Paralelização em *software* do Algoritmo de Miller

Diego F. Aranha[1], **Julio López**

[1] Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
CEP 13083-970 – Campinas – SP - Brazil

{dfaranha,jlopez}@ic.unicamp.br

Abstract. *In this paper we devise a parallelization of Miller's Algorithm to compute bilinear pairings. Our method provides a generic parallel algorithm for pairing computation which is independent of the pairing definition. The performance of the algorithm is illustrated with the parallel implementation of the R-ate asymmetric pairing and the η_T symmetric pairing in a computer with two Intel Core Quad processors. Both pairings are instantiated with parameters compatible with the AES-128 security level. We obtain performance gains of 10% with the parallel execution of the asymmetric pairing in 2 processor cores and 81% with the parallel execution of the symmetric pairing in 8 processor cores. The speedup with 8 cores is two times the best speedup obtained by previous works with the same parameters.*

Resumo. *Neste trabalho, uma paralelização do Algoritmo de Miller para o cálculo de emparelhamentos bilineares é derivada. Este método fornece um algoritmo paralelo genérico independente da definição do emparelhamento. O desempenho do algoritmo é ilustrado a partir da implementação paralela do emparelhamento assimétrico* R-ate *e do emparelhamento simétrico η_T em um computador com dois processadores Intel Core Quad. Os emparelhamentos são instanciados com parâmetros compatíveis com o nível de segurança AES-128. A execução paralela do emparelhamento* R-ate *em 2 processadores fornece 10% de ganho de desempenho e a execução paralela do emparelhamento η_T em 8 processadores resulta em uma aceleração de 81%. A aceleração com 8 processadores é o dobro da melhor aceleração obtida por trabalhos anteriores com os mesmos parâmetros.*

1. Introdução

A criptografia baseada em emparelhamentos possibilitou a flexibilização das primitivas criptográficas conhecidas e ampliou de forma considerável os cenários de aplicação de criptografia assimétrica. Entretanto, o desempenho de sistemas baseados em emparelhamentos ainda representa um obstáculo. Tipicamente, o cálculo de um emparelhamento ainda é mais caro do que uma execução de um protocolo variante do RSA e uma ordem de magnitude menos eficiente do que uma multiplicação de ponto em curvas elípticas [1]. Isto é natural, visto que os métodos mais estabelecidos de criptografia assimétrica puderam receber maior esforço de pesquisa para ganho de eficiência. Em criptografia baseada em emparelhamentos [2], esforço similar resultou em novos algoritmos [3, 4, 5] e novas curvas para instanciação [6].

[1]Financiado por FAPESP, Processo No. 2007/06950-0.

O obstáculo de desempenho é ainda mais relevante em dispositivos embarcados, especialmente em níveis altos de segurança [7]. Considerando a tendência tecnológica recente da indústria de computação em migrar o projeto de processadores para arquiteturas multiprocessadas, mesmo no espaço de projeto de processadores embarcados [8], algoritmos paralelos para o cálculo de emparelhamentos em arquiteturas com múltiplas unidades de processamento de baixo poder computacional são desejáveis, como sugerido em [9] e [10].

As contribuições deste trabalho, que visam preencher esta lacuna, são:

- Uma *paralelização do Algoritmo de Miller* é derivada a partir de uma propriedade elementar de funções de Miller. Esta paralelização fornece ganho de desempenho escalável com o número de processadores disponíveis e é completamente independente da definição do emparelhamento, permitindo escalabilidade em um número variado de processadores e podendo ser aplicada a qualquer emparelhamento calculado a partir de funções de Miller. A paralelização também não requer custo de armazenamento por processador superior ao exigido por uma execução serial do algoritmo;
- Uma *técnica para balanceamento estático de carga* entre processadores distintos é utilizada para determinar a partição que minimiza o tempo de execução do algoritmo paralelo. A mesma técnica pode ser empregada para determinar partições com frações não-ótimas controladas do custo total do algoritmo;
- *Resultados teóricos e experimentais de desempenho* para a paralelização de duas instanciações otimais [11] de emparelhamentos, uma simétrica e outra assimétrica. Instanciações sobre corpos finitos de característica pequena apresentam os resultados mais promissores, permitindo uma aceleração de 5 vezes com a execução paralela do emparelhamento η_T no nível de segurança AES-128 em uma arquitetura com 8 unidades de processamento.

As características de independência da instanciação do emparelhamento e balanceamento flexível de carga são especialmente importantes em dispositivos embarcados, visto que estes dispositivos frequentemente empregam escalonamento dinâmico de tarefas para obter compromissos entre tempo de execução e consumo de energia.

2. Definições preliminares

O Algoritmo de Miller para o cálculo de emparelhamentos requer diversos fundamentos matemáticos que serão brevemente apresentados nesta seção. Referências mais completas para estes fundamentos incluem [2] e [3].

Seja $\mathbb{F}_q$ o corpo finito de ordem q. Uma curva elíptica E é o conjunto de soluções (x, y) que satisfazem a equação de $Weiertra\beta$ na forma $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, com coeficientes $a_i \in \mathbb{F}_q$. Os pontos racionais de E, em conjunto com uma regra de adição, forma um grupo abeliano aditivo cuja identidade é o ponto no infinito $\mathcal{O}$. A *ordem* $\#E$ da curva é o número de pontos racionais. A *condição de Hasse* enuncia que $\#E = q + 1 - t$, onde t é o *traço de Frobenius* limitado por $|t| \leq 2\sqrt{q}$. Curvas em que o traço t é múltiplo da característica $char(\mathbb{F}_q)$ são chamadas curvas *supersingulares*.

Seja n a ordem da curva E. A *ordem* de um ponto $P \in E(\mathbb{F}_q)$ é o menor inteiro $r > 0$ tal que $rP = \mathcal{O}$ e sempre divide n. O conjunto de pontos de torção r de $E(\mathbb{F}_q)$, denotado

por $E(\mathbb{F}_q)[r]$, é o conjunto $\{P \in E(\mathbb{F}_q) | rP = \mathcal{O}\}$. Destas definições, segue que $\langle P \rangle$, o grupo de pontos gerado por P, é um subgrupo de $E(\mathbb{F}_q)[r]$, que por sua vez é um subgrupo de $E(\mathbb{F}_q)[n]$. Seja P um ponto de $E(\mathbb{F}_q)$ de ordem prima r. O subgrupo $\langle P \rangle$ possui *grau de mergulho* k, para algum $k > 0$, se k é o menor inteiro tal que $r|(q^k - 1)$.

Um *divisor* é uma soma formal de pontos $P_i \in E$ com coeficientes inteiros d_i denotado por $\mathcal{D} = \sum_{P_i \in E} d_i(P_i)$. O *grau* de um divisor é a soma dos coeficientes inteiros $deg(\mathcal{D}) = \sum_{P_i \in E} d_i$. O conjunto de divisores forma um grupo abeliano $Div(E)$ com regra de adição $\sum_{P_i \in E} a_i(P_i) + \sum_{P_i \in E} b_i(P_i) = \sum_{P_i \in E}(a_i + b_i)(P_i)$. A adição repetida de um divisor a si mesmo $n - 1$ vezes é dada por $n\mathcal{D} = \sum_{P \in E} nd_i(P_i)$. O *suporte* de um divisor é o conjunto de pontos P_i com coeficientes não-nulos.

O divisor de uma função racional não-nula $f : E(\mathbb{F}_{q^k}) \rightarrow \mathbb{F}_{q^k}$ na curva E, denotado por $div(f)$, é chamado de *divisor principal* e definido por $div(f) = \sum_{P_i \in E} ord_{P_i}(P_i)$, onde $ord_{P_i}(f)$ é a *multiplicidade* de f em P_i. Se $\mathcal{D}$ é um divisor principal, então $deg(\mathcal{D}) = 0$ e $\sum_{P_i \in E} d_i P_i = \mathcal{O}$. Dois divisores $\mathcal{C}$ e $\mathcal{D}$ são equivalentes ($\mathcal{C} \sim \mathcal{D}$) se a diferença $\mathcal{C} - \mathcal{D}$ é um divisor principal. Quando $div(f)$ e $\mathcal{D}$ possuem suportes disjuntos, a função f pode ser avaliada em $\mathcal{D}$ pelo cálculo de $\prod_{P_i \in E} f(P_i)^{d_i}$. Seja um ponto $P \in E(\mathbb{F}_q)[r]$ com $mdc(r, q) = 1$ e seja $\mathcal{D}$ um divisor equivalente a $(P) - (\mathcal{O})$. Como $rP = \mathcal{O}$ e $deg(\mathcal{D}) = 0$, o divisor $\mathcal{D}$ é principal e existe uma função $f_{r,P}$ tal que $div(f_{r,P}) = r(P) - r(\mathcal{O})$.

Um *emparelhamento bilinear admissível* é um mapa eficientemente computável $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ que mapeia elementos de grupos de pontos $\mathbb{G}_1$ e $\mathbb{G}_2$ a elementos não-triviais de um grupo multiplicativo $\mathbb{G}_T$ relacionado à $\mathbb{F}_q$. Se $\mathbb{G}_1 = \mathbb{G}_2$, o emparelhamento é dito simétrico. A propriedade de bilinearidade, $e(aU, bW) = e(U, W)^{ab}$, é fundamental para o projeto de protocolos criptográficos. O cálculo do emparelhamento $e(P, Q)$ requer a construção e avaliação de $f_{r,P}$ em um divisor equivalente a $(Q) - (\mathcal{O})$. Miller [12] constrói $f_{r,P}$ em estágios utilizando um método de duplicação e adição. Seja $g_{U,V} : E(\mathbb{F}_{q^k}) \rightarrow \mathbb{F}_{q^k}$ a equação da linha que passa pelos pontos U e V de $E(\mathbb{F}_q)$, com k o grau de mergulho. Se $U = V$, a linha $g_{U,V}$ é a tangente à curva em U. Se $V = -U$, a linha vertical $g_{U,-U}$ é abreviada por g_U. Uma *função de Miller* é qualquer função $f_{c,P}$ com divisor $div(f_{c,P}) = c(P) - (cP) - (c - 1)(\mathcal{O})$, $c \in \mathbb{Z}$. A propriedade a seguir é verdadeira para quaisquer inteiros a e b [3, Teorema 2]:

$$f_{a+b,P}(\mathcal{D}) = f_{a,P}(\mathcal{D}) \cdot f_{b,P}(\mathcal{D}) \cdot \frac{g_{aP,bP}(\mathcal{D})}{g_{(a+b)P}(\mathcal{D})}. \quad (1)$$

Corolários diretos são [3]:

(i) $f_{1,P}(\mathcal{D}) = 1$;
(ii) $f_{a,P}(\mathcal{D}) = f_{a-1,P}(\mathcal{D}) \cdot \frac{g_{(a-1)P,P}(\mathcal{D})}{g_{aP}(\mathcal{D})}$;
(iii) $f_{2a,P}(\mathcal{D}) = f_{a,P}(\mathcal{D})^2 \cdot \frac{g_{aP,aP}(\mathcal{D})}{g_{2aP}(\mathcal{D})}$.

Utilizando esta propriedade, Miller propôs o Algoritmo 1 [12]. O trabalho de Barreto *et al.* [3] posteriormente demonstrou como utilizar a exponenciação final empregada pelo emparelhamento de Tate para eliminar os denominadores envolvidos e avaliar f sobre o

ponto Q no lugar do divisor $\mathcal{D}$. Otimizações adicionais na literatura são direcionadas a minimizar o custo computacional do laço principal do algoritmo, ou seja, reduzir a magnitude de r mantendo seu peso de Hamming baixo [4, 13, 5].

Algoritmo 1 Algoritmo de Miller [12].

Entrada: $r = \sum_{i=0}^{\log_2(r)} r_i 2^i, P, Q + R, R.$
Saída: $f_{r,P}(\mathcal{D})$.

1: $T \leftarrow P, f \leftarrow 1$
2: **for** $i = \log_2(r) - 1$ **downto** 0 **do**
3: $\quad T \leftarrow 2T$
4: $\quad f \leftarrow f^2 \cdot \frac{g_{T,T}(Q+R)g_{2T}(R)}{g_{2T}(Q+R)g_{T,T}(R)}$
5: $\quad$ **if** $r_i = 1$ **then**
6: $\quad\quad T \leftarrow T + P$
7: $\quad\quad f \leftarrow f \cdot \frac{g_{T,P}(Q+R)g_{T+P}(R)}{g_{T+P}(Q+R)g_{T,P}(R)}$
8: $\quad$ **end if**
9: **end for**
10: **return** f

3. Trabalhos relacionados

O foco deste trabalho é investigar paralelizações do Algoritmo de Miller que independam da definição matemática do emparelhamento. Desta forma, a escalabilidade dos algoritmos não é restringida pela instanciação do emparelhamento. Limites práticos para escalabilidade de algoritmos paralelos sempre existirão quando o custo de comunicação for dominante ou quando o custo da paralelização (etapas de divisão e conquista) tornar-se mais alto do que o custo computacional do algoritmo propriamente dito.

Alguns autores já tentaram desenvolver estratégias paralelas para o cálculo de emparelhamentos, com resultados diversos. O trabalho de Grabher *et al.* [9] analisa duas abordagens distintas: exploração de paralelismo na aritmética de extensão e paralelização do laço principal do Algoritmo de Miller quando aplicado ao emparelhamento de Tate. Este trabalho conclui que a exploração de paralelismo na aritmética de extensão fornece bons resultados, mas esta abordagem possui escalabilidade claramente limitada pelo grau de mergulho empregado. Na paralelização do Algoritmo de Miller proposta para dois processadores, todas as avaliações de linha do Algoritmo 1 são pré-computadas e as iterações do laço para $r_i = 0$ são executadas pelo processador 1 enquanto as iterações para $r_i = 1$ são executadas pelo processador 2. Como este método depende do pré-cálculo das avaliações de linha para todas as $\log_2(r)$ iterações do algoritmo e já que r frequentemente possui um peso de Hamming baixo, o método resulta em perda de desempenho quando comparado com a execução serial.

O trabalho de Mitsunari [14] propõe uma versão especializada do emparelhamento $\eta_T(P, Q)$ em curvas supersingulares sobre corpos de característica 3 para execução paralela em dois processadores. Nesta versão, os pontos $3^i P$ e $(3)^{-i} Q$ necessários para a avaliação de linha em cada interação i são pré-calculados e o balanceamento de carga é automaticamente

atingido pela divisão das iterações do laço principal em porções iguais entre os dois processadores. Como o cálculo destes pontos em curvas supersingulares exige apenas o cálculo de quadrados e raízes quadradas que são eficientes em corpos com característica pequena, esta abordagem atinge acelerações entre 1.61 e 1.76 em dois níveis de segurança distintos. Entretanto, há um custo significativo de armazenamento, já que $2 \cdot \log_2(r)$ pontos precisam ser pré-calculados e armazenados. Esta abordagem é generalizada e estendida para um número maior de processadores no trabalho de Beuchat *et al.* [15], onde resultados são apresentados para corpos de característica 2 e 3 no nível de segurança AES-128. Para característica 2, as acelerações atingem 1.75, 2.53 e 2.57 para 2, 4 e 8 processadores, respectivamente. Para característica 3, as acelerações são de 1.65, 2.26 e 2.79, respectivamente. Estes resultados representam o estado da arte de implementações paralelas de emparelhamentos.

Cesena [16] propõe uma técnica nova para o cálculo de emparelhamentos sobre variedades de traço zero construídas com curvas elípticas supersingulares e extensões de graus $a = 3$ ou $a = 5$. Esta abordagem permite que o cálculo de um emparelhamento seja dividido em a laços curtos pela ação da a-ésima potência do mapa de Frobenius. Estes laços curtos podem ser implementados em paralelo de forma simples e com baixo custo de paralelização. A limitação desta abordagem é novamente a escalabilidade restringida pela instanciação do emparelhamento (grau da extensão a). A partir dos dados de implementação fornecidos em [17], a aceleração atingida com 3 processadores é de 1.11 em relação a uma implementação serial do emparelhamento η_T no mesmo nível de segurança.

4. Paralelização do algoritmo

Nesta seção, a paralelização do Algoritmo de Miller é derivada. Esta paralelização pode ser utilizada tanto para acelerar implementações seriais de emparelhamentos quanto para ampliar a escalabilidade de paralelizações restringidas pela definição do emparelhamento.

Como visto na Seção 2, o Algoritmo de Miller avalia $f_{r,P}$ com divisor $(rP) - r(\infty)$ em um divisor $\mathcal{D}$ com um total de $\log_2(r)$ iterações. Intuitivamente, uma paralelização do algoritmo precisa dividir estas iterações igualmente entre um número π de processadores. Para isto, uma propriedade de fácil verificação de funções de Miller é empregada [11]:

Lema 4.1. *Sejam P, Q pontos na curva E, $\mathcal{D} \sim (Q) - (\infty)$ e $f_{c,P}$ uma função de Miller. Para quaisquer $a, b \in \mathbb{Z}$, $f_{a \cdot b,P}(\mathcal{D}) = f_{b,P}(\mathcal{D})^a \cdot f_{a,bP}(\mathcal{D})$.*

Esta propriedade é útil porque a Equação (1) apenas divide o laço principal do Algoritmo de Miller calculado em $\log_2(r)$ iterações em dois laços com no mínimo $\log_2(r) - 1$ iterações. Se r puder ser representado como um produto $r_0 \cdot r_1$, o Lema 4.1 permite calcular $f_{r,P}$ com dois laços de $\frac{\log_2(r)}{2}$ iterações. O único problema com esta abordagem é que em algumas instanciações r é um número primo. Pode-se então escrever r como $2^w r_1 + r_0$, com $w \approx \frac{\log_2(r)}{2}$ e calcular:

$$f_{r,P}(\mathcal{D}) = f_{2^w r_1 + r_0,P}(\mathcal{D}) = f_{2^w r_1,P}(\mathcal{D}) \cdot f_{r_0,P}(\mathcal{D}) \cdot \frac{g_{(2^w r_1)P, r_0 P}(\mathcal{D})}{g_{rP}(\mathcal{D})}. \quad (2)$$

O Lema 4.1 fornece duas opções para desenvolver o termo $f_{2^w r_1,P}(\mathcal{D})$:

- $f_{2^w r_1,P}(\mathcal{D}) = f_{r_1,P}(\mathcal{D})^{2^w} \cdot f_{2^w,r_1 P}(\mathcal{D})$;
- $f_{2^w r_1,P}(\mathcal{D}) = f_{2^w,P}(\mathcal{D})^{r_1} \cdot f_{r_1,2^w P}(\mathcal{D})$.

A escolha é realizada com base na eficiência: calcular w quadrados consecutivos no corpo de extensão $\mathbb{F}_{q^k}^*$ e uma multiplicação de ponto pelo inteiro r_1; ou calcular uma exponenciação por r_1 no corpo de extensão e w duplicações de ponto consecutivas. A opção mais eficiente irá depender da curva e do grau de mergulho. Como graus de mergulho mais altos são desejáveis para emparelhamentos em níveis altos de segurança, assume-se que a exponenciação no corpo de extensão tem custo superior à multiplicação de ponto. Se r possui peso de Hamming baixo, as duas opções devem ter custos similares e envolver basicamente quadrados na extensão e duplicações de ponto. A primeira opção será adotada:

$$f_{r,P}(\mathcal{D}) = f_{2^w r_1 + r_0,P}(\mathcal{D}) = f_{r_1,P}(\mathcal{D})^{2^w} \cdot f_{2^w,r_1 P}(\mathcal{D}) \cdot f_{r_0,P}(\mathcal{D}) \cdot \frac{g_{(2^w r_1)P,r_0 P}(\mathcal{D})}{g_{rP}(\mathcal{D})}. \quad (3)$$

Esta fórmula é adequada para execução paralela em 3 processadores, pois cada função de Miller pode ser calculada em $\frac{\log_2(r)}{2}$ iterações. Para os casos considerados, entretanto, r terá peso de Hamming baixo e w pode ser ajustado para que r_0 seja pequeno. Assim, a função $f_{r,P}(\mathcal{D})$ pode ser calculada em dois laços de Miller com aproximadamente $\frac{\log_2(r)}{2}$ iterações. O parâmetro w pode ser aprimorado para que os custos de paralelização (w quadrados no corpo de extensão e multiplicação de ponto por r_1) sejam próximos nos dois processadores.

Esta mesma técnica pode ser aplicada recursivamente para o cálculo de $f_{r_1,P}$ e $f_{2^w,r_1 P}$ para obter uma paralelização adequada para qualquer número de processadores. Também não há restrições para o número de processadores devido à maneira flexível em que r é representado e o custo de armazenamento do algoritmo paralelo permanece similar ao custo do algoritmo serial. Um detalhe importante é que uma implementação paralela só terá aceleração significativa se o custo do laço principal no Algoritmo de Miller for dominante sobre o custo de comunicação e o custo de paralelização. Também é importante notar que o custo de paralelização cresce com o número de processadores, limitando a escalabilidade do algoritmo. A escalabilidade irá então depender do custo dos quadrados no corpo de extensão, do custo de multiplicações de ponto por r_i e do comprimento efetivo do laço principal. Felizmente, estes parâmetros são constantes e podem ser analisados estaticamente. Se P é um ponto fixo (uma chave privada, por exemplo), os múltiplos $r_i P$ podem também ser pré-calculados e armazenados com baixa sobrecarga.

5. Emparelhamentos paralelos

Nesta seção, o ganho de desempenho teórico de implementações paralelas de emparelhamentos é investigado para instanciações simétricas e assimétricas discutidas em [10]. É importante tratar as duas instanciações porque emparelhamentos simétricos e assimétricos possuem funcionalidades distintas. O emparelhamento assimétrico é o emparelhamento *R-ate* [5] sobre curvas Barreto-Naehrig (BN) [18] com grau de mergulho $k = 12$ e o emparelhamento simétrico é o emparelhamento η_T [4] sobre curvas supersingulares binárias com grau de mergulho $k = 4$. Estas duas instanciações são otimais [11].

5.1. Emparelhamento assimétrico

A curva BN selecionada é a curva $E_1/\mathbb{F}_p : y^2 = x^3+22$, com $p = 36x^4+36x^3+24x^2+6x+1$ parametrizado por $x = -0\text{x}4080000000000001$ [19]. Para esta curva, $\mathbb{G}_2$ é comumente escolhido como o *twist* sêxtuplo da curva elíptica sobre a extensão $\mathbb{F}_{p^2}$. Nesta instanciação, p tem 256 *bits* e a dificuldade do problema do logaritmo discreto nos grupos $\mathbb{G}_1$ e $\mathbb{G}_2$ é compatível com a dificuldade do problema do logaritmo discreto em $\mathbb{G}_T$ e com o nível de segurança AES-128. O emparelhamento ***R-ate***, representado por $r : \mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$, é uma generalização do emparelhamento *ate* [13] proposta por Lee, Lee e Park [5]. Quando instanciado sobre esta curva BN, o emparelhamento ***R-ate*** é definido por:

$$r(Q,P) = (f \cdot (f \cdot g_{aQ,Q}(P))^p \cdot g_{\phi(aQ+Q),aQ}(P))^{\frac{(p^{12}-1)}{r}},$$

onde $a = -6x - 3$, $f = f_{a,Q}(P)$ e $\phi(x,y) = (x^p, y^p)$ é o mapa de Frobenius. O emparelhamento ***R-ate*** pode ser calculado a partir do Algoritmo 2.

Algoritmo 2 Emparelhamento ***R-ate*** [5].

Entrada: $a = \sum_{i=0}^{\lfloor \log_2(a) \rfloor} a_i 2^i, P \in \mathbb{G}_1, Q \in \mathbb{G}_2$
Saída: $r(Q,P)$.

1: $T \leftarrow P$
2: $f \leftarrow 1$
3: **for** $i = \lfloor \log_2(a) \rfloor - 2$ **downto** 0 **do**
4: $\quad T \leftarrow 2T$
5: $\quad f \leftarrow f^2 \cdot g_{T,T}(P)$
6: $\quad$ **if** $a_i = 1$ **then**
7: $\quad\quad T \leftarrow T + Q$
8: $\quad\quad f \leftarrow f \cdot g_{T,Q}(P)$
9: $\quad$ **end if**
10: **end for**
11: $f \leftarrow f \cdot (f \cdot g_{T,Q}(P))^p \cdot g_{\phi(T+Q),T}(P)$
12: **return** $f^{(p^{12}-1)/r}$

Paralelização

Para esta instanciação de emparelhamento assimétrico, basta aplicar a fórmula paralela descrita na Equação (3) com $a = \lfloor \frac{a}{2^w} \rfloor * 2^w + 3$ para o cálculo de $f_{a,Q}(P)$:

$$f_{a,Q}(P) = f_{\lfloor \frac{a}{2^w} \rfloor * 2^w + 3,Q}(P) = f_{\lfloor \frac{a}{2^w} \rfloor,Q}(P)^{2^w} \cdot f_{2^w,\lfloor \frac{a}{2^w} \rfloor Q}(P) \cdot f_{3,Q}(P) \cdot \frac{g_{\lfloor \frac{a}{2^w} \rfloor 2^w Q,3Q}(P)}{g_{aQ}(P)}.$$

Inicialmente, será considerada a execução paralela em $\pi = 2$ processadores. O processador 1 calculará a primeira função de Miller e o processador 2 calculará as duas funções de Miller restantes e a acumulação da avaliação de linha associada.

Balanceamento de carga

É preciso determinar um valor de w que equilibre a carga nos dois processadores. Seja m o custo de uma multiplicação ou quadrado em $\mathbb{F}_p$ e i o custo de uma inversão. Seguindo a análise de desempenho presente em [10] para o Algoritmo 2, cada execução da linha 4 custa $17m$, cada execução da linha 5 custa $15m + 36m + 39m = 90m$, cada execução da linha 7 custa $30m$ e cada execução da linha 8 custa $10m+39m = 49m$. A linha 11 e a exponenciação final custam $260m + i$ e $6220m$, respectivamente.

O processador 1 calculará as linhas 4 e 5 por $(63 - w)$ vezes e as linhas 7 e 8 por 5 vezes, referentes aos *bits* com valor 1 localizados na parte alta de a. O custo de paralelização do processador 1 é $(w \cdot 36m)$ para o cálculo de w quadrados no corpo de extensão. O processador 2 calculará as linhas 4 e 5 por w vezes e as linhas 4, 5, 7 e 8 por uma única vez durante o cálculo de $f_{3,Q}(P)$. A contribuição da avaliação de linha adicional custa $10m$ e a acumulação dessa contribuição em f custa $39m$. O custo de paralelização do processador 2 é de $(63 - w) \cdot 17m + 5 \cdot 30m$ para a multiplicação $\lfloor \frac{a}{2^w} \rfloor Q$. Combinar os resultados dos dois processadores requer uma multiplicação na extensão de custo $54m$. O custo total para o processador 1 é, portanto, $c_2(1) = 107m(63 - w) + 395m + 36mw$ e o custo do processador 2 é, portanto, $c_2(2) = 107mw + 235m + 17m(63 - w) + 150m$. Resolvendo $c_2(1) = c_2(2)$ para w, obtém-se um valor aproximado $w = 35$ para a partição ótima entre 2 processadores. A linha 11 do algoritmo e a exponenciação final são executadas serialmente. Representando o custo do algoritmo quando executado serialmente por $c_1(1) = 14067m + i$ e considerando o custo de uma inversão obtido experimentalmente $i = 278m$, a aceleração obtida com a execução paralela em dois processadores é:

$$s(2) = \frac{c_1(1)}{c_2(1) + 54m + 260m + 6220m + i} = \frac{14067m + 278m}{4651m + 6534m + 278m} = 1.25$$

Devido ao alto custo de paralelização para esta instanciação específica de emparelhamento, tentativas adicionais de paralelização para um número de processadores superior a 2 resultaram em perdas de desempenho.

5.2. Emparelhamento simétrico

Seja E a curva supersingular com grau de mergulho $k = 4$ definida sobre $\mathbb{F}_{2^m}$ com equação $E/\mathbb{F}_{2^m} : y^2 + y = x^3 + x + b$. A ordem de E é $2^m + 1 \pm 2^{\frac{m+1}{2}}$. Uma extensão quártica é construída sobre $\mathbb{F}_{2^m}$ com base $\{1, s, t, st\}$, onde $s^2 = s + 1$ e $t^2 = t + s$. Um mapa de distorção associado $\psi : E(\mathbb{F}_{2^m})[r] \rightarrow E_2(\mathbb{F}_{2^{4m}})$ é definido por $\psi : (x, y) \rightarrow (x + s^2, y + sx + t)$. Sejam $P, Q \in E(\mathbb{F}_{2^m})$. Para esta família de curvas, Barreto *et al.* [4] definiu o emparelhamento $\eta_T : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ a partir de otimizações do emparelhamento de Tate:

$$\eta_T(P, Q) = f_{T',P'}(\psi(Q))^M,$$

onde $T' = (-v)(2^m - \#E_2(\mathbb{F}_{2^m}))$, $P' = (-v)P$ e $M = (2^{2m} - 1)(2^m + 1 \pm 2^{\frac{m+1}{2}})$ para um parâmetro $v \in \{-1, 1\}$ dependente de b.

Para esta instanciação satisfazer o nível de segurança AES-128, o corpo binário precisa ter $m = 1223$ *bits* [10]. Seja $E_2/\mathbb{F}_{2^m} : y^2 + y = x^3 + x$ a curva de ordem $5r = 2^{1223} + 2^{612} + 1$ definida sobre o corpo binário representado com a base polinomial $z^{1223} + z^{255} + 1$. Para esta curva, $T' = 2^{612} + 1$, $P' = -P$ e $M = (2^{2446} - 1)(2^{1223} - 2^{\frac{612}{2}} + 1)$.

Paralelização

Aplicando a fórmula paralela presente na Equação (3), o cálculo do emparelhamento pode ser decomposto em:

$$f_{T',P'}(\psi(Q))^M = \left(f_{2^{612-w},P'}(\psi(Q))^{2^w} \cdot f_{2^w,2^{612-w}P'}(\psi(Q)) \cdot \frac{g_{2^{612}P',P'}(\psi(Q))}{g_{T'P'}(\psi(Q))} \right)^M$$

Como o cálculo de quadrados no corpo de extensão $\mathbb{F}_{2^{4m}}$ é eficiente [20] e a duplicação de pontos em curvas supersingulares binárias requer apenas quadrados, esta instanciação apresenta baixo custo de paralelização. Entretanto, aprimoramentos ainda são possíveis. Em [4], Barreto *et al.* propuseram uma fórmula fechada para o cálculo do emparelhamento η_T utilizando uma abordagem reversa com raízes quadradas. Esta otimização elimina os quadrados no corpo de extensão, e consequentemente reduz o custo de paralelização. Beuchat *et al.* [20] encontraram pequenas otimizações adicionais e propuseram o Algoritmo 3.

Algoritmo 3 Abordagem reversa para o cálculo do emparelhamento η_T. [4, 20]

Entrada: $P = (x_P, y_P), Q = (x_Q, y_Q) \in E(\mathbb{F}_{2^m}[r])$.
Saída: $\eta_T(P, Q) \in \mathbb{F}^*_{2^{4m}}$.
Nota: α, β, δ são parâmetros dependentes de m e b.

1: $y_P \leftarrow y_P + 1 - \delta$
2: $u \leftarrow x_P + \alpha, v \leftarrow x_Q + \alpha$
3: $g_0 \leftarrow u \cdot v + y_P + y_Q + \beta$
4: $g_1 \leftarrow u + x_Q, g_2 \leftarrow v + x_P^2$
5: $G \leftarrow g_0 + g_1 s + t$
6: $L \leftarrow (g_0 + g_2) + (g_1 + 1)s + t$
7: $F \leftarrow L \cdot G$
8: **for** $i \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
9: $\quad x_P \leftarrow \sqrt{x_P}, y_P \leftarrow \sqrt{y_P}, x_Q \leftarrow x_Q^2, y_Q \leftarrow y_Q^2$
10: $\quad u \leftarrow x_P + \alpha, v \leftarrow x_Q + \alpha$
11: $\quad g_0 \leftarrow u \cdot v + y_P + y_Q + \beta$
12: $\quad g_1 \leftarrow u + x_Q$
13: $\quad G \leftarrow g_0 + g_1 s + t$
14: $\quad F \leftarrow F \cdot G$
15: **end for**
16: **return** $F^{(2^{2m}-1)(2^m+1\pm 2^{\frac{m+1}{2}})}$

Pode-se obter uma paralelização direta para este algoritmo aplicando a fórmula derivada acima e eliminando-se os quadrados no corpo de extensão. O algoritmo final consistirá simplesmente em dividir as iterações do laço principal em processadores distintos com o custo de paralelização reduzido às multiplicação de ponto pelos inteiros r_i. Este algoritmo é apresentado como o Algoritmo 4.

Algoritmo 4 Abordagem paralela reversa para o cálculo do emparelhamento η_T.

Entrada: $P = (x_P, y_P), Q = (x_Q, y_Q) \in E(\mathbb{F}_{2^m}[r])$.
Saída: $\eta_T(P, Q) \in \mathbb{F}^*_{2^{4m}}$.

$y_P \leftarrow y_P + 1 - \delta$
parallel section(processador i)
if $i = 1$ **then**
  $u_i \leftarrow x_P + \alpha, v_i \leftarrow x_Q + \alpha$
  $g_{0_i} \leftarrow u_i \cdot v_i + y_P + y_Q + \beta$
  $g_{1_i} \leftarrow u_i + x_Q, g_{2_i} \leftarrow v_i + x_P^2$
  $G_i \leftarrow g_{0_i} + g_{1_i} s + t$
  $L_i \leftarrow (g_{0_i} + g_{2_i}) + (g_{1_i} + 1)s + t$
  $F_i \leftarrow L_i \cdot G_i$
else
  $F_i \leftarrow 1$
end if
$x_{Q_i} \leftarrow (x_Q)^{2^{w_i}}, y_{Q_i} \leftarrow (y_Q)^{2^{w_i}}$
$x_{P_i} \leftarrow (x_P)^{\frac{1}{2^{w_i}}}, y_{P_i} \leftarrow (y_P)^{\frac{1}{2^{w_i}}}$
for $j \leftarrow w_i$ **to** $w_{i+1} - 1$ **do**
  $x_{P_i} \leftarrow \sqrt{x_{P_i}}, y_{P_i} \leftarrow \sqrt{y_{P_i}}, x_{Q_i} \leftarrow x_{Q_i}^2, y_{Q_i} \leftarrow y_{Q_i}^2$
  $u_i \leftarrow x_{P_i} + \alpha, v_i \leftarrow x_{Q_i} + \alpha$
  $g_{0_i} \leftarrow u_i \cdot v_i + y_{P_i} + y_{Q_i} + \beta$
  $g_{1_i} \leftarrow u_i + x_{Q_i}$
  $G_i \leftarrow g_{0_i} + g_{1_i} s + t$
  $F_i \leftarrow F_i \cdot G_i$
end for
end parallel
$F \leftarrow \prod_{i=1}^{\pi} F_i$
return F^M

Balanceamento de carga

Cada processador inicia o laço a partir do ponto de início w_i, calculando w_i quadrados e raízes quadradas como custo de paralelização. Sem quadrados no corpo de extensão para compensar duplicações de ponto, faz sentido atribuir a primeira avaliação de linha do algoritmo para o processador 1 e aumentar as porções do laço executadas por processadores com pontos de início w_i pequenos. A etapa de combinação dos resultados parciais dos π processadores pode ser implementada em pelo menos duas formas diferentes: combinação serial com $\pi - 1$ multiplicações no corpo de extensão em um único processador; ou combinação logarítmica paralela com $\lceil \log_2(\pi) \rceil$ multiplicações no corpo de extensão. Como o número de processadores disponíveis é geralmente pequeno, a primeira forma será adotada por simplicidade.

O processador 1 tem um custo de inicialização de 3 multiplicações e 2 quadrados. O custo de paralelização do processador i é de $2w_i$ quadrados e $2w_i$ raízes quadradas. O custo para a combinação de resultados é $\pi - 1$ multiplicações no corpo de extensão. Cada multiplicação na extensão custa 9 multiplicações no corpo base. Cada iteração do algoritmo

executa 2 raízes quadradas, 2 quadrados, 1 multiplicação e 1 multiplicação pelo elemento esparso G_i, que por sua vez custa 6 multiplicações no corpo base. A exponenciação final requer 26 multiplicações, 9 quadrados, 612 quadrados no corpo de extensão e 1 inversão. Cada quadrado no corpo de extensão custa 4 quadrados no corpo base [20].

Sejam $\widetilde{m}, \widetilde{s}, \widetilde{r}, \widetilde{i}$ os custos de multiplicação, quadrado, extração de raiz quadrada e inversão em $\mathbb{F}_{1223}$, respectivamente. Verificou-se experimentalmente que $\widetilde{r} \approx \widetilde{s}$, $\widetilde{m} \approx 30\widetilde{s}$ e $\widetilde{i} \approx 10\widetilde{m}$. Seja $c_\pi(i)$ o custo computacional do processador $0 < i \leq \pi$ ao executar a sua porção do algoritmo paralelo. Para $\pi = 2$, temos:

$$\begin{aligned} c_2(1) &= (3\widetilde{m} + 2\widetilde{s}) + (7\widetilde{m} + 4\widetilde{s})w_2 = 92\widetilde{s} + (214\widetilde{s})w_2 \\ c_2(2) &= (4\widetilde{s})w_2 + (7\widetilde{m} + 4\widetilde{s})\,(611 - w_2) = (4\widetilde{s})w_2 + 214\widetilde{s}\,(611 - w_2) \end{aligned}$$

Naturalmente, $w_1 = 0$. Resolvendo $c_2(1) = c_2(2)$ para w_2, obtém-se $w_2 = 308$ ótimo. Para $\pi = 4$ processadores, pode-se resolver $c_4(1) = c_4(2) = c_4(3) = c_4(4)$ e obter $w_2 = 157, w_3 = 311$ e $w_4 = 462$. Sempre é possível balancear a carga entre os π processadores, obtendo-se um laço principal no Algoritmo de Miller com latência $c_\pi(1)$. Seja $c_1(1)$ o custo de uma implementação serial. Considerando o custo adicional de $9(\pi-1)$ multiplicações para combinar os resultados e um custo da exponenciação final de $26\widetilde{m}+(9+2448)\widetilde{s}+\widetilde{i} = (3537)\widetilde{s}$, a aceleração teórica da execução paralela para $\pi = 2^l (l > 0)$ processadores é:

$$s(\pi) = \frac{c_1(1)}{c_\pi(1) + 9(\pi - 1)\widetilde{m} + (3537)\widetilde{s}} = \frac{92 + 214 \cdot 611 + 3537}{c_\pi(1) + 270(\pi - 1) + 3537}.$$

A Tabela 5.2 apresenta as acelerações teóricas para várias escolhas do número π de processadores quando os valores w_i são determinados para uma partição ótima. A paralelização proposta apresenta boa escalabilidade até 8 processadores, resultando em uma aceleração de pelo menos 5 vezes. O ganho de desempenho é saturado em $\pi = 16$ processadores e para $\pi > 16$, o custo de paralelização começa a se tornar proibitivo.

Tabela 1. Aceleração teórica para paralelização proposta do emparelhamento η_T sobre curvas supersingulares binárias no nível de segurança AES-128.

Número de processadores	**1**	**2**	**4**	**8**	**16**	**32**
Aceleração teórica	1	1.92	3.54	5.87	7.92	7.77

6. Resultados experimentais

A aritmética envolvida no cálculo de emparelhamentos foi implementada de forma que os custos relativos entre as operações satisfizessem as estimativas utilizadas durante as análises de desempenho. A plataforma utilizada foi um computador com dois processadores Intel Core Quad com frequência de 2.4GHz. A implementação foi realizada na linguagem C com o compilador GCC 4.1.2. O corpo finito primo foi implementado sobre a camada de baixo nível da biblioteca GMP[1] e o corpo finito binário foi implementado de forma eficiente com instruções vetoriais da família SSE. Para implementação das seções paralelas, a tecnologia

[1]GNU Multiple Precision Arithmetic Library. `http://www.gmplib.org`

OpenMP[2] foi empregada e verificou-se que os custos de sincronização e comunicação na plataforma eram desprezíveis em relação ao tempo de execução dos emparelhamentos.

Os resultados são apresentados na Tabela 2 e tomados como a média aritmética de 100 execuções consecutivas do emparelhamento. A Tabela encontra-se dividida em três seções, onde a primeira seção apresenta os resultados experimentais deste trabalho, a segunda apresenta os resultados experimentais que compõem o estado da arte anterior e a terceira apresenta o ganho em aceleração deste trabalho em relação à maior aceleração presente na segunda seção. As acelerações são sempre calculadas como a razão entre o tempo de execução serial e o tempo de execução paralela para o número correspondente de processadores. Os resultados do estado da arte são apresentados para duas plataformas distintas: uma plataforma Intel Core2 Quad com freqüência de 2.4GHz (P1) e uma máquina octoprocessada Intel Core i7 com freqüência de 2.9GHz (P2). Resultados paralelos anteriores para o emparelhamento *R-ate* não são apresentados por não haver propostas de paralelização publicadas na literatura.

Tabela 2. Resultados experimentais para a execução paralela dos emparelhamentos. Os tempos de execução são apresentados em milissegundos e as acelerações são calculadas como a razão entre o tempo serial e o tempo paralelo de uma mesma implementação. A última linha representa o ganho em aceleração da paralelização proposta quando comparada ao estado da arte anterior.

	Emparelhamento					
	R-ate		η_T			
Número de processadores	**1**	**2**	**1**	**2**	**4**	**8**
Proposta – Tempo de execução em ms	5.73	5.17	8.45	4.66	2.71	1.61
Proposta – Aceleração (*speedup*)	-	1.11	-	1.81	3.12	5.25
Hankerson *et al.* [10] em P1 – Tempo	4.17	-	16.25	-	-	-
Beuchat *et al.* [15] em P1 – Tempo	-	-	11.19	6.72	4.22	-
Beuchat *et al.* [15] em P1 – Aceleração	-	-	-	1.67	2.65	-
Beuchat *et al.* [15] em P2 – Tempo	-	-	7.94	4.53	3.13	3.08
Beuchat *et al.* [15] em P2 – Aceleração	-	-	-	1.75	2.53	2.57
Melhoria na aceleração	-	-	-	3.4%	17.7%	104.3%

A partir da tabela, pode-se observar que implementações reais podem obter até 89% do limite teórico de aceleração calculado anteriormente. A diferença é explicada por uma estimativa otimista dos custos da porção serial do Algoritmo de Miller, em especial da exponenciação final, que descarta algumas operações de custo relativamente baixo mas que tornam-se relevantes para o tamanho de parâmetros considerado. Um exemplo deste tipo de operação é a adição no corpo de extensão [10]. Houve ganho de aceleração em todos os casos considerados, com a vantagem adicional de que a paralelização proposta não introduz custo significativo de armazenamento. Em particular, a execução paralela do emparelhamento η_T em 8 processadores permitiu uma aceleração de 5 vezes que corresponde ao dobro da melhor aceleração obtida anteriormente neste mesmo cenário.

[2]Open Multi-Processing. `http://www.openmp.org`

7. Conclusões e trabalhos futuros

O cálculo de emparelhamentos bilineares é a operação de maior custo computacional em criptografia baseada em emparelhamentos, chegando a ser proibitiva em dispositivos embarcados. Neste trabalho, uma paralelização do Algoritmo de Miller adequada para implementação em dispositivos embarcardos multiprocessados foi derivada. Esta paralelização é genérica e pode ser aplicada a qualquer emparelhamento, permitindo execução paralela escalável sem introduzir custo significativo de armazenamento.

Para ilustrar a paralelização, uma instanciação simétrica e outra assimétrica de emparelhamentos foram paralelizadas e suas acelerações teóricas foram calculadas. Observou-se que instanciações de emparelhamentos sobre corpos com característica pequena possuem baixo custo de paralelização e fornecem boa aceleração quando executadas em arquiteturas multiprocessadas. Instanciações sobre corpos primos, entretanto, possuem custo de paralelização significativo e apresentaram baixa aceleração. Os resultados experimentais demonstraram que as acelerações teóricas podem ser alcançadas na prática e foram obtidos ganhos de desempenho de 10% para a instanciação assimétrica executada em 2 processadores e de 81% para a instanciação simétrica executada em 8 processadores. Apesar do ganho de desempenho obtido com a instanciação assimétrica ser pequeno, este resultado é o primeiro a obter aceleração utilizando uma abordagem paralela sem recorrer à exploração do paralelismo na aritmética do corpo de extensão. A instanciação simétrica quando executada em 8 processadores permitiu o dobro do ganho em aceleração fornecido pela melhor paralelização proposta na literatura.

Com enfoque na minimização da região executada serialmente e na redução dos custos de paralelização, como trabalhos futuros sugere-se a investigação de novos algoritmos eficientes para o cálculo de: (i) quadrados e raízes quadradas consecutivas em corpos binários; (ii) quadrados consecutivos em corpos de extensão; (iii) duplicações repetidas em curvas primas (ou curvas BN); (iv) exponenciação final de emparelhamentos. Outra alternativa é investigar a combinação entre a abordagem proposta e a exploração de paralelismo no nível de aritmética da extensão ou emprego de variedades de traço zero. Possivelmente partes desde trabalho podem ser utilizadas para acelerar implementações seriais convencionais de emparelhamentos. O exame desta possibilidade é também deixado como trabalho futuro.

Referências

[1] M. Scott. Implementing cryptographic pairings. Tech Report, 2009. `ftp://ftp.computing.dcu.ie/pub/resources/crypto/pairings.pdf`.

[2] P. S. L. M. Barreto, B. Lynn, and M. Scott. Efficient Implementation of Pairing-Based Cryptosystems. *Journal of Cryptology*, 17(4):321–334, 2004.

[3] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO '02*, pages 354–368, London, UK, 2002. Springer.

[4] P. S. L. M. Barreto, S. Galbraith, C. Ó hÉigeartaigh, and M. Scott. Efficient Pairing Computation on Supersingular Abelian Varieties. *Design, Codes and Cryptography*, 42(3):239–271, 2007.

[5] H. Lee E. Lee and C. Park. Efficient and Generalized Pairing Computation on Abelian Varieties. *IEEE Transactions on Information Theory*, 55(4):1793–1803, 2009.

[6] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 2009.

[7] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In *European Conference on Wireless Sensor Networks (EWSN'08)*, pages 305–320. Springer-Verlag, LNCS 4913, 2008.

[8] F. Zhao N. B. Priyantha and D. Lymberopolous. mPlatform: A reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. Tech Report, 2006. `http://research.microsoft.com/pubs/70353/tr-2006-142.pdf`.

[9] P. Grabher, J. Groszschaedl, and D. Page. On Software Parallel Implementation of Cryptographic Pairings. In *Selected Areas in Cryptography (SAC '08)*, pages 34–49. Springer Verlang, LNCS 5381, 2008.

[10] D. Hankerson, A. Menezes, and M. Scott. *Identity-Based Cryptography*, chapter 12, pages 188–206. IOS Press, 2008.

[11] F. Vercauteren. Optimal pairings. Cryptology ePrint Archive, Report 2008/096, 2008.

[12] V. S. Miller. The Weil Pairing, and Its Efficient Calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[13] F. Hess, N. P. Smart, and F. Vercauteren. The eta pairing revisited. *IEEE Transactions on Information Theory*, 52:4595–4602, 2006.

[14] S. Mitsunari. A Fast Implementation of η_T Pairing in Characteristic Three on Intel Core 2 Duo Processor. Cryptology ePrint Archive, Report 2009/032, 2009.

[15] J. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari, and F. Rodríguez-Henríquez. Multi-core Implementation of the Tate Pairing over Supersingular Elliptic Curves. Cryptology ePrint Archive, Report 2009/276, 2009.

[16] E. Cesena. Pairing with Supersingular Trace Zero Varieties Revisited. Cryptology ePrint Archive, Report 2008/404, 2008.

[17] E. Cesena and R. Avanzi. Trace Zero Varieties in Pairing-based Cryptography. In *CHiLE '09*. `http://inst-mat.utalca.cl/chile2009/Slides/Roberto_Avanzi_2.pdf`.

[18] P. S. L. M. Barreto and M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In *Selected Areas in Cryptography*, pages 319–331, 2005.

[19] Y. Nogami, M. Akane, Y. Sakemi, H. Kato, and Y. Morikawa. Integer Variable χ – Based Ate Pairing. In *Pairing '08*, pages 178–191. Springer-Verlag, 2008.

[20] J. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodríguez-Henríquez. A Comparison Between Hardware Accelerators for the Modified Tate Pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$. In *Pairing '08*, pages 297–315, 2008.