Felipe de Azevedo Piovezan

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

Tese 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 Tese 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

Banca Examinadora:

Prof. Dr. Daniel Santana de Freitas
Coorientador

Prof. Dr. Luiz Cláudio Villar dos Santos

Prof. Dr. José Luís Almada Güntzel

Prof. Dr. Daniel Santana de Freitas

AGRADECIMENTOS

TBC

“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 limitados.

Durante o desenvolvimento dos algoritmos assimétricos, os principais nodos da rede eram grandes servidores e desktops. Recentemente, porém, a escala dos dispositivos que estabelecem comunicação segura pela internet está diminuindo consistentemente: dos laptops e celulares, até sensores e etiquetas RFID, todos necessitam de um canal seguro. Essa miniaturização, no entanto, veio acompanhada de restrições de processamento e de suprimento energético, o que conflita com o fato que algoritmos de PKC são computacionalmente caros: como, então, utilizá-los em ambientes com recursos limitados? 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, bem como o impacto que técnicas de otimização têm na eficiência energética.

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 in 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 environment.

During the development of the first asymmetric algorithms, most nodes in a network were either desktops or servers. Recently, however, the scale of the devices partaking in secure communations over the internet has been steadily decreasing: from laptops and cellphones to sensors and RFID tags, they all require a secure channel. 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. What then is the best way to employ those algorithms in such constrained devices? By means of simulations, this work will evaluate the amount of energy spent in the memory system when processors execute elliptic curve algorithms, as well as the impact of certain code optimization techniques on energy efficiency.

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. 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*. Alguns problemas em comum podem ser encontrados nesses dispositivos, como 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 a última deficiência entra diretamente em conflito com as primeiras.

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 (como (RESNER; FRÖHLICH, 2015)), 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, 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, 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. De acordo com (LENSTRA; VERHEUL, 2000), a equivalência entre os tamanhos de chaves para as duas técnicas pode ser resumida na Tabela 1. Pode-se notar que a diferença chega próxima de uma ordem de magnitude em muitos casos e, de fato, este é um dos argumentos utilizados pela RFC que adiciona algoritmos de ECC ao protocolo TLS (BLAKE-WILSON et al., 2006).

ECC	RSA
163	1024
233	2048
283	3072
409	7680

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

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 algum 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...,).

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

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), parâmetros do algoritmo (como as curvas utilizadas e o tamanho de chaves) e aplicando-se técnicas de otimização voltadas à eficiência energética (como, por exemplo, pré-carga de instruções ou similar). Assim, será possível relacionar a influência de tais parâmetros no consumo de energia pelo sistema de memória.

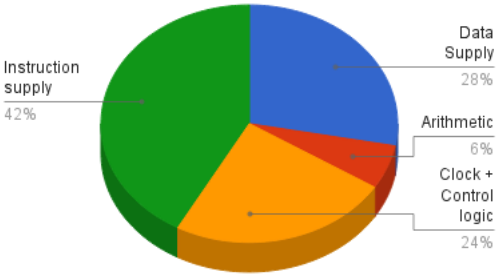

Figura 1: Consumo de energia em um processador embarcado (DALLY et al., 2008)

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. Na versão completa desta monografia, serão incluídos dois capítulos adicionais, um deles contendo os experimentos realizados, a metodologia adotada e resultados obtidos, o outro concluindo o traba-

lho e levantando 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

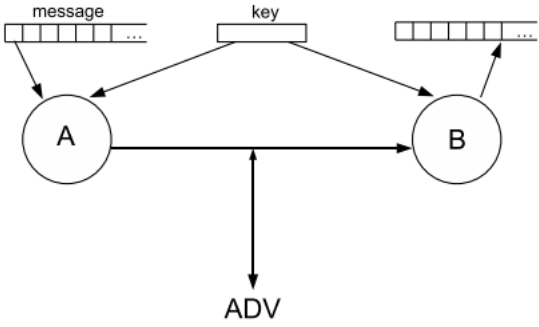

Figura 2: Modelo de uma sessão segura (RACKOFF, 2014)

A aplicação mais tradicional de criptografia, aquela descrita por um usuário comum de informática, encontra-se no estabelecimento de **sessões seguras** (*secure sessions*). 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**, *plaintext*) 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 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****:*

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

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

[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 $n > m$, 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. 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 manipulando 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 um elemento 0 e 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. *0 é o elemento neutro de $+$*
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 (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. Está 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 algum 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.

Pode-se também criar uma curva elíptica sobre $GF(2^m)$, cuja definição é dada a seguir, embora os demais conceitos desse capítulo sejam tratados apenas para o caso de $GF(p)$ para evitar demasiadas repetições. É importante lembrar, porém, que a escolha do corpo tem implicações tanto no modo como seus elementos são representados em memória como na definição de operações e algoritmos.

Definição 8 *Uma curva elíptica E sobre o corpo finito $GF(2^m)$, é o conjunto:*

$$E = \{(x, y) \quad | \quad y^2 + xy = x^3 + ax^2 + b \\ \wedge \quad x, y, a, b \in GF(2^m) \quad \wedge \quad b \neq 0\} \cup \{O\}$$

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 9 *O número de elementos de uma curva elíptica E é chamado de **ordem** de E e é denotado por $\#E(GF(q))$.*

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 $GF(13)$ e, para comparação, a mesma curva é desenhada sobre os reais na figura 4. Como se pode observar, a troca de domínio muda consideravelmente a aparência da curva.

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 SOMA, conforme (IEEE..., 2000).

As linhas 1– 4 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 9– 10 calculam o coeficiente angular da reta secante que passa por P_1 e P_2, enquanto calcula-se o coeficiente da reta tangente a curva em um dos pontos quando $P_1 = P_2$ (linha 14). Caso

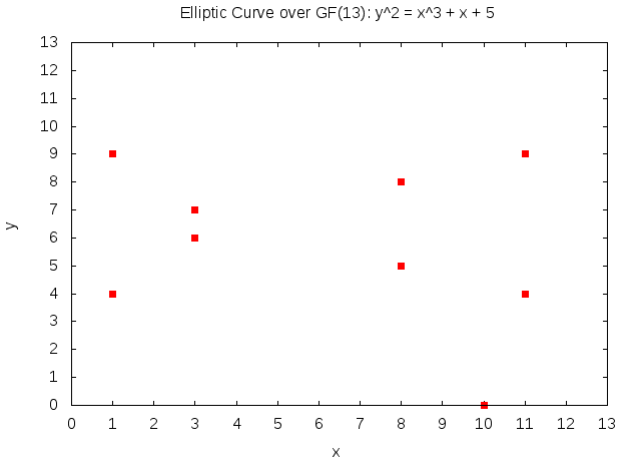

Figura 3: Conjunto de pontos que representa uma curva elíptica sobre $GF(13)$

os pontos difiram apenas na coordenada y, então temos $P_1 = -P_2$ e o algoritmo retorna o elemento neutro (linhas 12– 13). Por fim, as linhas 15– 16 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 SOMA 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.

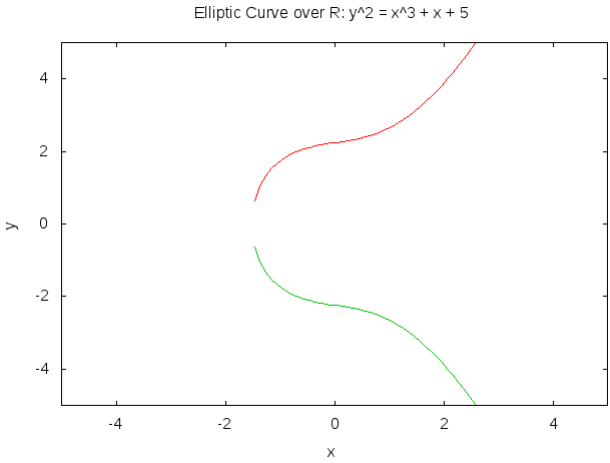

Figura 4: Conjunto de pontos que representa uma curva elíptica sobre $\mathbb{R}$

SOMA(P_1, P_2)

if $P_1 = O$
    output P_2 and stop.
if $P_2 = O$
    output P_1 and stop.
Set $x_1 = P_1.x$
Set $y_1 = P_1.y$
Set $x_2 = P_2.x$
Set $y_2 = P_2.y$
if $x_1 \neq x_2$
    Set $\lambda = (y_1 - y_2)/(x_1 - x_2)$
else
    if $y_1 \neq y_2$ or $y_2 = 0$
        Output O and stop.
    Set $\lambda = (3{x_2}^2 + a)/(2y_2)$
Set $x_3 = \lambda^2 - x_1 - x_2 \mod p$
Set $y_3 = (x_2 - x_3)\lambda - y_2 \mod p$
output (x_3, y_3)

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 5 ilustra *informalmente* a adição dos pontos $P_1 = (10, 0)$ e $P_2 = (8, 5)$. Primeiro, a reta que sai de P_1 e 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.

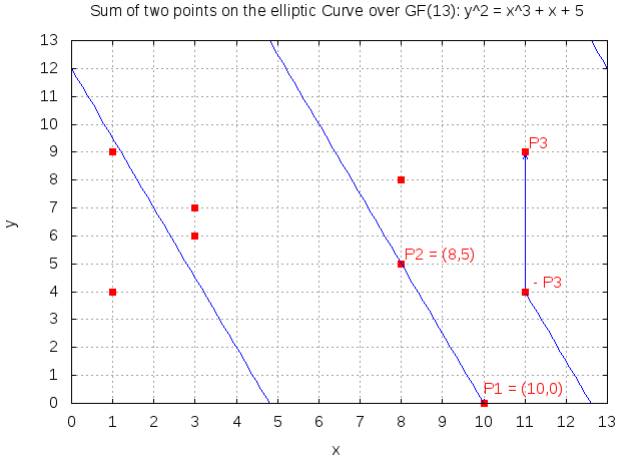

Figura 5: Somando pontos em uma curva elíptica sobre $GF(13)$

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 10 *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, $k \equiv l \mod p$.

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. Enquanto no logaritmo tradicional essa pergunta está sempre bem definida (e possui resposta) para valores positivos de b e x, 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 o envolvendo os conceitos de ordem de ponto as quais permitem resposta afirmativa.

Definição 11 *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 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 12 *Considere uma curva elíptica E e um ponto $P \in E$ de ordem n. Dado um ponto $Q \in \langle P \rangle = \{0, P, 2P, \ldots (n-1)P\}$, o* ***problema do logaritmo discreto sobre curvas elípticas*** *(ECDLP) consiste em computar $0 \leq l \leq n-1$ tal que $Q = lP$.*

O algoritmo ingênuo para resolver uma instância de ECDLP consiste em computar os múltiplos de P (isto é, os membros de $\langle P \rangle$) até que se encontre Q, 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, November 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.

As etapas, embora simples, merecem diversas considerações adicionais. Os parâmetros de domínio não podem ser gerados de forma

[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] Mais recentemente, as pesquisas em algoritmos baseados em Retículos (*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.

totalmente aleatória: diversas curvas anômalas precisam ser evitadas 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 11 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, difícil. Por fim, aplica-se alguma função sobre o valor x_P para gerar uma string 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 relacionados a investigar 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. Tais conjuntos são os temas dos capítulos 3.1 e 3.2.

3.1 TRABALHOS VOLTADOS A PROCESSADORES COM CACHE

As primeiras publicações as quais investigaram a performance 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 por escalar em curvas, e o tempo de execução em um processador de 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 Digistal Signature Algorithm*), o qual utiliza as mesmas primitivas que 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 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 não se encontrou nenhuma técnica utilizando ela na prática.

Em uma continuação desse trabalho, (BRANOVIC; GIORGI; MARTINELLI, 2003) realizaran uma análise da distribuição dos tipos de instrução executadas pelo algoritmo, onde pode-se observar que instruções de ALU totalizam 60% do total de executadas (Figura 6), resultado confirmado pelos trabalhos de (GOUVÊA; LOPÉZ, 2009) e (GURA et al., 2004). Outra contribuição consiste na comparação do *miss rate* entre algoritmos simétricos e assimétricos, onde pode-se observar que os algoritmos de ECC possuem uma taxa de faltas na cache de dados muito

inferior quando comparados ao AES (figura 7 à direita), embora detalhes da implementação deste não sejam reportados. Além disso, a diferença entre o *miss rate* na cache de dados e na de instruções (para os algoritmos de ECC) chega a uma ordem de magnitude, conforme a figura 7.

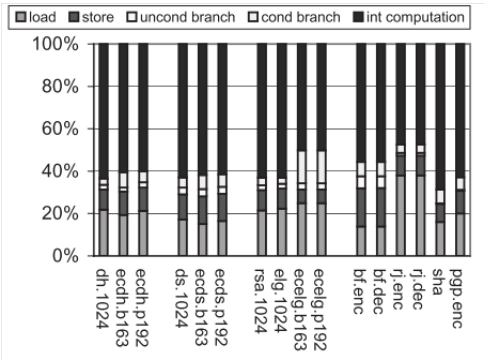

Figura 6: Distribuição dos tipos de instruções executadas durante algoritmos de ECC (segunda e terceira colunas) (BRANOVIC; GIORGI; MARTINELLI, 2003)

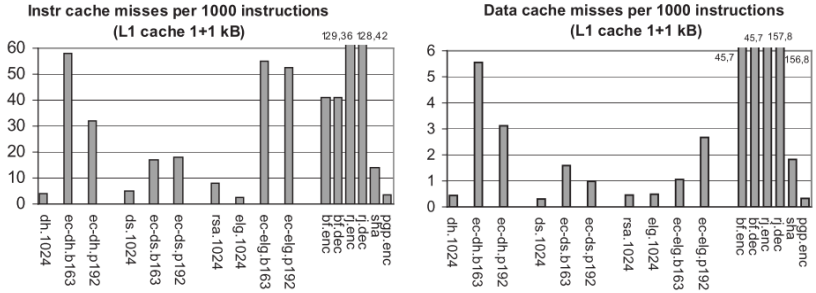

Figura 7: Comparação entre os acertos em cache dos algoritmos de ECC (colunas ec-dh) e AES (colunas rj) (BRANOVIC; GIORGI; MARTINELLI, 2003)

Como mostra a Figura 6, 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 7 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 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. Um exemplo de técnica com esse potencial é a pré-carga de instruções por software (software prefetching) (WUERGES; OLIVEIRA; SANTOS, 2013).

3.2 TRABALHOS VOLTADOS A 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 operada bem como a presença de multiplicador em hardware, atributo o qual não está 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 (precisos em ciclo) 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 o estado da arte em termos de velocidade encontra-se no trabalho publicado por (LIU et al., 2015), o qual, embora não utilize curvas padronizadas, afirma trabalhar sobre 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 opção 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 segunda opção abre a possibilidade de investigar 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 adota a primeira opção, em função do interesse em técnicas de compilação para otimização do uso de memória e também porque é para a segunda opção que se dispõe, no momento, de infraestrutura experimental adequada.

Ao se fazer esta opção, os resultados da Figura 7 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 DE CURVAS ELÍPTICAS

A infraestrutura experimental desenvolvida consiste na escolha de: uma implementação das primitivas de curvas elípticas, um protocolo baseado em ECC para ser executado, uma plataforma alvo, um simulador para tal plataforma e uma ferramenta capaz de fornecer dados energéticos relacionados ao sistema de memórias. Este capítulo é iniciado com o detalhamento e justificava de cada uma de tais escolhidas e, em seguida, apresenta-se os resultados experimentais obtidos bem como uma análise do comportamento do sistema de memórias e o seu impacto na eficiência energética dos algoritmos.

4.1 MODELAGEM E ESTIMATIVA ENERGÉTICA

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 *Instruction Set Architectures* (ISA) e sistemas 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 compilado para uma arquitetura desejada. Com base nisso, o programa teste foi executado através do simulador e, para cada nível de memória, foram coletados dados referentes ao número de leituras (n_L), de escritas (n_W), acertos e falhas. Além disso, foram coletadas informações relativas ao número de ciclos executados, ou o tempo total de execução do programa (t).

De modo a construir uma caracterização energética dos algoritmos de ECC, são necessárias dados sobre o custo de cada uma das operações em memória, i.e., o custo de um leitura (c_L), uma escrita (c_W) e o gasto estático (P). Para tal, utilizou-se a ferramenta CACTI (MURALIMANOHAR; BALASUBRAMONIAN; JOUPPI, 2009). Deste modo, o custo energético C_M de um componente M do sistema de memórias pode ser modelado por:

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

4.2 DOMÍNIO DE APLICAÇÃO, APLICAÇÃO ESPECÍFICA E CLASSES DE PROCESSADORES

4.2.1 Algoritmos criptográficos

Por ser de código aberto, utilizada tanto por indústria quanto academia e considerada segura, 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* e 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 (conforme discutido no capítulo 2.3) foi adotado. O programa desenvolvido executa os passos descritos em tal capítulo para ambas as partes, sem simular a troca de mensagens entre elas, efetivamente dobrando o número de operações executadas. Os parâmetros iniciais adotados, isto é, a curva e 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).

4.2.2 Classe de processadores

Por estarem presentes em grande parte do mercado atual de sistemas embarcados, os processadores ARM foram adotados neste trabalho. Eles estão divididos em 3 grandes grupos: Cortex-A, Cortex-R e Cortex-M. Cada um possui diferenças consideráveis em seus respectivos sistemas de memória, fato o qual justifica uma breve discussão sobre os grupos neste capítulo.

Os processadores do tipo Cortex-A são empregados normalmente em aplicações de usuários ou aplicações que demandam alta performance. São capazes de operar tanto em 32 quanto 64 bits, e normal-

[1] Preferiu-se, quando possível, utilizar um *fork* do OpenSSL chamado BoringSSl, o qual simplificou consideravelmente a biblioteca e atualmente é mantido pela Google, sendo utilizado, por exemplo, no navegador *Chrome*

mente acompanhados de 3 níveis de memória: uma cache L1, uma L2 e memória principal de alguns GBs. O principal processador de dispositivos como tablets, smartphones e, mais recentemente, servidores é desse grupo.

O grupo dos Cortex-R está focado em aplicações de tempo real, e estão um pouco abaixo dos Cortex-A em termos de perfomance. Nem sempre possuem unidades de ponto flutuante e suas caches são consideravelmente mais simples: caches L1 e uma memória principal entre 256kB e 512kB. Aplicações típicas incluem freios automotivos, modems, controladores para WiFi e 3g, entre outros.

Por fim, a classe Cortex-M é caracterizada pelo baixo consumo de energia e é ideal para microcontroladores e sensores. Seu sistema de memória é bem mais simples que os dois anteriores, normalmente não possuem caches e sua memória tem capacidade 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. Essa escolha é justificada pela presença dos mesmos em diversos dispositivos móveis atuais, como no hardware de banda base de celulares, e por possuírem segurança como uma das aplicações alvo. Além disso, seu sistema de memória oferece um balanço interessante entre a complexidade da categoria Cortex-A e a simplicidade de um Cortex-M. Todas as justificativas citadas estão de acordo com a discussão realizada no capítulo 3.3.

4.3 AVALIAÇÃO EXPERIMENTAL COMPATÍVEL COM CORTEX-R

4.3.1 Configuração experimental da memória

Com base na análise de diversas implementações dos processadores Cortex-R disponíveis no mercado, parâmetros de memória condizentes foram escolhidos. Deste modo, optou-se por 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 *2-way associative*.

A ferramenta CACTI foi configurada para utilizar o modelo UCA (*Uniform Cache Access*), a tecnologia itrs-lstp (*Low Standby Power*), 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. 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 memórias durante a execução dos algoritmos de ECC.

Da figura 8 pode-se concluir que a taxa de faltas é muito baixa para todas as curvas, sendo que, com uma cache de apenas 4kB, esse valor é inferior a 1%. Além disso, o parâmetro de segurança não causa grande impacto na cache dados, o que é esperado: a diferença entre o espaço necessário para armazenar pontos em P224 e P521 é muito pequena. Tal observação ilustra uma das vantagens de ECC sobre o algoritmo RSA, este último trabalha com números muito maiores para um mesmo nivel de segurança, conforme discutido no capítulo 1.

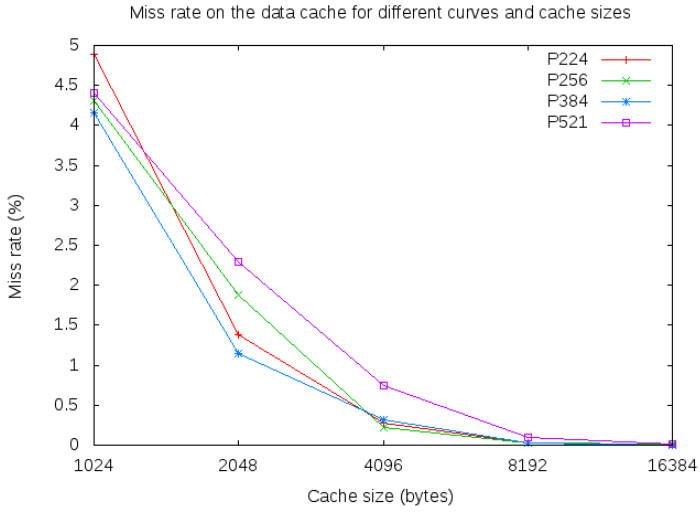

Figura 8: Taxa de falta na cache de dados

A cache de instruções, no entanto, possui um comportamento muito mais interessante e muito mais faltas que a de dados. A figura 9 revela que as duas maiores curvas (P521 e P384) possuem a menor

taxa de faltas em todos os tamanhos de cache, sugerindo, então, que o aumento do parâmetro de segurança implica um maior número de iterações de um ponto do código com um melhor comportamento em cache.

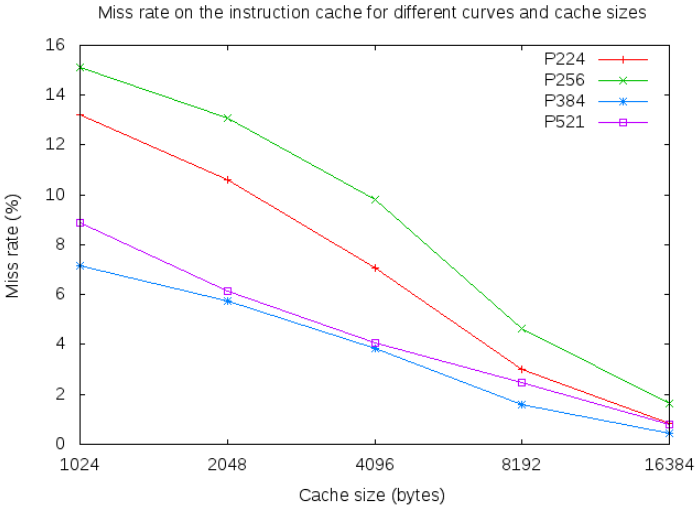

Figura 9: Taxa de falta na cache de instruções

Uma análise com a ferramenta Valgrind parece confirmar essa hipótese. Para a curva P256, a rotina com mais acessos na cache de instruções foi responsável por 7 milhões de leituras das quais 74 mil foram faltas, enquanto em P521 foram 75 milhões de leituras e 240 mil faltas. Ou seja, um aumento de 10x no número de acessos foi acompanhado de um aumento de 3x no número de faltas. Já segunda função com mais referências produziu 3.5 milhões e 378 mil acessos e falhas, respectivamente, em P256 enquanto em P521 foram 41 milhões e 4.4 milhões respectivamente. Ou seja, nessa segunda rotina houve um aumento de 10x tanto em leituras quanto em faltas. Isso parece sugerir que, durante as chamadas à primeira rotina, seu código já se encontrava em cache. Os números parecem apontar que o mesmo não deve ser válido para a segunda, podendo ser um alvo interessante para uma otimização como prefetching.

4.3.3 Avaliação do consumo em memória

A partir da figura 10, pode-se concluir que o parâmetro de segurança tem um impacto muito grande no custo energético total dos algoritmos, mesmo embora as maiores curvas tenham as melhores taxas de faltas na cache de instruções. Esse elevado custo fica mais acentuado nas maiores caches, onde as taxas de faltas de todas as curvas são praticamente idênticas. Observa-se, também, que as menores curvas mantêm uma mesma eficiência energética para todos os tamanhos de cache, o que não é verdade para as duas maiores. Isso pode ser explicado novamente pela figura 9, visto que as menores curvas se beneficiam mais do aumento da cache. Tais observações são interessantes pois mostram que, para P224 e P256, o gasto energético total independe (ou depende muito pouco) do tamanho das caches.

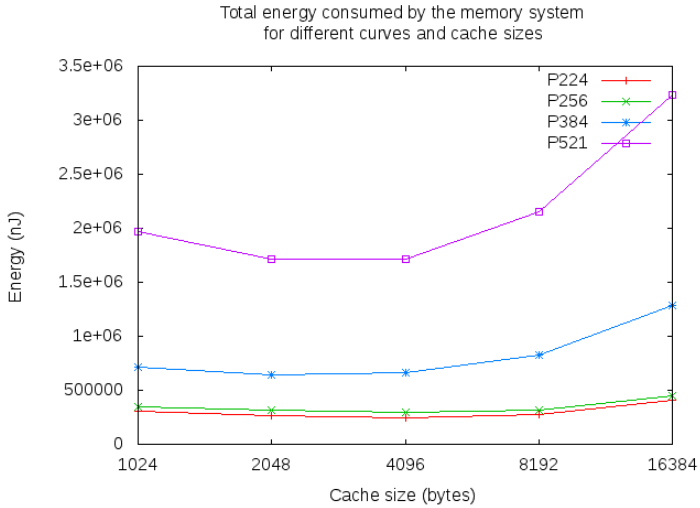

Figura 10: Energia total consumida pelo sistema de memória

Para viabilizar uma análise de quais componentes têm o maior impacto no custo energético, a figura 11 divide, para P224, os componentes de energia entre: energia total na cache de dados, energia total na cache de instruções, energia da memória principal devido acessos à instruções, energia da memória principal devido a dados e energia estática em memória principal. É possível perceber que, embora as

parcelas relativas às duas caches sejam similares, grande parte do uso da memória principal vem de acessos à instruções, novamente um indicativo de que a cache de instruções é um alvo importante para otimizações. Além disso, muito embora tenha-se concluido anteriormente que o custo total dos algoritmos para P224 independa do tamanho da cache, uma eventual otimização teria impacto maior nas menores caches, visto que apenas nelas seria possível melhorar consideravelmente a taxa de faltas. Aliado à esse fato, ganha-se em energia estática ao utilizar-se uma cache menor.

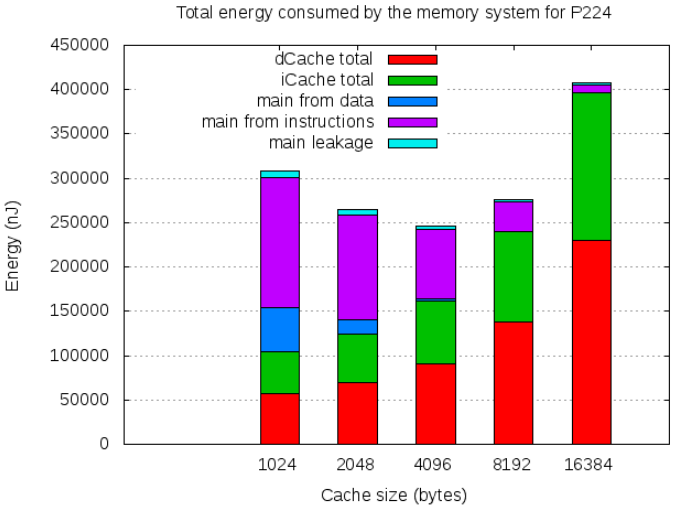

Figura 11: Energia total separada em componentes para a curva P224

O uso de diferentes tecnologias também possui um impacto a ser considerado, conforme exibido na figura 12. Em todas as curvas existe uma diferença de uma ordem de magnitude ao comparar-se as tecnologias de 32nm e 90nm, ou seja, o desenvolvimento de qualquer dispositivo com restrições energética deve começar com a escolha adequada desse parâmetro. Para explicar a maior taxa de crescimento das curvas P521 e P384, é necessário observar que tanto o custo de um acesso quanto o custo estático crescem conforme o nodo tecnológico aumenta. Conforme a figura 13 mostra, o número de acessos em memória não cresce linearmente com o parâmetro de segurança, portanto é esperado que a tecnologia tenha um impacto maior nas maiores curvas.

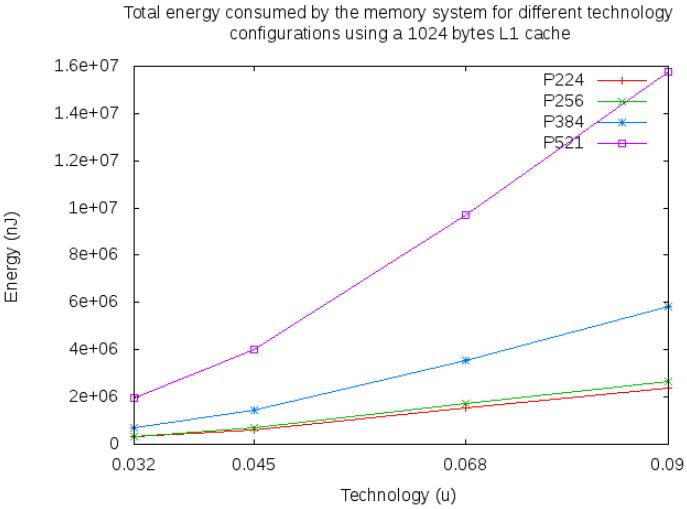

Figura 12: Energia total em diferentes tecnologias e curvas

4.3.4 Avaliação da eficiência energética em memória

A eficiência energética de cada uma das combinações de curva e tamanho de cache é exibida na figura 14, a qual está de acordo com todos os resultados obtidos. Nota-se novamente que as maiores curvas operam no maior número de operações por nJ, muito embora seu custo total seja muito acima das demais. O gráfico obtido traça um paralelo direto com aquele exibido na figura 10: o pico de eficiência acontece nas caches de 2kB ou 4kB. Essa afirmação considera que o número de acessos em memória permanece inalterado ao variar-se o tamanho da cache, o que não é verdade, porém uma aproximação razoável conforme a figura 15 comprova.

4.3.5 Conclusões

O problema de como trabalhar com criptografia de curvas elípticas em hardware com recursos limitados já foi amplamente estudado na literatura. Porém muitos dos esforços encontram-se em otimizações matemáticas, no método de representação dos elementos de grupo, na

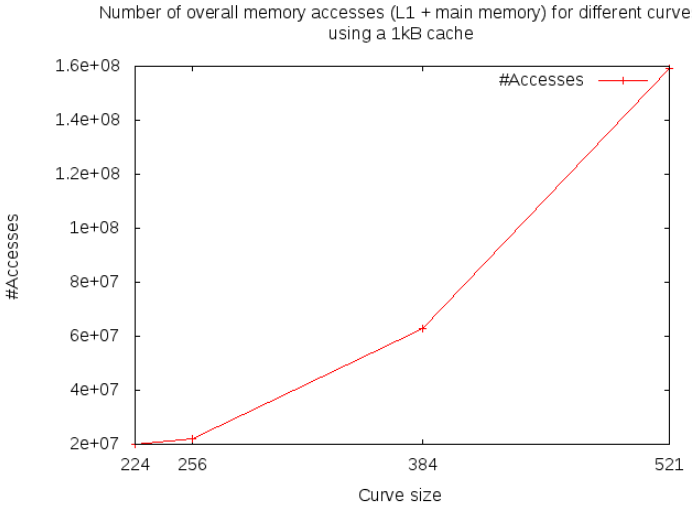

Figura 13: Número de acessos à memória por curva

escolha da curva ou na escolha do melhor algoritmo para cada operação envolvida no ECDH. Com este trabalho foi possível realizar uma abordagem diferente para o problema: fixando-se os parâmetros citados previamente através do uso de uma biblioteca conhecida e genérica, identificar novos pontos passíveis de otimização.

A partir da análise de um parâmetro simples como a taxa de faltas nas caches, foi possível mapear quais aspectos do algoritmo poderiam ser investigados em maior detalhe. De um lado, a cache de dados, muito embora represente uma parcela considerável da energia total, possue um comportamento difícil de ser melhorado visto que possui baixas taxas de falta. No outro, a cache de instruções aparece como a maior fatia do consumo de energia e, ao mesmo tempo, com altas taxas de faltas. Além disso, o fato de que as maiores curvas possuem menores taxas de faltas levam-nos a desconfiar de que, dentre os trechos de código mais executados, alguns podem um comportamento muito melhor que os demais. Ao comparar-se a menor e a maior curva, essa hipótese foi reforçada, sendo possível identificar um trecho responsável por um número maior de faltas que os demais.

Já a observação da energia total gasta em cada uma das curvas permitiu identificar que, para algumas, o custo energético total é quase

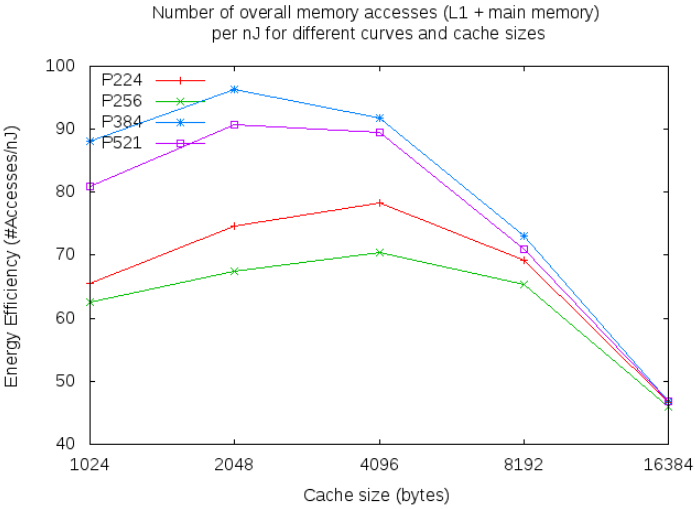

Figura 14: Eficiência energética em número de acessos em memória por nJ

invariante com o tamanho da cache, porém a distribuição das parcelas é muito diferente para cada tamanho. Isso abre espaço para combinar dois possíveis fatores positivos: utilizar menos recursos (tamanho menor de cache) e, ao mesmo tempo, investir em uma técnica que diminua o número de faltas na mesma. Tal técnica pode ser aplicada exatamente nos pontos identificados anteriormente.

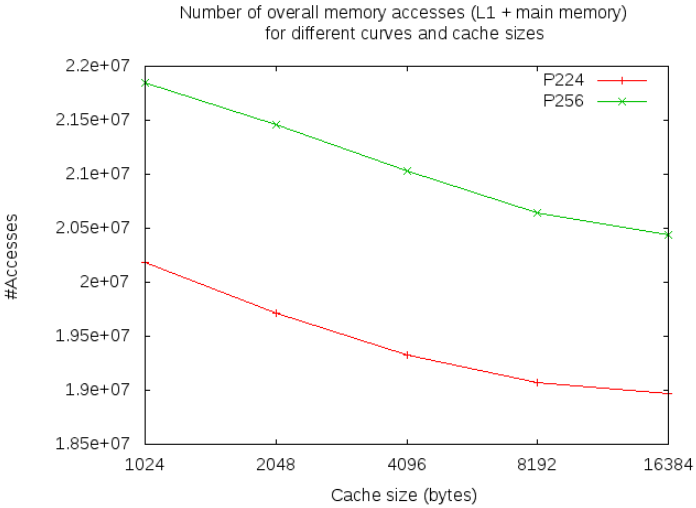

Figura 15: Número de acessos à memória variando-se o tamanho da cache

REFERÊNCIAS

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. November 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 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>.

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

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

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/>.

RESNER, D.; FRÖHLICH, A. A. Key Establishment and Trustful Communication for the Internet of Things. In: **4th International Conference on Sensor Networks (SENSORNETS 2015)**. Angers, France: [s.n.], 2015. p. 197–206. ISBN 978-989-758-086-4. Disponível em: <http://www.lisha.ufsc.br/pub/Resner_SENSORNETS_2015.pdf>.

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>.