Felipe de Azevedo Piovezan

ANÁLISE DA EFICIÊNCIA ENERGÉTICA DE ALGORITMOS DE CRIPTOGRAFIA BASEADOS EM CURVAS ELÍPTICAS

Monografia submetida ao Curso de Bacharelado em Ciência da Computação para a obtenção do Grau de Bacharel em Ciência da Computação.
Orientador: Prof. Dr. Luiz Cláudio Villar dos Santos
Coorientador: Prof. Dr. Daniel Santana de Freitas

Florianópolis - Santa Catarina

2015

Ficha de identificação da obra elaborada pelo autor através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

A ficha de identificação é elaborada pelo próprio autor

Maiores informações em:
http://portalbu.ufsc.br/ficha

Felipe de Azevedo Piovezan

ANÁLISE DA EFICIÊNCIA ENERGÉTICA DE ALGORITMOS DE CRIPTOGRAFIA BASEADOS EM CURVAS ELÍPTICAS

Esta Monografia foi julgada aprovada para a obtenção do Título de "Bacharel em Ciência da Computação", e aprovada em sua forma final pelo Curso de Bacharelado em Ciência da Computação.

Florianópolis - Santa Catarina, 02 de Dezembro 2015.

Prof. Dr. Renato Cislaghi
Coordenador de Projetos

Prof. Dr. Luiz Cláudio Villar dos Santos
Orientador

Banca Examinadora:

Prof. Dr. Daniel Santana de Freitas
Coorientador

Prof. Dr. José Luís Almada Güntzel
Presidente

AGRADECIMENTOS

Aos meus pais, Denize e Almir, e irmã, Aline, por me sempre me apoiarem em todas as minhas decisões e por tornarem possível a minha dedicação exclusiva à graduação.

Aos meus amigos Fábio e Guilherme que, sem nem estarem cientes disso, me influenciaram a abandonar a engenharia e ir para a computação.

Ao meu coorientador, Daniel Santana, por me mostrar, já na primeira fase, o que realmente significa Ciência da Computação. Por estimular a maratona de programação na UFSC e me dar a chance de participar em tal evento.

Ao meu orientador, Luiz Cláudio, e ao professor Güntzel, por permitirem que eu faça parte do ECL e pela valiosa orientação nas atividades de pesquisa.

Aos meus colegas de laboratório, pelas discussões e a aprendizado propiciados. Em especial ao Gabriel e ao Rodrigo, os quais participaram comigo de um trabalho precursor a este.

Ao meu colega, Tarcísio, com o qual aprendi muito.

“Nevermore”

The Raven

RESUMO

A criptografia de chaves públicas (PKC) ou criptografia assimétrica, uma das bases para a comunicação segura pela Internet, fornece as primitivas para a troca de chaves, autenticação de usuários e assinatura digital. A maioria dos algoritmos de PKC usados atualmente são baseados em conjecturas de Teoria dos Números, como a dificuldade da fatoração de inteiros ou a dificuldade de alguma versão do logaritmo discreto. Este último, quando usado na versão de curvas elípticas, permite o uso de chaves e assinaturas menores para o mesmo nível de segurança de outros algoritmos, o que é interessante em ambientes com recursos computacionais limitados. Recentemente, a escala dos dispositivos que estabelecem comunicação segura pela Internet está diminuindo consistentemente. Essa miniaturização, no entanto, veio acompanhada de restrições de processamento e de suprimento energético, o que conflita com o fato de que algoritmos de PKC são computacionalmente caros. O presente trabalho pretende, a partir de simulações, avaliar o gasto energético do sistema de memórias de processadores executando algoritmos de criptografia de curvas elípticas. Experimentos com diversas curvas e configurações de memória são feitos para medir a taxa de faltas nas caches de dados e de instruções e para avaliar como as localidades espacial e temporal são capturadas por cada cache. Deste modo, será explicado por que o acesso à instruções é responsável pela maior parcela do consumo energético e por que a parcela relativa ao acesso de dados é pouco promissora em relação a otimizações. Como um resultado, serão identificadas quais técnicas de otimização terão um maior impacto no consumo energético do subsistema de memória.

Palavras-chave: eficiência energética, criptografia, otimização de código

ABSTRACT

The realm of public-key (asymmetric) cryptography (PKC) is considered to be one the pillars of secure communication over the Internet, since it includes user authentication, key exchange and digital signatures. PKC algorithms in use are mostly based on certain number-theoretic conjectures, like the difficulty of integer factorization or the difficulty of some discrete-log problem. The latter, when used in the elliptic curve setting, allows for shorter keys and signatures while maintaining the same security level of other algorithms, which is interesting in a resource constrained computing environment. Recently, the scale of the devices partaking in secure communications over the Internet has been steadily decreasing. This downscaling is responsible for new constraints on the processing power and battery supply available to each node, which directly conflicts with the fact that PKC algorithms are computationally expensive.

By means of simulations, this work will evaluate the amount of energy spent in the memory subsystem when processors execute elliptic curve algorithms. By taking into account different curves and memory configurations, it will be possible to evaluate parameters such as the miss rate of both data and instructions caches and to asses how temporal and spatial localities are captured by each cache. This, in turn, will reveal why instruction access accounts for the biggest share of energy consumption and why the contribution from data accesses is unlikely to be reduced. As a result, a class of optimization techniques which could significantly impact the energy efficiency of the memory subsystem will be identified.

Keywords: energy efficiency, cryptography, code optimization

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LISTA DE SÍMBOLOS

SUMÁRIO

1 INTRODUÇÃO

Com a expansão da Internet, não apenas servidores e *desktops* passam a fazer parte da rede, mas também um grande número de pequenos dispositivos como PDAs, celulares e sensores inteligentes. Acoplados aos processadores de tais equipamentos, encontram-se transmissores de rádio que permitem a comunicação com outros dispositivos, estejam eles em uma rede local ou à longa distância conectados por algum *gateway*. Esses dispositivos possuem limitações na capacidade de processamento, quantidade de memória e disponibilidade de bateria, além de estarem potencialmente expostos a diversos usuários maliciosos. O desafio, então, surge ao observar-se que a solução para contornar a última deficiência entra diretamente em conflito com aquelas limitações.

Os protocolos utilizados para comunicação segura, sejam eles aqueles utilizados por *browsers* (como o TLS), por terminais (como o SSH) ou até mesmo em aplicações da Internet das Coisas, dependem de duas grandes classes de algoritmos criptográficos: os simétricos e os assimétricos. Embora as mensagens sejam cifradas com algoritmos simétricos, como o AES, o esbelecimento de um canal seguro é, de modo geral, iniciado com o uso da classe assimétrica, ou Criptografia de Chaves Públicas (PKC), a qual permite a definição de um chave secreta conhecida apenas pelos participantes da conversa. Essa chave passa a ser utilizada pelo resto da sessão segura em algoritmos simétricos.

De modo geral, observa-se que algoritmos simétricos possuem implementações extremamente simples, podendo ser otimizados para diferentes configurações de processadores. É o caso do *Rijndael* (DAEMEN; RIJMEN, 1999), vencedor da competição que elegeu o *Advanced Encryption Standard* (AES) , cuja implementação pode ser feita executando-se operações matemáticas ou com o uso de *look-up tables* que variam de tamanho: 256 bytes, 1kB ou 4kB. Desse modo, permite-se a execução do AES em processadores com diferentes configurações de memória. Tamanha flexibilidade não ocorre por acaso: o algoritmo foi desenvolvido com esse objetivo, visto que a competição incluiu critérios como quantidade de memória requerida, eficiência computacional e simplicidade para avaliação dos candidatos (NECHVATAL et al., 2000).

Por outro lado, os algoritmos assimétricos possuem implementações complexas pela natureza do problema com que lidam: muitos deles estão associados ao uso de conjecturas da Teoria dos Números sob domínios de ordem muito grande, isto é, trabalham com conjuntos cuja

cardinaliade precisa de centenas ou até milhares de bits para ser representada. Um dos algoritmos mais famosos dessa classe, o RSA (*Rivest, Shamir, Adleman*) , é baseado na dificuldade da fatoração de inteiros e trabalha com números na ordem de 1024 bits. Pode-se perceber, então, um dos possíveis inconvenientes que esse algoritmo traz: a aritmética de números desse tamanho não é trivial de ser implementada, principalmente se considerarmos que processadores utilizados em, por exemplo, redes de sensores trabalham com registradores de 8 ou 16 bits[1].

Nesse cenário, as técnicas baseadas em curvas elípticas (ECC) mostraram-se muito interessantes por propiciarem um mesmo nível de segurança para um menor tamanho de chave. A Tabela 1 resume a equivalência entre os tamanhos de chaves para as duas técnicas, de acordo com (LENSTRA; VERHEUL, 2000). Pode-se notar que a diferença é próxima de uma ordem de magnitude em muitos casos e, de fato, este é um dos argumentos que motivaram a adição de algoritmos de ECC ao protocolo TLS (BLAKE-WILSON et al., 2006).

Tabela 1: Tamanho de chave (em bits) necessário para que os algoritmos ECC e RSA forneçam segurança equivalente (LENSTRA; VERHEUL, 2000).

ECC	RSA
163	1024
233	2048
283	3072
409	7680

Apesar de os algoritmos de ECC possibilitarem uma redução no tamanho das chaves, a implementação eficiente de criptografia de chaves públicas em ambientes com recursos limitados necessitou de muitos esforços da academia, conforme relatado por (WENGER; UNTERLUGGAUER; WERNER, 2013). Tamanho investimento é justificado pelo crescente número de aplicações que executam em tais ambientes, como *smart cards*, redes de sensores sem fio ou etiquetas RFID. Para o caso de redes de sensores com processadores de 8 bits, apenas em 2004 demonstrou-se a viabilidade do uso de algoritmos de ECC em tais plataformas (GURA et al., 2004). De modo similar, a primeira implementação em hardware com consumo energético baixo o suficiente para ser alimentado passivamente foi demonstrada em 2008 (HEIN; WOLKERSTORFER; FELBER, 2009). Atualmente, diversos dispositivos

[1] É o caso do *Atmel ATmega* (8 bits) ou do *Texas Instruments MSP430* (16 bits)

médicos, como marcapassos, são configurados através de um canal sem fio e possuem longevidade de 5 a 15 anos. Nesse cenário, algoritmos de curvas elípticas são empregados por proverem a seguraça e eficiência necessárias (FAN et al., 2013).

1.1 MOTIVAÇÃO E TRABALHO PROPOSTO

Não é incomum em aplicações embarcadas que uma parcela considerável da computação seja efetuada por circuitos especializados, devido às demandas de desempenho e eficiência. De fato, segundo (DALLY et al., 2008), um ASIC (*application-specific integrated circuit*) pode atingir uma eficiência de $5pJ/op$ na tecnologia CMOS de 90-nm. Por outro lado, processadores embarcados altamente eficientes atingem apenas $250pJ/op$, uma perda de 50x. É evidente, então, a troca da flexibilidade de um processador de propósito geral pela eficiência propiciada por ASICs.

Nota-se, porém, que o tempo de projeto de um ASIC pode ser considerável e que muitas aplicações não se adequam à inflexibilidade propiciada por essa solução. Soluções flexíveis são necessárias em classes de problemas onde o desenvolvimento de novos algoritmos ou o surgimento repentino de uma demanda inesperada são comuns. Em criptografia, um exemplo de tal demanda pode ser a descoberta de um novo ataque a um cifrador, o que poderia requerer o aumento de algum parâmetro, como tamanho de chave, ou até mesmo a troca por outro algoritmo, o que é facilitado em implementações via software. Pode-se citar um caso onde uma especificação de hardware não acompanhou a evolução de algoritmos criptográficos: em 2011, o conjunto de instruções ARMv8 para processadores ARM foi anunciado, trazendo instruções específicas para os algoritmos SHA1 e SHA2. O primeiro foi considerado inseguro e seu uso descontinuado a partir de 2010, enquanto o segundo já teve seu sucessor escolhido via competição em 2012 (NIST, 2012).

A busca por soluções eficientes em software para criptografia, então, não deve ser desconsiderada. A Figura 1 ilustra os fatores mais impactantes na eficiência energética de um processador embarcado. Observe que o fornecimento de dados e instruções pode requerer cerca de 70% da energia total consumida, tornando-se alvos importantes para otimizações. Note, também, que a Figura 1 se refere ao consumo de energia nas memórias integradas junto ao processador (caches) e, essencialmente, independe da taxa de faltas (embora dependa

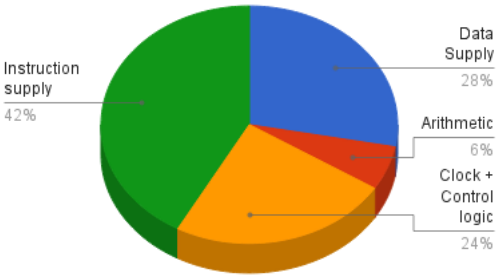

Figura 1: Consumo de energia em um processador embarcado (DALLY et al., 2008)

do número de acessos e do tamanho das caches). Por outro lado, o consumo em memória principal (não computado na Figura 1) é bastante significativo e dependente da taxa de faltas nas caches. Isso motiva o uso de otimizações de código capazes de reduzir a taxa de faltas a níveis inferiores aos obtidos com a mera exploração de localidade espacial e temporal.

Com base nisso, o presente trabalho irá analisar o comportamento do sistema de memórias de um processador embarcado executando algoritmos de curvas elípticas, variando-se parâmetros do processador (como tamanho e associatividade das caches) e parâmetros do algoritmo (como as curvas utilizadas e o tamanho de chaves). Assim, será possível relacionar a influência de tais parâmetros no consumo de energia pelo sistema de memória e identificar quais técnicas de otimização terão maior impacto na eficiência energética.

O trabalho está dividido da seguinte maneira: o Capítulo 2 define os conceitos básicos em criptografia e situa o contexto em que algoritmos de ECC são necessários, o Capítulo 3 detalha os trabalhos correlatos. O Capítulo 4 contém os experimentos realizados, a metodologia adotada e resultados obtidos. O Capítulo 5 conclui o trabalho e levanta possibilidades de trabalhos futuros.

2 CONCEITOS BÁSICOS EM CRIPTOGRAFIA

Este capítulo pretende introduzir o leitor aos conceitos básicos de criptografia, iniciando por definições clássicas, seguido dos conceitos relacionados à criptografia simétrica. O material desta seção é baseado principalmente no trabalho de Goldreich (GOLDREICH, 2006) e nas notas de aula do professor Charles Rackoff (RACKOFF, 2014), apresentando apenas uma síntese do conteúdo. O leitor interessado na bela teoria que suporta a criptografia é incentivado a consultar as referências indicadas. Embora não seja fundamental ao entendimento do resto deste trabalho, a primeira parte do capítulo é importante por situar o leitor no contexto em que a criptografia assimétrica é necessária.

Em seguida, o conceito de curva elíptica é apresentado, assim como operações necessárias à definição do logaritmo discreto. Um conhecimento básico de Teoria de Grupos é assumido, embora os conceitos mais importantes sejam revisados. Muitos dos exemplos e definições utilizados são baseados no anexo A do padrão IEEE 1363, (IEEE, 2000). Conclui-se o capítulo mostrando como todos os conceitos definidos podem ser combinados para formar um método seguro de troca de chaves, o que é conhecido como a versão de curvas elípticas de Diffie-Hellman, o qual é utilizado em todos os experimentos desse trabalho.

2.1 SESSÕES SEGURAS

A aplicação mais tradicional de criptografia, aquela descrita por um usuário comum de informática, encontra-se no estabelecimento de **sessões seguras**. Isto é, duas pessoas, A e B, tendo um segredo em comum (daqui em diante chamado de **chave**), gostariam de se comunicar através de um canal potencialmente inseguro. Com alguma frequência, A gostaria de mandar algo[1] (**texto em claro**) para B; denotar-se-á por **mensagem** tudo aquilo que A gostaria de enviar para B, embora A esteja restrito a enviar uma fração de tal mensagem por vez. Considera-se também a existência de um adversário ADV computacionalmente irrestrito, limitado a ouvir tudo que A envia pelo canal. Informalmente, A gostaria de cifrar a mensagem de modo que ADV não possa descobrir nenhuma informação sobre a mesma.

Tal cenário é representado pela Figura 2 e motiva a primeira

[1] Tratamos de comunicação unidirecional aqui por simplicidade, todos os conceitos podem ser expandidos para um canal bidirecional.

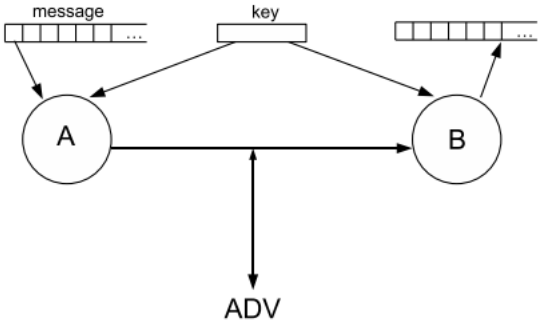

Figura 2: Modelo de uma sessão segura (RACKOFF, 2014)

definição aqui apresentada:

Definição 1 *Define-se que uma chave é um* $K \in \{0,1\}^n$*, onde* $K = K_1K_2 \ldots K_n$ *, e uma mensagem é um* $M \in \{0,1\}^m$*, onde* $M = M_1M_2 \ldots M_m$[2]*.*

Pode-se, então, definir uma função responsável por mapear uma mensagem e uma chave para uma nova string, a qual espera-se que seja "aleatória" (uma definição precisa para esse termo será dada adiante). De modo similar, precisa-se de uma função que desfaça o trabalho da primeira, isto é, uma função que receba a string "aleatória" junto com a chave utilizada para gerá-la e produza a mensagem original. Tais idéias são capturas pelas seguintes definições:

Definição 2 *Uma* encryption function *é uma função*

$$Enc : \{0,1\}^m \times \{0,1\}^n \rightarrow \{0,1\}^*$$

Uma decryption function *é uma função*

$$Dec : \{0,1\}^* \times \{0,1\}^n \rightarrow \{0,1\}^m$$

Requer-se do par (Enc, Dec) *a seguinte* ***condição de corretude****:*

$$\forall M \in \{0,1\}^m, \forall K \in \{0,1\}^n Dec(Enc(M,K),K) = M$$

[2] A notação $\{0,1\}^n$ representa uma string de exatamente n caracteres do conjunto $\{0,1\}$, enquanto $\{0,1\}^*$ denota uma string de caracteres desse mesmo conjunto, porém de tamanho arbitrário

Informalmente, $Enc(M, K)$ é entendida como a mensagem M cifrada que A envia a B, o qual aplica Dec para retornar ao texto em claro. É importante insistir na condição de corretude, uma vez que ela garante que qualquer mensagem cifrada por Enc será corretamente produzida por Dec, desde que a mesma chave seja usada em ambas as funções[3].

Pode-se agora descrever o que seria uma função segura. Informalmente, A e B escolheram de algum modo uma chave aleatória K, a qual foi utilizada para transmitir uma mensagem (cifrada) de A para B através de um canal inseguro. Idealmente, um adversário que queira descobrir a mensagem original, tendo visto o que foi enviado pelo canal e dispondo de quanto tempo quiser, não ganha nenhuma informação sobre a mesma. Isto é, sua melhor estratégia ainda consiste em fazer um palpite aleatório. Formalmente, diz-se que:

Definição 3 *O par (Enc, Dec) é dito **perfeitamente seguro** se, para toda $f : \{0,1\}^* \to \{0,1\}^m$, a seguinte proposição é válida:*

Considere o experimento em que M é escolhida de modo aleatório de $\{0,1\}^m$ e K é escolhida de modo aleatório de $\{0,1\}^n$. Então deve ser verdade que:

$$P[f(Enc(M, K)) = M] = 1/2^m$$

Note que o fato do adversário estar ilimitado computacionalmente é representado pelo fato de que ele é modelado como uma função arbitrária na definição.

No caso em que $n \geq m$, é possível criar um par (Enc, Dec) perfeitamente seguro: as funções $Enc(M, K) = M_1 \oplus K_1 M_2 \oplus K_2 \ldots M_m \oplus K_m$ e $Dec(X, K) = X_1 \oplus K_1 X_2 \oplus K_2 \ldots X_m \oplus K_m$ satisfazem a definição[4]. Na prática, porém, essa definição não é muito interessante, visto que gostaríamos de trabalhar com chaves pequenas (algumas centenas de bits) e mensagens de tamanho arbitrário[5]. Essa necessidade entra diretamente em conflito com o seguinte teorema:

Teorema 1 *Se $m > n$, isto é, se o tamanho da chave for superior ao tamanho da mensagem, então nenhum (Enc, Dec) é perfeitamente*

[3] É comum nos referirmos ao par (Enc, Dec) quando queremos discutir alguma propriedade criptográfica, visto que as duas funções geralmente estão relacionadas para conferir corretude. É um exercício interessante pensar como seria fácil obter segurança sem o requerimento de corretude.

[4] Tal função é conhecida como *one-time pad* na literatura.

[5] Embora o termo arbitrário seja utilizado, na prática espera-se que esse número seja muito inferior a 2^n

seguro.

A prova, embora simples, é omitida por estar fora do escopo do trabalho e pode ser consultada nas referências indicadas. Para contornar esse teorema poderoso, adicionaremos a restrição de que o adversário deve ser um algoritmo o qual executa em tempo polinomial (em relação a n), isto é, ele possui um tempo consideravalmente limitado para executar. Com esta restrição adicional, pode-se criar algoritmos (geradores de função) que, dado um chave, produzam uma função capaz de mapear mensagens para strings de modo aleatório, sem permitir que tais adversários limitados possam descobrir algo sobre a chave ou sobre a mensagem original. A definição formal dessa idéia é um tanto complexa, e é apresentada aqui para o leitor interessado.

Definição 4 *Um* ***gerador de funções*** *(*Function Generator*) F associa a cada $n \in \mathbb{N}$ e a cada $k \in \{0,1\}^n$ uma função $F_k : \{0,1\}^n \to \{0,1\}^n$ de modo que exista um algoritmo com tempo de execução polinomial (em n) o qual compute $F_k(x)$.*

Um gerador de funções é ***pseudo-aleatório*** *se um adversário não consegue distinguir entre um F_k (para um k escolhido aleatoriamente) e uma função $f : \{0,1\}^n \to \{0,1\}^n$ escolhida aleatoriamente com probabilidade superior a $1/n^c$ para todo c e para n suficientemente grande.*

Assume-se que os algoritmos AES e DES se comportem como geradores de funções pseudo-aleatórios[6]. O primeiro, por exemplo, associa uma chave k de 128 (196 ou 256) bits com a função AES_k e é esta função que A tipicamente usa para enviar sua mensagem para B. De fato, esses algoritmos são construídos de modo que as funções geradas sejam também permutações, permitindo que se utilize $Enc = AES_k$ e $Dec = AES_k^{-1}$. A razão pela qual simplesmente *assumimos* que geradores de função pseudo-aleatórios existam é devido ao fato dessa ser uma afirmação mais forte que $P \neq NP$, e somos incapazes de provar isso. De fato, uma condição necessária para a segurança de toda a criptografia moderna é que $P \neq NP$, embora essa não seja uma condição suficiente.[7]

[6] Um pequeno abuso é feito nessa afirmação, visto que esses algoritmos são definidos para alguns poucos valores de n.

[7] O leitor intrigado por essa observação pode notar que não basta ser difícil quebrar um sistema criptográfico no pior caso, e é essa noção que é capturada pelo problema $P \neq NP$. É possível construir um sistema criptográfico para o qual o problema de distinguir Enc é NP-Completo, embora exista um algoritmo eficiente que o resolva em 99% dos casos (GOLDREICH, 2006).

2.2 CURVAS ELÍPTICAS

Até esse momento, assumiu-se que A e B haviam previamente combinado uma chave para efetuar a comunição segura. Tal hipótese pode ser suficiente para ambientes em que poucos pares precisam se comunicar, porém não é adequada para a maioria dos outros cenários. Essa observação motiva o principal objetivo da criptografia assimétrica: como fazer com que duas pessoas possam trocar alguns poucos parâmetros através de um canal inseguro e, ao final da comunicação, ambas compartilhem uma mesma chave desconhecida a qualquer adversário escutando o canal?

Atualmente, esse objetivo é atingido com o uso de algumas conjecturas da Teoria dos Números. Uma delas é a dificuldade (de alguma versão) do problema do logaritmo discreto. Neste trabalho, estamos interessados na variação de curvas elípticas do problema, o assunto da presente seção. Definir-se-á o significado de uma curva elíptica, bem como operações que podem ser realizadas em seus elementos e, por fim, o que é o logaritmo discreto neste contexto.

Uma curva elíptica será definida como um conjunto de pontos com algumas propriedades. Ao contrário de pontos habituais, porém, as coordenadas dos pontos da curva não são número reais, mas sim inteiros de um certo conjunto, com algumas operações e propriedades. Esse tipo de conjunto é definido a seguir.

Definição 5 *Um conjunto F com duas operações binárias $+$ e $\cdot$ é um* ***corpo finito*** *(*finite field*, ou* Galois field*) se as seguintes condições forem verdadeiras:*

1. *F é um conjunto finito*
2. *$+$ é uma operação associativa*
3. *$+$ é uma operação comutativa*
4. *Existência de um elemento neutro, denotado por* 0*, para $+$*
5. *Cada elemento de F possui um inverso para $+$ em F*
6. *$\cdot$ é uma operação associativa*
7. *$\cdot$ é uma operação comutativa*
8. *Distributividade de $\cdot$ em relação a $+$ (à esquerda e à direita)*
9. *Existência de um elemento neutro, denotado por* 1*, para $\cdot$*

10. *Todo elemento de F diferente de 0 possui uma inversa para $\cdot$ em F*

O leitor certamente está habituado com tais conjuntos: os números reais com as operações de soma e multiplicação habituais satisfaz a definição, exceto pela primeira condição. Um outro conjunto menos conhecido, o qual será utilizado nos conceitos de curvas elípticas, também possui todas as propriedades necessárias:

Definição 6 *Dado um número primo p, o conjunto dos números inteiros módulo p com as operações habituais de soma e multiplicação módulo p é um corpo finito, denotado $GF(p) = \{0, 1, \ldots, p-1\}$.*

Menciona-se que, para um primo q e um inteiro positivo n, pode-se definir um corpo finito $GF(q^m)$ com o uso de polinomiais, embora sua definição precisa seja omitida por estar fora do escopo deste trabalho. Esta observação é importante pois tal conjunto pode ser usado como alternativa à $GF(p)$ em curvas elípticas. Dado um desses conjuntos, os pontos de uma curva elíptica possuem duas coordenadas (x e y), ambas em tal conjunto. Além disso, x e y estão relacionados por alguma equação, conforme a definição a seguir. Por razões discutidas mais adiante, será interessante poder somar dois pontos, o que requer um ponto especial que funcione como o zero da adição, chamado de ponto no infinito.

Definição 7 *Uma curva elíptica E sobre o corpo finito $GF(p)$, onde p é um número primo ímpar, é o conjunto:*

$$\begin{aligned} E = \{(x,y) \quad | \quad & y^2 = x^3 + ax + b \\ & \wedge \quad x, y, a, b \in GF(p) \\ & \wedge \quad 4a^3 + 27b^2 \neq 0 \ (mod\ p)\} \cup \{O\} \end{aligned}$$

onde a e b são constantes relacionadas a curva e O é chamado de ponto no infinito.

O tamanho do conjunto que define a curva será importante a diante, portanto recebe um nome especial.

Definição 8 *O número de elementos de uma curva elíptica E (sobre $GF(p)$) é chamado de* ***ordem*** *de E e é denotado por $\#E(GF(p))$.*

Para ilustrar as definições, considere a curva E dada por

$$y^2 = x^3 + x + 5$$

sobre $GF(13)$. Os pontos de E são:

$$\{O, (1,4), (1,9), (3,6), (3,7), (8,5), (8,8), (10,0), (11,4), (11,9)\}$$

Além disso, $\#E(GF(13)) = 10$.
A Figura 3 ilustra os pontos da curva sobre $GF(13)$.

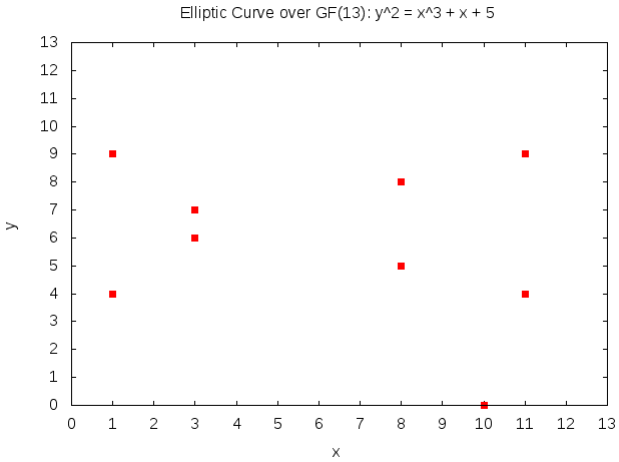

Figura 3: Conjunto de pontos que representa uma curva elíptica sobre $GF(13)$

Os próximos parágrafos são destinados a definir uma operação de soma $(+)$ para pontos de uma curva elíptica, o que viabilizará a definição de um problema conhecido como o logaritmo discreto em curvas elípticas. Tal problema é o cerne da criptografia de curvas elípticas, portanto merece uma explicação detalhada.

Intuitivamente, tem-se que a soma de dois pontos $P_1 + P_2$ é o ponto P_3 com a propriedade que P_1, P_2 e $-P_3$ são colineares. Para um ponto $P = (x, y)$, define-se $-P = (x, -y)$. Por clareza, este trabalho define a soma de dois elementos de E para o caso de $GF(p)$ através do Algoritmo 1, conforme (IEEE, 2000).

As linhas 1– 6 tratam o ponto no infinito, O, como o elementro neutro da operação. No caso em que os pontos não possuam a mesma coordenada x, as linhas 8–9 calculam o coeficiente angular da reta secante que passa por P_1 e P_2 ou, quando $P_1 = P_2$, calcula-se o coeficiente da reta tangente a curva em um dos pontos (linha 14). Caso os pontos

difiram apenas na coordenada y, então temos $P_1 = -P_2$ e o algoritmo retorna o elemento neutro (linhas 11–13). Por fim, as linhas 16–17 utilizam o coeficiente mencionado anteriormente para calcular as coordenadas de P_3.

Note que, embora termos como coeficiente angular e reta tangente sejam utilizados, os quais estão normalmente relacionados a funções sobre o domínio dos reais, o algoritmo opera sobre $GF(p)$, logo todas as operações de soma, subtração, multiplicação ou divisão que aparecem no Algoritmo 1 devem ser realizadas *mod p*. Isso ilustra uma das primeiras dificuldades de implementação de algoritmos relacionados a curvas elípticas: as operações de multiplicação e inversão em $GF(p)$ não são triviais.

Algoritmo 1: SOMA

```
   input  : P1, P2
   output: P3
 1 if P1 = O then
 2 |  return P2
 3 end if
 4 if P2 = O then
 5 |  return P1
 6 end if
 7 x1 ← P1.x,  y1 ← P1.y,  x2 ← P2.x,  y2 ← P2.y;
 8 if x1 ≠ x2 then
 9 |  λ ← (y1 − y2)/(x1 − x2)
10 else
11 |  if y1 ≠ y2 or y2 = 0 then
12 |  |  return O
13 |  end if
14 |  λ ← (3x2^2 + a)/(2y2)
15 end if
16 x3 ← λ^2 − x1 − x2   mod p ;
17 y3 ← (x2 − x3)λ − y2   mod p ;
18 return (x3, y3)
```

Com isto em mente, a intuição dada pelo domínio real ajuda a visualizar a soma em $GF(p)$, onde a idéia de colinearidade ainda pode ser representada graficamente para fins didáticos. Utilizando a mesma curva do exemplo anterior, a Figura 4 ilustra *informalmente* a adição dos pontos $P_1 = (10, 0)$ e $P_2 = (8, 5)$. Primeiro, a reta que sai de P_1 e

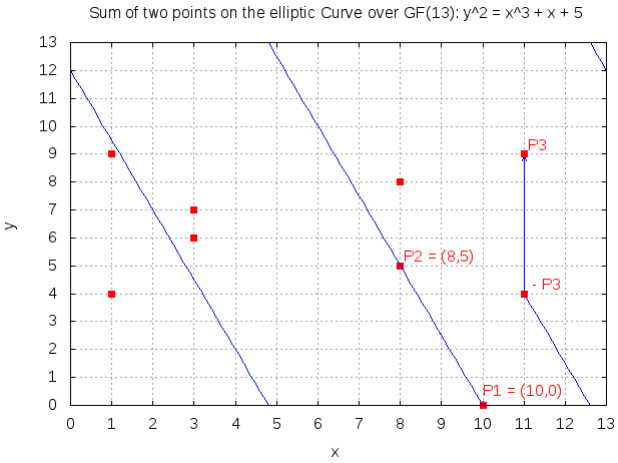

Figura 4: Somando pontos em uma curva elíptica sobre $GF(13)$

passa por P_2 é traçada. Quando esta reta atinge os limites do gráfico, isto é, uma de suas coordenadas atinge o valor 13 (ou 0), pode-se entender que a operação de módulo é aplicada (note que a operação de módulo é aplicada apenas na coordenada que atingiu um valor inteiro), efetivamente continuando a reta no lado oposto do gráfico. Tal procedimento é repetido até que um ponto de E, aqui chamado de $-P_3$, seja encontrado, de modo que P_1, P_2 e $-P_3$ sejam colineares. A soma de P_1 e P_2 é dada então por P_3. Note ainda que, caso dois pontos tenham a mesma coordenada x, a reta que passa por ambos está na vertical e, portanto, nunca tocará um terceiro ponto de E que não seja o ponto no infinito, O.

Algebricamente, teríamos que:

$$\begin{aligned}\lambda &= (y_1 - y_2)/(x_1 - x_2) \quad mod\ p \\ &= (0-5)/(10-8) \quad mod\ 13 \\ &= -5/2 \quad mod\ 13 \\ &= 8 * 7 \quad mod\ 13 \\ &= 4\end{aligned}$$

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \quad mod\ p \\ &= 4^2 - 10 - 8 \quad mod\ 13 \\ &= 11 \end{aligned}$$

$$\begin{aligned} y_3 &= (x_2 - x_3)\lambda - y_2 \quad mod\ p \\ &= (8 - 11)4 - 5 \quad mod\ 13 \\ &= 9 \end{aligned}$$

O que é equivalente às coordenadas de P_3 obtidas geometricamente.

Como consequência da operação de adição, pode-se definir a multiplicação de um ponto por um escalar, exatamente como se faz com inteiros: através de somas sucessivas. Se $k \in \mathbb{Z}$ e $P \in E$, então:

$$kP = \begin{cases} \text{O} & : k = 0 \\ \text{(-k)(-P)} & : k < 0 \\ \text{P + (k-1)P} & : k > 0 \end{cases}$$

Um observação importante é a de que, dado um ponto da curva, pode-se somá-lo repetidamente até que o elemento neutro O seja obtido. Isto acontece para qualquer ponto da curva e a quantidade de adições realizadas recebe um nome especial.

Definição 9 *A **ordem** de um ponto P em uma curva elíptica E é o menor inteiro positivo r tal que*

$$rP = O$$

Representa-se o conjunto formado pela repetida soma de um ponto P como $\langle P \rangle = \{0, P, 2P, \ldots (n-1)P\}$. Note que este é um conjunto finito.

De um resultado encontrado na Álgebra, tem-se que a ordem de um ponto sempre existe e divide a ordem da curva, $\#E(GF(p))$. Além disso, se k e l são inteiros, então $kP = lP$ se, e somente se, $k \equiv l \quad mod\ p$.

Pode-se agora definir o logaritmo discreto de um ponto em uma curva elíptica, porém é interessaante refletir sobre o logaritmo tradicional primeiro: enquanto $log_b x$ responde a pergunta de quantas vezes b

deve ser **multiplicado** por ele mesmo até que se encontre x, o logaritmo discreto de P em relação a G trabalha com quantas vezes precisa-se **somar** G até que se encontre P. No logaritmo tradicional essa pergunta está sempre bem definida (e possui resposta) para valores positivos de b e x, porém o leitor pode se perguntar se o mesmo é válido para todos os valores de G e P. A resposta é negativa, embora existam condições envolvendo os conceitos de ordem de ponto as quais permitem resposta afirmativa.

Definição 10 *Suponha que um ponto G em uma curva E tenha ordem r, onde r^2 não divida a ordem da curva $\#E(GF(p))$. Então um ponto P satisfaz $P = lG$ para algum inteiro l se, e somente se, $rP = O$. O coeficiente l é chamado de* ***logaritmo discreto*** *de P em relação ao ponto base G.*

Em termos de um problema computacional, a formulação mais tradicional é dada a seguir (TILBORG; JAJODIA, 2011):

Definição 11 *Considere uma curva elíptica E e um ponto $G \in E$ de ordem n. Dado um ponto $P \in \langle G \rangle = \{0, G, 2G, \ldots (n-1)G\}$, o* ***problema do logaritmo discreto sobre curvas elípticas*** *(ECDLP) consiste em computar $0 \leq l \leq n-1$ tal que $P = lG$.*

O algoritmo ingênuo para resolver uma instância de ECDLP consiste em computar os múltiplos de G (isto é, os membros de $\langle G \rangle$) até que se encontre P, o que pode levar até n adições, impraticável para valores muito altos de n. Atualmente, os melhores algoritmos conhecidos são baseados em variações do método *Pollard's* ρ e executam em tempo $O(\sqrt{n})$ (TESKE, 1998)(WIENER; ZUCCHERATO, 1999).

2.3 *ELLIPTIC CURVE DIFFIE HELLMAN*

O leitor deve lembrar o cenário em que a Seção 2.1 foi encerrada: A e B conseguem se comunicar de modo seguro dado que possuam uma chave secreta em comum. É exatamente esta pré-condição que será discutida a seguir, e não deve causar surpresa que o uso do ECDLP será crucial nessa tarefa.

Em 1976, Whitfield Diffie e Martin Hellman publicaram um método para a troca de chaves em um canal inseguro (DIFFIE; HELLMAN, 1976) o qual é baseado no problema do logaritmo discreto de inteiros em $GF(p)$. Embora não forneça autenticação (isto é, A não

possui garantias de que está de fato conversando com B[8]), ele é a base de protocolos com autenticação. Pouco tempo depois, em 1978, a dificuldade da fatoração de números inteiros foi utilizada por R.L. Rivest, A. Shamir, e L. Adleman para atingir o mesmo propósito, no algoritmo conhecido como RSA (RIVEST; SHAMIR; ADLEMAN, 1978). Em 1985, dois autores propuseram independentemente o uso de curvas elípticas para a troca de chaves ((KOBLITZ, 1987), (MILLER, 1986)) com tais métodos se popularizando a partir de 2004[9].

Descreve-se aqui um dos protocolos padronizados em (IEEE, 2000), lá referenciado por DL/ECKAS-DH1, ou *Discrete Logarithm and Elliptic Curve Key Agreement Scheme, Diffie-Hellman version.* A e B devem executar os seguintes passos para o estabelecimento de uma chave:

1. Em comum acordo, estabelecer um conjunto de parâmetros do domínio a ser trabalhado: os coeficientes a e b da curva, o número primo p e um ponto G de ordem r.

2. Cada uma das partes escolhe um número inteiro aleatório s no intervalo $[1, r-1]$ e computa o ponto $W = sG$. s e W são chamados de chave privada e pública, respectivamente.

3. Obter da outra parte a chave pública dela, W'.

4. Cada parte pode computar $P = sW' = (x_P, y_P)$. x_P será considerado o segredo compartilhado.

5. Aplicar uma função de derivação de chave sobre x_P para gerar a chave K.

O nome tradicional deste protocolo é ECDH (*Elliptic Curve Diffie-Hellman*).

As etapas, embora simples, merecem diversas considerações adicionais. Os parâmetros de domínio não podem ser gerados de forma totalmente aleatória: diversas curvas anômalas precisam ser evitadas

[8] Nenhum dos protocolos apresentados aqui possuirá essa característica, visto que é algo difícil de ser alcançado sem uma Infraestrutura de Chaves Públicas (*Public Key Infrastructure*) previamente estabelecida. Atualmente, isto é feito através de certificados e autoridades certificadoras. Ainda assim, tais protocolos são realistas e são utilizados em versões autenticadas dos mesmos.

[9] Recentemente, as pesquisas em algoritmos baseados em Retículados (*Lattices*) ressurgiram, visto que essa é uma classe de algoritmos resistentes a ataques por computadores quânticos, ao contrário daqueles baseados em Diffie-Hellman e RSA.

por permitirem a simplificação do ECDLP, assim como impõe-se restrições sobre a fatoração de r para evitar o mesmo problema. Tipicamente, utilizam-se parâmetros pré-estabelecidos por padrões, como as curvas padronizadas pelo NIST (*National Institute of Standards and Technology*) (BARKER; JOHNSON; SMID, 2007)[10].

Ao receber de B o valor W', A pode utilizar o critério da Definição 10 para verificar que W' é, de fato, um múltiplo de G. Nota-se também que, até esse momento, todas as mensagens foram trocadas em um canal inseguro: qualquer adversário pode ter acesso aos parâmetros de domínio, bem como a W e W' (o primeiro é enviado por A para B e o segundo por B para A). Porém apenas A e B conseguem computar facilmente o valor $P = sW' = s'W = ss'G$, assumindo que o problema ECDLP seja, de fato, computacionalmente difícil. Por fim, aplica-se alguma função sobre o valor x_P para gerar uma chave K de tamanho apropriado, a qual será utilizada em funções como o AES_K.

[10]Essas curvas foram escolhidas, em teoria, por possuírem características adequadas para o ECDLP e por permitirem implementações eficientes das operações de grupo. Existem, no entanto, algoritmos para gerar parâmetros seguros caso o uso de curvas estabelecidas por organizações seja um inconveniente.

3 TRABALHOS CORRELATOS

Os trabalhos que investigam a viabilidade e a eficiência energética de algoritmos de ECC podem ser divididos em dois grandes conjuntos: os voltados a processadores com cache e os voltados a processadores sem cache.

3.1 FOCO EM PROCESSADORES COM CACHE

As primeiras publicações que investigaram o desempenho de sistemas de criptografia baseados em curvas elípticas possuiam um grande enfoque em tempo de execução. Em (HANKERSON; HERNANDEZ; MENEZES, 2000), realizou-se um levantamento extensivo de diversas implementações para as operações aritméticas em $GF(2^m)$, assim como algoritmos para multiplicação de pontos por escalar e o tempo de execução em um processador de um *desktop* da época foi reportado. (AYDOS; YANIK; KOC, 2001) apresentaram uma biblioteca para um processador ARM de 32 bits com operações para curvas em $GF(p)$, reportando tempo de execução para o ECDSA (*Elliptic Curve Digital Signature Algorithm*), o qual utiliza as mesmas primitivas que o ECDH.

O trabalho de (BARTOLINI et al., 2004), a partir do simulador Superscalar para a arquitetura ARM, foi o primeiro a reportar dados do comportamento do sistema de memória para algoritmos de ECC, embora para uma única configuração de cache. Além disso, encontram-se dados sobre CPI (ciclos por intrução) e tempo de execução por procedimento do algoritmo, bem como a proposta de uma instrução para multiplicação de polinomiais e o impacto que isso teria no algoritmo. Tal proposta é interessante visto que, anos depois, o conjunto de instruções ARMv8 trouxe uma instrução similar, embora nenhuma técnica que a utilize tenha sido encontrada.

Em uma continuação desse trabalho, (BRANOVIC; GIORGI; MARTINELLI, 2003) realizaram uma análise da distribuição dos tipos de instrução executadas pelo algoritmo, onde pode-se observar que instruções de ALU totalizam 60% das instruções executadas, como ilustra a Figura 5. Esse resultado foi confirmado pelos trabalhos de (GOUVÊA; LOPÉZ, 2009) e (GURA et al., 2004). Outra contribuição do trabalho é apresentada pela Figura 6, onde compara-se a taxa de faltas nas caches de dados e instruções de algoritmos simétricos e assimétricos. Para os algoritmos de ECC, pode-se observar que a diferença entre a taxa de

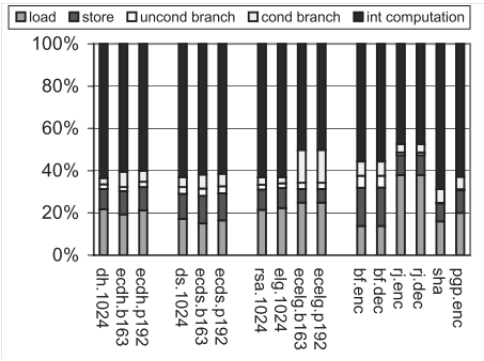

Figura 5: Distribuição dos tipos de instruções executadas durante algoritmos de ECC (segunda e terceira colunas) (BRANOVIC; GIORGI; MARTINELLI, 2003)

faltas na cache de dados e na de instruções chega a uma ordem de magnitude. Além disso, os algoritmos de ECC possuem uma taxa de faltas na cache de dados muito inferior quando comparados ao AES (Figura 6 (b)), embora detalhes da implementação deste não sejam reportados

Como mostra a Figura 5, a proporção das instruções que corresponde a desvios é bastante baixa (menos de 10%). Ora, isso significa que o tamanho médio dos blocos básicos é de 10 instruções (sucessivas), o que resulta em um grande potencial de captura de localidade espacial pelo mero uso de busca sob demanda (*on-demand fetching*), mecanismos clássico embutido na maioria dos controladores de cache. Portanto, técnicas que reorganizem o código para aumentar sua localidade espacial parecem ter pouco potencial de impacto mesmo para caches de instruções com mais de, digamos, 8 palavras por bloco. Como, para caches de dados, a Figura 6 mostra uma taxa de faltas que é uma ordem de magnitude inferior à da cache de instruções, técnicas de otimização baseadas no aumento da localidade espacial de dados seriam inócuas. Como a melhor captura de localidade temporal requereria o uso de caches com maior associatividade, a redução do consumo em memória principal (devido à redução da taxa de faltas) provocaria um aumento do consumo em cache (devido à maior associatividade). Este raciocínio mostra que o impacto energético em cache ao executar algoritmos baseados em curvas elípticas merece uma investigação experimental para uma ampla faixa de parâmetros compatível com a classe de aplicações alvo (dispositivos pessoais móveis). Ademais, o desafio de economizar

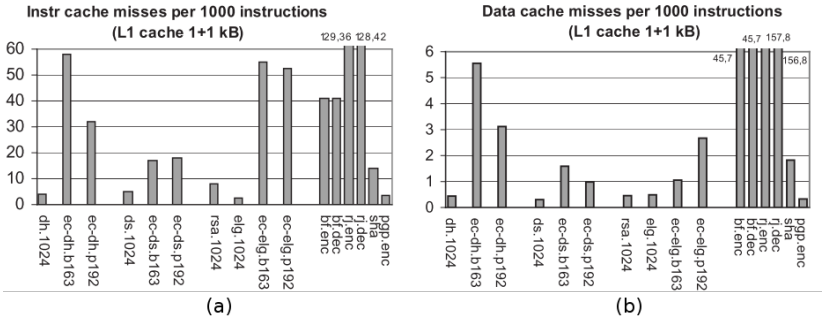

Figura 6: Comparação entre os acertos em cache dos algoritmos de ECC (colunas ec-dh) e AES (colunas rj) (BRANOVIC; GIORGI; MARTINELLI, 2003)

energia em algoritmos baseados em curvas elípticas parece residir em técnicas de otimização que busquem reduzir a taxa de faltas sem buscar aumentar a localidade espacial para um dado grau de associatividade. Uma técnica com esse potencial é a pré-carga de instruções por software (*software prefetching*) (MUCHNICK, 1997), a qual já foi utilizada para aumento de eficiência energética (WUERGES; OLIVEIRA; SANTOS, 2013).

3.2 FOCO EM PROCESSADORES SEM CACHE

Outro conjunto de trabalhos passa a focar em processadores utilizados em sensores, como (GURA et al., 2004), o qual comparou o tempo de execução, uso de memória e tamanho do código para os algoritmos RSA e ECC em um ATmega128. As conclusões apresentadas em tal trabalho indicam que os algoritmos baseados em RSA podem ser até uma ordem de magnitude mais lentos, embora espere-se que essa diferença seja menor para processadores com tamanho de palavra maior. Uma implementação detalhada para plataforma de sensores MICA, a qual utiliza o processador ATmega128, é descrita em (MARY et al., 2007), embora não sejam informados dados de eficiência enérgica. Um trabalho por (GOUVÊA; LOPÉZ, 2009) compara esquemas de ECC com esquemas baseados em identidades e pareamentos, onde conclui-se que os primeiros possuem um desempenho superior. Uma segunda contribuição do trabalho consiste em demonstrar a importância do uso apropriado de

otimizações voltadas a arquitetura alvo, visto que os autores foram os primeiros a considerar a instrução de *multiply and accumulate* do processador MSP430 ao desenvolverem sua implementação. Nessa mesma plataforma, (MANE; SCHAUMONT, 2013) reporta o consumo energético ao variar-se a frequência de operação bem como a presença de multiplicador em hardware, atributo não presente em todas as versões do MSP430.

Quatro anos após a prova em silício de que algoritmos de ECC podem ser utilizados em RFID (HEIN; WOLKERSTORFER; FELBER, 2009), apresentou-se uma implementação em software para a plataforma RFID conhecida como WISP (PENDL; PELNAR; HUTTER, 2012). Neste trabalho, apenas a menor curva padronizada pelo NIST é utilizada, reportando tempo de execução de 1.6 segundos para multiplicação de um ponto da curva por um escalar.

O trabalho de (CLERCQ et al., 2014) avalia o uso de curvas elípticas com o recém lançado processador Cortex M0+ da ARM (32 bits), propondo um novo algoritmo para multiplicação em $GF(q)$. Além disso, o custo enérgico de cada instrução executada pelo processador é avaliado e o desempenho dos algoritmos criptográficos calculado com base em tais valores. Infelizmente, o trabalho se concentra em curvas especiais e não naquelas padronizadas. De modo similar, (LIU et al., 2015) explora duas formas especiais de curvas elípticas (*Montgomery* e *Twisted Edwards*) bem como a equivalência existente entre elas para diminuir o custo das operações de grupo. A importância do uso de curvas padronizadas é discutida em (WENGER; UNTERLUGGAUER; WERNER, 2013), onde os autores desenvolvem clones VHDL (com precisão de ciclos) dos processadores ATmega128, MSP430 e Cortex M0+ e simulam o uso de tais curvas. Destaca-se que o processador ARM alcança a melhor eficiência energética e tempo de execução enquanto o MSP430 ocupa a menor área em silício e demanda a menor potência. Por fim, deve-se mencionar que (LIU et al., 2015) representa o estado da arte em termos de velocidade. Embora não utilize curvas padronizadas, aquele trabalho afirma adotar um corpo finito mais compatível com outras implementações de ECC.

3.3 DISCUSSÃO

A julgar pela literatura, há duas possibilidades abertas para um trabalho sobre o uso energeticamente eficiente de memória em algoritmos baseados em curvas elípticas: 1)Criptografia em dispositivos pesso-

ais móveis (que utilizam processadores com caches, e.g. ARMv7/ARMv8); 2) Criptografia em redes de sensores (que utilizam microcontroladores sem caches, e.g. ATMega). A segunda linha de investigação tem foco na codificação eficiente dos algoritmos e no uso de *scratchpads*. O desafio está na viabilidade das curvas elípticas quando se usam pequenos processadores, sobretudo em face dos requisitos da *Internet of Things*. Por outro lado, a primeira linha de investigação abre a possibilidade de explorar técnicas de compilação para otimização do uso energeticamente eficiente do subsistema de memória. O desafio reside em se reduzir o consumo em memória para algoritmos que possuem uma taxa de faltas já bastante pequena quando comparada à observada para os algoritmos simétricos. Este trabalho aborda a primeira opção, em função do interesse em técnicas de compilação para otimização do uso de memória e também porque é para a primeira opção que se dispõe, no momento, de infraestrutura experimental adequada.

Ao se fazer a opção pela primeira linha de investigação, os resultados da Figura 6 sugerem que este trabalho tenha seu foco na otimização da cache de instruções, devido ao seu maior impacto potencial na redução do consumo energético em memória principal.

4 AVALIAÇÃO EXPERIMENTAL

Para compor a infraestrutura experimental, escolheu-se uma implementação das primitivas de curvas elípticas, um caso de uso de ECC que invoca essas primitivas, uma plataforma alvo, um simulador para tal plataforma e uma ferramenta capaz de fornecer dados energéticos relacionados ao sistema de memória. Este capítulo detalha e justifica cada uma de tais escolhas. Em seguida, apresentam-se os resultados experimentais obtidos e faz-se uma análise do comportamento do sistema de memória e seu impacto na eficiência energética.

4.1 APLICAÇÃO E CLASSES DE PROCESSADORES

4.1.1 Primitivas criptográficas e caso de uso

Por disponibilizar código aberto e ser utilizada tanto pela academia quanto pela indústria, a biblioteca *OpenSSL*[1] foi adotada neste trabalho para fornecer a implementação das operações de curvas elípticas e de *big numbers*. Por ser uma biblioteca extensa, apenas o código necessário à simulação foi compilado (sem suporte a *threads*). Utilizou-se apenas código C, evitando-se otimizações escritas em *assembly*. O *cross-compiler* adotado foi o GCC versão 4.9 configurado pela ferramenta *Buildroot*.

Para analisar a eficiência energética de algoritmos de curvas elípticas, o método baseado em Diffie-Hellman foi adotado como caso de uso. O programa desenvolvido executa os passos descritos na Seção 2.3 para ambas as partes, sem simular a troca de mensagens entre elas[2].

[1] Preferiu-se, quando possível, utilizar uma derivação do OpenSSL chamada BoringSSl, a qual simplificou consideravelmente a biblioteca e atualmente é mantida pela Google, sendo utilizado, por exemplo, no navegador *Chrome*.

[2] Como explicado na Seção 2.3, essa simplicação efetivamente dobra o número de operações executadas. Ora, a principal rotina do ECDH consiste na multiplicação de um ponto da curva por uma constante. Uma execução típica do algoritmo é dada por duas multiplicações desse tipo, isto é, a mesma rotina é executada duas vezes em sequência. A segunda execução poderia se beneficiar mais da localidade temporal imposta pela primeira. No experimento realizado, a multiplicação é executada quatro vezes em sequência, sendo que apenas a terceira se beneficia indevidamente da localidade temporal induzida pelas anteriores. Isso poderia fazer com que os resultados deste trabalho sejam otimistas em relação à taxa de faltas na cache de instruções. Porém, muitas subrotinas possuem blocos básicos com tamanho em torno de 400 instruções, o que faz com que a cache seja frequentemente reescrita.

Os parâmetros iniciais adotados, isto é, as constantes necessárias, são baseados em curvas padronizadas e amplamente utilizadas. Mais especificamente, as curvas P224, P256, P384 e P521 (NIST, 2013) foram simuladas, sendo que o número referenciado no nome da curva indica o nível de segurança da mesma (i.e. o número de bits necessários para representar o primo associado).

4.1.2 Classe de processadores

Para selecionar a classe de processadores a ser usada nos experimentos, este trabalho restringe-se a processadores ARM, devido ao seu uso bastante difundido em Computação Móvel e Computação Embarcada. Eles estão divididos em 3 principais classes: Cortex-A, Cortex-R e Cortex-M. Os sistemas de memória diferem consideravelmente de uma classe para outra, como brevemente discutido a seguir.

Os processadores Cortex-A são empregados normalmente em aplicações de usuários. Admitem conjuntos de instruções de 32 (ARMv7) e 64 bits (ARMv8). Normalmente utilizam uma hierarquia de três níveis de memória: caches L1 distintas para instruções e dados, cache L2 unificada e uma memória principal de alguns GBs. O principal processador de um dispositivo pessoal móvel (e.g. *smartphones* e *tablets*) pertence a esta classe. Também a ela pertencem os processadores de servidores recentemente lançados (AMD, 2015).

A classe Cortex-R é orientada a aplicações de tempo real. Nem sempre possuem unidades de ponto flutuante e a memória se organiza em uma hierarquia de dois níveis: caches L1 e uma memória principal entre 256kB e 512kB. Aplicações típicas incluem freios automotivos, modems, controladores para WiFi, 3G, etc.

Por fim, a classe Cortex-M consiste de microcontroladores com baixo consumo de energia para serem usados, por exemplo, em sensores inteligentes. Seu sistema de memória é bem mais simples: normalmente não possuem caches e sua capacidade de memória é muito inferior (em torno de 128kB).

Neste trabalho, as configurações de memória foram escolhidas de modo a serem similares àquelas presentes em processadores do tipo Cortex-R. A justificava para esta escolha ampara-se em quatro argumentos principais: 1) processadores Cortex-R já são utilizados no processamento em banda-base de celulares, 2) uma das aplicações alvo que

---

Assim, o impacto das multiplicações adicionais não parece levar a uma estimativa excessivamente otimista da taxa de faltas.

lhes podem ser atribuídas é justamente segurança computacional, 3) há uma carência de resultados de desempenho e eficiência energética para a classe Cortex-R, 4) tais processadores têm o potencial de oferecer uma solução de compromisso entre o desempenho de um Cortex-A e o consumo energético de um Cortex-M.

4.2 MODELAGEM DO CONSUMO ENERGÉTICO

Para obter-se dados quantitativos sobre o número de acessos a cada nível de memória, bem como as taxas de acerto em cada um deles, utilizou-se o framework de simulação GEM5 (BINKERT et al., 2011). Com suporte a diversos conjuntos de instruções e diferentes organizações de memória, o GEM5 faz parte de uma iniciativa conjunta entre academia e indústria para o desenvolvimento de uma ferramenta capaz de simular a execução de um programa binário para uma arquitetura desejada. Com base nisso, o caso de uso foi compilado e executado através do simulador e, para cada nível de memória, foram coletados dados referentes ao número de leituras (n_L), de escritas (n_W), de acertos e de faltas. Além disso, foram coletados o número de ciclos e/ou o tempo total de execução do programa (t).

Para construir uma caracterização energética dos algoritmos de ECC, precisa-se obter o consumo energético de cada leitura (c_L), e de cada escrita (c_W), bem como o consumo estático de potência (P). Para tal, utilizou-se a ferramenta CACTI (MURALIMANOHAR; BALASUBRAMONIAN; JOUPPI, 2009). Deste modo, o consumo energético C_M de um componente M do sistema de memória pode ser modelado por:

$$C_M = n_L * c_L + n_W * c_W + t * P$$

4.3 AVALIAÇÃO EXPERIMENTAL

4.3.1 Configuração experimental da memória

Escolheram-se parâmetros de memória compatíveis com implementações de processadores Cortex-R disponíveis no mercado. Como resultado, adotou-se um sistema de memória de dois níveis: uma memória principal com 512kB e duas caches L1 (uma para instruções e uma para dados) com tamanho variando entre {1kB, 2kB, 4kB, 8kB, 16kB} e associatividade *2-way*.

A ferramenta CACTI foi configurada para utilizar o modelo UCA (*Uniform Cache Access*), a tecnologia itrs-lstp (*Low Standby Power*), um tamanho de bloco de 32 bytes para as duas menores caches e 64 bytes nas demais. Todos os resultados são reportados para 32nm, exceto onde explicitamente especificado em contrário. O simulador GEM5 foi utilizado com o modelo "arm-detailed" e modelo de memória "simple".

4.3.2 Avaliação do desempenho em cache

A análise da taxa de faltas nas caches de dados e de instruções oferece um panorama do comportamento do sistema de memória durante a execução dos algoritmos de ECC.

A Figura 7 representa a taxa de faltas na cache de dados para diferentes configurações e curvas. Note que, com uma cache de 4kB, a taxa de faltas é inferior a 1% para qualquer curva. Observe também que o tamanho da curva não afeta significativamente a taxa de faltas para caches de 4kB ou de maior capacidade. Isso deve-se ao fato de que a diferença entre o espaço necessário para armazenar pontos em P224 e P521 é muito pequena quando comparada, por exemplo, ao espaço requerido pelo algoritmo RSA para suprir os mesmos níveis de segurança. Isso ilustra uma das vantagens de ECC discutidas no Capítulo 1.

A Figura 8 reporta a taxa de faltas na cache de instruções. Note que as duas maiores curvas (P521 e P384) possuem a menor taxa de faltas para todos os tamanhos de cache. Isso sugere que o aumento do parâmetro de segurança resulta em um maior número de iterações de um trecho do código cuja maior localidade temporal induz uma menor taxa de faltas.

Para investigar melhor esse comportamento, um experimento adicional foi realizado com a ferramenta Valgrind[3]. A Tabela 2 lista as duas rotinas com mais acessos na cache de instruções para as curvas P256 e P521 e caches de 1kB. Note que *bn_mul_add_words* exibe, de uma curva para a outra, um aumento de 10x no número de referências e de apenas 3x no número faltas. Ora, para que a taxa de faltas permanecesse a mesma de P256 para P521, seria necessário que o número de faltas crescesse na mesma proporção que o número de acessos, o que não acontece nesta rotina. Isso demonstra a existência de uma localidade temporal a qual poderia ser melhor explorada em P256.

[3]Valgrind é uma ferramenta de *profiling* a qual, dentre outras funcionalidades, executa um programa dado e identifica as rotinas com maior número de acessos a caches de dados e instruções.

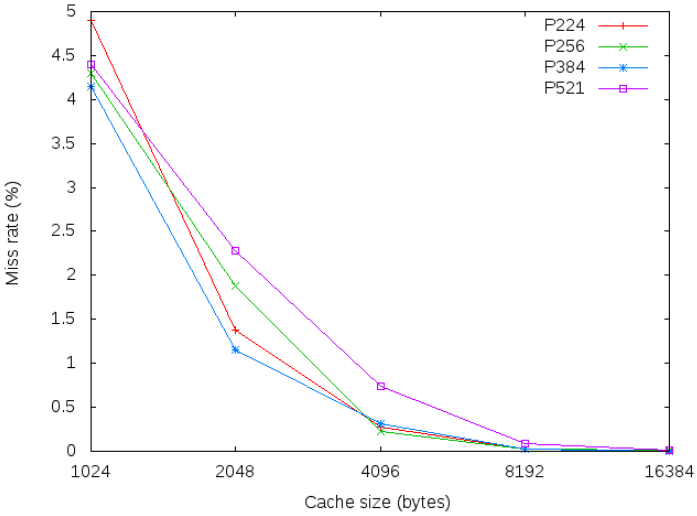

Figura 7: Taxa de faltas na cache de dados

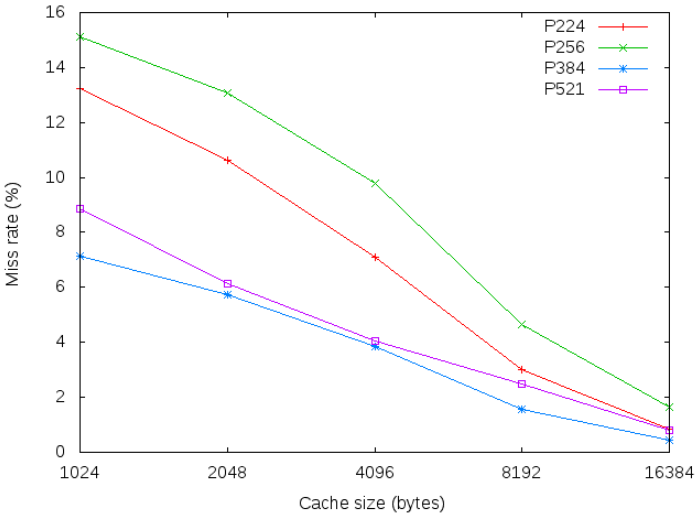

Figura 8: Taxa de faltas na cache de instruções

Este experimento também identificou uma rotina (*bn_mul_comba8*) a qual sofre um aumento de 10x tanto em referências quanto em faltas. Considerando que seu código é um grande bloco básico (em torno de 400 instruções), ela exibe uma alta localidade espacial, capturada pelo mecanismo de *on-demand-fetching*. Logo, pode-se concluir que a maior parcela das faltas nesta rotina são faltas compulsórias, isto é, faltas devido a primeira referência a um endereço de memória.

Tabela 2: Número de acessos e faltas para a cache de instruções para as duas rotinas com maior número de instruções executadas.

Curva	Rotina	Número de acessos	Número de faltas
P256	*bn_mul_add_words*	7×10^6	74×10^5
P521	*bn_mul_add_words*	75×10^6	240×10^5
P256	*bn_mul_comba8*	3.5×10^6	378×10^5
P521	*bn_mul_comba8*	41×10^6	4.4×10^6

4.3.3 Avaliação do consumo em memória

A Figura 9 reporta o consumo energético do sistema de memória para diferentes curvas e configurações de cache. Note que o aumento do tamanho da curva causa um aumento de 4x no consumo energético (entre P224 e P521) em caches de 1kB ou de 6x em caches 16k. Para um dado tamanho de cache, esse aumento pode ser explicado pelo crescimento não linear no número de acessos à memória em relação ao tamanho da curva, como ilustra a Figura 10.

Para cada uma das curvas, nota-se a existência de um tamanho de cache acima do qual o consumo energético cresce. Para explicar esse fato, a Figura 11 mostra, para P224, os seguintes componentes energéticos: energia total na cache de dados, energia total na cache de instruções, energia da memória principal devido aos acessos a instruções, energia da memória principal devido a dados e energia estática em memória principal. Observe que, com o aumento do tamanho da cache, diminui o consumo em memória principal, mas aumenta o consumo em cache. Portanto, existe um tamanho ótimo para a cache (neste caso 4kB) além do qual o consumo total passa a crescer. Em memória principal, o consumo é dominado pelo acesso a instruções, um forte indício de que, em todos os tamanhos adotados, a localidade dos dados é capturada. Ou seja, otimizações voltadas à cache de dados dificilmente terão impacto relevante. Por outro lado, a localidade de instruções só é

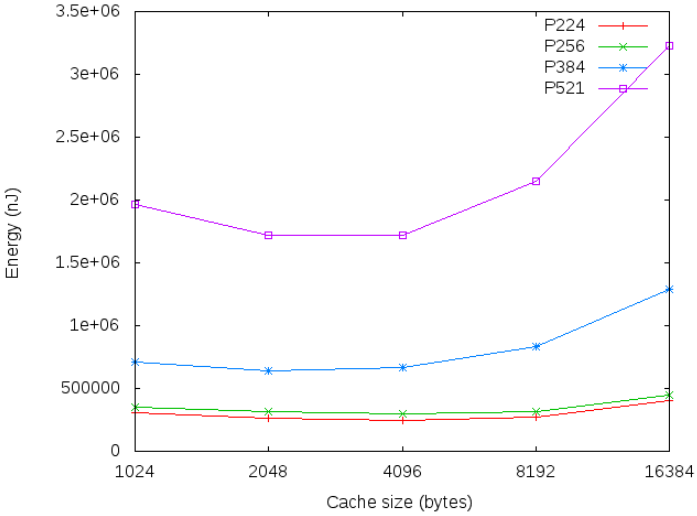

Figura 9: Energia total consumida pelo sistema de memória

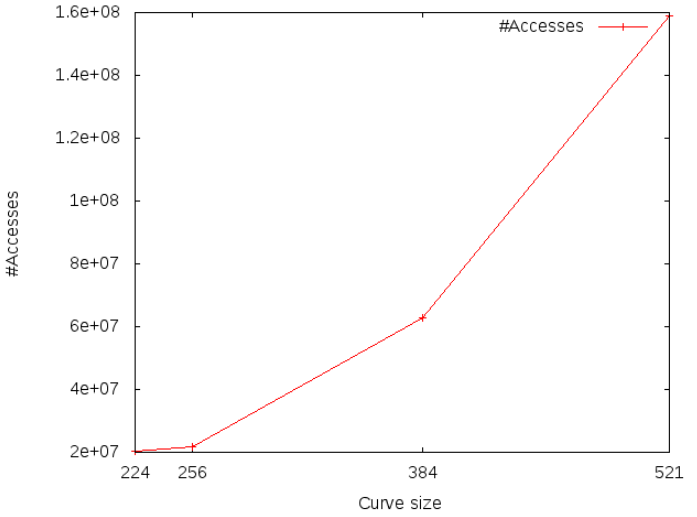

Figura 10: Número de acessos à memória por curva

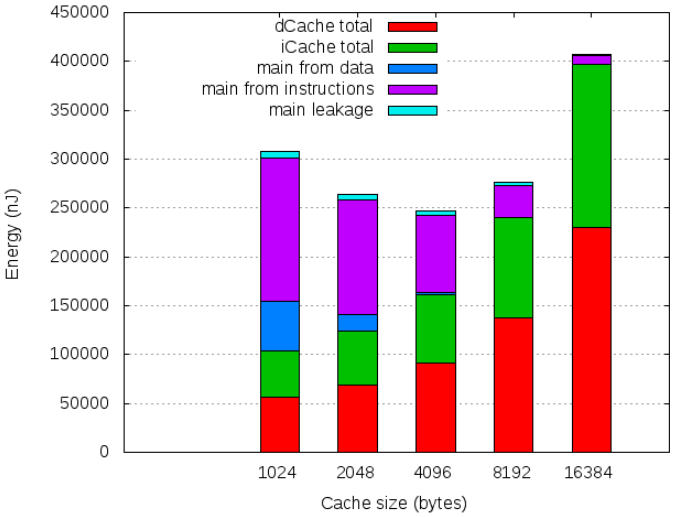

Figura 11: Componentes energéticos para a curva P224

totalmente capturada pelas maiores caches, cujo elevado consumo por acesso faz com que não sejam o tamanho ótimo mencionado anteriormente.

A Figura 12 reporta o impacto que diferentes tecnologias de cache têm no consumo energético. Em todas as curvas existe uma diferença de uma ordem de magnitude entre 32nm e 90nm, ou seja, a escolha adequada desse parâmetro representa uma das maiores economias de energia que se pode alcançar. O maior crescimento de consumo em P521 é explicado pela combinação de: 1) o número de acessos em memória é 10x maior em P521 em relação a P224, 2) o custo de cada acesso e de potência estática aumentam conforme o nó tecnológico aumenta.

4.3.4 Avaliação da eficiência energética em memória

A Figura 13 exibe a eficiência energética de cada uma das combinações de curva e tamanho de cache expressa em número de acessos à memória por nJ. Observe que, para P521, o aumento da cache causa (de 1kB para 2kB) um aumento da eficiência. Aumentos ainda maiores,

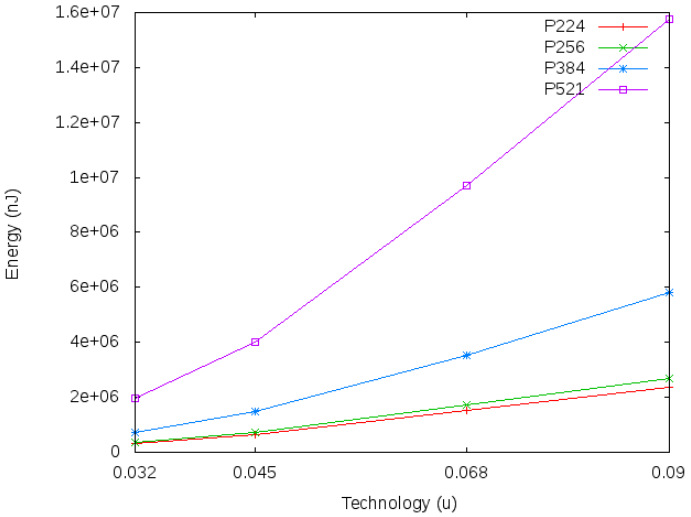

Figura 12: Energia total em diferentes tecnologias e curvas

porém, diminuem a eficiência devido aos alto custos de cada acesso em cache. Note novamente que as maiores curvas operam no maior número de operações por nJ, muito embora seu consumo total seja muito acima das demais. O gráfico obtido traça um paralelo direto com aquele exibido na Figura 9: o pico de eficiência acontece nas caches de 2kB (P224, P256) ou 4kB (P384, P521)[4]. Para esses tamanhos de cache, lembre que: 1) a Figura 8 mostra que a taxa de faltas é superior a 10%, 2) a Figura 11 mostra que 30% do consumo energético é devido a acessos à memória principal para busca de instruções. Isso é um indício de que existe uma oportunidade para otimização exatamente na configuração que possui maior eficiência energética.

[4]Essa afirmação considera que o número de acessos à memória permanece inalterado ao variar-se o tamanho da cache, o que não é verdade. Porém é uma aproximação razoável: os dados levantados mostram que existe uma diferença de 6% entre o número de acessos na maior e na menor cache.

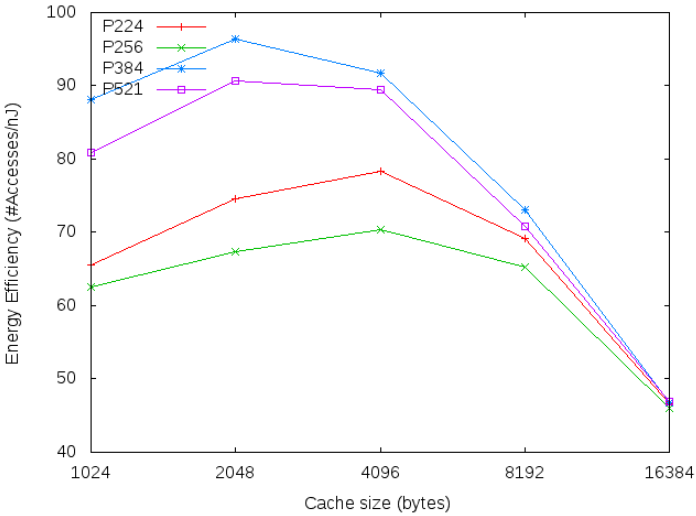

Figura 13: Número de acessos em memória por nJ

5 CONCLUSÕES E PERSPECTIVAS

5.1 OPORTUNIDADES DE OTIMIZAÇÃO

O problema de como viabilizar criptografia de curvas elípticas em hardware com recursos bastante limitados já foi amplamente estudando na literatura. Porém, a maioria dos trabalhos concentra-se em otimizações para atingir um desempenho compatível com os requisitos de uma dada classe de aplicações. Os principais aspectos explorados por esses trabalhos incluem otimizações matemáticas e escolhas de método de representação dos elementos de grupo, da curva e do algoritmo. Este trabalho insere-se em uma linha de investigação complementar: otimizar o código da aplicação criptográfica para reduzir o consumo energético sem degradar o desempenho, isto é, otimizar a eficiência energética da computação criptográfica.

Este trabalho propôs uma avaliação energética preliminar como ponto de partida para identificar otimizações com potencial de aumentar a eficiência energética da computação criptográfica. Ele pressupõe que o subsistema de memória é um dos principais consumidores de energia em Computação Embarcada e Computação Móvel e que, portanto, otimizar a eficiência energética no acesso ao subsistema de memória tem impacto significativo na eficiência energética do sistema como um todo.

Dado um protocolo criptográfico, uma representação de elementos de grupo, uma curva elíptica e um algoritmo que a implementa, este trabalho avaliou o consumo energético no subsistema de memória. Para fixar os parâmetros da implementação utilizou-se uma biblioteca criptográfica conhecida e genérica. Para estabelecer os parâmetros do subsistema de memória, foram escolhidos valores compatíveis com a família ARM Cortex-R. A análise dos resultados experimentais permitiu a identificação de oportunidades de otimização de eficiência energética. Dois tipos principais de experimentos auxiliaram na identificação de oportunidades de otimização. No primeiro, mediram-se as taxas de faltas nas caches para diferentes configurações; no segundo, mediram-se os diferentes componentes do consumo energético.

O primeiro tipo de experimento permitiu identificar oportunidades de redução de consumo de energia na memória principal com melhoria de desempenho. A taxa de faltas no acesso a instruções revelou-se bem maior do que a taxa de faltas no acesso a dados (para a curva P521, por exemplo, a primeira é de 2 a 4 vezes maior do que segunda para

caches entre 1KB e 4KB). Isso significa que, para uma configuração fixa de cache, é mais impactante otimizar a localidade do acesso a instruções do que otimizar a localidade do acesso a dados. Como a localidade espacial do acesso a instruções é bastante alta em curvas elípticas (devido a blocos básicos com centenas de instruções), para reduzir significativamente o consumo em memória principal com significativa melhoria de desempenho, deveriam ser investigadas otimizações que aumentem a localidade temporal no acesso a instruções. O fato de que as maiores curvas exibiram as menores taxas de faltas no acesso a instruções é um indício complementar dessa oportunidade de otimização: parecem haver trechos de código frequentemente referenciados para as curvas menores, mas a frequência de invocação não é suficientemente alta para permitir a exploração da localidade temporal via *on-demand fetching* em caches pequenas.

O segundo tipo de experimento permitiu identificar uma otimização que pode reduzir o consumo em cache mesmo que não aumente significativamente o desempenho, desde que viabilize o uso de caches menores, como explicado a seguir. Como era de se esperar a partir das taxas de faltas (Seção 4.3.2), o principal consumidor em memória principal é o acesso a instruções (Seção 4.3.3). Por outro lado, o principal consumidor em cache é o acesso a dados, pois o número de acessos a dados é ligeiramente superior ao número de acessos a instruções (e o número de acessos à cache não depende da taxa de faltas, como é o caso do acesso à memória principal). Como a taxa de faltas a dados é inferior à de instruções, o uso de caches de mesmo tamanho para dados e instruções não é eficientemente energético, pois aumenta o consumo dos dados em cache, sem reduzir significativamente o consumo no acesso a dados em memória principal. Ora, uma otimização como *software prefetching*(MUCHNICK, 1997) pode viabillizar o uso de uma I-cache menor do que a seria requerida para sustentar uma determinada taxa de faltas apenas com o uso de *on-demand fetching* (WUERGES; OLIVEIRA; SANTOS, 2013). Assim, um sistema energeticamente eficiente onde a D-cache e a I-cache têm a mesma capacidade poderia utilizar *on-demand fetching* na D-cache, mas deveria usar *prefetching* na I-cache. Ou seja, o simples uso de *instruction prefetching* parece ser suficiente para prover eficiência energética a aplicações criptográficas baseadas em curvas elípticas, sem a necessidade de *data prefetching*.

5.2 TRABALHOS FUTUROS

As conclusões deste trabalho estão obviamente limitadas aos resultados obtidos para o caso de uso utilizado. Sua validade para outros protocolos criptográficos baseados em curvas elípticas precisaria ser investigada experimentalmente, repetindo os experimentos e a análise para um maior número de casos de uso representativos dessa classe de aplicações. A ampliação da base experimental será objeto de trabalho futuro.

Assumindo que, à luz de um conjunto mais amplo de experimentos, as conclusões deste trabalho continuem válidas, um tópico de pesquisa promissor seria a avaliação experimental do impacto de otimizações de código em computação criptográfica, restringindo-se a otimizações que melhorem a localidade temporal no acesso a instruções e realizem a pré-carga antecipada de instruções através da inserção de instruções de *prefetch* no código (isto é, *software prefetching*). Inicialmente, seria pragmático avaliar o impacto de otimizações disponíveis em compiladores-padrão (como o gcc). Se tais otimizações não se revelarem satisfatórias, pode ser interessante adaptar ou desenvolver novas otimizações que possam resultar na maior eficiência energética da computação criptográfica.

REFERÊNCIAS

AMD. **AMD Opteron A1100 Processor**. out. 2015. Disponível em: <www.amd.com/en-us/products/server/opteron-a-series/a1100>.

AYDOS, M.; YANIK, T.; KOC, C. High-speed implementation of an ecc-based wireless authentication protocol on an arm microprocessor. **IEE Proceedings-Communications**, IET, v. 148, n. 5, p. 273–279, 2001.

BARKER, E. B.; JOHNSON, D.; SMID, M. E. **SP 800-56A. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)**. Gaithersburg, MD, United States, 2007.

BARTOLINI, S. et al. A performance evaluation of arm isa extension for elliptic curve cryptography over binary finite fields. In: **Computer Architecture and High Performance Computing, 2004. SBAC-PAD 2004. 16th Symposium on**. [S.l.: s.n.], 2004. p. 238–245. ISSN 1550-6533.

BINKERT, N. et al. The gem5 simulator. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, v. 39, n. 2, p. 1–7, ago. 2011. Disponível em: <http://doi.acm.org/10.1145/2024716.2024718>.

BLAKE-WILSON, S. et al. **Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS), RFC4492**. 2006.

BRANOVIC, I.; GIORGI, R.; MARTINELLI, E. A workload characterization of elliptic curve cryptography methods in embedded environments. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, v. 32, n. 3, p. 27–34, set. 2003. ISSN 0163-5964. Disponível em: <http://doi.acm.org/10.1145/1024295.1024299>.

CLERCQ, R. de et al. Ultra low-power implementation of ecc on the arm cortex-m0+. In: **Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE**. [S.l.: s.n.], 2014. p. 1–6.

DAEMEN, J.; RIJMEN, V. Aes proposal: Rijndael. 1999.

DALLY, W. J. et al. Efficient embedded computing. **Computer**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 41, n. 7, p. 27–32, jul. 2008. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/MC.2008.224>.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. **IEEE Trans. Inf. Theor.**, IEEE Press, Piscataway, NJ, USA, v. 22, n. 6, p. 644–654, set. 1976. ISSN 0018-9448. Disponível em: <http://dx.doi.org/10.1109/TIT.1976.1055638>.

FAN, J. et al. Low-energy encryption for medical devices: Security adds an extra design dimension. In: ACM. **Proceedings of the 50th Annual Design Automation Conference**. [S.l.], 2013. p. 15.

GOLDREICH, O. **Foundations of Cryptography: Volume 1**. New York, NY, USA: Cambridge University Press, 2006. ISBN 0521035368.

GOUVÊA, C. P. L.; LOPÉZ, J. Software implementation of pairing-based cryptography on sensor networks using the msp430 microcontroller. In: ROY, B.; SENDRIER, N. (Ed.). **Progress in Cryptology - INDOCRYPT 2009**. Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5922). p. 248–262. ISBN 978-3-642-10627-9. Disponível em: <http://dx.doi.org/10.1007/978-3-642-10628-6_17>.

GURA, N. et al. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In: JOYE, M.; QUISQUATER, J.-J. (Ed.). **Cryptographic Hardware and Embedded Systems - CHES 2004**. Springer Berlin Heidelberg, 2004, (Lecture Notes in Computer Science, v. 3156). p. 119–132. ISBN 978-3-540-22666-6. Disponível em: <http://dx.doi.org/10.1007/978-3-540-28632-5_9>.

HANKERSON, D.; HERNANDEZ, J. L.; MENEZES, A. Software implementation of elliptic curve cryptography over binary fields. In: KOÃ§, Ã.; PAAR, C. (Ed.). **Cryptographic Hardware and Embedded Systems - CHES 2000**. Springer Berlin Heidelberg, 2000, (Lecture Notes in Computer Science, v. 1965). p. 1–24. ISBN 978-3-540-41455-1. Disponível em: <http://dx.doi.org/10.1007/3-540-44499-8_1>.

HEIN, D.; WOLKERSTORFER, J.; FELBER, N. Selected areas in cryptography. In: AVANZI, R. M.; KELIHER, L.; SICA, F. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. ECC Is Ready for

RFID — A Proof in Silicon, p. 401–413. ISBN 978-3-642-04158-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-04159-4_26>.

IEEE. **IEEE Standard Specifications for Public-Key Cryptography**. [S.l.], Aug 2000. 1-228 p.

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

LENSTRA, A.; VERHEUL, E. Selecting cryptographic key sizes. In: IMAI, H.; ZHENG, Y. (Ed.). **Public Key Cryptography**. Springer Berlin Heidelberg, 2000, (Lecture Notes in Computer Science, v. 1751). p. 446–465. ISBN 978-3-540-66967-8. Disponível em: <http://dx.doi.org/10.1007/978-3-540-46588-1_30>.

LIU, Z. et al. Efficient implementation of ecdh key exchange for msp430-based wireless sensor networks. In: **Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security**. New York, NY, USA: ACM, 2015. (ASIA CCS '15), p. 145–153. ISBN 978-1-4503-3245-3. Disponível em: <http://doi.acm.org/10.1145/2714576.2714608>.

MANE, D.; SCHAUMONT, P. Energy-architecture tuning for ecc-based rfid tags. In: HUTTER, M.; SCHMIDT, J.-M. (Ed.). **Radio Frequency Identification**. Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, v. 8262). p. 147–160. ISBN 978-3-642-41331-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-41332-2_10>.

MARY, W. et al. Wm-ecc: an elliptic curve cryptography suite on sensor motes. In: CITESEER. **In**. [S.l.], 2007.

MILLER, V. Use of elliptic curves in cryptography. In: WILLIAMS, H. (Ed.). **Advances in Cryptology - CRYPTO '85 Proceedings**. Springer Berlin Heidelberg, 1986, (Lecture Notes in Computer Science, v. 218). p. 417–426. ISBN 978-3-540-16463-0. Disponível em: <http://dx.doi.org/10.1007/3-540-39799-X_31>.

MUCHNICK, S. S. **Advanced Compiler Design Implementation**. [S.l.]: Morgan Kaufmann Publishers, 1997. ISBN 9781558603202.

MURALIMANOHAR, N.; BALASUBRAMONIAN, R.; JOUPPI, N. P. Cacti 6.0: A tool to model large caches. **HP Laboratories**, p. 22–31, 2009.

NECHVATAL, J. et al. Report on the development of the advanced encryption standard (aes). 2000.

NIST. **NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition**. 2012. Acesso em 17-06-2015. Disponível em: <http://www.nist.gov/itl/csd/sha-100212.cfm>.

NIST. **FIPS PUB 186-4 Digital Signature Standard (DSS)**. 2013.

PENDL, C.; PELNAR, M.; HUTTER, M. Elliptic curve cryptography on the wisp uhf rfid tag. In: JUELS, A.; PAAR, C. (Ed.). **RFID. Security and Privacy**. Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7055). p. 32–47. ISBN 978-3-642-25285-3. Disponível em: <http://dx.doi.org/10.1007/978-3-642-25286-0_3>.

RACKOFF, C. **Fundamentals of Cryptography**. 2014. University Lecture. Disponível em: <http://www.cs.toronto.edu/~rackoff/2426f14/>.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. **Commun. ACM**, ACM, New York, NY, USA, v. 21, n. 2, p. 120–126, fev. 1978. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/359340.359342>.

TESKE, E. Speeding up pollard's rho method for computing discrete logarithms. In: **Proceedings of the Third International Symposium on Algorithmic Number Theory**. London, UK, UK: Springer-Verlag, 1998. (ANTS-III), p. 541–554. ISBN 3-540-64657-4. Disponível em: <http://dl.acm.org/citation.cfm?id=648184.749879>.

TILBORG, H. C. A. van; JAJODIA, S. (Ed.). **Encyclopedia of Cryptography and Security, 2nd Ed**. Springer, 2011. ISBN 978-1-4419-5905-8. Disponível em: <http://dx.doi.org/10.1007/978-1-4419-5906-5>.

WENGER, E.; UNTERLUGGAUER, T.; WERNER, M. 8/16/32 shades of elliptic curve cryptography on embedded processors. In: **Proceedings of the 14th International Conference on Progress in Cryptology — INDOCRYPT 2013 - Volume 8250**. New York, NY, USA: Springer-Verlag New York, Inc., 2013. p. 244–261. ISBN 978-3-319-03514-7. Disponível em: <http://dx.doi.org/10.1007/978-3-319-03515-4_16>.

WIENER, M.; ZUCCHERATO, R. Faster attacks on elliptic curve cryptosystems. In: TAVARES, S.; MEIJER, H. (Ed.). **Selected Areas in Cryptography**. Springer Berlin Heidelberg, 1999, (Lecture Notes in Computer Science, v. 1556). p. 190–200. ISBN 978-3-540-65894-8. Disponível em: <http://dx.doi.org/10.1007/3-540-48892-8_15>.

WUERGES, E.; OLIVEIRA, R. S. de; SANTOS, L. C. V. dos. Reconciling real-time guarantees and energy efficiency through unlocked-cache prefetching. In: **Proceedings of the 50th Annual Design Automation Conference**. New York, NY, USA: ACM, 2013. (DAC '13), p. 146:1–146:9. ISBN 978-1-4503-2071-9. Disponível em: <http://doi.acm.org/10.1145/2463209.2488915>.