais utilizado em cifras.

Para um corpo F com identidade multiplicativa 1, a característica p é tal que:

$$\underbrace{1+1+1+...+1}_{p\ vezes} = 0 \quad \textbf{[Equação 1]}$$

Exemplo 1: Característica de um número

Os conjuntos dos números racionais, reais e complexos têm característica 0 porque $\underbrace{1+1+1+...+1}_{p\ vezes} \neq 0$ seja qual for o valor de p.

Os corpos finitos do tipo F_p, onde p é um número primo, têm característica p.

Os corpos finitos binários têm esse nome porque possuem uma característica 2. São representado por F_{2^n}, e na prática significa ser possível recorrer a representações de bases (como por exemplo a base polinomial ou normal) tornando a implementação das operações de soma, multiplicação e inversão mais rápidas.

Exemplo 2: Corpos Finitos

O corpo finito primo F_{29} tem como elementos F=$\{0,1,2...28\}$.

As operações de soma e multiplicação são módulo 29. Por exemplo, 24+21=45 mod 29=16.

Para cada elemento não nulo existe sempre um inverso porque a dimensão é um número primo. Por exemplo, $24^{-1} = 5$.

Normalmente, os elementos de um grupo abeliano definido por $(F,+,*,0,1)$ podem ser representados por polinómios com os coeficientes de F. Os termos $z, z^2, ..., z^p$ não são membros de F sendo apenas usados para representação dos elementos de F e para facilitar as operações de soma e multiplicação, onde z é a raiz do polinómio.

Exemplo 3: Representação polinomial

Um corpo binário, definido por F_{2^4}, é composto pelos 16 polinómios de grau menor ou igual a 3:

$$\begin{array}{cccc} 0 & z^2 & z^3 & z^3+z^2 \\ 1 & z^2+1 & z^3+1 & z^3+z^2+1 \\ z & z^2+z & z^3+z & z^3+z^2+z \\ z+1 & z^2+z+1 & z^3+z+1 & z^3+z^2+z+1 \end{array}$$

A operação de soma é trivial, basta adicionar os termos módulo 2. Por exemplo, $(z^2+1)+(z^3+z^2) = z^3+1$.

A operação de multiplicação exige a determinação do resíduo do polinómio primitivo do produto dos termos. Por exemplo, adoptando o polinómio primitivo z^4+z+1 tem-se $(z^2+1)(z^3+z^2) = z^5+z^4+z^3+z^2 \bmod z^4+z+1 = z^3+1$.

2.2 Equação de Weierstrass

Karl Weierstrass foi um matemático que dedicou-se ao estudo de curvas elípticas. Definiu uma equação geral, em que os coeficientes a_i podem ser escolhidos de forma a ser possível usar a equação para aplicações e corpos específicos. Esta equação é definida por:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad \textbf{[Equação 2]}$$

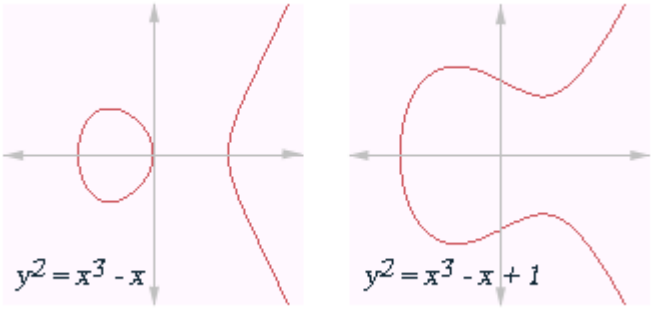


Figura 1: Exemplos gráficos de duas equações de Weierstrass

Na Figura 1 são representadas graficamente duas equações de Weierstrass, por forma a se visualizar o formato típico das curvas elípticas. As curvas podem ser descontínuas (tal como na equação da esquerda) ou contínuas (tal como na equação da direita).

2.2.1 Discriminante

Dada uma determinada curva elíptica "E" pode-se definir os seguintes parâmetros:

$$\begin{cases} \Delta = -d_2^2.d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 = a_1^2 + 4a_2 \\ d_4 = 2a_4 + a_1a_3 \\ d_6 = a_3^2 + 4a_6 \\ d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_a^2 \end{cases} \quad \textbf{[Equação 3]}$$

O Δ é o discriminante da equação de Weierstrass. Se $\Delta \neq 0$, a curva elíptica é suave (nenhum ponto da curva tem dois ou mais pontos numa linha tangente) e a curva é não singular e torna-se fácil a quebra da cifra.

2.2.2 Simplificações

Dada a equação de Weierstrass, esta pode ser simplificada, usando para isso substituições de variáveis. Se E for definida sobre um corpo K, temos que:

Se a característica do corpo K for diferente de dois ou três:

$$\begin{cases} E(x,y): y^2 = x^3 + ax + b \\ \Delta = -16(4a^3 + 27b^2) \end{cases}, a,b \in K \quad \textbf{[Equação 4]}$$

Se a característica do corpo K for igual a dois, pode acontecer dois casos:

Caso 1: $a_1 \neq 0$, Curva não-supersingular

$$\begin{cases} E(x,y): y^2 + xy = x^3 + ax^2 + b \\ \Delta = b \end{cases}, \; a,b \in K \quad \textbf{[Equação 5]}$$

Caso 2: $a_1 = 0$, Curva supersingular

$$\begin{cases} E(x,y): y^2 + cy = x^3 + ax^2 + b \\ \Delta = c_4 \end{cases}, \; a,b \in K \quad \textbf{[Equação 6]}$$

Ainda existe o caso em que a característica do corpo é igual a três:

Caso 1: $a_1^2 \neq -a_2$

$$\begin{cases} E(x,y): y^2 = x^3 + ax^2 + b \\ \Delta = -a^3 b \end{cases} \quad \textbf{[Equação 7]}$$

Caso 2: $a_1^2 = -a_2$

$$\begin{cases} E: y^2 = x^3 + ax + b \\ \Delta = -a^3 \end{cases} \quad \textbf{[Equação 8]}$$

2.3 Bases de representação

Uma base, segundo a álgebra linear, é um conjunto de vectores, linearmente independentes, que geram um determinado espaço. A utilização destas transformações permite que um determinado elemento de um corpo possa ser escrito de uma maneira que seja mais fácil, e mais rápida de executar as operações aritméticas. No âmbito deste trabalho, existem duas bases que serviram para representar o corpo finito escolhido, sobre o qual está definido a curva elíptica, a saber: a base polinomial e a base normal. É no entanto possível, converter elementos entre diferentes bases através de algoritmos ou matrizes de conversão, sendo este tipo de conversão computacionalmente pesada (eficiência quadrática).

A implementação foi implementada usando os dois tipos de bases que serão descritas em pormenor em seguida.

2.3.1 Representação em Base Polinomial

A aritmética baseada numa representação polinomial é uma abordagem simples porque é baseada em polinómios. Para que seja válida esta base é necessário definir as quatro operações básicas: soma, multiplicação, subtracção e divisão (inverso).

A base normal polinomial representa os elementos do corpo sob a forma $\{1, \alpha, \alpha^2, ..., \alpha^{n-1}\}$, onde α é a raiz do polinómio primitivo de grau m sobre o corpo F_{p^m}. De seguida são descritas sumariamente com funcionam as operações aritméticas num corpo finito utilizando uma base polinomial:

2.3.1.1 Adição

Por se recorrer ao uso de polinómios, a adição nesta base é simples. Usa uma adição de modulo p, sendo p a característica do corpo. Para um corpo F_{p^m} a soma de dois números a e b é dada por:

$$a + b = a + b(\text{mod } p) \quad \textbf{[Equação 9]}$$

No caso de ser um corpo finito binário (ordem 2) a soma e subtracção é feita em modulo 2, ou seja é o mesmo que realizar uma operação lógica ou-exclusivo (XOR), o que torna a sua implementação em Hardware bastante rápida.

Exemplo 4: Adição de dois números utilizando uma base polinomial

Considerando os números $x_1 = x^3 + x^2 + 1$ e $x_2 = x^3 + 1$ a operação adição tem como resultado $x_1 + x_2 = x^2$

2.3.1.2 Multiplicação

Apesar de ser uma operação fácil de compreender, computacionalmente pode tornar-se bastante pesada, visto serem precisos usar m^2 multiplicadores num corpo F_p. Primeiro é feita uma multiplicação parcial, uma soma de expoentes na base do corpo (que caso seja binário representa a operação ou-exclusiva dos vários expoentes) e depois é aplicada a redução de módulo, utilizando para isso um polinómio irredutível na base.

Exemplo 5: Multiplicação de dois números utilizando uma base polinomial

Por exemplo, pretende-se multiplicar dois números num corpo finito F_{2^5} onde existe o polinómio irredutível $p = x^5 + x^3 + 1$. A multiplicação de $x_1 = x^3 + x^2 + 1$ e $x_2 = x^3 + 1$ é feita, como dito anteriormente, por duas fases. A multiplicação de expoentes (em módulo 2) com o resultado $x_1 * x_2 = x^6 + x^5 + x^2 + 1$, e depois aplica-se o módulo com o polinómio irredutível (para que seja um elemento do conjunto do corpo), ficando no resultado final $x_1 * x_2 (\text{mod } p) = x^4 + x^3 + x^2$.

Para além desta multiplicação, existem outros métodos que se propõem serem mais rápidos e eficientes neste tipo de base, como por exemplo os algoritmos de Karatsuba-Ofman [Karatsuba-Ofman, 62] ou Linear Feedback Shift Register [LFSR, 97]; a computação de subcorpos; multiplicadores de pipelines ou sistólicos; ou mesmo recorrer a tabelas de resultados (solução pouco eficiente se existir pouca memória no sistema).

2.3.1.3 Inversão

A divisão de polinómios foi implementada utilizando o algoritmo de inversão euclidiano estendido. Este algoritmo é descrito como:

Algoritmo 1: Inversão Euclidiano estendido

```
Entradas: number, prime

Saídas: inver_number

b ← 1
c ← 0
u ← number
v ← prime

Enquanto: u diferente 1
        j ← GrauPolinomio(u) - GrauPolinomio(v)
        Se j<0 Então
                aux ← u
                u ← v
                v ← aux
                aux ← c
                c ← b
                b ← aux
                j = -j
        u ← u + DeslocacaoEsquerdaBits(v, j)
        b ← b + DeslocacaoEsquerdaBits(c, j)
inver_number ← b

Devolve inver_number
```

Por ser tão demorada, e à semelhança da multiplicação, também existem outros métodos possíveis para implementar a inversão, como os algoritmos de quase inverso de Schroeppel [Schroeppel, 99] ou o algoritmo de inversão Itoh-Tsujii [Itoh-Tsujii, 88]; repetição de multiplicação de quadrados e os já referenciados na secção anterior: tabelas de resultados, inversão de subcorpos e tabelas de resultados.

A vantagem do uso da base polinomial é que a multiplicação é relativamente simples. Existe também outra base, chamada base normal também muito usada em sistemas de criptologia, mas mais virada para a implementação em hardware.

2.3.2 Representação em Base Normal

Se o objectivo for obter uma solução dedicada, feita em hardware, a base preferencial seria a base normal. Esta representação é baseada em operações lógicas simples como o E (AND), ou-exclusivo (XOR) e operações mais complexas como o SHIFT e o ROTATE.

Sendo um corpo F_{p^m}, um elemento β do conjunto pode ser representado na base normal por:

$$\{\beta^{p^{m-1}},...,\beta^{p^2},\beta^{p},\beta\}$$ **[Equação 10]**

Um elemento é representado por:

$$e = e_{m-1}\beta^{2^{m-1}} + ... + e_2\beta^{2^2} + e_1\beta^2 + e_0\beta = \sum_{i=0}^{m-1} e_i\beta^{2^i}$$ **[Equação 11]**

As operações necessárias no trabalho, utilizando a base normal, são descritas em seguida.

2.3.2.1 Adição

A adição é feita da mesma maneira que a base polinomial, usando a operação lógica ou-exclusivo (XOR).

2.3.2.2 Quadrado

Um quadrado de um elemento é o mesmo que fazer a deslocação (shifting) de cada coeficiente para o próximo termo e fazer a rotação do ultimo coeficiente para a posição zero. (shift ciclico).

2.2.2.3 Multiplicação

A multiplicação em base normal é complexa sendo óptima quando se considera uma base normal óptima. Este tipo de sub base utiliza algumas simplificações tornando a multiplicação mais rápida. Como para cada tipo diferente é necessário um algoritmo optimizado diferente, optou-se nesta tese escolher o algoritmo geral que funciona para qualquer base. No trabalho foi implementado o algoritmo apresentado no documento regulado pelo governo americano [federal-government, 99].

Sabendo qual o tipo da base normal, é necessário fazer um pré-processamento (feito quando a base é construída) onde é calculado os vários valores da matriz lambda de tamanho $p = Tipo * nbits + 1$ do corpo finito $F_{2^{nbits}}$. Esta matriz é construída segundo o próximo algoritmo:

Algoritmo 2: Construção da matriz lambda

```
Entradas: Tipo, nbits, order
Saídas: lambda

p ← Tipo * nbits +1
u ← order
w ← 1

De: j=0 Até: T-1
        n ← w
        De: i=0 Até: nbits-1
                F(n) ← i
                n ← (2 * n) mod p
        w ← (u * w) mod p

Devolve F(u,v) = Σ_{k=1}^{p-2} u_{F(k+1)} v_{F(p-k)}
```

A multiplicação usa a matriz lambda e é feita segundo o próximo algoritmo:

Algoritmo 3: Multiplicação em base normal

```
Entradas: a, b, nbits
Saídas: c

u ← a
v ← b

De: k=0 Até: nbits-1
        ck ← F(u,v)
        u ← DeslocacaoCircularEsquerdaBits(u,1)
        v ← DeslocacaoCircularEsquerdaBits(v,1)

Devolve c
```

2.3.2.4 Inversão

A inversão na representação em base normal é dada pelo seguindo algoritmo:

Algoritmo 4: Inversão em base normal

```
Entradas: number, nbits
Saídas: invert_number

s ← Log2(nbits-1) - 1
m ← nbits - 1

Enquanto: s>=0 Faz:
        r ← DeslocamentoDireita(m, s)
        shift ← C
        De: rsft=0 Até: rsft < (r/2)
                DeslocamentoEsquerda(shift,1)

        temp ← C * shift;
        Se r impar Então

                DeslocamentoEsquerda(temp,1)
                C ← temp * number
        Se Não:

                C ← temp
        s--

DeslocamentoEsquerda(C,1)

Devolve: C
```

2.4 Algoritmos

Depois de visto como são feitas as operações de soma, multiplicação e inversão nas duas representações que serão adoptadas na implementação da cifra de curva elíptica, também é importante explicar alguns algoritmos auxiliares (cifra, cálculo de valores de dispersão e alteração de representação) que serão utilizados nesta dissertação.

2.4.1 AES/Rijndael

O algoritmo AES (Advance Encryption Standard) [Daemen&Rijmen, 02], também conhecido por Rijndael é usado num dos protocolos implementados. Caracteriza-se por ser uma cifra em blocos com chaves de tamanhos fixos (podendo ser de 128, 192 ou 256 bits).

O uso deste algoritmo simétrico permite acelerar as operações de cifra dos dados, mantendo a utilização da curva elíptica.

Algoritmo 5: Inversão em base normal

```
        Entradas: plainText, CipherKEy, Nr //Número de Rounds

        Saídas: chipherText

        AddRoundKey(CipherKey, ExpandedKey)

        De i=1 Até i<Nr Faz
              Round (state, ExpandedKey + Nb*i)

        FinalRound(state, ExpandedKey + Nb*Nr)

//AddRoundKey
//a subchave é combinada com o estado actual        do algoritmo. Para cada estado a
//subchave é derivada do chave principal, sendo construída através da combinação de cada
//byte do estado com os bytes da chave principal.

//Round
//É composto pelas operações SubBytes (substituição não linear de cada byte), ShiftRows
//(deslocamento cíclico) e MixColumns (opera em cada coluna de cada estado, misturando 4
//bytes)

//FinalRound
//Composto pela sequencia de operações SubBytes, ShiftRows e AddRoundKey
```

Uma alternativa ao uso deste algoritmo no protocolo ECIES é o uso do Triple-DES [IEEE P1363, 99] .

2.4.2 HMACSHA1

O algoritmo HMAC [RFC2104, 97] gera uma mensagem de integridade de código utilizando para isso uma função de dispersão. No projecto irá ser usado a função SHA1 [SHA1, 95]. O resultado é de tamanho fixo de 160 bits (20 bytes). Este mecanismo permite criar um selo para um conjunto de dados que tenha de ser transmitido por um canal inseguro, permitindo assim, haver um mecanismo que comprove que a informação não foi alterada. Qualquer alteração feita provoca que o resultado do HMAC seja diferente obrigando à rejeição dos dados.

Este algoritmo é apresentado, em pseudo-código, de seguida:

Algoritmo 6: Algoritmo HMACSHA1

```
Entradas: key, message

Saídas: HMAC(key,message)

opad = [0x5c * blocksize] // blocksize=64

ipad = [0x36 * blocksize] // blocksize=64

Se (Tamanho(key) > blocksize) Então

    key = hash(key)

De i=0 Até Tamanho(key) Faz

    ipad[i] ^= key[i]

    opad[i] ^= key[i]

Devolve hash(opad || hash(ipad || message)) // Concatenação de strings de bytes
```

2.4.3 KDF2

As funções de derivação de chaves (KDF) [RFC2898, 00] são funções baseadas em funções de dispersão que permitem criar um mecanismo adicional de segurança. A partir de chaves, informação comum ou de segredos partilhados é criado uma chave adicional. O uso das KDF permite aumentar o nível de segurança nos casos onde é usada uma chave considerada fraca.

O algoritmo usado no trabalho é a implementação do algoritmo KDF2 descrito no documento IEE P1363A:

Algoritmo 7: Algoritmo KDF2

```
Entradas: sharedvalue(Z), keydatalength, Sharedinfo(opcional)
Saídas: keyData

Hashlen <- 160
Counter <- 1

De i=0 Até j=ceiling(keydatalen/hashlen Faz
    Hash(i) <- Hash (Z || counter [|| sharedinfo]
    counter <- counter + 1

keydata <- Hash_1 || Hash_2 || … || Hash_j-1 || Hash_j

Devolve keydata
```

2.4.4 SHA1

O SHA1 é uma função de dispersão. A partir de um conjunto de bytes (dados, documentos, etc.) gera uma cadeia de bits de tamanho predefinido. É uma função que trabalha sobre os dados em blocos. O resultado tem sempre o mesmo tamanho. Se não for múltiplo então é preciso juntar bits à informação (padding).

Algoritmo 8: Algoritmo SHA1

```
h0 := 0x67452301; h1 := 0xEFCDAB89; h2 := 0x98BADCFE; h3 := 0x10325476; h4 := 0xC3D2E1F0


Por cada chunk //de 512bits
    break chunk into sixteen 32-bit big-endian words w(i), 0 ≤ i ≤ 15


    De i Até 16 to 79
        w(i) := (w(i-3) xor w(i-8) xor w(i-14) xor w(i-16)) DeslocamentoEsquerda 1
    a := h0;     b := h1;     c := h2;     d := h3;     e := h4
    De i até 0 to 79
        Se 0 ≤ i ≤ 19 Então
            f := (b and c) or ((not b) and d)
            k := 0x5A827999
        Se não, Se 20 ≤ i ≤ 39
            f := b xor c xor d
            k := 0x6ED9EBA1
        Se não, Se 40 ≤ i ≤ 59
            f := (b and c) or (b and d) or (c and d)
            k := 0x8F1BBCDC
        Se não, Se 60 ≤ i ≤ 79
            f := b xor c xor d
            k := 0xCA62C1D6
        temp := (a leftrotate 5) + f + e + k + w(i)
        e := d
        d := c
        c := b leftrotate 30
        b := a
        a := temp


   h0 := h0 + a;     h1 := h1 + b ;     h2 := h2 + c;     h3 := h3 + d;     h4 := h4 + e


Devolve: digest = hash = h0 || h1 || h2 || h3 || h4
```

2.4.5 NAF (not adjacent form)

Por vezes, fazer a multiplicação directa de um número pode não ser a melhor solução. Assim, partindo em partes o multiplicador, ganha-se vantagens face ao número de multiplicações necessárias. O algoritmo NAF [NAF, 06] transforma um número, converte-o para a sua representação binária e produz um resultado em termo de somas e subtracções, onde o resultado (na representação binária) não possui nenhum bit consecutivo.

Algoritmo 9: Algoritmo NAF

```
Entradas: numero
Saídas: numeroNAF

numeroNAF ← ''

Enquanto: number diferente 0 Faz:
        Se number impar Faz
                u ← 2-( numero  % 4)
                numeroNAF ← numeroNAF + u
                Se u < 0 Faz
                        numero++
        Se Não, Faz:
                numeroNAF ← numeroNAF + 0
        numero ← numero/2

Devolve numeroNAF
```

Exemplo 6: NAF

Considerado o número 29, ao aplicar-mos o algoritmo NAF ficamos com $29 = 32\ 4\ +1$.

Os valores dados pelo algoritmo passo a passo, são os seguintes:

1º Ciclo: $number = 29$ **:** número impar => $u = 2 - 29\%4 = 1$

numeroNAF = $\{1\}$

2º Ciclo: $number = 29/2 = 14$ **:** número par

$numeroNAF = \{0, 1\}$

3º Ciclo: $number = 14/2 = 7$ **:** número impar => $u = 2 - 7\%4 = -1$

$numeroNAF = \{-1, 0, 1\}$, $number = 7 + 1 = 8$

4º Ciclo: $number = 8/2 = 4$ **:** número par

$numeroNAF = \{0, -1, 0, 1\}$

5º Ciclo: $number = 4/2 = 2$ **:** número par

$numeroNAF = \{0, 0, -1, 0, 1\}$

6º Ciclo: $number = 2/1 = 1$ **:** número impar => $u = 2 - 2\%4 = 1$

$numeroNAF = \{1, 0, 0, -1, 0, 1\}$

O número dado é (transportando para a representação binária): $1.2^5 + 0.2^4 + 0.2^3 - 1.2^2 + 0.2^1 + 1.2^0 = 32 - 4 + 1$

2.5 Notas finais sobre a descrição teórica

Neste capítulo foram dadas as noções básicas matemáticas para a boa compreensão da parte teórica que está por detrás da equação de curvas elípticas. O capítulo 3 fala em pormenor dos diferentes tipos de curva elíptica, dando maior foco às curvas que serão implementadas na parte prática da tese.

3. Curva Elíptica

O presente capítulo aborda a descrição teórica da cifra de curva elíptica. Este capítulo explica os vários protocolos utilizados, como por exemplo, como serão feitas as trocas de chaves e quais as mensagens necessárias antes de se transferir a informação propriamente dita.

Serão explicados como serão integrados os corpos finitos binários, assim como a utilização correcta da curva elíptica e as operações definidas sob a essa curva. Na parte final deste capítulo serão identificados quais os protocolos implementados e descritos no capítulo 4 desta dissertação.

3.1 Operações Aritméticas

Para criar operações sob um corpo finito de uma curva elíptica, é necessário primeiro verificar as regras da Lei de Grupos:

Primeiro é definido um elemento identidade, designado por O_∞ (ponto no infinito):

- $P + O_\infty = P$ **[Equação 12]**

Dados um ponto $P = (x, y)$ então o seu inverso é dado por:

- $-P = (x, -y)$ **[Equação 13]**

Uma das operações definidas numa curva elíptica é a adição de dois pontos. Considerando $P = (x_1, y_1)$ e $Q = (x_2, y_2)$, graficamente temos que a soma é dada baseando-se na ideia da linha que une os dois pontos que irá intersectar na curva no inverso da soma. Fazendo o inverso do ponto de intersecção obtém-se $P + Q$. **[figura 2]**

O dobro de um número P é um valor importante porque a multiplicação P*Q pode ser calculada por P+P+...+P (Q vezes) = 2P + P + ... P (Q-1 vezes). O dobro é obtido pela mesma ideia que a adição, apesar de que neste caso, a união é feita apenas do ponto P, que irá intersectar a curva e fazendo o inverso no cruzamento, obtém-se o dobro **[figura 2].** O ponto de cruzamento existe sempre, porque num grupo a operação de soma é sempre fechada.

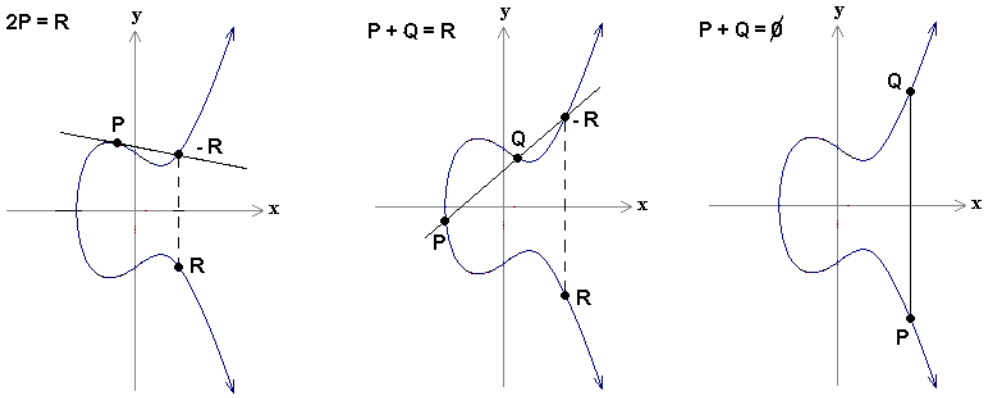


Figura 2: Representação das operações Soma e dobro numa curva elíptica

Na Figura 2 são representadas graficamente as operações de duplicação (à esquerda), soma (ao meio) e inverso (à direita) de pontos em curvas elípticas.

3.1.1 Soma e Dobro em curvas sob números reais

Dado dois pontos $P = (x_1, y_1)$ e $Q = (x_2, y_2)$, a soma é dada analiticamente por:

$$R = P + Q = (x_3, y_3) \quad \textbf{[Equação 14]}$$

Caso $P \neq Q$, a equação 14 fica:

$$x_3 = \theta^2 - x_1 - x_2 \;;\; y_3 = \theta(x_1 + x_3) - y_1 \text{ com } \theta = \frac{y_2 - y_1}{x_2 - x_1} \quad \textbf{[Equação 15]}$$

Se $P = Q = (x, y) = 2P$, ou seja, o dobro:

$$x_3 = \theta^2 - x_1 - x_2 \;;\; y_3 = \theta(x_1 + x_3) - y_1 \textbf{ com } \theta = \frac{3x^2 + a_4}{2y} \quad \textbf{[Equação 16]}$$

O negativo de um número é dado por:

$$-P = (x, -y) \text{ onde } P = (x, y) \quad \textbf{[Equação 17]}$$

3.1.2 Soma e Dobro em corpos de Galois 2^n

Dado dois pontos $P=(x_1,y_1)$ e $Q=(x_2,y_2)$, a soma é dada analiticamente pela equação 14

Caso $P \neq Q$, a equação 14 fica:

$$x_3=\theta^2+\theta+x_1+x_2+a_2 \;;\; y_3=\theta(x_1+x_3)+x_3+y_1 \text{ com } \theta=\frac{y_2-y_1}{x_2-x_1} \quad \textbf{[Equação 18]}$$

Se $P=Q=(x,y)=2P$, ou seja, o dobro:

$$x_3=\theta^2+\theta+a_2 \;;\; y_3=x^2+(\theta+1)x_3 \text{ com } \theta=x+\frac{y}{x} \quad \textbf{[Equação 19]}$$

Nos cálculos efectuados na implementação da cifra, a soma e a divisão são efectuadas em corpos de *Galois* 2^n.

3.1.3 Ponto Base (gerador)

Para operações criptográficas, é definido o ponto base G, também designado como gerador. Um elemento gerador é um dos elementos de F_{2^m} a partir do qual é possível gerar todo o conjunto, calculando as potências desse elemento.
A ordem de *G* é o menor número não negativo *n* tal que:

$$nG=O \quad \textbf{[Equação 20]}$$

Exemplo 7: Geração dos elementos a partir do ponto base *em* F_p

Considerado os seguintes parâmetros:

O corpo $F_{2^4}=\{g^0,g^1,g^2,g^3,g^4,g^5,g^6,g^7,g^8,g^9,g^{10},g^{11},g^{12},g^{13},g^{14},g^{15}\}$

Ponto $G=(0010)_2=\{x\}$

Polinómio irredutível: $f(x)=x^4+x+1$

Os elementos de F_{2^4} podem ser obtidos pela multiplicação do ponto *G*:

$g^0=(0001)_2=\{1\}=\{0x^3+0x^2+0x+1\}$

$g^1=(0010)_2=\{x\}=\{0x^3+0x^2+1x+0\}$

$$g^2 = (0100)_2 = \{x^2\} = \{0x^3 + 1x^2 + 0x + 0\}$$

$$g^3 = (1000)_2 = \{x^2\} = \{1x^3 + 0x^2 + 0x + 0\}$$

$$g^4 = (0011)_2 = \{x^4 \bmod f(x) = -x - 1\} = \{1x^3 + 0x^2 - 1x - 1\}$$

$$g^5 = (0110)_2 = \{x^5 \bmod f(x) = -x^2 - x\} = \{1x^3 - x^2 - 1x + 0\}$$

$$g^6 = (1100)_2 = \{x^6 \bmod f(x) = -x^3 - x^2\} = \{-x^3 - x^2 + 0x + 0\}$$

$$g^7 = (1011)_2 = \{x^7 \bmod f(x) = -x^3 + x + 1\} = \{-x^3 + 0x^2 + x + 1\}$$

$$g^8 = (0101)_2 = \{x^8 \bmod f(x) = x^2 + 2x + 1\} = \{0x^3 + 1x^2 + 2x + 1\}$$

$$g^9 = (1010)_2 = \{x^9 \bmod f(x) = x^3 + 2x^2 + 1\} = \{1x^3 + 2x^2 + 1x + 0\}$$

$$g^{10} = (0111)_2 = \{x^{10} \bmod f(x) = 2x^3 + x^2 - x - 1\} = \{2x^3 + x^2 - x - 1\}$$

$$g^{11} = (1110)_2 = \{x^{11} \bmod f(x) = x^3 - x^2 - 3x - 2\} = \{x^3 - x^2 - 3x - 2\}$$

$$g^{12} = (1111)_2 = \{x^{12} \bmod f(x) = -x^3 - 3x^2 - 3x - 1\} = \{-x^3 - 3x^2 - 3x - 1\}$$

$$g^{13} = (1101)_2 = \{x^{13} \bmod f(x) = -3x^3 - 3x^2 + 1\} = \{-3x^3 - 3x^2 + 0x + 1\}$$

$$g^{14} = (1001)_2 = \{x^{14} \bmod f(x) = -3x^3 + 4x + 3\} = \{-3x^3 + 0x^2 + 4x + 3\}$$

$$g^{15} = (0001)_2 = \{x^{15} \bmod f(x) = 4x^2 + 6x + 3\} = \{0x^3 + 4x^2 + 6x + 3\}$$

Exemplo 8: Cálculo de múltiplos do ponto base.

Considerado os seguintes parâmetros:

A curva elíptica $E(F_{751}) = y^2 = x^3 - x + 188 \bmod 751$ com $a = -1$ e $b = 188$

Ponto $G = (0;376)$

Os múltiplos kG são:

G = (0, 376)
2G = (1, 376)
3G = (750, 375)
4G = (2, 373)
5G = (188, 657)
6G = (6, 390)
7G = (667, 571)
8G = (121, 39)
9G = (582, 736)
10G = (57, 332)
...
761G = (565, 312)
762G = (328, 569)
763G = (677, 185)
764G = (196, 681)
765G = (417, 320)
766G = (3, 370)
767G = (1, 377)
768G = (0, 375)
769G = O (ponto no infinito)

Todos os pontos da curva elíptica considerados para a cifra de dados são apenas os que satisfazem a equação 20.

$$y^2 \bmod p = x^3 + ax + b \bmod p \quad \textbf{[Equação 21]}$$

3.2 Protocolos de Cifra

De seguida são descritos os vários algoritmos que foram implementados para cifrar/decifrar informação.

3.2.1 *Elliptic Curve Diffie-Hellman (ECDH)*

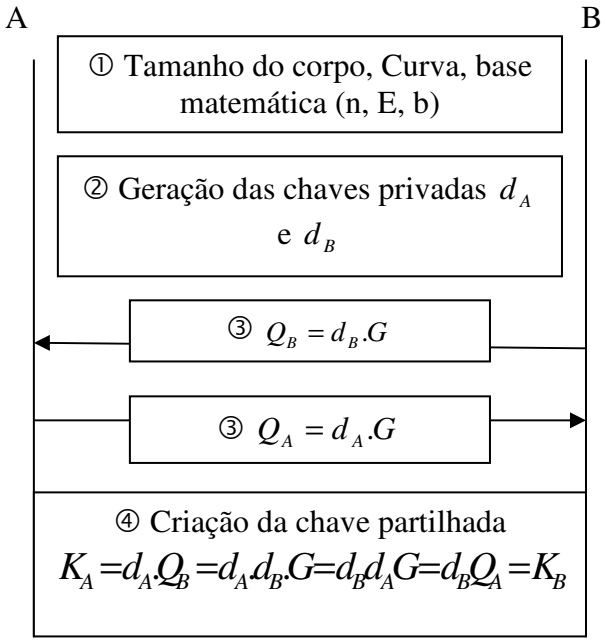
O protocolo *Elliptic Curve Diffie-Hellman* [ECDH, 00] é um protocolo de troca de chaves que, na sua forma mais básica, é susceptível ao ataque do Homem do meio (*Man in the middle*). Este algoritmo é usado no projecto apenas para estabelecer rapidamente a troca de chaves publicas entre entidades, não tendo por isso, sido considerado este tipo de falhas no protocolos.

Existe no entanto várias formas de ultrapassar esta falha como por exemplo o uso de ligações seguras [SSH, 05] ou o uso de certificados [Stallings, 03] que caem fora do âmbito desta dissertação.

Descrição do Protocolo passo-a-passo

1. Primeiro é necessário estabelecer quais os parâmetros da curva a usar, parâmetros estes que são especificados no **Anexo 1**. Nesta fase ocorre a escolha do número de bits da chave, da curva elíptica a usar e qual o seu ponto base, sendo o número de bits e o ponto base G públicos.
2. Cada entidade (**A** e **B** na imagem **protocolo 1**) gera a sua chave privada (d_A e d_B). Esta chave é um ponto aleatório na curva elíptica.
3. A entidade **A** envia a sua chave pública que consiste no ponto da curva $Q_A = d_A.G$ e recebe a chave pública de **B**, $Q_B = d_B.G$, onde G é o ponto base.
4. Com a troca de chaves, é possível ambas as entidades calcularem o ponto que servirá de chave pública partilhada. A entidade **A** calcula $K_A = d_A.Q_B$ enquanto que a entidade **B** calcula $K_B = d_B.Q_A$, sendo este dois pontos iguais, ou seja, $K_A = d_A.Q_B = d_A.d_B.G = d_B d_A G = d_B Q_A = K_B$.

Graficamente, o protocolo ECDH está definido no diagrama "Protocolo 1":



Protocolo 1: Diffie-Hellman

Actualmente não são conhecidos algoritmos computacionalmente eficientes para o cálculo dos valores de d a partir de G e d*Q. Tal se deve ao facto da multiplicação na curva elíptica não ser linear.

3.2.2 ECIES

O protocolo ECIES (*Elliptic Curve Integrated Encryption Scheme*) [ECIES, 01] permite acelerar a geração da cifra, em comparação ao método tradicional onde só é usado operações sobre a curva. Este algoritmo é baseado no algoritmo de *Diffie-Hellman* (troca de chaves entre os intervenientes).

O ECIES gera um conjunto de dados (V, C, T) que serão enviados onde o V é a chave pública do emissor, C é a mensagem cifrada (usando o XOR, AES ou 3DES) e o T é uma *tag* de autenticação gerada a partir da mensagem M e da chave publica do destinatário (MAC).

A implementação deste protocolo serão usados os algoritmos KDF2, HMACSHA1, AES (Advanced Encryption Standard) especificados no final do capítulo anterior. O algoritmo AES (*Symetric encryption Scheme*) [Daemen&Rijmen, 02] poderia ser substituído pela operação XOR (Stream Cipher , para mensagens pequenas e um nível de segurança inferior) ou pelo algoritmo triple-DES **[3DES, 04].**

O protocolo de cifra de uma mensagem é dado por:

1. Z = Diffie-Hellman Primitive (troca de chaves)
2. K = KDF (Z) [com tamanho de enkeylen [20octetos] + mackeylen [20octetos])
    a. K de tamanho k1 (mais à esquerda) (tamanho chave AES-CBC que vai ser usada 128bit) e k2 (mais à direita) (tamanho da mensagem de autenticação MAC de 128bit)
3. MaskedEncData = (message $\oplus$ K1) (ou AES(message,K1) ou AES(message,K2)
4. MacTag = HMAC_SHA1 (K2, MaskedEncData)
5. Enviar (Z || MaskedEncData || MacTag) = (V, C, T) = (publicKey/secret, AES, MAC)

O protocolo de decifra de uma mensagem é dado por:

1. Z = Diffie-Hellman Primitive
2. K = KDF2 (Z) (igual ao anterior)
3. Decifrar com o algoritmo AES (ou XOR ou Triple-DES)
4. MacTag1 = HMAC_SHA1 (K2, MaskedEncData)
5. Confirmar MacTag2 com a recebida

3.3 Assinaturas Digitais

As assinaturas digitais apareceram porque surgiu a necessidade de criar um mecanismo para certificar documentos. Esta necessidade surgiu devido à crescente vaga de insegurança. Desta maneira, pode-se garantir a integridade da mensagem (apesar de não ser absolutamente seguro, visto ser susceptível a ataques como por exemplo o ataque por data de nascimento). Desta maneira qualquer alteração dos dados é facilmente reconhecida.

Após a criação da mensagem, esta é passada por uma função de dispersão (como por exemplo o SHA ou MD5) e depois é cifrada com a chave pública do subscritor. As funções de dispersão são unidireccionais, ou seja, não é possível obter o resultado inicial.

Neste trabalho será implementada um tipo de assinatura digital denominada de ECDSA.

3.3.1 ECDSA

O algoritmo ECDSA [ECDSA, 05] é uma variante de assinatura digital que permite, recorrendo às curvas elípticas, o uso de chaves mais pequenas para o mesmo grau de segurança.

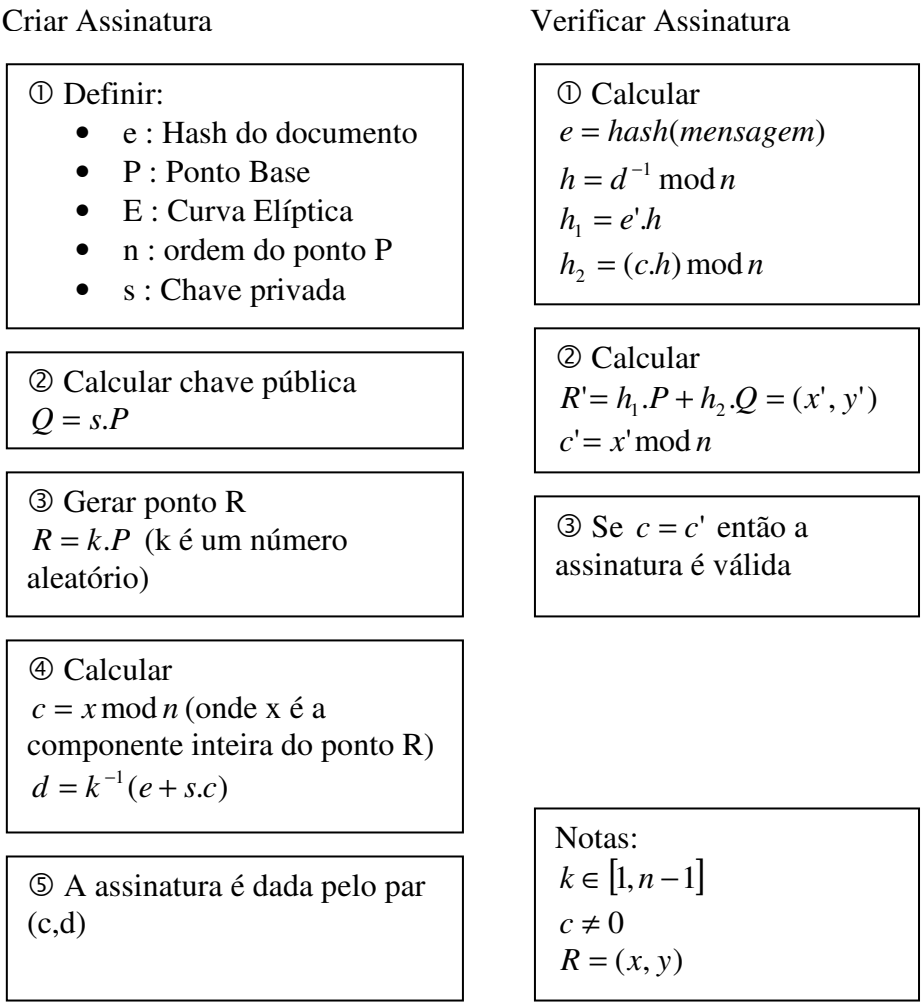
A assinatura é criada pelos seguintes passos:

1. É necessário definir os parâmetros do protocolo ECDSA, nomeadamente qual a função de dispersão a usar, a curva elíptica (tamanho da chave e o ponto base da curva) e a chave privada a usar (designada por s)
2. Calcular a chave pública $Q = s.P$
3. Calcular o ponto R dada pela expressão: $R = k.P$, onde k é um número aleatório.
4. Fazer o cálculo $c = x \bmod n$, onde x é a componente inteira do ponto R, e calcular $d = k^{-1}(e + s.c)$
5. Por fim, define-se a assinatura pelo par (c, d)

A Assinatura é verificada pelos seguintes passos:

1. Após a definição dos mesmos parâmetros usados na criação da assinatura, calcula-se os seguintes valores: $e = hash(mensagem)$, $h = d^{-1} \bmod n$, $h_1 = e'.h$ e por fim $h_2 = (c.h) \bmod n$
2. Com os valores obtidos no ponto anterior, calcula-se $R' = h_1.P + h_2.Q = (x', y')$ e $c' = x' \bmod n$
3. A assinatura do documento é válida se $c = c'$.

Graficamente, o protocolo está definido no diagrama **"Protocolo 2"**.



Protocolo 2: Algoritmo ECDSA

3.4 Notas sobre as implementações realizadas

Com este capítulo pretendeu-se explicar como funcionam os algoritmos que serão implementados na parte prática do trabalho. Deu-se a conhecer como são feitas as operações numa curva e que algoritmos são necessários a cada um dos protocolos de cifra e assinatura digital.

O próximo capítulo irá tratar de todos os aspectos práticos a ter em conta na execução da cifra. Será explicado a estrutura do código, requisitos e limitações físicas, assim como a descrição do programa e exemplos de aplicação.

Por falta de tempo, não foi possível implementar todos os protocolos que estavam previstos. Foram implementadas as primitivas básicas de operações artiméticas sobre elementos da curva elíptica (soma e produto) e o protocolo ECIES, ficando para futuros aperfeiçoamentos os protocolos ElGamal [ElGamal, 85], Menezes-Qu-Vanstone (MQV simples e MQV Hash) [MQV, 03] e a assinatura digital Nyberg-Rueppel [Nyberg&Rueppel, 99].

4. Implementação

O capítulo 4, destina-se a explicar toda a implementação prática do trabalho. Neste capítulo é feita uma breve descrição sobre a estrutura de classes usadas, tecnologias e parâmetros usados.

4.1 Ambiente de desenvolvimento

O sistema de segurança com base em curvas elípticas foi construído usando a linguagem de programação C# [csharp, 05] no ambiente de desenvolvimento *Visual Studio* 2005. Poderiam ser usadas também outras linguagens, como por exemplo o C++ [C++, 00] ou Java [Java, 98] mas a linguagem C# foi escolhida de forma a utilizar ao máximo as funcionalidades da plataforma *Framework*. Foram implementadas duas versões do programa mas com o mesmo núcleo, que são as classes utilizadas para cifrar/decifrar os dados. As duas versões existem visto que a tecnologia permitida no dispositivo móvel ser bastante limitativa e para efeitos de depuração (debug) foi criado um programa mais sofisticado que funciona exclusivamente no PC/Windows.

Para o software produzido para PC/WINDOWS, foi utilizada a *framework* 3.0 (apesar de ser apenas usadas funções existentes na versão 2), desenvolvido num PC HP D 530, com um processador *Intel*® *Pentium*® IV2.8GHz com 512MB de RAM. Este computador também foi usado para os testes, assim com um portátil (equipado com bluetooth e infra-vermelhos) ACER 4001 *TravelMate WLMi* com um processador Centrino *Pentium* M, 1.5GHz e 512 MB de RAM.

O programa para PDA foi desenvolvido também com a ferramenta Visual Studio 2005, com a componente "*Smart Device*" activa, utilizando o *Compact Framework* 2.0, compatível com o sistema operativo Windows CE 5. Para os testes, foi usado o PDA HTC p3600, processador *Samsung* 2442 a 400MHz com 64 MB de RAM e 128 MB de Flash.

Para além da representação do programa em diagrama UML (**Anexo 2**), é também interessante observar a estrutura do programa em camadas. A implementação é separada em três grandes blocos, como descrito na figura 3:

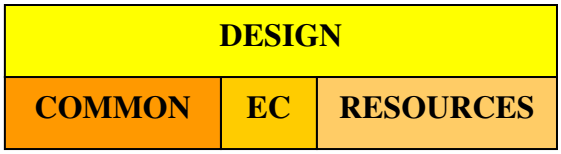


Figura 3: Estrutura em blocos do programa

- O bloco *Design* corresponde à parte do código que desenha o programa, assim como todas as instruções a nível de execução.
- O bloco *Common* a todo o código genérico que pode ser acrescentado novas funcionalidades (tornando o sistema o mais aberto possível).
- O bloco EC com funções específicas da cifra de curva elíptica que é previsto nunca ser alterada porque contém as operações básicas deste tipo de cifra.
- O bloco *Resources* apenas guarda todas as imagens necessárias para as apresentações gráficas do programa.

Esta divisão do programa por módulos permite dividir tarefas, facilitando o desenvolvimento. Sempre que for necessário fazer alguma alteração, basta apenas modificar o módulo correspondente.
As secções 4.2 e 4.3 descrevem cada bloco e as classes existentes em cada um deles.
Para desenvolver o programa, foi necessário numa primeira fase uma procura exaustiva de informação (que demorou dois a três meses, numa média de duas a três horas por dia) e uma segunda fase, a implementação propriamente dita num período de seis a sete meses com uma média de quatro horas diárias, esforço este feito pelo autor da tese.

4.2 Bloco Design

Neste bloco estão agrupados as funções que desenham o ambiente gráfico do programa (que diferem nas versões *PC/WINDOWS* e *PocketPC*, devido às suas características próprias).

4.2.1 Classe Form1

Esta classe agrupa os vários objectos que constroem o ambiente gráfico das duas versões. Contém também o código funcional que permite a selecção das várias opções. É importante portanto descrever os vários métodos/eventos mais relevantes:

- **Form1_Load**: Inicializa as variáveis globais usadas no âmbito do programa. Preenche as várias caixas de selecção com os dados dos ficheiros XML (protocolos, tipos de dados, tipo de conexão, etc.);
- **bttStart_Click**[*PC/WINDOWS*]/**picStart_Click**[*PocketPC*]: Inicializa o protocolo (variáveis necessárias, criação de chaves publicas e privadas). Notar que os parâmetros (tipo de base, numero de bits e tipo de protocolo) têm que estar já escolhidos antes da execução do programa. Caso contrário, o método *Start* não é inicializado, escrevendo um erro no Log (caso esteja activo esta opção);
- **bttEnd_Click**[*PC/WINDOWS*]/**picEnd_Click**[*PocketPC*]: Faz um *reset* ao estado do programa (elimina as opções como por exemplo, o tipo de base);
- **bttSend_Click**[*PC/WINDOWS*]/**picSend_Click**[*PocketPC*]: Transforma os dados numa cadeia de bytes e envia para o destino. Neste projecto assumiu-se que era necessário a troca de chaves neste método e não no Start.
- **bttReceive_Click**[*PC/WINDOWS*]/**picReceive_Click**[*PocketPC*]: Recebe um conjunto de dados e decifra-o (caso a opção *cipher* esteja ligada);

4.3 Bloco Common

Este bloco é composto por quatro partes, partes estas, que agrupam os aspectos variáveis do programa. É esta a parte de código aberto, para que se no futuro seja possível implementar outras funcionalidades (novas representações, protocolos de cifra, novos meios de comunicação), sendo assim relativamente fácil de modificar o código, não alterando o funcionamento base. Como se verifica na figura 4, este bloco é composto pelos sub blocos Base (uma classe de constantes), *Comunication* (responsável pelo envio/recepção de dados), *Protocol* (implementação dos vários protocolos de cifra propriamente ditos) e *Element* (um classe abstracta que serve de classe "mãe" a todas as implementações de representações/bases matemáticas):

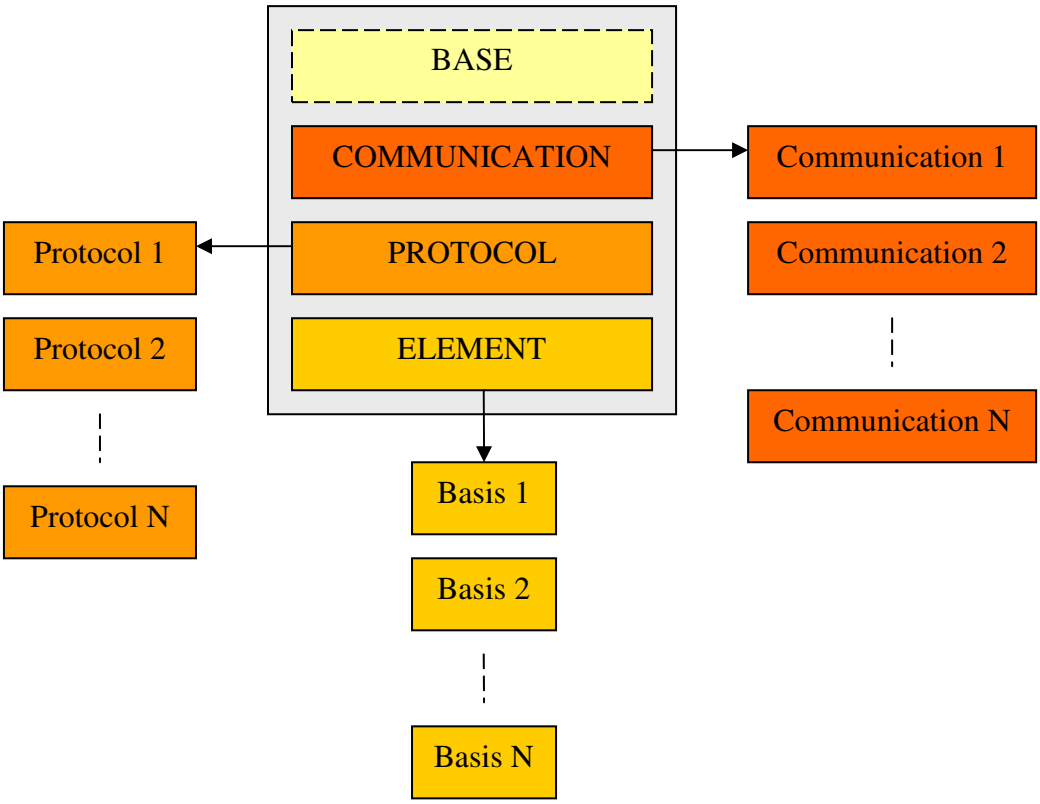


Figura 4: Esquemático do bloco *Common*

Nas secções 4.3.1-4.3.4, são descritos os métodos e características principais de cada classe, que basicamente são as implementações em código de algoritmo especificado nos capítulos anteriores.

4.3.1 Classe Base

A classe Base é uma das classes principal que guarda os parâmetros genéricos que são usados ao longo do programa. Contem as definições de funcionamento (TYPECOMMUNICATION, TYPEBASIS, TYPEPROTOCOL e TYPEDISPLAY) que são acedidos por diversos métodos e que são constantes ao logo do processo de cifra.

Nesta classe também estão presentes as várias constantes relativas a cada base. Cada constante é gerada sempre que é chamado o método *Initialize*, evocado quando se altera as opções (tipo de base).

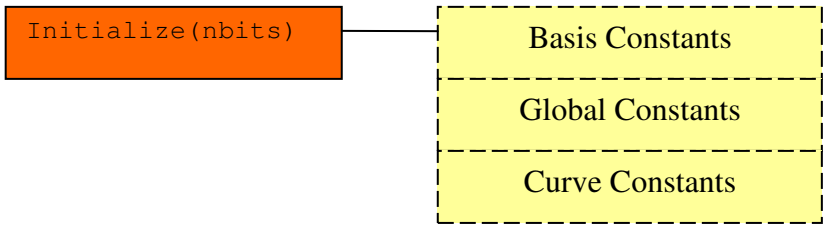


Figura 5: Esquemático da classe Base

4.3.2 Classe *Communication*

Classe responsável por enviar e receber uma cadeia de bytes. Contém dois métodos que podem ser chamados pelo programador: *Send* (envia) e o *Receive* (recebe). Com estes métodos, torna-se transparente o tipo de protocolo quando são chamados, porque internamente na classe é que são definidos os vários mecanismos de transferência de dados (por exemplo cabo, infravermelhos, *bluetooth* ou *Wi-fi*). É uma classe também abstracta, porque permite criar novos mecanismos de comunicação sem ter que mudar a interface e assim evitar alterações de fundo no programa.

Existe também as métodos *SendPublicKey* e *ReceivePublicKey*, que são métodos auxiliares troca de chaves em que no primeiro byte, é enviado o número de bits, de forma a evitar conflitos quando se tenta cifrar/decifrar os dados.

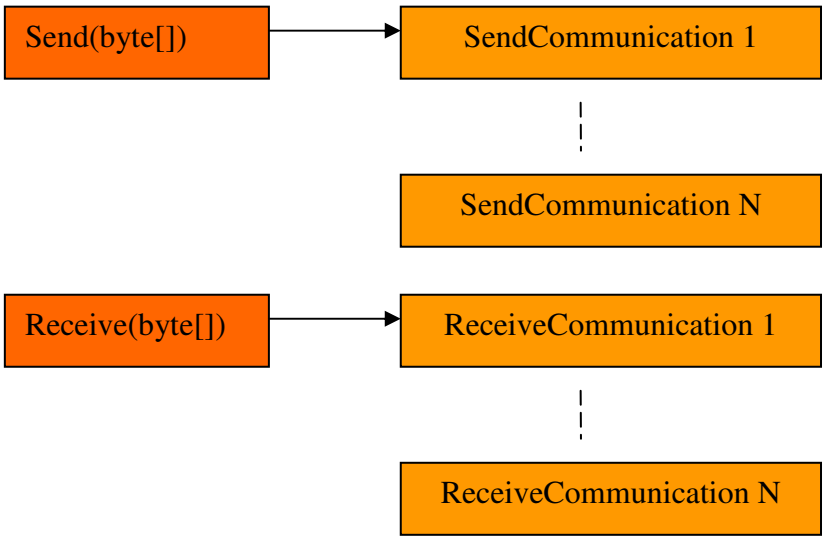


Figura 6: Esquemático da classe *Communication*

4.3.3 Comunicação Porta Série/USB

O módulo de envio/recepção por USB é diferente nas versões PC/WINDOWS e PocketPC, porque a comunicação por sockets que foi usada neste modulo só permite que o PC/WINDOWS actue como Servidor. Cada vez que é necessário enviar/receber é preciso criar um serviço Cliente-Servidor via socket, e enviar por protocolo TCP os dados. Para esta transação é necessário que os dois dispositivos (PC/WINDOWS e PocketPC) estejam ligados por cabo USB e no PC/WINDOWS estar instalado o programa Microsoft ActiveSync. A comunicação é feita como mostra a **Figura 7**:

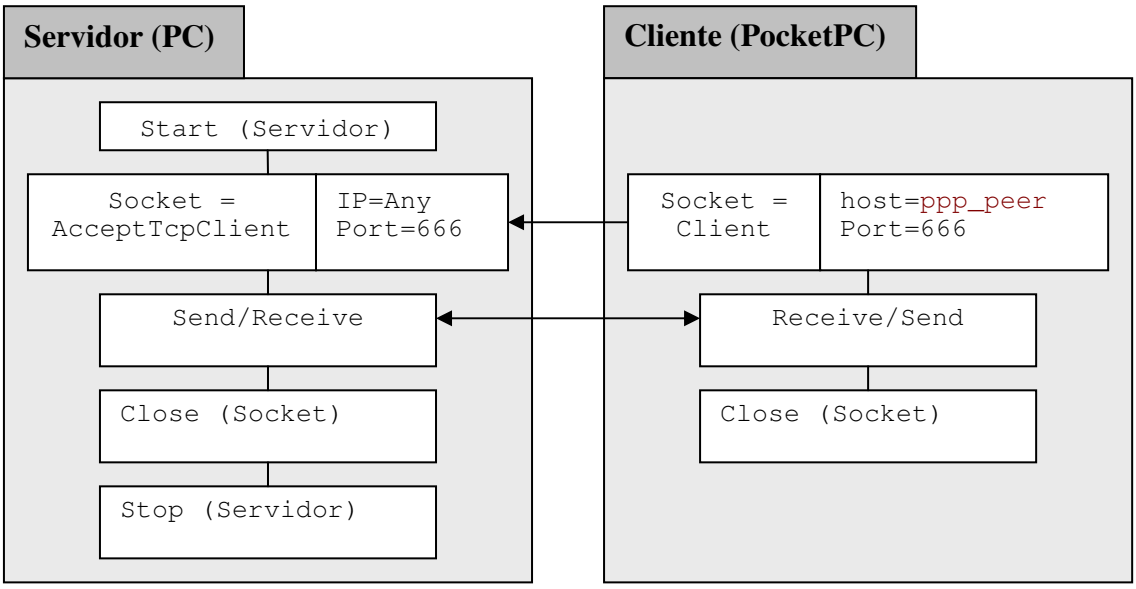


Figura 7: Comunicação USB entre PC/WINDOWS (Servidor) e PocketPC (Cliente)

Novamente, por falta de tempo, não foi possível a implementação dos protocolos de comunicação por infravermelhos e Bluetooth, apesar da aplicação estar preparada para essa utilização bastando implementar os métodos *Send* e *Receive* das respectivas classes.

4.3.3 Classe *Protocol*

Estão agrupados neste bloco todos os protocolos baseados em Curva Elíptica. Cada um está especificado no ficheiro config.xml e implementado numa classe. Existem interfaces comuns como o *encrypt* e o *decrypt*. São implementados também neste tipo de classes os algoritmos auxiliares necessários ao funcionamento de cada um. Esta é

uma classe abstracta que faz selecção dos vários algoritmos conforme o protocolo escolhido, tornando transparente para o utilizador/programador a implementação. Caso seja pretendido criar um novo protocolo, basta juntar a nova classe e introduzir no ficheiro config.xml.

4.3.3.1 Classe ECDF

Esta classe é responsável pela implementação do protocolo de troca de chaves Diffie-Hellman especificado no capítulo 3. Este protocolo também serve de base de troca de chaves para os protocolos ECIES e ECDSA implementados no trabalho .

Após especificado a base e o número de bits, é gerada a curva (pelas recomendações já estabelecidas pelas entidades competentes) e criadas a chave privada (que é um numero inteiro aleatório, 4 bytes que posteriormente é convertido num objecto *Element*) e a chave pública obtida através de $Q = d.B$ (onde P é a chave publica, K a chave privada e B o ponto base da curva). Depois são usadas as funções genéricas da classe *Communication* para enviar e receber as chaves entre as duas entidades.

4.3.3.2 Classe ECIES

O protocolo ECIES (*Elliptic Curve Integrated Encryption Scheme*) é implementado nesta classe. Nesta classe (que tem descrito o funcionamento no capítulo 4), também estão implementados os algoritmos HMAC_SHA1, ANSI_X9_93, Rijndael (Encrypt e Decrypt).

É preciso executar o método Initialize da classe (onde é criada as chaves e inicializado o algoritmo de Rijndael).

Como já tinha sido referenciado anteriormente, devido à falta de tempo, não foi possível implementar outro tipo de protocolos, como por exemplo o protocolo ElGamal ou MQV.

4.3.4 Classe *Element*

Esta classe é responsável pela criação de um elemento genérico numa representação abstracta. Esta classe abstracta contém todos os métodos que tem de ser implementados quando é criada uma nova representação. Por se abstracta, mantêm a abertura do código

a novas implementados, deixando novamente uma forma transparente de manusear os dados. Uma nova base deve estender a classe *Element* e implementar os vários métodos abstractos que foram definidos nesta classe para as operações serem usadas na classe *EllipticCurve* de forma genérica.

A descrição da classe é descrita na **Figura 8**:

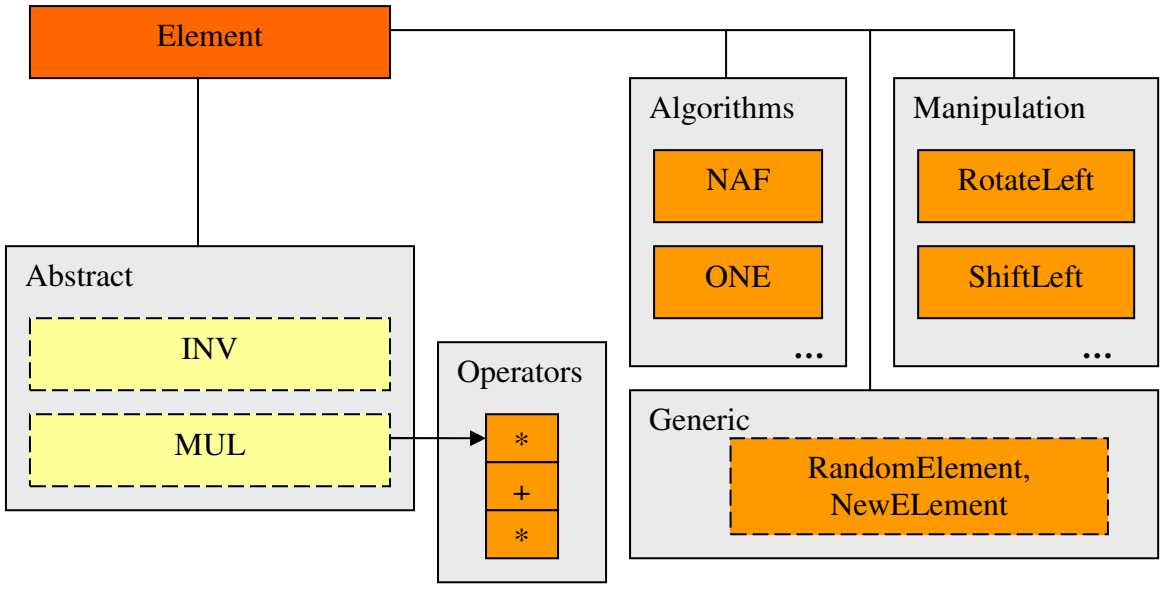


Figura 8: Esquemático da Classe Element

Para criar uma nova representação de base, basta estende-la da classe Element, definir os métodos abstractos de inversão e multiplicação. Como a soma (função lógica XOR) era igual nas duas bases implementadas, esta operação ficou definida na classe mãe. Todas os métodos de manipulação (rotação de bits, deslocamentos, etc.) e algoritmos (NAF, One que transforma todos os bits a um, etc.) ficam também definidos nesta classe.

4.3.4.1 Classe *Normal*

É nesta classe que estão definidas as operações baseadas na representação em base normal. Implementa as funções abstractas definidas na classe superior (Element) segundo os algoritmos descritos na tese.

4.3.4.2 Classe *Polynomial*

Classe responsável pela implementação dos elementos em representação polinomial.

4.4 Bloco EC

Este bloco funcional é, à partida um sistema fechado. Fazem parte deste bloco tudo o que seja relativo às operações sobre a curva elíptica. É composto pela directoria XML (ficheiros de configuração config.xml e ec_parameters.xml), pela classe EllipticCurve (onde estão descritas as operações de soma, dobro, e multiplicação de pontos sob a curva, assim como o método que embute um ponto sobre a curva), a classe Curve (que define uma curva com os elementos a e b descritos na secção anterior) e a classe Point (dois elementos que caracterizam uma coordenada no plano da curva).

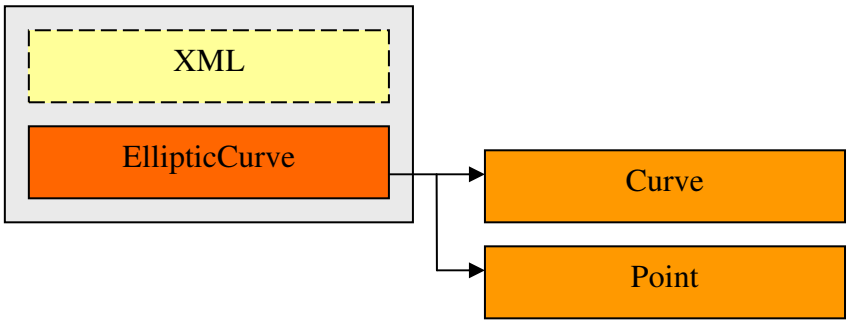


Figura 9: Esquemático do bloco EC

4.4.1 XML (config.xml e ec_parameters.xml)

Estes dois ficheiros em linguagem XML fazem com que o programa carregue automaticamente a sua configuração (composta pelos valores que podem ser escolhidos pelo utilizador, nomeadamente o número de bits, os protocolos, as bases matemáticas e de representação e o diferentes tipos de comunicação) e os parâmetros necessários à representação da curva elíptica no ficheiro ec_parameters.xml, parâmetros estes que baseiam-se na lista de recomendações que está em anexo.

4.4.2 Classe *EllipticCurve*

Esta classe serve apenas para especificar todas as operações efectuadas sob a curva elíptica. As operações aritméticas (soma, multiplicação, dobro), assim como a operação de embutir um ponto na curva definida como base, estão implementadas nesta classe.

4.4.2.1 Classe *Curve*

Classe representativa de uma curva elíptica. Visto estar a trabalhar no âmbito de programação por objectos, é necessário haver uma representação dos dados. Assim, surge a classe Curve que tem como atributos os parâmetros A e B da curva.

4.4.2.2 Classe *Point*

A classe Point representa um ponto da curva, definido pelas suas coordenadas sob o referencia cartesiano (x,y), sendo este dois elementos da classe Element.

4.5 Funcionamento

O programa, para além do *core* que corresponde aos blocos lógicos Common e EC, tem também definido um ambiente gráfico (GUI) que permite perceber facilmente como funcionam todos os algoritmos e de uma maneira mais simples, introduzir, validar, cifrar e decifrar dados.

Visto existir uma diferença significativa entre as duas plataformas PC/WINDOWS e PocketPC (existência de objectos como por exemplo RichTextBox que permite a adição de texto com cores que só existe na framework PC/WINDOWS), assim como uma diferença em termos de desempenho (processador, tamanho do monitor ou memória por exemplo), houve necessidade de criar dois ambientes gráficos, um para cada plataforma, permitindo assim um melhor supervisionamento da cifra de curva elíptica na versão PC.

De seguida surge a explicação detalhada de cada um dos ambientes, e um exemplo de aplicação genérico na secção 5.1. No próximo capítulo, são explicados os exemplos concretos que foram usados, desempenhos e conclusões que se podem tirar do trabalho.

4.5.1 Versão PC/WINDOWS

Como foi explicado na secção anterior, a plataforma Framework para PC/WINDOWS é bastante mais avançada e completa que a Compact (dispositivos móveis). Como a tela de um computador também é maior, permite ter um ambiente mais largo e apelativo que num PDA.

O programa é composto por duas áreas. A área “Main” e a área “Options”. A primeira, como pode ser visto na imagem 10 é a área onde estão os dados a serem enviados

(cifrados ou não) e os dados recebidos (novamente estes podem estar cifrados ou não). Os textos cifrados são compostos por partes coloridas respeitantes aos vários tipos de dados (chaves privadas, dados propriamente ditos e Tags de verificação) consoante o tipo de protocolo. Neste separador também estão descritos os parâmetros da curva a serem usados, e os botões Start (inicialização do protocolo após ter sido escolhido as várias opções), Send (envio de dados), Receive (recepção de dados) e o botão End (reset ao protocolo). Existe também a opção de enviar um ficheiro (em vez de dados escritos pelo utilizador). É também disponibilizado um campo de status que indica em que estado está o programa (start, send, etc.)

Legenda do separador Main (**Figura 10**):

1. Campos a serem enviados (texto plano e cifrado
2. Dados recebidos (texto cifrado e decifrado)
3. Parâmetros que descrevem a equação da curva
4. Polinómio irredutível
5. O tipo de base normal óptima (quando escolhido a base Normal)
6. Ponto Base da curva e chaves públicas e privadas
7. Botões de intervenção (iniciar, enviar, receber e parar)
8. Tipo de dados a enviar (caixa de texto ou ficheiro)
9. Nome do ficheiro a enviar (quando a opção File está activa)
10. Botão que abre o explorador de ficheiros
11. Caixa que descreve qual o processo que está a ser processado (envio, cifra, modo debug, entre outros)

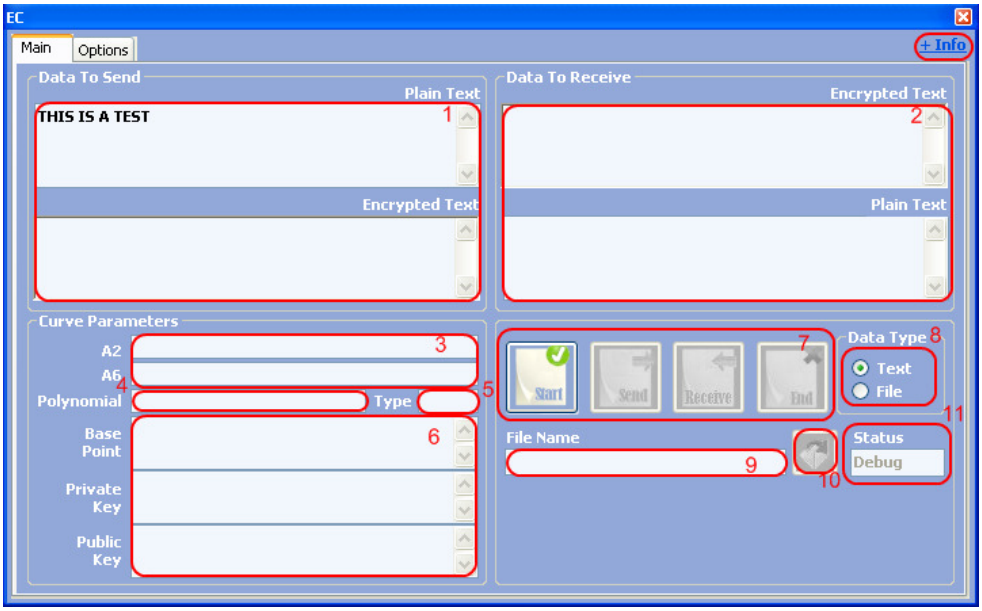


Figura 10: Separador Main

O segundo separador, é o separador de "Options" (Opções). É nele que é descrito todas as opções possíveis de serem alteradas pelo utilizador (tipo de protocolo, de comunicação, número de bits, tipo de display e tipo de representação). É também possível ver o Log do programa (caso esteja activo), assim como escolher se pretende cifrar dados ou se a ligação entre dispositivos seja síncrona. Uma imagem desta área está na **Figura 11**.

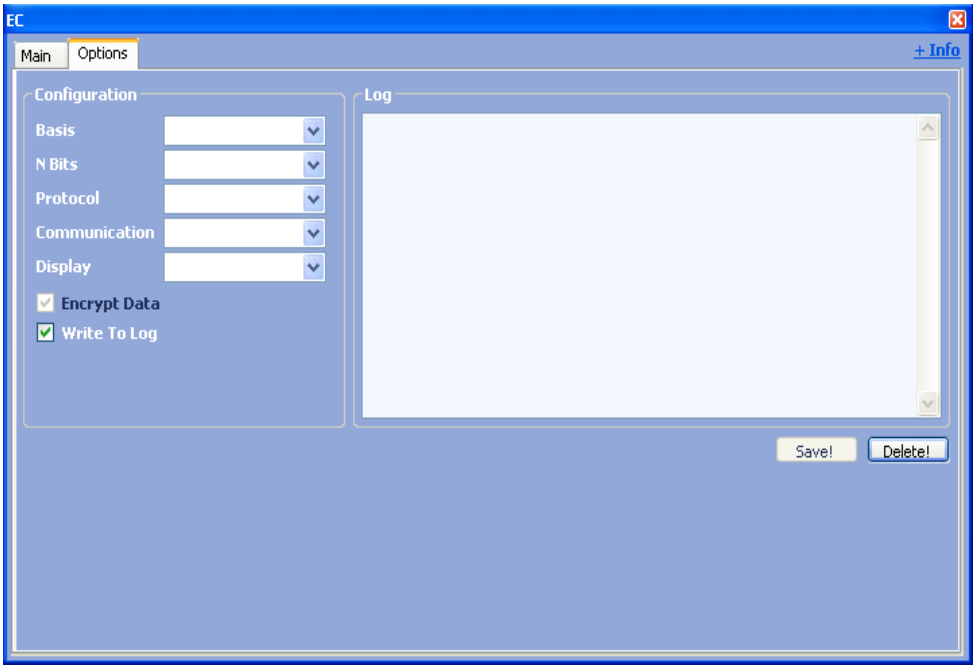


Figura 11: Separador Options

4.5.2 Versão PocketPC

A versão PocketPC é descrita em cinco separadores, descritos de seguida:

O primeiro, é o separador "Main". Este separador apenas consiste na visualização dos parâmetros da curva que são preenchidos após a escolha no separador "Options". É composto também pelos botões Start (inicialização do protocolo), Data Type (tipo de dados, texto ou ficheiro) e o botão End (fim de protocolo, também presente nos separadores Send e Receive).

O campo Type (tipo) só faz sentido quando a base de representação escolhida é a base normal. Os corpos PrivateKey e PublicKey só aparecem após o inicio (Start) do protocolo.

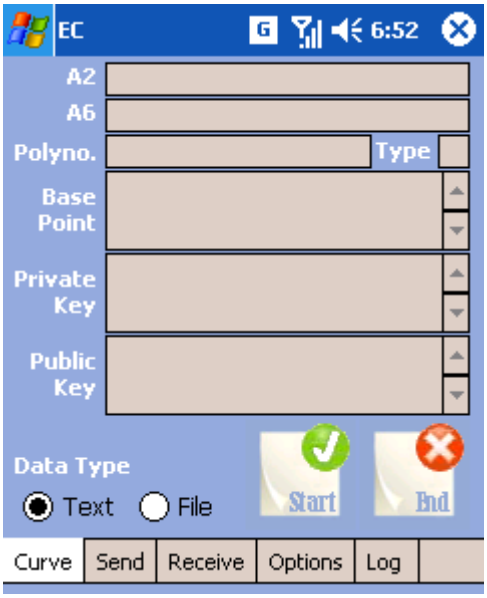


Figura 12 – Separador Main (PocketPC)

No separador Send, existe uma caixa de texto que permite escrever algo para ser enviado (cifrado). Após carregar no botão Start (separador Curve) é possível carregar no botão Send. Assim que for carregado no botão Send, surge na caixa "Encrypted Text" os dados cifrados (que pode ser um ficheiro, caso seja essa a opção do utilizador) e os mesmos são enviados ao destinatário que deverá carregar no botão receive (separador Receive).

A qualquer momento é possível interromper o processo de cifra, carregando para isso no botão End.

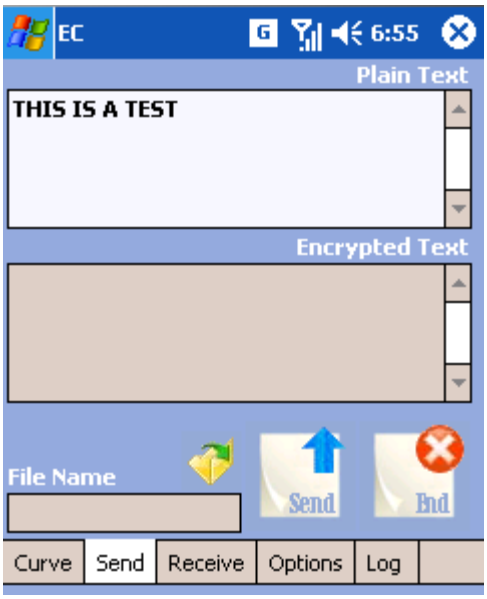


Figura 13 – Separador Send (PocketPC)

Para receber dados é necessário carregar no botão receive (separador receive) e esperar que os dados sejam carregados no ecrã. É também possível especificar um ficheiro para que seja guardado directamente os dados já decifrados. Também como no separador anterior existe o botão End.

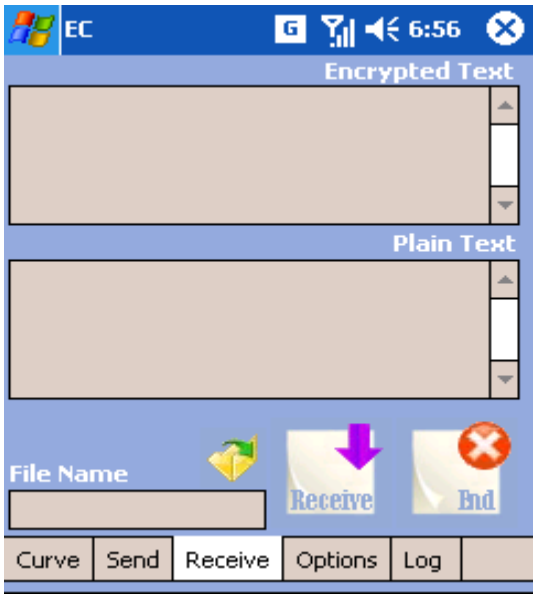


Figura 14 – Separador Receive (PocketPC)

Este separador é o local onde são definidos todos os parâmetros do programa. Inicialmente é necessário definir na caixa de selecção o tipo de base (Basis), o número de bits (N Bits) e qual o protocolo de cifra (Protocol).

É possível também especificar se o registo de acontecimentos está ou não activo (Write to Log), como se queremos enviar dados cifrados ou não (útil para enviar ficheiros entre dispositivos) ou se a comunicação será síncrona ou não.

O tipo de comunicação também é definido no separador "Options" assim como a representação dos dados (Decimal, hexadecimal ou binário).

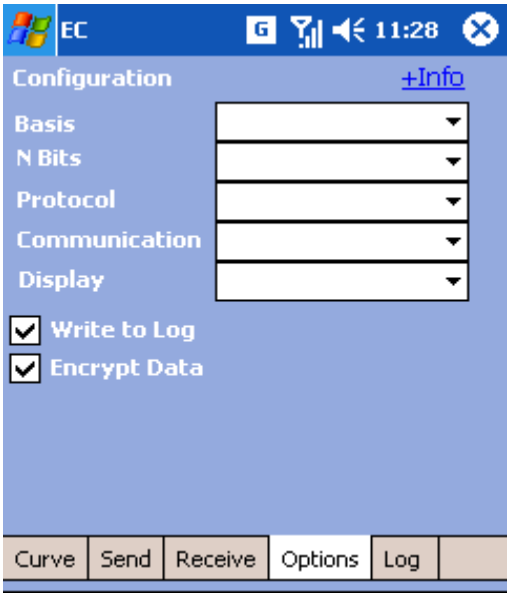


Figura 15 – Separador Options (PocketPC)

O Separador Log é apenas o registo de eventos. Regista todas as operações efectuadas ao longo do programa, desde inicializações de bases, envio ou recepção de dados

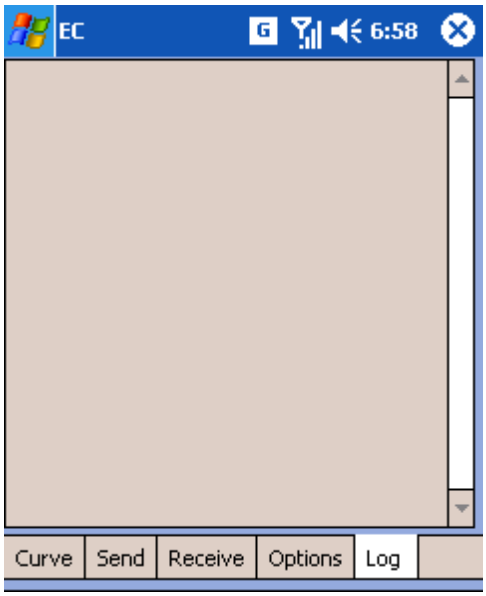


Figura 16 – Separador Options (PocketPC)

4.6 Contribuições

Para implementar o protocolo de assinatura digital ECDSA foi necessário recorrer a uma biblioteca que fornece um diverso conjunto de métodos para manipular números de grande dimensão, designados em inglês como "BigInteger". Esta classe está inserida no bloco lódico *Common*. O seu criador, Chew Keong TAN publicou esta biblioteca em regime de *open source* no portal http://www.codeproject.com não estando por isso fechada nem impedida de ser modificada.

4.7 Notas sobre a estrutura da implementação

Após a explicação funcional da implementação prática do trabalho, é necessário agora com dados concretos (partindo de cadeias de caracteres ou ficheiros), fazer testes mais exaustivos de maneira a mostrar que um solução de cifra por meio de curva elíptica é também uma opção viável em termos de segurança e rapidez num dispositivo móvel que à partida é mais limitado (em termos de desempenho) que um computador de secretária. No capítulo 5 serão apresentados os vectores de teste e as conclusões que se podem tirar após este trabalho.

5. Resultados e Conclusões

O último capítulo da tese, pretende mostrar que a aplicação da cifra de curva elíptica é de facto uma solução viável e que funciona correctamente num dispositivo móvel. Será dado um exemplo concreto de funcionamento, descrevendo passo a passo o seu funcionamento. Este procedimento será também comentado com registos de performance a nível de tempos de processamento para um determinado conjunto de informação (um ficheiro e uma cadeia de caracteres.

Na secção 5.2 descreve a possível continuação do trabalho, com ideias do que poderia ainda ser feito para completar o projecto.

5.1 Exemplo do funcionamento da Aplicação

Neste exemplo, será explicado passo a passo como cifrar/decifrar a cadeia de caracteres "THIS IS A TEST", enviando esta informação de um computador (PC/WINDOWS) para um PDA (PocketPC). As características físicas destes equipamentos são listadas na secção 4.1 .

Passo 1: Configuração

Antes de enviar qualquer tipo de dados é necessário primeiro estabelecer quais os parâmetros que vão ser usados na cifra. Tanto na plataforma *PC/WINDOWS* como no *PocketPC* é preciso indicar qual o tipo de Base de representação, o tamanho da chave e qual o protocolo a usar. Neste exemplo, serão usados os seguintes parâmetros, que são alterados no separador *Options* (PC/WINDOWS e PocketPC):

Basis: Polynomial

N Bits: 163, 233, 283 e 409

Protocol: ECIES

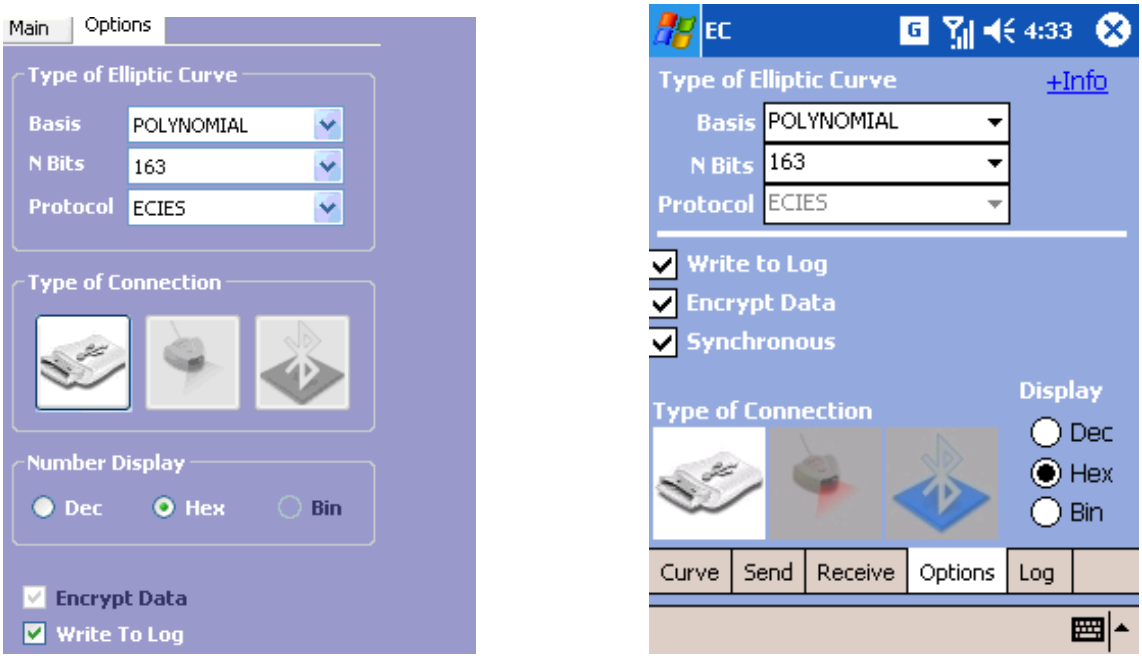


Figura 17: Alteração de Parâmetros (Versão PC/WINDOWS à esquerda e PocketPC à direita)

Passo 2: "Start"

Após escolher os parâmetros é necessário carregar no botão "Start" (separador Main em ambas as versões). Esta acção vai fazer com que sejam criadas as chaves públicas e privadas.

Passo 3: "Send" e "Receive"

Para enviar um ficheiro, o tipo "Data File" tem de estar seleccionada na opção "File" (Separador Main na versão PC/WINDOWS e separador Curve na versão pocketPC).

O dispositivo que envia o ficheiro(PC/WINDOWS), terá de carregar no botão "Send" (Separador Main") e o dispositivo móvel carrega no botão "Receive" (separador Receive). Estas acções fazem com que o ficheiro seja cifrado segundo o protocolo escolhido (Após o "Send") e decifrado no destinatário (acção "Receive"). Antes da troca de dados propriamente dita, existe primeiro a troca de chaves entre as duas entidades.

Passo 4: Visualização

Após a correcta troca de dados, o receptor decifra os dados que recebeu e apresenta na caixa de texto os bytes que recebeu e a informação que decifrou. Por ter sido um ficheiro a ser enviado, também é gravado na raiz do programa o mesmo ficheiro com o nome "file.txt".

5.2 Conclusões, desempenho e futuros melhoramentos

Pelos testes efectuados, tomando como referencia também o exemplo anterior, esta cifra pode ser implementada de facto num dispositivo móvel mas com alguma limitação, não tanto em termos de memória mas ao nível de processamento. A inicialização das operações é um processo moroso assim como a aplicação de uma cifra onde se utilize puramente as operações sob a curva elíptica. Na próxima tabela, é possível verificar os tempos que demora o processo de cifra e decifra para o algoritmo ECIES:

Tabela 1 (Tempos obtidos através do objecto DateTime)

		Portatil/Windows ($T_{Windows}$)	PocketPC/WinCE5 (T_{WinCE})	
		ECIES		$Tdif = \frac{T_{WinCE}}{T_{Windows}}$
Geração da chave publica	163	2212	43000	19,44
Cifrar Mensagem		2259	42000	18,59
Decifrar Mensagem		2171	42000	19,35
Total		6642	127000	19,12
Geração da chave publica	233	7207	149000	20,67
Cifrar Mensagem		7212	148000	20,52
Decifrar Mensagem		7148	148000	20,71
Total		21567	445000	20,63
Geração da chave publica	283	14149	242000	17,10
Cifrar Mensagem		13876	242000	17,44
Decifrar Mensagem		14054	243000	17,29
Total		42079	727000	17,28
Geração da chave publica	409	45523		
Cifrar Mensagem		45932		
Decifrar Mensagem		46002		
Total		137457		
		(Tempo em milissegundos)		**Média:** 19,01

Conforme seria de esperar, o desempenho dos algoritmos de curva elíptica no *PocketPC* é substancialmente inferior a um PC, à volta de 20 vezes mais lento.

Ainda existe um longo caminho a percorrer na optimização dos vários algoritmos, caminho esse que não foi possível percorrer neste trabalho devido à falta de tempo.
A implementação de novos algoritmos e de novos protocolos é possível tendo o mínimo de trabalho devido à estrutura da aplicação, que utiliza um nível de abstracção de modo a permitir futuros melhoramentos.
Quanto ao desempenho em si, a cifra para tamanhos de chaves superiores a 163 bits tornam o processo de cifra impraticáveis, visto que as operações de multiplicação (as mais recorrentes) tornam o processamento muito pesado.
É também importante referir que a maior parte do processamento dá-se quando são geradas as chaves, e não no processo de cifra propriamente dito (dados cifrados segundo o algoritmo AES). Como nos métodos *Encrypt* e *Decrypt* são calculadas as chaves partilhadas ($Chave_{Partilhada} = Chave_{privada} * Chave_{pública}$,) os tempos da geração da chave pública e destes métodos são semelhantes.
Em suma, a curva elíptica é uma solução viável para dispositivos com poder de processamento reduzido mas só se forem aplicados algoritmos que acelerem a cifra e toda a aplicação terá que estar optimizada para que se note essa fluidez no processo de cifrar/decifrar dados.
A maior dificuldade de eficiência na cifra está relacionada com a multiplicação num curva elíptica. Para um valor de K muito grande é necessário um elevado número de operações.

Referências:

[Koblitz & Miller, 85]: Victor S. Miller, "*Use of elliptic curves in cryptography*"; CRYPTO 85; 1985; Neal Koblitz, "*Elliptic curve cryptosystems*, in *Mathematics of Computation";* MIT ; 1987

[Rivest&Shamir&Adleman, 78]: R. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, 21(2), pp 120-126, Fev 1978

[Karatsuba-Ofman, 62]: A. Karatsuba and Yu Ofman, "*Multiplication of Many-Digital Numbers by Automatic Computers*"; Doklady Akad. Nauk SSSR; 1962

[LFSR, 97]: Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone; "Handbook of Applied Cryptography"; CRC Press; 1997

[Schroeppel, 99]: Çetin K. Koç, Christof Paar; "Cryptographic Hardware and Embedded Systems"; Springer; 1999

[Itoh-Tsujii, 88]: T. Itoh, S. Tsujii; "A Fast Algorithm for Computing Multiplicative Inverses in GF(2m) Using Normal Bases"; Information and Computation, 78, pp 171-177; 1988

[federal-government, 99]: "RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE"; 1999

[Daemen & Rijmen 02]: Joan Daemen, Vincenr Rijmen; "The Design of Rijndael: AES - The Advanced Encryption Standard"; Springer-Verlag; 2002

[IEEE P1363, 99]: IEEE P1363, "Standard Specifications for Public Key Cryptography: Additional Techniques"; IEEE; 1999

[RFC2104, 97]: H. Krawczyk, M. Bellare, R. Canetti; RFC 2014; 1997

[SHA, 95]: NIST, "FIPS PUB 180-1: Secure Hash Standard", April 1995

[RFC2898, 00]: B. Kaliski; "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898; IETF RSA; Sep 2000

[NAF, 06]: Roberto M. Avanzi, Henri Cohen, Gerhard Frey; "Handbook of Elliptic and Hyperelliptic Curve Cryptography"; CRC Press, 2006

[ECDH, 00]: Certicom Research, WStandards for efficient cryptography, SEC 1: Elliptic Curve Cryptography", 2000.

[ECIES, 01]: Victor Shoup, A proposal for an ISO standard for public key encryption; 2001

[3DES, 04]: NIST, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, 2004

[ECDSA, 05]: Daniel R. L. Brown, Generic Groups, Collision Resistance, and ECDSA, Designs, Codes and Cryptography, 2005

[ElGamal, 85]: Taher ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", Springer-Verlag, 1985

[MQV, 03]: Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, Scott A. Vanstone; "An Efficient Protocol for Authenticated Key Agreement", Cryptography 28(2): pp119–134, 2003.

[Nyberg&Rueppel, 99]: Masahiro Mambo, Yuliang Zheng; "Information Security: Second International Workshop", ISW'99; Springer, 1999

[csharp, 05]: Jesse Liberty, "Programming C#", O'Reilly, 2005

[Magalhães, 01]: Luís T. Magalhães; "Álgebra Linear como Introdução a Matemática Aplicada"; Texto Editora; 2001

[Solinas, 98]: Jerome A. Solinas; "Na Improved Algorithm for Arithmetic on Family of Elliptic Curve"; National Security Agency; 1998

[Hankerson&Menezes&Vanstone, 04]: Darrel HanKerson, Alfred Menezes, Scott Vanstone; "Guide to Elliptic Curve Cryptography"; Springer; 2004

[Choi&Kim&Lee, 05]: Yong-Je Choi, Moo-Seop Kim, Hang-Rok Lee and Ho-Won Kim ; "Implementation and Analysis of Elliptic Curve Cryptosystems over Polynomial basis and ONB"; Worlds Enformatika Society; 2005

[Rosing, 99]: Michael Rosing; "Implementing Elliptic Curve Cryptography"; Manning Publications Co; 1999

[Knuth, 98]: D. E. Knuth; "Seminumerical Algorithms: The Art of Computer Programming Vol. 2"; Addison-Wesley; 1998

[SSH, 05]: Daniel J. Barrett, Richard E. Silverman, Robert; "SSH, The Secure Shell: The Definitive Guide"; O'Reilly; 2005

[Stallings, 03]: William Stallings; "Cryptography and Network Security: principles and practices"; Prentice-Hall; 2003

[C++, 00]: Bjarne Stroustrup; "The C++ Programming Language"; Addison-Wesley; 2000

[Java, 98]: Ken Arnold, James Gosling; “The Java Programming Language”; Addison-Wesley; 1998

Apontadores

1. http://www.google.com
2. http://www.codeproject.com
3. http://www.msdn.com
4. http://www.wikipedia.org
5. http://www.msdn.microsoft.com
6. http://www.32feet.net
7. http://www.library.umaine.edu/theses
8. http://www-fs.informatik.uni-tuebingen.de/~reinhard/krypto/English/2.e.html
9. http://www.dragongate-technologies.com

Anexo 1

RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE

July 1999

This collection of elliptic curves is recommended for Federal government use and contains choices of private key length and underlying fields.

1. Parameter Choices

1.1 Choice of Key Lengths

The principal parameters for elliptic curve cryptography are the elliptic curve E and a designated point G on E called the *base point*. The base point has order r, a large prime. The number of points on the curve is $n = fr$ for some integer f (the *cofactor*) not divisible by r. For efficiency reasons, it is desirable to take the cofactor to be as small as possible.
All of the curves given below have cofactors 1, 2, or 4. As a result, the private and public keys are approximately the same length. Each length is chosen to correspond to the cryptovariable length of a common symmetric cryptologic. In each case, the private key length is, at least, approximately twice the symmetric cryptovariable length.

1.2 Choice of Underlying Fields

For each cryptovariable length, there are given two kinds of fields.

- A *prime field* is the field $GF(p)$ which contains a prime number p of elements. The elements of this field are the integers modulo p, and the field arithmetic is implemented in terms of the arithmetic of integers modulo p.
- A *binary field* is the field $GF(2^m)$ which contains 2^m elements for some m (called the *degree* of the field). The elements of this field are the bit strings of length m, and the field arithmetic is implemented in terms of operations on the bits.

The following table gives the sizes of the various underlying fields. By ||p|| is meant the length of the binary expansion of the integer p.

Symmetric CV Length	Example Algorithm	Prime Field	Binary Field
80	SKIPJACK	$\|\|p\|\| = 192$	$m = 163$
112	Triple-DES	$\|\|p\|\| = 224$	$m = 233$
128	AES Small	$\|\|p\|\| = 256$	$m = 283$
192	AES Medium	$\|\|p\|\| = 384$	$m = 409$
256	AES Large	$\|\|p\|\| = 521$	$m = 571$

1.3 Choice of Basis

To describe the arithmetic of a binary field, it is first necessary to specify how a bit string is to be interpreted. This is referred to as choosing a *basis* for the field. There are two common types of bases: a *polynomial basis* and a *normal basis*.

- A polynomial basis is specified by an irreducible polynomial modulo 2, called the *field polynomial*. The bit string $(a_{m-1} \ldots a_2\ a_1\ a_0)$ is taken to represent the polynomial
$$a_{m-1}t^{m-1} + \ldots + a_2 t^2 + a_1 t + a_0$$
over $GF(2)$. The field arithmetic is implemented as polynomial arithmetic modulo $p(t)$, where $p(t)$ is the field polynomial.
- A normal basis is specified by an element θ of a particular kind. The bit string $(a_0\ a_1\ a_2\ \ldots\ a_{m-1})$ is taken to represent the element
$$a_0\theta + a_1\theta^2 + a_2\theta^{2^2} + a_{m-1}\theta^{2^{m-1}}.$$
Normal basis field arithmetic is not easy to describe or efficient to implement in general, but is for a special class called *Type T low-complexity* normal bases. For a given field degree m, the choice of T specifies the basis and the field arithmetic (see Appendix 2).

There are many polynomial bases and normal bases from which to choose. The following procedures are commonly used to select a basis representation.

- *Polynomial Basis*: If an irreducible *trinomial* $t^m + t^k + 1$ exists over $GF(2)$, then the field polynomial $p(t)$ is chosen to be the irreducible trinomial with the lowest-degree middle term t^k. If no irreducible trinomial exists, then one selects instead a *pentanomial* $t^m + t^a + t^b + t^c + 1$. The particular pentanomial chosen has the following properties: the second term t^a has the lowest degree m; the third term t^b has the lowest degree among all irreducible pentanomials of degree m and second term t^a; and the fourth term t^c has the lowest degree among all irreducible pentanomials of degree m, second term t^a, and third term t^b.
- *Normal Basis*: Choose the Type T low-complexity normal basis with the smallest T.

For each binary field, the parameters are given for the above basis representations.

1.4 Choice of Curves

Two kinds of curves are given:

- *Pseudo-random* curves are those whose coefficients are generated from the output of a seeded cryptographic hash. If the seed value is given along with the coefficients, it can be verified easily that the coefficients were indeed generated by that method.
- *Special curves* whose coefficients and underlying field have been selected to optimize the efficiency of the elliptic curve operations.

For each size, the following curves are given:

→ A pseudo-random curve over $GF(p)$.

→ A pseudo-random curve over $GF(2^m)$.

→ A special curve over $GF(2^m)$ called a *Koblitz curve* or *anomalous binary curve*.

The pseudo-random curves are generated via the SHA-1 based method given in the ANSI X9.62 and IEEE P1363 standards. (The generation and verification processes are given in Appendices 4 through 7.)

1.5 Choice of Base Points

Any point of order r can serve as the base point. Each curve is supplied with a sample base point $G = (G_x , G_y)$. Users may want to generate their own base points to ensure cryptographic separation of networks.

2. Curves over Prime Fields

For each prime p, a pseudo-random curve

$$E : \; y^2 \equiv x^3 - 3x + b \; (mod \; p)$$

of prime order r is listed[1]. (Thus, for these curves, the cofactor is always $f = 1$.) The following parameters are given:

- The prime modulus p
- The order r
- the 160-bit input seed s to SHA-1 based algorithm
- The output c of the SHA-1 based algorithm
- The coefficient b (satisfying $b^2 c \equiv -27 \; (mod \; p)$)
- The base point x coordinate G_x
- The base point y coordinate G_y

The integers p and r are given in decimal form; bit strings and field elements are given in hex.

Curve P-192

p = 6277101735386680763835789423207666416083908700390324961279

r = 6277101735386680763835789423176059013767194773182842284081

s = 3045ae6f c8422f64 ed579528 d38120ea e12196d5

c = 3099d2bb bfcb2538 542dcd5f b078b6ef 5f3d6fe2 c745de65

b = 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1

G_x = 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012

G_y = 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811

Curve P-224

p = 26959946667150639794667015087019630673557916260026308143510066298881

r = 26959946667150639794667015087019625940457807714424391721682722368061

s = bd713447 99d5c7fc dc45b59f a3b9ab8f 6a948bc5

c = 5b056c7e 11dd68f4 0469ee7f 3c7a7d74 f7d12111 6506d031 218291fb

b = b4050a85 0c04b3ab f5413256 5044b0b7 d7bfd8ba 270b3943 2355ffb4

G_x = b70e0cbd 6bb4bf7f 321390b9 4a03c1d3 56c21122 343280d6 115c1d21

[1] The selection $a \equiv -3$ for the coefficient of x was made for reasons of efficiency; see IEEE P1363.

G_y = bd376388 b5f723fb
4c22dfe6 cd4375a0 5a074764 44d58199 85007e34

Curve P-256

p = 115792089210356248762697446949407573530086143415290314195533631308867097853951

r = 115792089210356248762697446949407573529996955224135760342422259061068512044369

s = c49d3608 86e70493 6a6678e1 139d26b7 819f7e90

c = 7efba166 2985be94 03cb055c
75d4f7e0 ce8d84a9 c5114abc af317768 0104fa0d

b = 5ac635d8 aa3a93e7 b3ebbd55
769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b

G_x = 6b17d1f2 e12c4247 f8bce6e5
63a440f2 77037d81 2deb33a0 f4a13945 d898c296

G_y = 4fe342e2 fe1a7f9b 8ee7eb4a
7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5

Curve P-384

p = 39402006196394479212279040100143613805079739270465446667948293404245721771496870329047266088258938001861606973112319

r = 39402006196394479212279040100143613805079739270465446667946905279627659399113263569398956308151229491355443365394264 3

s = a335926a a319a27a 1d00896a 6773a482 7acdac73

c = 79d1e655 f868f02f
ff48dcde e14151dd b80643c1 406d0ca1 0dfe6fc5
2009540a 495e8042 ea5f744f 6e184667 cc722483

b = b3312fa7 e23ee7e4
988e056b e3f82d19 181d9c6e fe814112 0314088f
5013875a c656398d 8a2ed19d 2a85c8ed d3ec2aef

G_x = aa87ca22 be8b0537
8eb1c71e f320ad74 6e1d3b62 8ba79b98 59f741e0
82542a38 5502f25d bf55296c 3a545e38 72760ab7

G_y = 3617de4a 96262c6f
5d9e98bf 9292dc29 f8f41dbd 289a147c e9da3113
b5f0b8c0 0a60b1ce 1d7e819d 7a431d7c 90ea0e5f

Curve P-521

p = 6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151

r = 6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449

s = d09e8800 291cb853 96cc6717 393284aa a0da64ba

c = 0b4 8bfa5f42
0a349495 39d2bdfc 264eeeeb 077688e4 4fbf0ad8
f6d0edb3 7bd6b533 28100051 8e19f1b9 ffbe0fe9
ed8a3c22 00b8f875 e523868c 70c1e5bf 55bad637

b = 051 953eb961
8e1c9a1f 929a21a0 b68540ee a2da725b 99b315f3
b8b48991 8ef109e1 56193951 ec7e937b 1652c0bd
3bb1bf07 3573df88 3d2c34f1 ef451fd4 6b503f00

G_x = c6 858e06b7
0404e9cd 9e3ecb66 2395b442 9c648139 053fb521
f828af60 6b4d3dba a14b5e77 efe75928 fe1dc127
a2ffa8de 3348b3c1 856a429b f97e7e31 c2e5bd66

G_y = 118 39296a78
9a3bc004 5c8a5fb4 2c7d1bd9 98f54449 579b4468
17afbd17 273e662c 97ee7299 5ef42640 c550b901
3fad0761 353c7086 a272c240 88be9476 9fd16650

3. Curves over Binary Fields

For each field degree m, a pseudo-random curve is given, along with a Koblitz curve. The pseudo-random curve has the form

$$E: y^2 + x\,y = x^3 + x^2 + b,$$

and the Koblitz curve has the form

$$E_a: y^2 + x\,y = x^3 + ax^2 + 1$$

where $a = 0$ or 1.

For each pseudorandom curve, the cofactor is $f = 2$. The cofactor of each Koblitz curve is $f = 2$ if $a = 1$ and $f = 4$ if $a = 0$.

The coefficients of the pseudo-random curves, and the coordinates of the base points of both kinds of curves, are given in terms of both the polynomial and normal basis representations discussed in 1.3.

For each m, the following parameters are given:

Field Representation:

- The normal basis type T
- The field polynomial (a trinomial or pentanomial)

Koblitz Curve:

- The coefficient a
- The base point order r
- The base point x coordinate G_x
- The base point y coordinate G_y

Pseudo-random curve:

- The base point order r

Pseudo-random curve (Polynomial Basis representation):

- The coefficient b
- The base point x coordinate G_x
- The base point y coordinate G_y

Pseudo-random curve (Normal Basis representation):

- The 160-bit input seed s to the SHA-1 based algorithm
- The coefficient b (i.e., the output of the SHA-1 based algorithm)

- The base point x coordinate G_x
- The base point y coordinate G_y

Integers (such as T, m, and r) are given in decimal form; bit strings and field elements are given in hex.

Degree 163 Binary Field

$T = 4$

$p(t) = t^{163} + t^7 + t^6 + t^3 + 1$

Curve K-163

$a = 1$

$r = 5846006549323611672814741753598448348329118574063$

Polynomial Basis:

$G_x =$ 2 fe13c053 7bbc11ac aa07d793 de4e6d5e 5c94eee8

$G_y =$ 2 89070fb0 5d38ff58 321f2e80 0536d538 ccdaa3d9

Normal Basis:

$G_x =$ 0 5679b353 caa46825 fea2d371 3ba450da 0c2a4541

$G_y =$ 2 35b7c671 00506899 06bac3d9 dec76a83 5591edb2

Curve B-163

r = 5846006549323611672814742442876390689256843201587

Polynomial Basis:

b = 2 0a601907 b8c953ca 1481eb10 512f7874 4a3205fd

G_x = 3 f0eba162 86a2d57e a0991168 d4994637 e8343e36

G_y = 0 d51fbc6c 71a0094f a2cdd545 b11c5c0c 797324f1

Normal Basis:

s = 85e25bfe 5c86226c db12016f 7553f9d0 e693a268

b = 6 645f3cac f1638e13 9c6cd13e f61734fb c9e3d9fb

G_x = 0 311103c1 7167564a ce77ccb0 9c681f88 6ba54ee8

G_y = 3 33ac13c6 447f2e67 613bf700 9daf98c8 7bb50c7f

Degree 233 Binary Field

T = 2

$p(t) = t^{233} + t^{74} + 1$

Curve K-233

a = 0

r =3450873173395281893717377931138512760570940988862252 1\
26328087024741343

Polynomial Basis:

G_x = 172 32ba853a 7e731af1 29f22ff4 149563a4 19c26bf5 0a4c9d6e efad6126

G_y = 1db 537dece8 19b7f70f 555a67c4 27a8cd9b f18aeb9b 56e0c110 56fae6a3

Normal Basis:

G_x = 0fd e76d9dcd 26e643ac 26f1aa90 1aa12978 4b71fc07 22b2d056 14d650b3

G_y = 064 3e317633 155c9e04 47ba8020 a3c43177 450ee036 d6335014 34cac978

Curve B-233

r = 6901746346790563787434755862277025555839812737345013 5\
55379383634485463

Polynomial Basis:

b = 066 647ede6c 332c7f8c 0923bb58 213b333b 20e9ce42 81fe115f 7d8f90ad

G_x = 0fa c9dfcbac 8313bb21 39f1bb75 5fef65bc 391f8b36 f8f8eb73 71fd558b

G_y = 100 6a08a419 03350678 e58528be bf8a0bef f867a7ca 36716f7e 01f81052

Normal Basis:

s = 74d59ff0 7f6b413d 0ea14b34 4b20a2db 049b50c3

b = 1a0 03e0962d 4f9a8e40 7c904a95 38163adb 82521260 0c7752ad 52233279

G_x = 18b 863524b3 cdfefb94

G_y = f2784e0b 116faac5 4404bc91 62a363ba b84a14c5
049 25df77bd 8b8ff1a5
ff519417 822bfedf 2bbd7526 44292c98 c7af6e02

Degree 283 Binary Field

$T = 6$

$p(t) = t^{283} + t^{12} + t^7 + t^5 + 1$

Curve K-283

$a = 0$

r = 3885337784451458141838923813647037813284811733793061324295874997529815829704422603873

Polynomial Basis:

G_x = 503213f 78ca4488 3f1a3b81 62f188e5
53cd265f 23c1567a 16876913 b0c2ac24 58492836

G_y = 1ccda38 0f1c9e31 8d90f95d 07e5426f
e87e45c0 e8184698 e4596236 4e341161 77dd2259

Normal Basis:

G_x = 3ab9593 f8db09fc 188f1d7c 4ac9fcc3
e57fcd3b db15024b 212c7022 9de5fcd9 2eb0ea60

G_y = 2118c47 55e7345c d8f603ef 93b98b10
6fe8854f feb9a3b3 04634cc8 3a0e759f 0c2686b1

Curve B-283

r = 3890919652677004427778737869287177706755689029162836778476272940756265696259243769048

Polynomial Basis:

b = 27b680a c8b8596d a5a4af8a 19a0303f
ca97fd76 45309fa2 a581485a f6263e31 3b79a2f5

G_x = 5f93925 8db7dd90 e1934f8c 70b0dfec
2eed25b8 557eac9c 80e2e198 f8cdbecd 86b12053

G_y = 3676854 fe24141c b98fe6d4 b20d02b4
516ff702 350eddb0 826779c8 13f0df45 be8112f4

Normal Basis:

s = 77e2b073 70eb0f83 2a6dd5b6 2dfc88cd 06bb84be

b = 157261b 894739fb 5a13503f 55f0b3f1
0c560116 66331022 01138cc1 80c0206b dafbc951

G_x = 749468e 464ee468 634b21f7 f61cb700
701817e6 bc36a236 4cb8906e 940948ea a463c35d

G_y = 62968bd 3b489ac5 c9b859da 68475c31 5bafcdc4 ccd0dc90 5b70f624
46f49c05 2f49c08c

Degree 409 Binary Field

$T = 4$

$p(t) = t^{409} + t^{87} + 1$

Curve K-409

$a = 0$

$r = 3305279843951242994759576540163855199142023414821406\backslash$
$9642324395022880711289249191050673258457777458014096\backslash$
366590617731358671

Polynomial Basis:

$G_x =$ 060f05f 658f49c1 ad3ab189
0f718421 0efd0987 e307c84c 27accfb8 f9f67cc2
c460189e b5aaaa62 ee222eb1 b35540cf e9023746

$G_y =$ 1e36905 0b7c4e42 acba1dac
bf04299c 3460782f 918ea427 e6325165 e9ea10e3
da5f6c42 e9c55215 aa9ca27a 5863ec48 d8e0286b

Normal Basis:

$G_x =$ 1b559c7 cba2422e 3affe133
43e808b5 5e012d72 6ca0b7e6 a63aeafb c1e3a98e
10ca0fcf 98350c3b 7f89a975 4a8e1dc0 713cec4a

$G_y =$ 16d8c42 052f07e7 713e7490
eff318ba 1abd6fef 8a5433c8 94b24f5c 817aeb79
852496fb ee803a47 bc8a2038 78ebf1c4 99afd7d6

Curve B-409

$r = 661055968790248598951915308032771039828404682964281\backslash$
$9284648798304157774827374805208143723762179110965979\backslash$
867288366567526771

Polynomial Basis:

$b =$ 021a5c2 c8ee9feb 5c4b9a75
3b7b476b 7fd6422e f1f3dd67 4761fa99 d6ac27c8
a9a197b2 72822f6c d57a55aa 4f50ae31 7b13545f

$G_x =$ 15d4860 d088ddb3 496b0c60
64756260 441cde4a f1771d4d b01ffe5b 34e59703
dc255a86 8a118051 5603aeab 60794e54 bb7996a7

$G_y =$ 061b1cf ab6be5f3 2bbfa783
24ed106a 7636b9c5 a7bd198d 0158aa4f 5488d08f
38514f1f df4b4f40 d2181b36 81c364ba 0273c706

Normal Basis:

$s =$ 4099b5a4 57f9d69f 79213d09 4c4bcd4d 4262210b

$b =$ 124d065 1c3d3772 f7f5a1fe
6e715559 e2129bdf a04d52f7 b6ac7c53 2cf0ed06
f610072d 88ad2fdc c50c6fde 72843670 f8b3742a

$G_x =$ 0ceacbc 9f475767 d8e69f3b
5dfab398 13685262 bcacf22b 84c7b6dd 981899e7
318c96f0 761f77c6 02c016ce d7c548de 830d708f

$G_y =$ 199d64b a8f089c6 db0e0b61
e80bb959 34afd0ca f2e8be76 d1c5e9af fc7476df
49142691 ad303902 88aa09bc c59c1573 aa3c009a

Degree 571 Binary Field

$T = 10$

$p(t) = t^{571} + t^{10} + t^5 + t^2 + 1$

Curve K-571

$a = 0$

$r = 1932268761508629172347675945465993672149463664853217 4\backslash$
$9932861762572575957114478021226813397852270671183470 6\backslash$
$7128008253514612736749740666173119296824216170925035 5\backslash$
5733685276673

Polynomial Basis:

$G_x =$ 26eb7a8 59923fbc 82189631
f8103fe4 ac9ca297 0012d5d4 60248048 01841ca4
43709584 93b205e6 47da304d b4ceb08c bbd1ba39
494776fb 988b4717 4dca88c7 e2945283 a01c8972

$G_y =$ 349dc80 7f4fbf37 4f4aeade
3bca9531 4dd58cec 9f307a54 ffc61efc 006d8a2c
9d4979c0 ac44aea7 4fbebbb9 f772aedc b620b01a
7ba7af1b 320430c8 591984f6 01cd4c14 3ef1c7a3

Normal Basis:

$G_x =$ 04bb2db a418d0db 107adae0
03427e5d 7cc139ac b465e593 4f0bea2a b2f3622b
c29b3d5b 9aa7a1fd fd5d8be6 6057c100 8e71e484
bcd98f22 bf847642 37673674 29ef2ec5 bc3ebcf7

$G_y =$ 44cbb57 de20788d 2c952d7b
56cf39bd 3e89b189 84bd124e 751ceff4 369dd8da
c6a59e6e 745df44d 8220ce22 aa2c852c fcbbef49
ebaa98bd 2483e331 80e04286 feaa2530 50caff60

Curve B-571

$r = 3864537523017258344695351890931987344298927329706434 9\backslash$
$9865723525145151914228956042453614399938941577308313 3\backslash$
$8811219269444862468724628168130702345282883033324113 9\backslash$
3191105285703

Polynomial Basis:

$b =$ 2f40e7e 2221f295 de297117
b7f3d62f 5c6a97ff cb8ceff1 cd6ba8ce 4a9a18ad
84ffabbd 8efa5933 2be7ad67 56a66e29 4afd185a
78ff12aa 520e4de7 39baca0c 7ffeff7f 2955727a

$G_x =$ 303001d 34b85629 6c16c0d4
0d3cd775 0a93d1d2 955fa80a a5f40fc8 db7b2abd
bde53950 f4c0d293 cdd711a3 5b67fb14 99ae6003
8614f139 4abfa3b4 c850d927 e1e7769c 8eec2d19

$G_y =$ 37bf273 42da639b 6dccfffe
b73d69d7 8c6c27a6 009cbbca 1980f853 3921e8a6
84423e43 bab08a57 6291af8f 461bb2a8 b3531d2f
0485c19b 16e2f151 6e23dd3c 1a4827af 1b8ac15b

Normal Basis:

$s =$ 2aa058f7 3a0e33ab 486b0f61 0410c53a 7f132310

$b =$ 3762d0d 47116006 179da356
88eeaccf 591a5cde a7500011 8d9608c5 9132d434

G_x = 26101a1d fb377411 5f586623 f75f0000 1ce61198
3c1275fa 31f5bc9f 4be1a0f4 67f01ca8 85c74777
0735e03 5def5925 cc33173e
b2a8ce77 67522b46 6d278b65 0a291612 7dfea9d2
d361089f 0a7a0247 a184e1c7 0d417866 e0fe0feb
0ff8f2f3 f9176418 f97d117e 624e2015 df1662a8

G_y = 04a3642 0572616c df7e606f
ccadaecf c3b76dab 0eb1248d d03fbdfc 9cd3242c
4726be57 9855e812 de7ec5c5 00b4576a 24628048
b6a72d88 0062eed0 dd34b109 6d3acbb6 b01a4a97

APPENDIX 1: IMPLEMENTATION OF MODULAR ARITHMETIC

The prime moduli in the above examples are of a special type (called *generalized Mersenne numbers*) for which modular multiplication can be carried out more efficiently than in general. This appendix provides the rules for implementing this faster arithmetic, for each of the prime moduli appearing in the examples.

The usual way to multiply two integers (mod m) is to take the integer product and reduce it (mod m). One therefore has the following problem: given an integer A less than m^2, compute

$$B := A \bmod m.$$

In general, one must obtain B as the remainder of an integer division. If m is a generalized Mersenne number, however, then B can be expressed as a sum or difference (mod m) of a small number of terms. To compute this expression, one can evaluate the integer sum or difference and reduce the result modulo m. The latter reduction can be accomplished by adding or subtracting a few copies of m.

The prime moduli p for each of the five example curves is a generalized Mersenne number.

Curve P-192:

The modulus for this curve is $p = 2^{192} - 2^{64} - 1$. Every integer A less than p^2 can be written

$A = A_5 \cdot 2^{320} + A_4 \cdot 2^{256} + A_3 \cdot 2^{192} + A_2 \cdot 2^{128} + A_1 \cdot 2^{64} + A_0$,

where each A_i is a 64-bit integer. The expression for B is

$$B := T + S_1 + S_2 + S_3 \bmod p;$$

where the 192-bit terms are given by

$T = A_2 \cdot 2^{128} + A_1 \cdot 2^{64} + A_0$

$S_1 = A_3 \cdot 2^{64} + A_3$

$S_2 = A_4 \cdot 2^{128} + A_4 \cdot 2^{64}$

$S_3 = A_5 \cdot 2^{128} + A_5 \cdot 2^{64} + A_5$.

Curve P-224:

The modulus for this curve is $p = 2^{224} - 2^{96} + 1$. Every integer A less than p^2 can be written

$A = A_{13} \cdot 2^{416} + A_{12} \cdot 2^{384} + A_{11} \cdot 2^{352} + A_{10} \cdot 2^{320} + A_9 \cdot 2^{288} + A_8 \cdot 2^{256} + A_7 \cdot 2^{224} + A_6 \cdot 2^{192} + A_5 \cdot 2^{160} + A_4 \cdot 2^{128} + A_3 \cdot 2^{96} + A_2 \cdot 2^{64} + A_1 \cdot 2^{32} + A_0$,

where each A_i is a 32-bit integer. As a concatenation of 32-bit words, this can be denoted by

$$A = (A_{13} \,||\, A_{12} \,||\, \cdots \,||\, A_0).$$

The expression for B is

$$B := T + S_1 + S_2 - D_1 - D_2 \bmod p,$$

where the 224-bit terms are given by

$T = (A_6 \,||\, A_5 \,||\, A_4 \,||\, A_3 \,||\, A_2 \,||\, A_1 \,||\, A_0)$

$S_1 = (\ A_{10} \| A_9 \|\ A_8 \| A_7 \|\ 0\ \|\ 0\ \|\ 0\ )$

$S_2 = (\ 0\ \| A_{13} \| A_{12} \| A_{11} \|\ 0 \|\ 0\ \|\ 0\ )$

$D_1 = (A_{13}\ \| A_{12} \|\ A_{11} \| A_{10} \|\ A_9 \|\ A_8 \| A_7)$

$D_2 = (\ 0\ \|\ 0\ \|\ 0\ \|\ 0\ \| A_{13} \| A_{12} \|\ A_{11})$.

Curve P-256:

The modulus for this curve is $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$. Every integer A less than p^2 can be written

$A = A_{15} \cdot 2^{480} + A_{14} \cdot 2^{448} + A_{13} \cdot 2^{416} + A_{12} \cdot 2^{384} + A_{11} \cdot 2^{352} +$

$A_{10} \cdot 2^{320} + A_9 \cdot 2^{288} + A_8 \cdot 2^{256} + A_7 \cdot 2^{224} + A_6 \cdot 2^{192} + A_5 \cdot 2^{160} +$

$A_4 \cdot 2^{128} + A_3 \cdot 2^{96} + A_2 \cdot 2^{64} + A_1 \cdot 2^{32} + A_0,$

where each A_i is a 32-bit integer. As a concatenation of 32-bit words, this can be denoted by

$$A = (A_{15} \| A_{14} \| \cdots \| A_0).$$

The expression for B is

$$B := T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4 \bmod p,$$

where the 256-bit terms are given by

$T = (A_7 \| A_6 \| A_5 \| A_4 \| A_3 \| A_2 \| A_1 \| A_0)$

$S_1 = (A_{15} \| A_{14} \| A_{13} \| A_{12} \| A_{11} \| 0 \| 0 \| 0)$

$S_2 = (0 \| A_{15} \| A_{14} \| A_{13} \| A_{12} \| 0 \| 0 \| 0)$

$S_3 = (A_{15} \| A_{14} \| 0 \| 0 \| 0 \| 0 \| A_{10} \| A_9 \| A_8)$

$S_4 = (A_8 \| A_{13} \| A_{15} \| A_{14} \| A_{13} \| A_{11} \| A_{10} \| A_9)$

$D_1 = (A_{10} \| A_8 \| 0 \| 0 \| 0 \| 0 \| A_{13} \| A_{12} \| A_{11})$

$D_2 = (A_{11} \| A_9 \| 0 \| 0 \| A_{15} \| A_{14} \| A_{13} \| A_{12})$

$D_3 = (A_{12} \| 0 \| A_{10} \| A_9 \| A_8 \| A_{15} \| A_{14} \| A_{13})$

$D_4 = (A_{13} \| 0 \| A_{11} \| A_{10} \| A_9 \| 0 \| A_{15} \| A_{14})$.

Curve P-384:

The modulus for this curve is $p = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$. Every integer A less than p^2 can be written

$A = A_{23} \cdot 2^{736} + A_{22} \cdot 2^{704} + A_{21} \cdot 2^{672} + A_{20} \cdot 2^{640} + A_{19} \cdot 2^{608} +$

$A_{18} \cdot 2^{576} + A_{17} \cdot 2^{544} + A_{16} \cdot 2^{512} + A_{15} \cdot 2^{480} + A_{14} \cdot 2^{448} + A_{13} \cdot 2^{416} + A_{12} \cdot 2^{384} + A_{11} \cdot 2^{352} + A_{10} \cdot 2^{320} + A_9 \cdot 2^{288} + A_8$

$\cdot 2^{256} + A_7 \cdot 2^{224} +$

$A_6 \cdot 2^{192} + A_5 \cdot 2^{160} + A_4 \cdot 2^{128} + A_3 \cdot 2^{96} + A_2 \cdot 2^{64} + A_1 \cdot 2^{32} + A_0,$

where each A_i is a 32-bit integer. As a concatenation of 32-bit words, this can be denoted by

$$A = (A_{23} \| A_{22} \| \cdots \| A_0).$$

The expression for B is

$$B := T + 2S_1 + S_2 + S_3 + S_4 + S_5 + S_6 - D_1 - D_2 - D_3 \bmod p,$$

where the 384-bit terms are given by

$T = (A_{11} \| A_{10} \| A_9 \| A_8 \| A_7 \| A_6 \| A_5 \| A_4 \| A_3 \| A_2 \| A_1 \| A_0)$

$S_1 = (0 \| 0 \| 0 \| 0 \| 0 \| A_{23} \| A_{22} \| A_{21} \| 0 \| 0 \| 0 \| 0)$

$S_2 = (A_{23} \| A_{22} \| A_{21} \| A_{20} \| A_{19} \| A_{18} \| A_{17} \| A_{16} \| A_{15} \| A_{14} \| A_{13} \| A_{12})$

$S_3 = (A_{20} \| A_{19} \| A_{18} \| A_{17} \| A_{16} \| A_{15} \| A_{14} \| A_{13} \| A_{12} \| A_{23} \| A_{22} \| A_{21})$

$S_4 = (A_{19} \| A_{18} \| A_{17} \| A_{16} \| A_{15} \| A_{14} \| A_{13} \| A_{12} \| A_{20} \| 0 \| A_{23} \| 0)$

$S_5 = (0 \| 0 \| 0 \| 0 \| 0 \| A_{23} \| A_{22} \| A_{21} \| A_{20} \| 0 \| 0 \| 0 \| 0)$

$S_6 = (0 \| 0 \| 0 \| 0 \| 0 \| 0 \| A_{23} \| A_{22} \| A_{21} \| 0 \| 0 \| A_{20})$

$D_1 = (A_{22} \| A_{21} \| A_{20} \| A_{19} \| A_{18} \| A_{17} \| A_{16} \| A_{15} \| A_{14} \| A_{13} \| A_{12} \| A_{23})$

$D_2 = (0 \| 0 \| 0 \| 0 \| 0 \| 0 \| 0 \| A_{23} \| A_{22} \| A_2 \| A_{20} \| 0)$

$D_3 = (0 \| 0 \| 0 \| 0 \| 0 \| 0 \| 0 \| A_{23} \| A_{23} \| 0 \| 0 \| 0)$.

Curve P-521:

The modulus for this curve is $p = 2^{521} - 1$. Every integer A less than p^2 can be written

$$A = A_1 \cdot 2^{521} + A_0,$$

The expression for B is

$$B := A_0 + A_1 \bmod p$$

APPENDIX 2: NORMAL BASES

The elements of $GF(2^m)$ are expressed in terms of the type T normal basis [2] B for $GF(2^m)$, for some T. Each element has a unique representation as a bit string

$$(a_0 a_1 \ldots a_{m-1})$$

The arithmetic operations are performed as follows.

Addition: addition of two elements is implemented by bitwise addition modulo 2. Thus, for example,

$$(1100111) + (1010010) = (0110101).$$

Squaring: if

$$\alpha = (a_0 a_1 \ldots a_{m-1})$$

then

$$\alpha^2 = (a_{m-1} a_0 a_1 \ldots a_{m-2})$$

Multiplication: to perform multiplication, one first constructs a function $F(\underline{u},\underline{v})$ on inputs

$\underline{u} = (u_0 u_1 \ldots u_{m-1})$ and $\underline{v} = (v_0 v_1 \ldots v_{m-1})$

as follows.

1. Set $p \leftarrow Tm + 1$
2. Let u be an integer having order T modulo p
3. Compute the sequence $F(1); F(2),\ldots,F(p-1)$ as follows:
   - 3.1 Set $w \leftarrow 1$
   - 3.2 For j from 0 to T-1 do
     - Set $n \leftarrow w$
     - For i from 0 to m-1 do
       - Set $F(n) \leftarrow i$
       - Set $n \leftarrow 2n \bmod p$
     - Set $w \leftarrow uw \bmod p$
4. Output the formula

$$F(\underline{u}, \underline{v}) := \sum_{k=1}^{p-2} u_{F(k+1)} v_{F(p-k)}.$$

This computation need only be performed once per basis.

Given the function F for B, one computes the product

$$(c_0 c_1 \ldots c_{m-1}) = (a_0 a_1 \ldots a_{m-1}) \times (b_0 b_1 \ldots b_{m-1})$$

as follows.

1. Set $(u_0 u_1 \ldots u_{m-1}) \leftarrow (a_0 a_1 \ldots a_{m-1})$
2. Set $(v_0 v_1 \ldots v_{m-1}) \leftarrow (b_0 b_1 \ldots b_{m-1})$
3. For k from 0 to m - 1 do
   - 3.1 Compute
     $$c_k := F(\underline{u}, \underline{v})$$
   - 3.2 Set $u \leftarrow$ **LeftShift** (u) and $v \leftarrow$ **LeftShift** (v), where **LeftShift** denotes the circular left shift operation.
4. Output $c := (c_0 c_1 \ldots c_{m-1})$

[2] It is assumed in this section that m is odd and T is even, since this is the only case considered in this standard.

EXAMPLE. For the type 4 normal basis for $GF(2^7)$, one has $p = 29$ and $u = 12$ or 17. Thus the values of F are given by

$F(1) = 0$ $F(8) = 3$ $F(15) = 6$ $F(22) = 5$
$F(2) = 1$ $F(9) = 3$ $F(16) = 4$ $F(23) = 6$
$F(3) = 5$ $F(10) = 2$ $F(17) = 0$ $F(24) = 1$
$F(4) = 2$ $F(11) = 4$ $F(18) = 4$ $F(25) = 2$
$F(5) = 1$ $F(12) = 0$ $F(19) = 2$ $F(26) = 5$
$F(6) = 6$ $F(13) = 4$ $F(20) = 3$ $F(27) = 1$
$F(7) = 5$ $F(14) = 6$ $F(21) = 3$ $F(28) = 0$

Therefore

$$F(\underline{u}; \underline{v}) = u_0 v_1 + u_1 (v_0 + v_2 + v_5 + v_6) + u_2 (v_1 + v_3 + v_4 + v_5) + u_3 (v_2 + v_5) + u_4 (v_2 + v_6) + u_5 (v_1 + v_2 + v_3 + v_6) + u_6 (v_1 + v_4 + v_5 + v_6).$$

Thus, if

$a = (1\ 0\ 1\ 0\ 1\ 1\ 1)$ and $b = (1\ 1\ 0\ 0\ 0\ 0\ 1)$,

then

$c_0 = F((1\ 0\ 1\ 0\ 1\ 1\ 1), (1\ 1\ 0\ 0\ 0\ 0\ 1)) = 1,$

$c_1 = F((0\ 1\ 0\ 1\ 1\ 1\ 1), (1\ 0\ 0\ 0\ 0\ 1\ 1)) = 0,$

$\vdots$

$c_6 = F((1\ 1\ 0\ 1\ 0\ 1\ 1); (1\ 1\ 1\ 0\ 0\ 0\ 0)) = 1,$

so that $c = ab = (1\ 0\ 1\ 1\ 0\ 0\ 1)$.

APPENDIX 3: SCALAR MULTIPLICATION ON KOBLITZ CURVES

This appendix describes a particularly efficient method of computing the scalar multiple nP on the Koblitz curve E_a over $GF(2^m)$.

The operation τ is defined by

$$\tau(x, y) = (x^2, y^2)$$

When the normal basis representation is used, then the operation τ is implemented by performing right circular shifts on the bit strings representing x and y.

Given m and a, define the following parameters:

- C is some integer greater than 5.
- $\mu := (-1)^{1-a}$
- For $i = 0$ and $i = 1$, define the sequence $s_i(m)$ by

$$s_i(0) = 0,\ s_i(1) = 1 - i,$$
$$s_i(m) = \mu \bullet s_i(m - 1) - 2\, s_i(m - 2) + (-1)^i$$

- Define the sequence $V(m)$

$$V(0) = 2,\ V(1) = \mu$$
$$V(m) = \mu \bullet v(m - 1) - 2V(m - 2).$$

For the example curves, the quantities $s_i(m)$ and $V(m)$ are as follows.

Curve K-163:

$s_0(163) = 257938643911073165041953 7$
$s_1(163) = -755360064476226375461594$
$V(163) = -484546663253941077680431 7$

Curve K-233:

$s_0(233)$ = -27859711741434429761757834964435883

$s_1(233)$ = -44192136247082304936052160908934886

$V(233)$ = -13738154601110823539498729965136677 9

Curve K-283:

$s_0(283)$ = -66598153210904904110879553600159146928 0025

$s_1(283)$ = 11558600549091367751922810725916099139459 68

$V(283)$ = 77772448708728309992877919709628239775699 17

Curve K-409:

$s_0(409)$ = -18307510456002382137810317198756461378590542487556 86\
9338419259

$s_1(409)$ = -88930485261383040971966532418442126796265661009966 06\
444816790

$V(409)$=10457288737315625927447685387048320737638796957687 57\
5791173829

Curve K-571:

$s_0(571)$ = -37373194468764636924293858924761155671472939645961 3\
10241234064202352419167299832613 05

$s_1(571)$ = -31918577064464160995838145959489596741319689121485 64\
65861056511758982848515832612248752

$V(571)$=-14838092698169141389961914029705149036454257418049 3\
93623291233953420851682897311145984 3

The following algorithm computes the scalar multiple nP on the Koblitz curve E_a over $GF(2^m)$. The average number of elliptic additions and subtractions is at most ~ 1 + (m/3), and is at most ~ m/3 with probability at least 1 - 2^{5-C}.

For i = 0 to 1 do

$n' \leftarrow \lfloor n / 2^{a-C+(m-9)/2} \rfloor$

$g' \leftarrow s_i(m) \cdot n'$

$h' \leftarrow \lfloor g' / 2^m \rfloor$

$j' \leftarrow V(m) \cdot h'$

$l' \leftarrow \text{Round}((g' + j') / 2^{(m+5)/2})$

$\lambda_i \leftarrow l' / 2^C$

$f_i \leftarrow \text{Round}(\lambda_i)$

$\eta_i \leftarrow \lambda_i - f_i$

$h_i \leftarrow 0$

$\eta \leftarrow 2\,\eta_0 + \mu\,\eta_1$

If $\eta \geq 1$

then

if $\eta_0 - 3\,\mu\eta_1 < -1$

then set $h_1 \leftarrow \mu$

else set $h_0 \leftarrow 1$

else

if $\eta_0 + 4\,\mu\,\eta_1 \geq 2$

then set $h_1 \leftarrow \mu$

If $\eta < -1$

then

if $\eta_0 - 3\mu\eta_1 \geq 1$

then set $h_1 \leftarrow -\mu$

else set $h_0 \leftarrow -1$

else

if $\eta_0 + 4\mu\eta_1 < -2$

then set $h_1 \leftarrow -\mu$

$q_0 \leftarrow f_0 + h_0$

$q_1 \leftarrow f_1 + h_1$

$r_0 \leftarrow n - (s_0 + \mu s_1) q_0 - 2 s_1 q_1$

$r_1 \leftarrow s_1 q_0 - s_0 q_1$

Set $Q \leftarrow O$

$P_0 \leftarrow P$

While $r_0 \neq 0$ or $r_1 \neq 0$

If r_0 odd then

set $u \leftarrow 2 - (r_0 - 2 r_1 \bmod 4)$

set $r_0 \leftarrow r_0 - u$

if $u = 1$ then set $Q \leftarrow Q + P_0$

if $u = -1$ then set $Q \leftarrow Q - P_0$

Set $P_0 \leftarrow \tau P_0$

Set $(r_0, r_1) \leftarrow (r_1 + \mu r_0/2, -r_0/2)$

Endwhile

Output Q

APPENDIX 4: GENERATION OF PSEUDO-RANDOM CURVES (PRIME CASE)

Let l be the bit length of p, and define

$$v = \lfloor (l - 1)/160 \rfloor$$

$$w = l - 160v - 1$$

1. Choose an arbitrary 160-bit string s.
2. Compute h:= SHA-1(s).
3. Let h_0 be the bit string obtained by taking the w rightmost bits of h.
4. Let z be the integer whose binary expansion is given by the 160-bit string s.
5. For i from 1 to v do:
   5.1 Define the 160-bit string s_i to be binary expansion of the integer $(z + i) \bmod (2^{160})$.
   5.2 Compute h_i:=SHA-1(s_i).
6. Let h be the bit string obtained by the concatenation of $h_0, h_1, \ldots, h_v$ as follows:

$$h = h_0 \| h_1 \| \ldots \| h_v$$

7. Let c be the integer whose binary expansion is given by the bit string h.
8. If $c = 0$ or $4c + 27 \equiv 0 \pmod p$, then go to Step 1.
9. Choose integers $a, b \in GF(p)$ such that

$$c\, b^2 \equiv a^3 \pmod p.$$

(The simplest choice is $a = c$ and $b = c$. However, one may want to choose differently for performance reasons.)

10. Check that the elliptic curve E over $GF(p)$ given by $y^2 = x^3 + ax + b$ has suitable order. If not, go to Step 1.

APPENDIX 5: VERIFICATION OF CURVE PSEUDO-RANDOMNESS (PRIME CASE)

Given the 160-bit seed value s, one can verify that the coefficient b was obtained from s via the cryptographic hash function SHA-1 as follows.

Let l be the bit length of p, and define

$$v = \lfloor (l - 1) / 160 \rfloor$$

$$w = l - 160v - 1$$

1. Compute h := SHA-1(s).
2. Let h_0 be the bit string obtained by taking the w rightmost bits of h.
3. Let z be the integer whose binary expansion is given by the 160-bit string s.
4. For i from 1 to v do
   - 4.1 Define the 160-bit string s_i to be binary expansion of the integer $(z + i) \bmod (2^{160})$.
   - 4.2 Compute h_i :=SHA-1(s_i).
5. Let h be the bit string obtained by the concatenation of h_0 , h_1, . . . , h_v as follows:

$$h = h_0 \| h_1 \| \ldots \| h_v.$$

6. Let c be the integer whose binary expansion is given by the bit string h.
7. Verify that $b^2 c \equiv -27 \pmod{p}$.

APPENDIX 6: GENERATION OF PSEUDO-RANDOM CURVES (BINARY CASE)

Let:

$$v = \lfloor (m - 1) / B \rfloor$$

$$w = m - Bv$$

1. Choose an arbitrary 160-bit string s.
2. Compute h := SHA-1(s)
3. Let h_0 be the bit string obtained by taking the w rightmost bits of h.
4. Let z be the integer whose binary expansion is given by the 160-bit string s.
5. For i from 1 to v do:
   - 5.1 Define the 160-bit string s_i to be binary expansion of the integer $(z + i) \bmod (2^{160})$.
   - 5.2 Compute h_i :=SHA-1(s_i).
6. Let h be the bit string obtained by the concatenation of h_0 , h_1, . . . , h_v as follows:

$$h = h_0 \| h_1 \| \ldots \| h_v.$$

7. Let b be the element of $GF(2^m)$ which binary expansion is given by the bit string h.
8. Choose an element a of $GF(2^m)$.
9. Check that the elliptic curve E over $GF(2^m)$ given by $y^2 + xy = x^3 + ax^2 + b$ has suitable order. If not, go to Step 1.

APPENDIX 7: VERIFICATION OF CURVE PSEUDO-RANDOMNESS (BINARY CASE)

Given the 160-bit seed value s, one can verify that the coefficient b was obtained from s via the cryptographic hash function SHA-1 as follows.

Define

$v = \lfloor (m - 1) /160 \rfloor$

$w = m - 160v$

1. Compute $h := \text{SHA-1}(s)$
2. Let h_0 be the bit string obtained by taking the w rightmost bits of h.
3. Let z be the integer whose binary expansion is given by the 160-bit string s.
4. For i from 1 to v do
   4.1 Define the 160-bit string s_i to be binary expansion of the integer $(z + i) \bmod (2^{160})$
   4.2 Compute $h_i := \text{SHA-1}(s_i)$.
5. Let h be the bit string obtained by the concatenation of $h_0, h_1, \ldots, h_v$ as follows:

$$h = h_0 \| h_1 \| \ldots \| h_v.$$

6. Let c be the element of $GF(2^m)$ which is represented by the bit string h.
7. Verify that $c = b$.

APPENDIX 8: POLYNOMIAL BASIS TO NORMAL BASIS CONVERSION

Suppose that α an element of the field $GF(2^m)$. Denote by **p** the bit string representing α with respect to a given polynomial basis. It is desired to compute **n**, the bit string representing α with respect to a given normal basis. This is done via the matrix computation

$$\mathbf{p}\,\Gamma = \mathbf{n}$$

Where Γ is an m-by-m matrix with entries in $GF(2)$. The matrix Γ, which depends only on the bases, can be computed easily given its second-to-last row. The second-to-last row for each conversion is given in the table below.

Degree 163:

3 e173bfaf 3a86434d 883a2918 a489ddbd 69fe84e1

Degree 233:

0be 19b89595 28bbc490

038f4bc4 da8bdfc1 ca36bb05 853fd0ed 0ae200ce

Degree 283:

3347f17 521fdabc 62ec1551 acf156fb

0bceb855 f174d4c1 7807511c 9f745382 add53bc3

Degree 409:

0eb00f2 ea95fd6c 64024e7f

0b68b81f 5ff8a467 acc2b4c3 b9372843 6265c7ff

a06d896c ae3a7e31 e295ec30 3eb9f769 de78bef5

Degree 571:

7940ffa ef996513 4d59dcbf

e5bf239b e4fe4b41 05959c5d 4d942ffd 46ea35f3

e3cdb0e1 04a2aa01 cef30a3a 49478011 196bfb43

c55091b6 1174d7c0 8d0cdd61 3bf6748a bad972a4

Given the second-to-last row **r** of Γ, the rest of the matrix is computed as follows. Let β be the element of $GF(2^m)$ whose representation with respect to the normal basis is **r**. Then the rows of Γ, from top to bottom, are the bit strings representing the elements

$$\beta^{m-1}, \beta^{m-2}, \ldots, \beta^2, \beta, 1$$

with respect to the normal basis. (Note that the element 1 is represented by the all-1 bit string.)

Alternatively, the matrix is the inverse of the matrix described in Appendix 9.

More details of these computations can be found in Annex A.7 of the IEEE P1363 standard.

APPENDIX 9: NORMAL BASIS TO POLYNOMIAL BASIS CONVERSION

Suppose that α an element of the field $GF(2^m)$. Denote by **n** the bit string representing α with respect to a given normal basis. It is desired to compute **p**, the bit string representing α with respect to a given polynomial basis. This is done via the matrix computation

$$\mathbf{n}\,\Gamma = \mathbf{p}$$

where Γ is an m-by-m matrix with entries in $GF(2)$. The matrix Γ, which depends only on the bases, can be computed easily given its top row. The top row for each conversion is given in the table below.

Degree 163:

7 15169c10 9c612e39 0d347c74 8342bcd3 b02a0bef

Degree 233:

149 9e398ac5 d79e3685
59b35ca4 9bb7305d a6c0390b cf9e2300 253203c9

Degree 283:

31e0ed7 91c3282d c5624a72 0818049d
053e8c7a b8663792 bc1d792e ba9867fc 7b317a99

Degree 409:

0dfa06b e206aa97 b7a41fff
b9b0c55f 8f048062 fbe8381b 4248adf9 2912ccc8
e3f91a24 e1cfb395 0532b988 971c2304 2e85708d

Degree 571:

452186b bf5840a0 bcf8c9f0
2a54efa0 4e813b43 c3d41496 06c4d27b 487bf107
393c8907 f79d9778 beb35ee8 7467d328 8274caeb
da6ce05a eb4ca5cf 3c3044bd 4372232f 2c1a27c4

Given the top row **r** of Γ, the rest of the matrix is computed as follows. Let β be the element of $GF(2^m)$ whose representation with respect to the polynomial basis is **r**. Then the rows of Γ, from top to bottom, are the bit strings representing the elements

$$\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}$$

with respect to the polynomial basis.

Alternatively, the matrix is the inverse of the matrix described in Appendix 8.

More details of these computations can be found in Annex A.7 of the IEEE P1363 standard.

Anexo 2

UML Simplificado