Demonstrações Distribuídas de Caráter Primo de Larga Escala

Décio Luiz Gazzoni Filho e Taufik Abrão

***Resumo*— Descrevemos uma implementação distribuída do algoritmo ECPP devido a Atkin e Morain, para demonstração de caráter primo de inteiros de forma geral. Sugerimos uma nova arquitetura de rede para implementações distribuídas, e uma nova técnica para estimativa de precisão de um dos cálculos intermediários do algoritmo. Por fim, relatamos um resultado preliminar obtido com a atual implementação.**

***Palavras-Chave*— Número primos, curvas elípticas, demonstrações de caráter primo, computação distribuída**

***Abstract*— We describe a distributed implementation of Atkin and Morain's ECPP algorithm for primality proving of integers without special form. We suggest a new network architecture for distributed implementations, and a new technique for estimating the precision of one of the intermediate computations of ECPP. Finally, we relate a preliminary result obtained with the current implementation.**

***Keywords*— Prime numbers, elliptic curves, primality proofs, distributed computing**

I. Introdução

O grupo de pontos de uma curva elíptica tem se mostrado uma importante ferramenta na criptografia e teoria dos números computacional. Criptossistemas empregando o problema do logaritmo discreto sobre este grupo tem se mostrado imbatíveis em certas aplicações, como smart cards [1]. A fatoração de inteiros através do método ECM de Lenstra é o método mais eficiente conhecido [2] (exceto para módulos RSA [3]), e as demonstrações de caráter primo de inteiros de forma geral, através do algoritmo ECPP de Atkin e Morain, atingiram patamares de desempenho inéditos: já foram obtidos certificados de caráter primo para inteiros de até 7760 dígitos em uniprocessadores [4] e 15071 dígitos para implementações distribuídas [5].

Os números primos são uma ferramenta básica de construção de sistemas criptográficos. É possível argumentar que as demonstrações de caráter primo são supérfluas, em função da existência de algoritmos que indicam o caráter primo de um número com alta probabilidade, os chamados testes de caráter pseudoprimo (ver [6] para uma visão geral do assunto). No entanto, para certas aplicações onde exige-se o máximo possível de segurança, pode ser desejável obter um certificado de caráter primo, que os testes de caráter pseudoprimo não podem fornecer. E embora os números primos de interesse criptográfico estejam ao alcance de implementações monoprocessadas no momento, o desenvolvimento de novos criptossistemas, ou de algoritmos que simplifiquem a quebra dos criptossistemas atuais, pode requerer o uso de primos maiores. Uma implementação distribuída permite, por exemplo, que uma empresa certifique o caráter primo de um inteiro utilizando apenas recursos ociosos dos computadores de sua rede.

A proposta deste trabalho é uma implementação distribuída em larga escala do algoritmo ECPP (a implementação de [5] restringe-se a *clusters*) na tentativa de estabelecer novos recordes.

Na seção II, são introduzidos os conceitos necessários para compreensão do funcionamento do algoritmo ECPP, que é descrito na seção III. Na seção IV, é discutida a estratégia para paralelização do algoritmo e os compromissos necessários para implementações em larga escala. Na seção V, é discutida a implementação do algoritmo e outras subrotinas necessárias para sua operação (fatoração de inteiros, testes de caráter pseudoprimo etc.). Na seção VI, relatamos resultados preliminares obtidos com a atual implementação.

Universidade Estadual de Londrina, Departamento de Engenharia Elétrica, Caixa Postal 6001, 86051-990 - Londrina - PR. Emails: decio@decpp.net, taufik@uel.br

II. Curvas Elípticas

A equação geral de um polinômio de grau 3, com coeficientes provindos de um corpo F e igualada a zero, é

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + \\ + fxy + gy^2 + hx + iy + j = 0, \quad (1)$$

com $a, b, c, d \neq 0$. Os pares ordenados $(x, y) \in F \times F$ que satisfazem (1) são ditas as soluções *afim* da equação. Tais soluções são estritamente relacionadas às soluções *projetivas* da curva, que satisfazem

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2z + \\ + fxyz + gy^2z + hxz^2 + iyz^2 + jz^3 = 0. \quad (2)$$

Se estas curvas forem *suaves*, ou seja, nenhuma solução da equação apresentar as três derivadas parciais simultaneamente nulas, então a curva é dita uma *curva elíptica*. Impondo a condição adicional que a característica do corpo F deve ser diferente de 2 e 3, é sabido que toda equação da forma (2) pode ser transformada por uma mudança de variáveis numa equação da forma

$$y^2z = x^3 + axz^2 + bz^3, \ a, b \in F, \quad (3)$$

cuja forma afim

$$y^2 = x^3 + ax + b \quad (4)$$

é conhecida como *parametrização de Weierstrass*. Uma curva nesta forma é suave se e somente se $4a^3 + 27b^2 \neq 0$.

Independentemente de se estar empregando a forma afim ou projetiva da curva, será usada a notação O para denotar

o ponto no infinito que ocorre na forma projetiva da curva. Temos então:

Definição 2.1: Uma curva cúbica suave da forma (2), com coeficientes pertencentes a um corpo F e pelo menos uma solução com coordenadas em F não simultaneamente nulas, é dita uma curva elíptica sobre F. Se a característica de F não for 2 ou 3, a equação (3) também define uma curva elíptica sobre F, contanto que $4a^3 + 27b^2 \neq 0$. Denota-se por $E(F)$ o conjunto de pontos com coordenadas em F que satisfazem a equação, adicionado do ponto no infinito O. Ou seja,

$$E(F) = \{(x, y) \in F \times F : y^2 = x^3 + ax + b\} \cup \{O\}.$$

A. O Grupo de Pontos de uma Curva Elíptica

O interesse criptográfico nas curvas elípticas surge da definição de uma certa operação sobre os pontos do conjunto $E(F)$ dado pela Definição 2.1, dando origem a um grupo abeliano. Parte-se de uma interpretação geométrica para a operação do grupo quando $F = \mathbb{R}$, a partir da qual serão deduzidas expressões válidas em qualquer corpo, ainda que a interpretação geométrica perca o sentido. A operação do grupo sobre dois pontos distintos é feita traçando uma reta que contenha ambos os pontos, obtendo uma terceira interseção desta reta com a curva. Este ponto é então refletido através do eixo x. Quando os pontos são os mesmos, a reta empregada é a tangente à curva nesse ponto. A notação padrão na literatura para a operação deste grupo é aditiva. Após um pouco de geometria analítica e álgebra, obtém-se:

Definição 2.2: Seja $E(F)$ uma curva elíptica definida pela equação (3) sobre um corpo F de característica diferente de 2 e 3, e $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ dois pontos não necessariamente distintos sobre esta curva. Denotando por O o ponto no infinito, define-se uma operação comutativa $+$, com operação inversa $-$, como segue:

1. $-P_1 = (x_1, -y_1)$;
2. $P_1 + P_2 = (x_3, y_3)$, com

$$x_3 = m^2 - x_1 - x_2$$
$$y_3 = m(x_1 - x_3) - y_1,$$

onde a *inclinação* m é definida por

$$m = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, & \text{se } x_1 \neq x_2 \\ \frac{3x_1^2 + a}{2y_1}, & \text{se } x_1 = x_2. \end{cases}$$

É uma consequência surpreendente que a operação assim definida, em conjunto com $E(F)$, forme um grupo. Ademais, é conhecido um profundo teorema a respeito da ordem desse grupo:

Teorema 2.3 (Hasse): A ordem $\#E(F_{p^k})$ do grupo de pontos de uma curva elíptica satisfaz a desigualdade

$$|\#E - (p^k + 1)| \leq 2\sqrt{p^k}$$

Essencialmente, o teorema de Hasse afirma que a ordem do grupo é próxima da característica do corpo-base.

B. Curvas Elípticas com Multiplicação Complexa

Seja $-D$ um discriminante fundamental, i.e. $D \equiv 3, 4, 7, 8, 11, 15 \pmod{16}$ e D não possui fatores primos ímpares repetidos, e seja p um primo. Caso existam $u, v \in \mathbb{Z}$ tal que $4p = u^2 + Dv^2$ (o *algoritmo de Cornacchia* [7], [6] pode ser empregado para obter estes valores), então existem curvas elípticas tais que

$$\#E = \begin{cases} p + 1 \pm u, p + 1 \pm (u \pm 3v)/2 & \text{se } D = 3, \\ p + 1 \pm u, p + 1 \pm 2v & \text{se } D = 4, \\ p + 1 \pm u & \text{se } D > 4. \end{cases} \quad (5)$$

A teoria fornece então métodos eficientes para a construção de curvas com as ordens de grupo indicadas.

III. O Algoritmo ECPP de Atkin-Morain

O seguinte teorema, fruto do trabalho de Goldwasser e Kilian [8] numa primeira versão (teórica) do ECPP, é crucial para o algoritmo:

Teorema 3.1: Seja $n > 1$ um inteiro relativamente primo a 6 e $E(F_n)$ uma curva elíptica módulo n. Sejam m um inteiro e $P \in E(F_n)$ um ponto da curva satisfazendo as seguintes condições:

1. Existe um fator primo q de m tal que
$$q > \left(\sqrt[4]{n} + 1\right)^2.$$
2. $mP = O$.
3. $(m/q)P \neq O$.

Então n é primo.

Caso n não seja primo, a curva elíptica $E(F_n)$ não forma um grupo, e não será possível satisfazer as condições do teorema.

A estratégia do algoritmo ECPP é construir diversas curvas elípticas, até obter uma ordem de grupo $m = \#E(F_n)$ que seja fatorável na prática e que possua um fator pseudoprimo p_0 satisfazendo as condições do Teorema 3.1. Pela aplicação do teorema, obtém-se um certificado do tipo "se p_0 é primo, então n é primo". Aplica-se o procedimento recursivamente sobre p_0 para obter um certificado do tipo "se p_1 é primo, então p_0 é primo". Dessa maneira, constrói-se uma cadeia de certificados $\ldots \rightarrow p_2 \rightarrow p_1 \rightarrow p_0 \rightarrow n$, até atingir p_k's cujo tamanho permita uma demonstração de caráter primo por métodos mais simples, como o crivo de Eratóstenes.

Na versão atual do ECPP, a construção das curvas elípticas emprega o método de multiplicação complexa exposto acima [9]. O método envolve a aplicação do algoritmo de Cornacchia para obtenção da ordem do grupo, que sem modificações é relativamente custoso, por envolver uma raiz quadrada modular. Uma idéia atribuída a J. Shallit é a construção de uma base de fatores e raízes quadradas modulares precomputadas, combinando-se os elementos dessa base para a construção de discriminantes, e removendo o custo das raízes quadradas da aplicação do algoritmo de Cornacchia. Incidentalmente, discriminantes compostos de diversos fatores primos são desejáveis na construção da curva elíptica implícita no Teorema 3.1, por motivos descritos na seção V-B.

Um resumo do algoritmo é dado abaixo, para referência futura. A entrada é o pseudoprimo atual na cadeia de certificados, digamos p_i, e a saída é o próximo pseudoprimo na cadeia, p_{i+1}.

1. Construa a base de fatores dos discriminantes, $\mathcal{Q} = \{q_1^*, q_2^*, \ldots, q_r^*\}$ para um r qualquer, e precompute as raízes quadradas módulo p_i da base de fatores, $\mathcal{S} = \{\sqrt{q_1^*}, \sqrt{q_2^*}, \ldots, \sqrt{q_r^*}\}$.
2. Construa discriminantes $-D_j$ como produtos de subconjuntos de $\mathcal{Q}$, tentando representá-los na forma $4q_i = u^2 + D_j v^2$.
3. Para os discriminantes que possuem uma representação nessa forma, tente fatorar as possíveis ordens de grupo (5), buscando um inteiro da forma $x = mq_{i+1}$, onde o cofator q_{i+1} é um pseudoprimo que satisfaz as condições do Teorema 3.1.
4. Construa uma curva elíptica com a ordem de grupo obtida no item anterior, e verifique as condições do Teorema 3.1 para essa curva e um ponto escolhido ao acaso.
5. Se as condições do teorema forem satisfeitas, o algoritmo termina com resultado q_{i+1}.

Se o algoritmo esgotar os discriminantes possíveis, existem três saídas: repetir as computações sobre os discriminantes existentes (aplicando um esforço maior na busca de fatores grandes das ordens de grupo), estender a base de fatores, ou voltar atrás na cadeia de certificados.

IV. Paralelização do ECPP

Em teoria, o ECPP pode ser paralelizado sem esforço. É possível distribuir os subconjuntos do item 2 do algoritmo entre diversos computadores e, a rigor, não é necessário esperar a confirmação das condições do Teorema 3.1 para o procedimento do algoritmo, uma vez que uma falha naquele passo do algoritmo é rara e implicaria no término da tentativa de demonstração do caráter primo do número em questão.

Porém, surgem certas dificuldades de ordem prática para uma implementação de larga escala com servidor central, em particular a repetição das precomputações do item 1 em cada cliente, e o desperdício de computações durante a transição de um pseudoprimo para outro na cadeia de q_i's, uma vez que todos os computadores da rede precisam ser avisados quando da ocorrência dessa transição. Também é desejável evitar uma sobrecarga do servidor central para não limitar a escalabilidade da rede. Assim, propõe-se o uso de uma arquitetura de rede de múltiplos níveis, com um servidor central no topo, *proxies* nos níveis do meio e os clientes em baixo. As vantagens são múltiplas:

1. o ônus da transferência de dados é reduzido nos níveis superiores de *proxies*, uma vez que os mesmos só precisam se comunicar com outros *proxies* e não diretamente com os clientes;
2. relacionado a isso, torna-se viável propagar um aviso de transição na cadeia de certificação, ao invés de esperar nova comunicação provinda dos clientes;
3. o servidor central só precisa ficar exposto ao primeiro nível de *proxies* da rede (supostamente instalações confiáveis), aumentando a segurança do sistema.
4. as precomputações do passo 1 podem ser efetuadas nos proxies e transmitidas para os clientes;
5. é viável realizar verificações parciais dos resultados computados pelos clientes, na busca de máquinas com defeito ou tentativas de trapaça.

Até onde os autores têm conhecimento, essa arquitetura de rede não foi proposta na literatura especializada uma vez que não há trabalhos estudando implementações de larga escala como essa.

V. Implementação

O programa encontra-se em fase de implementação. Uma vez que o programa será distribuído sob a licença GNU GPL, é possível reutilizar código de outros projetos distribuídos sob esta licença. Isto é vantajoso, uma vez que é possível encontrar código de altíssima qualidade, como por exemplo o programa GMP-ECM [2], do qual foi empregado o código de fatoração de inteiros usando o algoritmo $p-1$, e a biblioteca Trolltech Qt [10], empregada para desenvolvimento da interface de usuário do programa. Outros projetos de software livre usados na implementação do programa incluem a biblioteca GNU Common C++ e o banco de dados PostgreSQL.

Expomos a seguir os algoritmos empregados na confecção do programa.

A. Fatorando a ordem de grupo

Para a tarefa de fatoração das potenciais ordens de grupo, os principais algoritmos disponíveis são *trial division*, Pollard rho [11], Pollard p-1 [12] e ECM [13]. A implementação de *trial division* gera o produto de diversos primos pequenos e calcula o MDC do inteiro sendo fatorado com este produto. A implementação de Pollard rho utiliza a técnica de busca de ciclos de Brent [14]. Como mencionado, foi empregada a implementação do programa GMP-ECM do algoritmo Pollard p-1. O algoritmo ECM, embora implementado no programa GMP-ECM, não é utilizado ainda.

A seleção de parâmetros e a divisão de tempo entre esses algoritmos é crucial para maximizar a eficiência do procedimento da fatoração das ordens de grupo. Como a implementação ainda não está completa, não foram realizados experimentos nesse sentido ainda.

B. Gerando curvas elípticas com multiplicação complexa

Para verificar as condições do teorema 3.1, é preciso construir uma curva elíptica sobre F_{q_i} com a ordem de grupo indicada, empregando a teoria de multiplicação complexa. O procedimento é o seguinte:

- construir o *polinômio de classe*, um polinômio com coeficientes inteiros cujas raízes geram o corpo quadrático imaginário $\mathbb{Q}(\sqrt{-D})$;
- Reduzir os coeficientes do polinômio módulo q_i;
- Encontrar uma raiz desse polinômio em F_{q_i};
- Calcular os coeficientes da curva a partir dessa raiz.

Para o passo 1, o procedimento tradicional é a construção do chamado polinômio de Hilbert. No entanto, os coeficientes deste polinômio tendem a ser muito grandes, exigindo computação em alta precisão, o que leva a um desperdício de tempo e espaço (memória). Na prática são empregados

outros polinômios, como os polinômios de Weber e outros, baseados em diferentes *invariantes de classe*. Nesse sentido, um conjunto de invariantes foi padronizado pelo IEEE no documento p1363, na qual baseamos nossa implementação.

Resumidamente, o processo envolve a geração das raízes complexas do polinômio, a partir dos invariantes de classe aplicados a cada elemento do grupo de classe do discriminante associado, e a posterior reconstrução desse polinômio a partir de suas raízes. Os cálculos são realizados em ponto flutuante de precisão múltipla, levantando duas grandes questões:

- qual é a precisão mínima requerida para o processo?
- quais os melhores algoritmos nessa faixa de precisão?

Quanto à primeira questão, embora existam estimativas na literatura, elas não são de valor prático. A técnica empregada em nossa implementação, que até onde sabemos é inédita, envolve o cálculo do polinômio em baixa precisão, que pode ser feito rapidamente, para obter uma estimativa da magnitude dos coeficientes do mesmo. Essa estimativa, somada a uma margem de segurança (cujo valor ótimo ainda está sendo estudado, mas parece ser da ordem de $\sqrt{h}$, o número de classe do discriminante em questão), é então utilizada para o cômputo final do polinômio.

A segunda questão é abordada a seguir.

1) Aritmética de polinômios com coeficientes complexos: A aritmética dos coeficientes é realizada pela própria biblioteca GMP, restando apenas a questão da aritmética de polinômios. As únicas operações necessárias para a reconstrução do polinômio a partir de suas raízes são a multiplicação, soma e subtração (as duas últimas como rotinas auxiliares para a rotina de multiplicação). As algoritmos de soma e subtração são os algoritmos clássicos, enquanto que para multiplicação, são empregados três algortimos diferentes: clássico, Karatsuba e convoluções por FFT.

O algoritmo clássico multiplica $a(x) = \sum_{i=0}^{m} a_i x^i$ por $b(x) = \sum_{j=0}^{n} b_j x^j$, resultando num polinômio $c(x) = a(x)b(x) \sum_{k=0}^{m+n} c_k x^k$ através da avaliação da expressão

$$c_k = \sum_{i=0}^{k} a_i b_{k-i}. \tag{6}$$

No entanto, o tempo de execução desse algoritmo é proporcional a mn, sendo deixado para trás pelo algoritmo de Karatsuba mesmo para pequenos valores de m e n. O algoritmo de Karatsuba é baseado na observação que é possível particionar um polinômio em duas partes, digamos $a_L(x) = \sum_{i=0}^{\lfloor m/2 \rfloor} a_i x^i$ e $a_H(x) = \sum_{j=0}^{\lceil m/2 \rceil} a_{j+\lfloor m/2 \rfloor+1} x^j$, e de maneira semelhante para $b_L(x)$ e $b_H(x)$. Seja $n = \lfloor m/2 \rfloor + 1$. O polinômio resultante $c(x)$ é calculado pela expressão

$$\begin{aligned} c &= a_H b_H x^{2n} + a_L b_L + \\ &+ ((a_H + a_L)(b_H + b_L) - a_L b_L - a_H b_H) x^n. \end{aligned} \tag{7}$$

Esta expressão envolve apenas três multiplicações de polinômios, e não as quatro esperadas, e pode ser aplicada recursivamente. O tempo de execução é então da ordem de $\max(m,n)^{\log_2 3} \approx \max(m,n)^{1,58}$. Durante a implementação, foi observado que o terceiro termo do lado direito da equação 7 gera o problema de cancelamento em ponto flutuante, o que leva a perda de precisão. Assim, pode ser desejável ajustar o limiar de transição em favor do algoritmo clássico (a despeito de seu tempo de execução maior), para evitar esse problema.

Existe também um algoritmo com tempo de execução proporcional a $\max(m,n) \log \max(m,n)$, baseado na observação de que convoluções no domínio da frequência podem ser efetudas de maneira bastante eficiente (multiplicação ponto-a-ponto dos elementos dos vetores sendo convoluídos), e que a conversão entre o domínio do tempo e o domínio da frequência também pode ser efetuado de maneira eficiente através do algoritmo FFT. Nossa implementação utiliza um FFT split-radix, uma das variantes mais eficientes, com planos para implementação de um FFT four-step no futuro, para permitir execução paralela em sistemas SMP.

Mencionamos de passagem que a literatura não menciona nenhuma implementação que utiliza FFTs para multiplicação de polinômios, embora nossa implementação tenha mostrado um ganho bastante significativo por conta disso. Naturalmente, os ganhos do FFT só vem a ser realizados para discriminantes maiores, de modo que outros implementadores podem estar satisfeitos com o desempenho do algoritmo de Karatsuba se a faixa de discriminantes for limitada.

2) Aritmética de polinômios com coeficientes de F_p: O passo 3 do algoritmo de construção de curvas acima exige a obtenção de uma raiz de um polinômio com coeficientes provindos de um corpo finito F_p. O algoritmo empregado é o de Cantor e Zassenhaus, que busca uma raiz de grau d do polinômio $W(x)$ através do cálculo de

$$\text{mdc}(W, T^{(p^d-1)/2} - 1),$$

onde T é um polinômio tomado aleatoriamente de $F_p[x]$, e a exponenciação indicada é realizada módulo W. Assim, é necessário implementar rotinas de aritmética de polinômios para a aplicação do algoritmo de Cantor e Zassenhaus, em particular algoritmos de soma e subtração, multiplicação, redução modular e mdc. Destes, apenas a escolha dos algoritmos de soma e subtração é trivial, uma vez que o desempenho dos algoritmos clássicos é satisfatório. A escolha do algoritmo de mdc também é imediata, empregando-se o algoritmo de Euclides.

Para o algoritmo de multiplicação, foi empregado um método conhecido por *segmentação binária* [6]. Neste método, são construído inteiros a partir dos coeficientes dos polinômios a serem multiplicados, e a multiplicação dos dois é feita usando uma rotina de multiplicação de inteiros, reconstruindo o polinômio produto a partir do resultado desta operação. A vantagem é que é possível utilizar a mesma rotina para multiplicação de inteiros e multiplicação de polinômios (nesse caso, a rotina do GMP), concentrando todos os esforços de otimização nessa rotina.

Resta então a rotina de exponenciação, que emprega redução modular como subrotina. Inicialmente foi implementado um algoritmo de exponenciação esquerda-direita [6], [7], embora um algoritmo mais eficiente deva ser implementado no futuro. Para redução modular, é empregada uma técnica remanescente dos métodos de Newton para divisão, descrita em [6, Alg. 9.6.4].

C. Projetos em andamento

Algumas melhoras cruciais para a eficiência do programa estão em fase de planejamento ou implementação. São elas:

- determinação da precisão mínima necessária para o cômputo dos polinômios de Weber;
- implementação de invariantes de classe alternativos, para obtenção de polinômios de menor grau;
- fatoração sobre o corpo de gênero, que reduz o grau do polinômio de Weber gerado [9];
- multiplicação mais eficiente de inteiros gigantescos ([15] relata uma melhora significativa pela substituição da rotinas de multiplicação da biblioteca GMP por multiplicação usando FFTs da biblioteca FFTW);
- ajustes nos parâmetros das rotinas de fatoração de inteiros.

VI. Resultados preliminares

Até este momento, os esforços dispendidos na implementação foram concentrados na geração dos polinômios de classe, a mais custosa subrotina envolvida na verificação das condições do teorema 3.1. Para verificação da correção da implementação, foram geradas diversas curvas elípticas com multiplicação complexa, e foi verificado que suas ordens de grupo coincidissem com os valores esperados através do sistema PARI/GP, que inclui facilidades para aritmética elíptica. Destes, o resultado mais interessante foi a geração de uma curva elíptica com multiplicação complexa por $\sqrt{-5000500423}$. Os autores têm conhecimento de apenas um esforço semelhante [16], porém com discriminante mais baixo (-1000004099). A característica do corpo base é 80181938509340352490501477954289294831064589795 7, a ordem do grupo é 80181938509340352490501436759261883081904996278 4 e a equação da curva é $y^2 = x^3 + 71155998340812443850119358884033186872540627802 3x - 63808115168408029555878465139204509629752896475 2$. A computação toda foi realizada em 27h25m em um computador Intel Pentium 4 2.6 GHz com 1 GB de memória, rodando o sistema operacional Linux.

Alguns dados a respeito desta computação: o número de classe do discriminante em questão é 29098, o mesmo grau do polinômio obtido. Inicialmente o polinômio foi calculado com precisão de 128 bits, levando menos de um minuto para tanto. Com base nesse cálculo, a precisão estimada para a computação real foi 42208 bits. A obtenção das raízes do polinômio não foi cronometrada, mas levou em torno de 20 horas. A construção do polinômio a partir de suas raízes levou 6h10m. O tempo de redução modular dos coeficientes do polinômio é insignificante, assim como o tempo de construção da curva dado uma raiz desse polinômio, sobrando um tempo de aproximadamente 1 hora para a extração dessa raiz pelo algoritmo de Cantor-Zassenhaus.

Referências

[1] Alfred J. Menezes. Elliptic curve cryptosystems. `ftp://ftp.rsa.com/pub/cryptobytes/crypto1n2.pdf`, 1995.

[2] Paul Zimmermann. The ECMNET project. `http://www.loria.fr/~zimmerma/records/ecmnet.html`.

[3] Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Walter M. Lioen, Peter L. Montgomery, Brian A. Murphy, and Herman J. J. te Riele et al. Factorization of a 512-bit RSA modulus. Technical Report MAS-R0007, Centrum voor Wiskunde en Informatica, February 2000.

[4] Mike Oakes. ECPP proof of 7760-digit Euler-irregular prime. Comunicação à lista de discussão NMBRTHRY, `http://listserv.nodak.edu/scripts/wa.exe?A2=ind0407&L=nmbrthry&F=&S=&P=%2292`.

[5] François Morain. A new large primality proof using fastECPP. Comunicação à lista de discussão NMBRTHRY, `http://listserv.nodak.edu/scripts/wa.exe?A2=ind0407&L=nmbrthry&F=&S=&P=%2165`.

[6] Richard Crandall and Carl Pomerance. *Prime Numbers: A Computational Perspective*. Springer-Verlag, New York, 2000.

[7] Henri Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer, Berlin, Heidelberg, New York, third edition, 1996.

[8] S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. In *Proceedings of the 18th Annual ACM Symposium on the Theory of Computing*, pages 316–329, 1986.

[9] A. O. L. Atkin and François Morain. Elliptic curves and primality proving. *Mathematics of Computation*, 61(203):29–68, 1993.

[10] Trolltech. Qt application framework. `http://www.trolltech.com`.

[11] John M. Pollard. A Monte Carlo method for factorization. *Nordisk Tidskr. Informationsbehandling (BIT)*, 15:331–334, 1975.

[12] John M. Pollard. Theorems on factorization and primality testing. In *Proceedings of the Cambridge Philosophical Society*, volume 76, pages 521–528, 1974.

[13] Hendrik W Lenstra Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 2:649–673, 1987.

[14] Richard P. Brent. An improved Monte Carlo factorization algorithm. *Nordisk Tidskr. Informationsbehandling (BIT)*, 20:176–184, 1980.

[15] Jens Franke, T. Kleinjung, François Morain, and T. Wirth. Proving the primality of very large numbers with fastECPP. `ftp://lix.polytechnique.fr/pub/submissions/morain/Preprints/large.ps.gz%`, 2004. Manuscript.

[16] Reynald Lercier. Elliptic curves with complex multiplication. Comunicação à lista de discussão NMBRTHRY, `http://listserv.nodak.edu/scripts/wa.exe?A2=ind0401&L=nmbrthry&F=&S=&P=%1098`.