Implementação eficiente de emparelhamentos bilineares sobre curvas elípticas na plataforma ARM

Victor Henrique Hisao Taira[1], **Diego F. Aranha**[1]

[1] Departamento de Ciência da Computação – Universidade de Brasília (UnB)
CEP 70910-900 – Brasília – DF - Brasil
victaira@gmail.com, dfaranha@unb.br

Abstract. *The discovery of cryptosystems based on elliptic curves produced a new revolution in cryptography, because of their low requirements for key storage and low computational cost for execution. Furthermore, the discovery of cryptographic techniques based on bilinear pairings on elliptic curves brought even new applications such as identity-based cryptographic protocols. However, the computational cost of these cryptosystems remains significantly higher than the traditional ones, representing an obstacle to their adoption, especially on devices with limited resources. Among such devices, smartphones and tablets have achieved great success in the market by ensuring access to increasingly large computing capacities combined with mobility and portability. Since most of these devices run on an ARM architecture , this work aims to improve the overall performance of cryptographic methods based on bilinear pairings over elliptic curves on ARM-based platforms, obtaining a 17% performance gain in comparison with the current state of the art.*

Resumo. *A descoberta de criptossistemas baseados em curvas elípticas produziu uma nova revolução na área de criptografia, por possuirem baixos requisitos para armazenamento de chaves e baixo custo computacional para execução. Além disso, a descoberta de técnicas criptográficas baseada em emparelhamentos bilineares sobre curvas elípticas trouxe ainda novas aplicações, como protocolos criptográficos baseados em identidades. No entanto, o custo computacional desses criptossistemas ainda permanece significativamente maior do que os tradicionais, representando um obstáculo para sua adoção, especialmente em dispositivos com recursos limitados. Dentre esse dispositivos, os smartphones e os tablets têm obtido grande sucesso no mercado por garantir acesso a capacidades cada vez maiores de computação aliadas a mobilidade e portabilidade. Como grande parte desses dispositivos rodam sobre uma arquitetura ARM, este trabalho tem como objetivo geral aprimorar o desempenho dos métodos de criptografia baseado em emparelhamentos bilineares sobre curvas elípticas em processadores ARM, obtendo um ganho de desempenho de 17% em comparação com o estado da arte atual.*

1. Introdução

Atualmente, técnicas criptográficas necessitam ser utilizadas em diversos dispositivos e sistemas móveis, que vão ganhando espaço cada vez maior no mercado. *Tablets, smartphones* e outros equipamentos armazenam e transmitem informação sensível que precisa ser protegida de forma análoga a computadores *Desktop*, o que termina por diversificar as plataformas com exigências de segurança e introduz requisitos específicos a esse contexto. Naturalmente,

métodos criptográficos precisam acompanhar o avanço desses dispositivos e ser rápidos e compatíveis com as características desse ambiente.

Em 1985, Neal Koblitz e Victor Miller propuseram independentemente a utilização de curvas elípticas na construção de sistemas criptográficos de chave pública [Hankerson et al. 2004]. A descoberta de criptossistemas de chave pública baseados em curvas elípticas produziu uma nova revolução na área de criptografia, pois costumam apresentar um desempenho superior, além de exigirem um tamanho de chaves inferior para um mesmo patamar de segurança que as demais técnicas de criptografia assimétrica, principalmente quando comparado com o algoritmo RSA. Por esses motivos, em cenários com limitações quanto ao poder de processamento e armazenamento computacional, como a computação móvel, este modelo tem se mostrado ideal.

Desde então, um esforço considerável de pesquisa tem sido dedicado ao desempenho de implementações de criptografia de curvas elípticas, sendo que a eficiência desses esquemas depende de inúmeros fatores como tamanho dos parâmetros, poder de processamento disponível, otimizações de *software* e *hardware* e de aprimoramentos algorítmicos [Menezes et al. 2001]. A descoberta de técnicas criptográficas baseadas em emparelhamentos bilineares sobre curvas elípticas trouxe ainda novas aplicações, como sistemas criptográficos baseados em identidades [Boneh and Franklin 2001] e acordos de chaves multi-parte com menor custo de comunicação [Sakai et al. 2001, Joux 2004]. No entanto, o custo computacional desses criptossistemas ainda permanece significativamente maior do que os tradicionais, representando um obstáculo para sua adoção, especialmente em dispositivos com recursos limitados.

Dentre esse dispositivos, os *smartphones* e os *tablets* têm obtido grande sucesso no mercado por garantir acesso a capacidades cada vez maiores de computação, aliadas a mobilidade e portabilidade. Como grande parte desses dispositivos possuem processadores ARM, este presente trabalho tem como objetivo aprimorar o desempenho dos métodos de criptografia baseado em emparelhamentos bilineares sobre curvas elípticas nessa plataforma. As contribuições principais deste trabalho consistem em:

- *Implementação eficiente de aritmética em* $\mathbb{F}_p$: são apresentadas versões otimizadas de algoritmos conhecidos para multiplicação e redução modular em corpos primos na plataforma ARM. As otimizações produzem algoritmos mais eficientes, ao atrasar o tratamento de *carries* e acumulações, reduzindo o número de instruções de adição;
- *Recorde de velocidade para o cálculo de emparelhamentos na plataforma ARM*: a utilização de aritmética mais eficiente produz a implementação mais rápida já proposta na literatura, oferecendo um ganho de 17% em relação ao estado da arte, sem considerar conjuntos de instruções vetoriais (NEON).

O artigo está organizado da seguinte forma. Uma breve introdução teórica é apresentada na Seção 2, mostrando alguns métodos utilizados para realizar a multiplicação. Na Seção 3 são apresentadas algumas características da plataforma ARM. Na Seção 4 apresenta-se a implementação proposta para aritmética em corpos finitos na plataforma-alvo e na Seção 5 comparam-se os resultados obtidos com trabalhos relacionados. A Seção 6 finaliza o artigo com as conclusões obtidas.

2. Revisão Teórica

2.1. Emparelhamentos bilineares

Sejam $\mathbb{G}_1$ e $\mathbb{G}_2$ grupos cíclicos aditivos e $\mathbb{G}_T$ um grupo cíclico multiplicativo tais que $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T|$. Seja P o gerador de $\mathbb{G}_1$ e Q o gerador de $\mathbb{G}_2$. Um mapeamento $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ é dito um emparelhamento bilinear se:

1. Dados $(V, W) \in \mathbb{G}_1 \times \mathbb{G}_2$, temos:

$$e(P, Q + Z) = e(P, Q) \cdot e(P, Z) \tag{1}$$

$$e(P + V, Q) = e(P, Q) \cdot e(V, Q) \tag{2}$$

2. $e(P, Q) \neq 1_{\mathbb{G}_T}$, onde $1_{\mathbb{G}_T}$ é a identidade do grupo $\mathbb{G}_T$
3. O mapeamento e deve ser computado com complexidade de ordem polinomial.

Emparelhamentos bilineares foram propostos inicialmente para atacar sistemas criptográficos de curvas elípticas ao reduzir os logaritmos da curva para logaritmos em corpos finitos [Menezes et al. 1993] aproveitando-se da estrutura algébrica mais rica. Porém, com a descoberta da utilidade do emparelhamento para realizar a cifração baseada em identidade [Sakai et al. 2001] um repertório de novos protocolos foi desenvolvido. A idéia de sistemas baseados em identidade é aproveitar-se da existência de informações públicas autênticas para simplificar a autenticação de chaves públicas.

O custo de técnicas criptográficas baseadas em emparelhamento bilineares ainda é significativamente mais alto que técnicas alternativas. Porém, os algoritmos para cálculo do emparelhamento ainda se encontram em aperfeiçoamento, estando longe de ter sido atingido um limite computacional, haja em vista o grande esforço em pesquisa que tem sido investido.

2.2. Métodos de Multiplicação

O desempenho do cálculo de emparelhamentos bilineares depende fundamentalmente do desempenho da operação de multiplicação modular em $\mathbb{F}_p$. Nesta subseção, algumas das estratégias para implementação dessa operação são descritas. Na Figura 1, os quatro métodos abaixo são ilustrados com a multiplicação de operandos de oito palavras. As setas indicam a acumulação dos produtos parciais.

Método *Schoolbook*

O método mais intuitivo para se implementar a operação de multiplicação é o algoritmo *Schoolbook*. Inicialmente, para realizar a multiplicação, todas as possíveis posições do vetor de palavras que armazena o resultado da operação são inicializados com o valor 0. Em seguida, devemos fixar um dígito do multiplicador (iteração externa) e processar os dígitos do multiplicando um a um, da direita para a esquerda (iteração interna), calculando a multiplicação entre dígitos e deslocando a posição do produto intermediário um dígito para a esquerda em relação aos produtos intermediários anteriores.

Método Comba

Este método foi proposto por Paul G. Comba [Comba 1990], como uma variação do algoritmo *Schoolbook*. Aqui, o processamento se dá pelas colunas do produto e o n-ésimo dígito do produto é computado acumulando-se todos os produtos de pares de dígitos cujo somatório dos índices seja equivalente a n. Uma maneira popular de implementar este método consiste em manter uma janela tripla de registradores para acumular resultados de dupla-precisão provenientes de multiplicações entre dígitos. Portanto, o terceiro registrador da janela é utilizado para acumular os *carries* das somas intermediárias.

O algoritmo Comba tende a ser mais eficiente do ponto de vista computacional do que o algoritmo Schoolbook, já que o primeiro executa menos operações de memória que o segundo, pois boa parte das operações podem ser realizadas em registradores quando os resultados intermediários são acumulados na janela tripla.

Método Híbrido

O método híbrido [Gura et al. 2004] para a multiplicação foi desenvolvido para combinar as vantagens dos métodos *Schoolbook* e Comba, mantendo o desempenho do método Comba e minimizando o instruções de leitura da memória utilizando registradores adicionais.

O método híbrido pode ser implementado utilizando dois laços, sendo o laço externo uma multiplicação *Schoolbook* e o laço mais interno uma multiplicação Comba, conforme a Figura 1. Veja que os produtos parciais são processados seguindo uma separação em blocos, sendo que em cada bloco os operandos são processados seguindo o algoritmo *Schoolbook*. Dessa forma, pode-se economizar algumas instruções de leitura da memória caso existam registradores disponíveis. No entanto, no laço externo do método híbrido os operandos são processados seguindo o método Comba. Dessa forma, entre dois blocos consecutivos não é possível aproveitar os operandos, sendo necessário propagar todos os operandos da memória novamente, pois não há compartilhamento dos mesmos. Entretanto, o método híbrido consegue reduzir o número de instruções de acesso à memória para leitura de operandos, o que aumenta o desempenho, sendo aconselhável o seu uso em arquiteturas que disponham de um número grande de registradores.

Método de reutilização de operandos

O método de reutilização de operandos[1] [Hutter and Wenger 2011] nada mais é do que a implementação da multiplicação Comba tendo o seu cálculo subdivido em linhas. A idéia principal desse método é reduzir o número de instruções de leitura da memória pela reutilização cuidadosa dos operandos. Esse método mostra que um pequeno aumento no número de instruções de escrita em memória, permite um economia significativa de instruções de leitura, reutilizando operandos que já foram carregados em registradores. Para

[1]Do inglês *operand-caching*.

isso, um conjunto de resultados parciais é também escrito em memória e precisa ser carregado a cada mudança de linha.

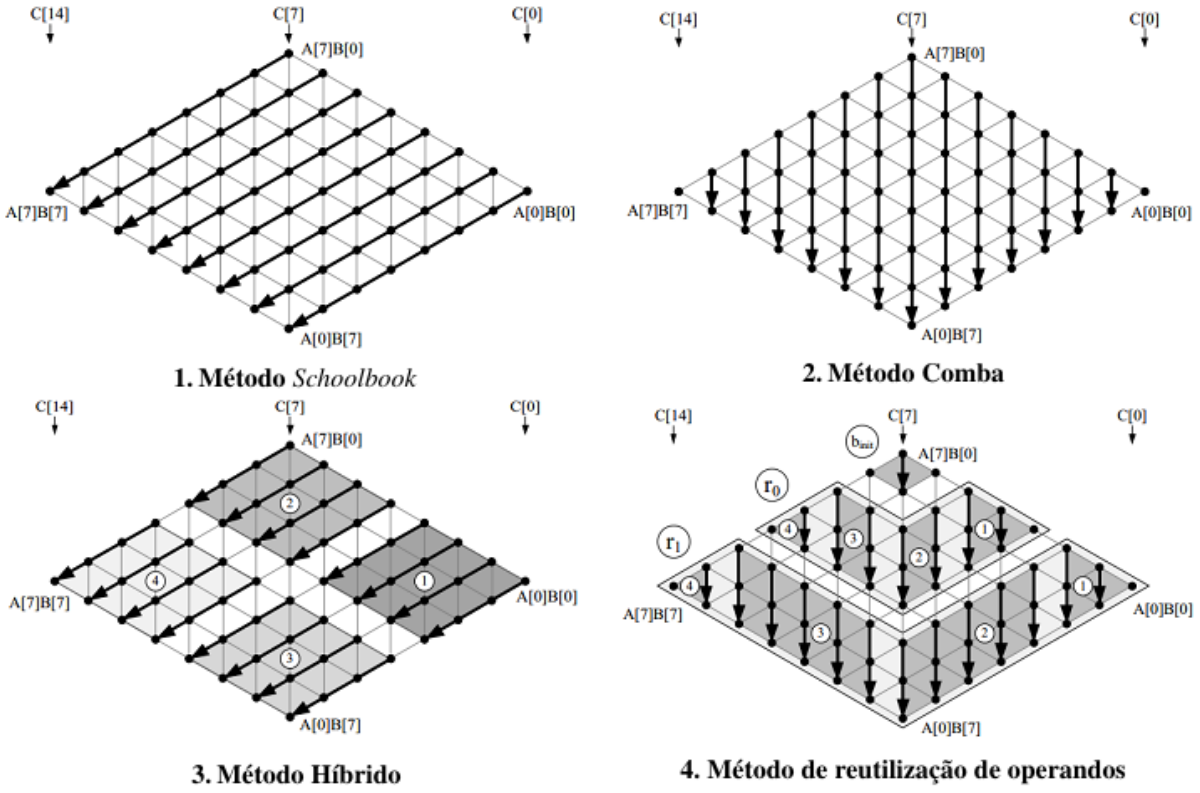

Figura 1. Diferentes métodos de multiplicação [Hutter and Wenger 2011].

3. Plataforma Alvo

Grande parte dos *smartphones* e *tablets* rodam sobre uma arquitetura denominada ARM (*Advanced RISC Machine*). Essa arquitetura possui uma interface de programação RISC (*Reduced Instruction Set Computer*) de 32 *bits*. Visando um melhor desempenho em sistemas embarcados, a arquitetura ARM não segue a risca todos os conceitos clássicos adotados pela arquitetura RISC [Sloss et al. 2004], mesmo porque o objetivo não é somente uma grande velocidade natural do processador, mas um desempenho eficiente do sistema como um todo enquanto se mantém controlado o consumo de energia.

Essa arquitetura dispõe de dezesseis registradores de uso geral, embora três possuam funções específicas. O controle de execução condicional é realizado por meio dos códigos de condição que determinam se o processador irá executar ou não determinada instrução. A grande maioria das instruções ARM possui condicionais mnemônicos que determinam se esta será executada de acordo com um conjunto de códigos de condição.

Para implementar a multiplicação foi utilizada a instrução `UMLAL` (*unsigned multiply accumulate long*) [ARM Limited 2003]. Esta instrução recebe quatro parâmetros, os dois

primeiros armazenam a parte alta e a parte baixa do resultado. Os dois parâmetros restantes referem-se aos valores a serem multiplicados. Note que esta instrução primeiro realiza a multiplicação dos dois últimos parâmetros e em seguida acumula o resultado nos dois primeiros parâmetros, sendo o resultado final armazenado nesses par de registradores. No entanto, a instrução `UMLAL` não escreve uma possível *flag* de *carry* que poderia ser resultado da acumulação.

4. Implementação

Neste trabalho, foi realizada a otimização e implementação de primitivas de operações aritméticas em corpos primos utilizando *Assembly* ARM. Utilizou-se como base a biblioteca RELIC versão 0.3.4 [Aranha and Gouvêa]. Como ambiente de programação foi utilizada a IDE Eclipse com as ferramentas de desenvolvimento para o sistema operacional *Android*, incluindo SDK (*Software Development Kit*) e NDK (*Native Development Kit*) para construção de aplicações. A utilização do NDK permite que o código de alto nível na linguagem Java faça chamadas a rotinas críticas para desempenho implementadas em *Assembly*. Para realizar as medições de tempo, foi utilizado o dispositivo Samsung Galaxy Note equipado com a versão 4.0.3 do sistema operacional.

Utilizou-se como parâmetro a curva elíptica Barreto-Naehrig $y^2 = x^3 + 2$ no nível de segurança de 128 *bits* e definida sobre um corpo primo $\mathbb{F}_p$, com $|p|$ = 254 *bits*. A curva é parametrizada com módulo $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ e ordem $n = 36u^4 + 36u^3 + 18u^2 + 6u + 1$, onde $u = -(2^{62} + 2^{55} + 1)$ [Aranha et al. 2011].

4.1. Adição e Subtração Modular

A implementação em *Assembly* ARM foi realizada desenrolando o laço de repetição, uma vez que o número de iterações é previamente conhecido. Como estamos utilizando inteiros de 254 *bits* são necessárias oito iterações. Com isso evitamos a utilização de instruções de desvio condicional e economizam-se registradores e instruções, pois não é preciso reservar um registrador para a variável de controle do laço nem realizar o incremento da mesma.

Na implementação dos algoritmos de adição e subtração modular foram utilizadas instruções de desvio condicional (*branch*), pois a introdução de adição ou subtração modular condicional teve impacto negativo no desempenho.

4.2. Multiplicação

As otimizações propostas nesse artigo foram implementadas no método de multiplicação com reutilização de operandos, mas também podem ser realizada na multiplicação Comba.

Propagação atrasada de *carries* e acumulação atrasada de resultados

A otimização consiste em atrasar a propagação de *carry* ou acumulação de resultado parcial a ser realizada ao final de uma coluna da multiplicação para a próxima adição utilizando o mesmo registrador. O trecho de código a seguir ilustra a otimização.

```
/* Multiplicacao de digitos em r6 e r7 */
UMLAL r4, r12, r6, r7
        /* Inicializacao de auxiliar */
        MOV r14, #0
        /* Leitura de operandos futuros */
        LDR r8,  [r1, #(4*5)]
        LDR r11, [r2, #(4*2)]
ADDS r3, r3, r12
/* ADC r5, r5, #0 - Instrucao eliminada */
/* Multiplicacao de digitos em r8 e r9 */
UMLAL r3, r14, r8, r9
        /* Escrita de resultado pronto */
        STR r4, [r0, #(4*4)]
        /* Inicializacao de auxiliares */
        MOV r4, #0
        MOV r12, #0
ADCS r5, r5, r14  /* Propagacao atrasada */
ADC r4, r4, #0
```

```
/* Multiplicacao de digitos em r6 e r7 */
UMLAL r3, r12, r6, r7
        /* Inicializacao de auxiliares */
        MOV r14, #0
        /* Recarga de linha anterior */
        LDR  r10, [r0, #(4*6)]
/* ADDS r4, r4, r12 - Instrucao eliminada */
/* ADC r5, r5, #0 - Instrucao eliminada */

/* Acumulacao na coluna seguinte */
ADDS r3, r3, r10 /* Acumulacao atrasada */
ADCS r4, r4, r12 /* Propagacao atrasada */
/* ADC  r5, r5, #0 - Instrucao eliminada */

/* Escrita de resultado pronto */
STR r3, [r0, #(4*6)]
```

No trecho à esquerda, a propagação de *carry* é atrasada de uma coluna para outra e na mesma linha do método de reutilização de operandos. No trecho à direita, tanto a acumulação de resultado parcial quanto a propagação de *carry* são atrasadas entre colunas consecutivas e linhas diferentes do método de reutilização de operandos. A janela de registradores utilizadas para acumular os resultados parciais no código em *Assembly* corresponde aos registradores `r3`, `r4` e `r5`. Os registradores `r12` e `r14` são utilizados como registradores auxiliares para a captura do código de condição de *carry*, lembrando que a instrução `UMLAL` não captura uma possível *flag* de *carry* que pode ser resultado da acumulação. Dessa forma, deve-se obter essa possível *flag* de *carry* de forma manual.

Para obter a *flag* de *carry*, antes da instrução `UMLAL`, o registrador auxiliar tem o seu valor zerado. E em seguida é utilizada a instrução `UMLAL` tendo o resultado dessa operação, parte alta e parte baixa, armazenado no registrador `r5` e no auxiliar, respectivamente. Repare que como o registrador auxiliar foi zerado no passo anterior, ele armazena somente a parte baixa da multiplicação. Não foi realizada ainda a operação de acumulação. Essa acumulação é realizada manualmente utilizando a instrução `ADDS`, tendo como operandos o conteúdo do registrador auxiliar e `r4`. Como foi utilizado o sufixo `S`, uma possível *flag* de *carry* será capturada nessa instrução. Finalmente, essa possível *flag* é adicionada ao último registrador da janela de acumuladores com a instrução `ADC`, finalizando o tratamento para captura manual da *flag* de *carry*.

A propagação atrasada de *carries* pode ser realizada porque na arquitetura ARM é possível controlar a atribuição de *flags* com a utilização do sufico `S`. Como nem a instrução de multiplicação `UMLAL` nem instruções de acesso à memória influenciam a *flag* de *carry*, e não existem instruções com sufixo `S` entre uma adição e outra, uma possível *flag de carry* é preservada e sua propagação oportunamente atrasada até uma instrução de soma posterior.

Escalonamento de instruções

O código acima demonstra não só as otimizações propostas, mas o escalonamento cuidadoso de instruções. O registrador `r14`, previamente utilizado como registrador para armazenar o

endereço de retorno de função (*lr - link register*), é armazenado no início da rotina na pilha de execução. Dessa forma pode-se utilizar o mesmo para funções gerais. Na implementação realizada, utiliza-se o registrador `r14` juntamente com o `r12` como registradores auxiliares para captura manual da *flag* de *carry* ao utilizar a instrução `UMLAL`.

No algoritmo de multiplicação implementado, é utilizada a instrução `UMLAL` armazenando o resultado em um dos registradores de acumulação `r3`, `r4`, `r5` e em um dos registradores auxiliares `r12`, `r14`. No entanto, logo em seguida é preciso propagar o resultado para os demais registradores de acumulação. Porém, o resultado do `UMLAL` pode levar de três a sete ciclos de *clock* para ficar pronto dependendo do tamanho das entradas [ARM Limited 2003]. Dessa forma, para aproveitar a latência, sempre que possível foi escalonada alguma instrução entre o `UMLAL` e a instrução de adição que utiliza seu resultado. As instruções mais utilizadas foram as de leitura de memória, carregando operando em registradores para serem usados posteriormente, e instruções para zerar algum registrador auxiliar, para a captura manual do *carry*.

4.3. Redução Modular

Para a redução modular, foi implementado o algoritmo de Montgomery utilizando a multiplicação Comba [Großschädl et al. 2005]. Na implementação realizada utilizou-se a propagação atrasada de *carries* e o escalonamento já descritos nas seções anteriores.

Além disso, foi utilizado o método da redução modular preguiçosa que tem como ideia principal reduzir o número de operações de redução modular. Para isso substituem-se as ocorrências de $((ab \mod p)+(cd \mod p))$ no cálculo do emparelhamento por $((ab+cd) \mod p))$, conforme proposto em [Aranha et al. 2011].

5. Resultados

Para a tomada de tempo foi utilizado como padrão o número de instruções (em milhões de ciclos) necessárias para execução do cálculo do emparelhamento bilinear sobre curvas elípticas. Para verificar a correção dos algoritmos implementados foi utilizado um módulo específico de testes de correção da biblioteca RELIC v0.3.4.

Foi realizada a comparação dos resultados obtidos neste trabalho com trabalhos relacionados (Tabela 1). Para uma comparação justa, foi considerado o mesmo emparelhamento bilinear no mesmo nível de segurança em todas as implementações. Comparando o resultado com a implementação [Sánchez and Rodríguez-Henríquez 2013] em C obteve-se uma redução em torno de 27,4% do tempo de execução. A implementação dos mesmos autores em NEON é mais eficiente que a do presente trabalho, no entanto possui um nível de complexidade maior por utilizar instruções vetoriais. Em relação à implementação [Grewal et al. 2012] em C, obteve-se uma melhoria de cerca de 26,7%. Comparando agora com a implementação em *Assembly* dos mesmos autores houve uma melhoria de 16,8%. Já em relação ao trabalho [Acar et al. 2013], houve uma melhoria de cerca de 80,6% no tempo do cálculo do emparelhamento.

Implementação	Tecnologia	Tempo	Ganho
[Acar et al. 2013], NVidia Tegra 2 Cortex-A9 1.0 GHz	C	51,01	80,6
[Sánchez and Rodríguez-Henríquez 2013], Galaxy Note Exynos 4 Cortex-A9 1.4 GHz	C	13,62	27,4
[Grewal et al. 2012], Galaxy Nexus TI OMAP 4460 Cortex-A9 1.2 GHz	C	13,49	26,7
[Grewal et al. 2012], Galaxy Nexus TI OMAP 4460 Cortex-A9 1.2 GHz	*Assembly*	11,89	16,8
[Sánchez and Rodríguez-Henríquez 2013], Galaxy Note Exynos 4 Cortex-A9 1.4 GHz	NEON	9,48	-4,1
Este trabalho, Galaxy Note Cortex-A9 1.4 GHz	*Assembly*	9,89	-

Tabela 1. Comparação com trabalhos relacionados. Tempo em milhões de ciclos e ganho de desempenho percentual.

6. Conclusão

Neste trabalho foram apresentadas otimizações em implementações de operações aritméticas em corpos primos com o objetivo de melhorar a eficiência de sistemas criptográficos baseados em emparelhamentos bilineares sobre curvas elípticas em processadores ARM. As otimizações propostas consistem em atrasar a propagação de *carries* e acumulação de resultados intermediários, com o objetivo de reduzir o número de instruções de adição envolvidas nas operações de multiplicação e redução modular. Com base na melhor implementação de complexidade similar disponível na literatura [Grewal et al. 2012], foi possível verificar, após a adição das acelerações em *Assembly* ARM, uma redução de cerca de 17% no tempo de execução do emparelhamento. Comparando com outras implementações também na plataforma ARM, a implementação proposta apresentou um ganho entre 26% e 80% de desempenho. Espera-se que estes resultados possam ampliar ainda mais a eficiência e a viabilidade da criptografia de emparelhamentos bilineares baseados em curvas elípticas em ambientes móveis, principalmente na plataforma ARM.

Referências

Acar, T., Lauter, K., Naehrig, M., and Shumow, D. (2013). Affine Pairings on ARM. In Abdalla, M. and Lange, T., editors, *5th International Conference on Pairing-Based Cryptography (PAIRING 2012)*, volume 7708 of *LNCS*, pages 203–209. Springer.

Aranha, D. F. and Gouvêa, C. P. L. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.

Aranha, D. F., Karabina, K., Longa, P., Gebotys, C. H., and López, J. (2011). Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In Patterson, K., editor, *30th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2011)*, volume 6632 of *LNCS*, pages 48–68. Springer.

ARM Limited (2003). ARM Architecture Reference Manual. ARM Doc No. DDI-0100.

Boneh, D. and Franklin, M. K. (2001). Identity-Based Encryption from the Weil Pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 2001)*, volume 2139 of *LNCS*, pages 213–229. Springer.

Comba, P. G. (1990). Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 29(4):526–538.

Grewal, G., Azarderakhsh, R., Longa, P., Hu, S., and Jao, D. (2012). Efficient Implementation of Bilinear Pairings on ARM Processors. In Knudsen, L. R. and Wu, H., editors, *Selected Areas in Cryptography (SAC 2012)*, volume 7707 of *LNCS*, pages 149–165. Springer.

Großschädl, J., Avanzi, R. M., Savas, E., and Tillich, S. (2005). Energy-Efficient Software Implementation of Long Integer Modular Arithmetic. In *7th international Conference on Cryptographic Hardware and Embedded Systems (CHES 2005)*, volume 3659 of *LNCS*, pages 75–90. Springer.

Gura, N., Patel, A., Wander, A., Eberle, H., and Shantz, S. C. (2004). Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In Joye, M. and Quisquater, J.-J., editors, *6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, volume 3156 of *LNCS*, pages 119–132. Springer.

Hankerson, D., Menezes, A. J., and Vanstone, S. (2004). *Guide to Elliptic Curve Cryptography*. Springer.

Hutter, M. and Wenger, E. (2011). Fast Multi-precision Multiplication for Public-Key Cryptography on Embedded Microprocessors. In Preneel, B. and Takagi, T., editors, *Cryptographic Hardware and Embedded Systems (CHES 2011)*, volume 6917 of *LNCS*, pages 459–474. Springer.

Joux, A. (2004). A One Round Protocol for Tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276.

Menezes, A., Okamoto, T., and Vanstone, S. (1993). Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Inform. Theory*, 39:1639–1646.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S. (2001). *Handbook of Applied Cryptography*. CRC Press.

Sakai, R., Ohgishi, K., and Kasahara, M. (2001). Cryptosystems Based on Pairing over Elliptic Curve. In *The 2001 Symposium on Cryptography and Information Security (SCIS 2001)*.

Sloss, A. N., Symes, D., and Wright, C. (2004). *F System Developer's Guide: Designing and Optimizing System Software*. Elsevier.

Sánchez, A. and Rodríguez-Henríquez, F. (2013). NEON Implementation of an Attribute-Based Encryption Scheme. In Jacobson, M., Locasto, M., Mohassel, P., and Safavi-Naini, R., editors, *Applied Cryptography and Network Security*, volume 7954 of *LNCS*, pages 322–338. Springer.