Segurança do bit menos significativo no RSA e em curvas elípticas

Dionathan Nakamura[1], **Routo Terada**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
São Paulo, SP – Brazil

{nakamura,rt}@ime.usp.br

Abstract. *The security of the least significant bit of the secret key in the Elliptic Curve Diffie-Hellman (and of the message in the RSA) is related to the security of the whole key (message). In this paper, algorithms are presented to invert these cryptographic systems making use of oracles that predict the LSB. We implement two of them, critical parameters are identified and the original sampling is changed. With the modified sampling we achieve an improvement in the execution times.*

Resumo. *A segurança do bit menos significativo da chave secreta no Diffie-Hellman sobre Curvas Elípticas (e da mensagem no RSA) está relacionada à segurança de toda a chave (mensagem). Neste artigo são apresentados algoritmos que conseguem inverter os criptossistemas citados fazendo uso de oráculos que predizem o LSB. Fazemos a implementação de dois desses algoritmos, identificamos parâmetros críticos e mudamos a amostragem do formato original. Com a modificação na amostragem conseguimos uma melhora nos tempos de execução.*

1. Introdução

Algoritmos criptográficos geralmente se baseiam em algum problema computacional sem solução eficiente conhecida, sendo assim considerado um problema difícil — podemos citar o problema do logaritmo discreto (PLD) e o problema da fatoração de inteiros (PFI). As relações entre esses algoritmos e seus problemas são frequentemente alvos de pesquisas, por exemplo, a verificação de equivalência entre os algoritmos e seus respectivos problemas subjacentes.

Dentro da criptografia há casos onde é necessário o uso de números pseudoaleatórios. Muitos estudos também são feitos entorno desses números e um alvo de pesquisa é verificar se um número gerado possa ser considerado criptograficamente seguro.

Algoritmos criptográficos e seus problemas computacionais subjacentes estão relacionados com geradores de números pseudoaleatórios criptograficamente seguros [Blum and Micali 1984]. Podemos citar, por exemplo, o gerador pseudoaleatório de Blum, Blum e Shub [Blum et al. 1986] que é um gerador pseudoaleatório baseado no algoritmo criptográfico Rabin [Rabin 1979]. Ele faz sucessivas encriptações Rabin e utiliza o bit menos significativo (LSB — *Least Significant Bit*) de cada criptograma para compor o número gerado. Um gerador semelhante para o RSA [Rivest et al. 1978] é descrito em [Menezes et al. 1996, Alg.5.35].

Pesquisas interessantes têm aparecido relacionando a segurança dos algoritmos criptográficos com os bits que compõem as chaves secretas ou as mensagens encriptadas por esses sistemas [Goldwasser et al. 1982, Ben-Or et al. 1983, Boneh and Venkatesan 1996, Alexi et al. 1988, Fischlin and Schnorr 2000, Boneh and Shparlinski 2001, Jetchev and Venkatesan 2008, Chevalier et al. 2009, Hofheinz and Kiltz 2009, Roh and Hahn 2010], em especial, com o LSB.

Esses estudos analisam o algoritmo RSA e o protocolo Diffie-Hellman [Diffie and Hellman 1976] sobre Curvas Elípticas (DHCE) — enquanto no RSA o interesse está no LSB da mensagem original, no DHCE está no LSB da chave secreta combinada. Nos estudos com RSA, o algoritmo Rabin também é abordado por sua similaridade. Então, verificar a segurança do LSB é o mesmo que verificar a segurança dos números pseudoaleatórios obtidos pelos geradores citados anteriormente.

Dos trabalhos relacionados, destacamos para o RSA o artigo de [Alexi et al. 1988] onde mostrou ser possível inverter o RSA com um oráculo com probabilidade de acerto de $(50+\varepsilon)\%$, sendo ε positivo e bem pequeno, todavia não negligenciável. Vamos chamar esse trabalho (e o respectivo algoritmo) de ACGS por brevidade.

Para o DHCE, destacamos o artigo de [Boneh and Shparlinski 2001] onde mostrou que o LSB da abscissa do ponto combinado é imprevisível, ou seja, a existência de um oráculo para o LSB implicaria a quebra do DHCE. Vamos chamar esse trabalho (e o respectivo algoritmo) de BS.

Em [Fischlin and Schnorr 2000], os autores mostraram como esses algoritmos podem ter seu tempo de execução demorado, a ponto de ultrapassar o tempo de execução de soluções para problemas difíceis. Aqui surgiu nossa motivação. Os trabalhos são apresentados apenas como artigos científicos, imaginávamos como eles se comportariam depois de implementados. Apesar de apresentarem convergência teórica, queremos verificar se ela se traduz na prática com o uso dos geradores de aleatoriedade que dispomos. Ainda, se o tempo real de computação é viável com recursos computacionais modestos.

Assim, neste trabalho nós implementamos o ACGS para o RSA e em seguida o BS para DHCE. Como parte secundária dos objetivos, queríamos identificar os parâmetros que mais influenciavam no tempo de execução, procurando encontrar onde erros eram superestimados e promover melhorias no modo como os oráculos eram utilizados.

As principais contribuições deste trabalho são as seguintes:

- provê uma implementação da segurança do LSB no algoritmo RSA;
- provê uma implementação da segurança do LSB no protocolo DHCE;
- identificação dos parâmetros relevantes para o tempo de execução;
- conseguimos uma redução no limitante da função de paridade;
- encontramos uma melhoria significativa no número de acessos ao oráculo;
- das melhorias, o ACGS passou a ser mais rápido que o PFI para valores práticos.

Organização do trabalho Na Seção 2 apresentamos os conceitos necessários para a compreensão da metodologia desenvolvida. Na Seção 3 são descritos os algoritmos dos trabalhos utilizados e as técnicas utilizadas para a implementação desses. Os resultados obtidos são expostos e discutidos na Seção 4. Por fim, na Seção 5, concluímos a pesquisa e propomos novas direções de trabalhos.

2. Conceitos preliminares

Consideramos nesse trabalho curvas elípticas E sobre um corpo finito $\mathbb{F}$ de característica prima $\mathrm{char}(\mathbb{F}) = p$, $p \neq 2$, $p \neq 3$ e sendo p grande. Apresentamos uma curva elíptica $E/\mathbb{F}$ na forma da equação simplificada de Weierstrass $E : y^2 = x^3 + ax + b$, onde $a, b \in \mathbb{F}$, ∞ é o ponto no infinito e o discriminante $\Delta = 4a^3 + 27b^2 \neq 0$.

Sobre a **propriedade multiplicativa** do RSA: considere o criptograma $y \equiv x^e \bmod N$, desejamos obter $z \equiv (cx)^e \bmod N$ para alguma constante c. Note ser possível obter o valor de z sem conhecer o valor da mensagem original x. Um outro fato importante, a divisão por 2 dentro da aritmética modular é simplesmente o produto com o inverso multiplicativo de 2 módulo N. A propriedade multiplicava e os modos de cálculo da divisão por 2 são utilizados no ACGS. Cálculos similares serão feitos em curvas elípticas.

Adotamos notar o módulo N com letra maiúscula e seu tamanho em bits $n = \lfloor \lg N \rfloor + 1$ com letra minúscula. Para facilitar a escrita, seja x um inteiro, denotamos $[x]_N$ como o resto de x módulo N, ou seja, $[x]_N = x \bmod N$.

O que chamamos de oráculo neste trabalho trata-se de um algoritmo que, ao receber como entrada parâmetros públicos de um criptossistema, retorna o LSB da parte secreta desse sistema em tempo polinomial.

Dizemos que um oráculo tem **vantagem** ε se ele acerta o LSB com uma probabilidade igual ou superior a $\frac{1}{2} + \varepsilon$. Por exemplo, para o DHCE em uma curva E, gerador G de ordem q e a, b distribuídos uniformemente em $[1, q-1]$, o algoritmo $\mathcal{A}$ é um oráculo com vantagem ε em predizer o LSB da coordenada x da função Diffie-Hellman e denotamos $\mathrm{Adv}^X_{E,G}(\mathcal{A}) = \left|\Pr_{a,b}[\mathcal{A}(E, G, aG, bG) = \mathrm{LSB}(x)] - \frac{1}{2}]\right| > \varepsilon$.

O máximo divisor comum (MDC, também GCD do inglês *Greatest Common Divisor*) é um algoritmo utilizado aqui para verificar se dois operandos b e c são primos entre si, ou seja, se $\mathrm{MDC}(b, c) = 1$.

Apresentamos uma versão chamada MDC binário baseado em Knuth [Knuth 1981, Sec.4.5.2 Alg.B p.321]. Nela são utilizadas as seguintes propriedades devidas a Stein [Stein 1967]:

- se $|b|$ e $|c|$ são ambos pares, então $\mathrm{MDC}(b, c) = 2 \cdot \mathrm{MDC}(\frac{b}{2}, \frac{c}{2})$;
- se $|b|$ é par e $|c|$ ímpar, então $\mathrm{MDC}(b, c) = \mathrm{MDC}(\frac{b}{2}, c)$;
- se $|b|$ e $|c|$ são ambos ímpares, então $\mathrm{MDC}(b, c) = 2 \cdot \mathrm{MDC}(\frac{b+c}{2}, \frac{b-c}{2})$;
- adicionalmente, no último caso, ou $\frac{b+c}{2}$ ou $\frac{b-c}{2}$ é divisível novamente por 2.

Essas propriedades foram utilizadas para se criar um MDC binário modificado em [Ben-Or et al. 1983] e no Brent Kung GCD [Brent and Kung 1984] utilizado no ACGS.

A Desigualdade de Chebyshev é uma ferramenta estatística que nos permite estabelecer limitantes para uma distribuição de probabilidade que não conhecemos. Sua importância vem do fato de só necessitar o conhecimento da esperança e da variância da distribuição. Ela é utilizada nas provas do ACGS.

Por outro lado, devemos levar em conta que ela pode não definir aproximações muito exatas de probabilidade. Vamos citar dois exemplos desse fato obtidos do livro de Ross [Ross 2006, Example 8.8b]. Em um exemplo para distribuição uniforme, a aproximação foi de 52%, enquanto que a probabilidade real era de 20%. Já em um segundo exemplo, para distribuição normal, a diferença foi de 25% para $4{,}5\%$.

3. Métodos e Implementação

Trabalhamos com dois criptossistemas, o RSA e o DHCE. Nesta seção, vamos apresentar métodos que se utilizam de um oráculo para tentar inverter esses criptossistemas. Ainda, daremos detalhes das implementações de tais métodos.

3.1. Algoritmo para o RSA

Suponha a existência de um oráculo $\mathcal{O}_N$ com vantagem ε em descobrir o LSB do RSA com módulo N. Onde a vantagem $0 < \varepsilon \leq \frac{1}{2}$ é pequena, porém não negligenciável. Vamos mostrar como o ACGS faz uso de $\mathcal{O}_N$ para inverter o RSA no Algoritmo 3.1.

O ACGS computa dois valores aleatórios $[ax]_N$ e $[bx]_N$, assegura que $[ax]_N$ seja ímpar e depois calcula o MDC com ajuda do Brent Kung GCD [Brent and Kung 1984], por brevidade chamaremos esse MDC binário de BKGCD.

Ao chegar na Parte 3 do algoritmo, a variável a contém um valor cujo produto com x é o MDC de $[ax]_N$ e $[bx]_N$. É neste momento que esperamos que o $[ax]_N$ e $[bx]_N$ escolhidos sejam coprimos, pois na verificação da Linha 27 teremos $\mathrm{RSA}(ax) = \pm 1$ e assim conseguimos recuperar x através do cálculo do inverso multiplicativo $[\pm a^{-1}]_N$. Caso não sejam coprimos, o algoritmo volta para Linha 2 e sorteia novamente a e b na esperança que esses valores resultem em $[ax]_N$ e $[bx]_N$ coprimos.

Por um famoso teorema de Dirichlet [Knuth 1981, Sec.4.5.2 Theo.D p.324], a probabilidade de dois inteiros aleatórios no intervalo $[-K, K]$ serem relativamente primos converge para $\frac{6}{\pi^2}$ conforme K tende a ∞. Isso é algo em torno de $60{,}8\%$. Assim, com duas tentativas em a e b já é esperado encontrar coprimos em $[ax]_N$, $[bx]_N$.

Outra condição que faz o algoritmo retornar à Linha 2 se refere à variável $limiteGCD$, que chamaremos limitante do BKGCD. Esse limitante é para o caso quando a função PAR é falha, assim o algoritmo não ficará executando indefinidamente. A função PAR é uma função de paridade, por exemplo, dado $\mathrm{PAR}(b, y)$, ela retorna a paridade de $[bx]_N$. Ela é apresentada no Algoritmo 3.2.

A função PAR faz uma quantidade m de amostras entorno da paridade de dx, denominadas **medidas-dx**. São somados os votos para a resposta, par ou ímpar, nas variáveis $contador_0$ e $contador_1$, respectivamente. O resultado é a maioria dos votos.

Cada voto de paridade é obtido verificando se $\mathrm{LSB}(rx) = \mathrm{LSB}(rx + dx)$. A igualdade se mantém se, e somente se, dx for par, isso porque um número par não altera a paridade de outro número na operação de adição.

O problema aqui é se na adição $rx + dx$ ocorrer uma redução modular (também chamada *overlapping* ou *wraparound*). Como N é ímpar, o voto é invertido. Por isso que desejamos que na escolha de a e b, $[ax]_N$ e $[bx]_N$ sejam "pequenos". Aqui, $[ax]_N$ pequeno quer dizer que $\mathrm{abs}_N(ax) < \frac{\varepsilon}{2}N$, onde:

$$\mathrm{abs}_N(x) \stackrel{def}{=} \begin{cases} [x]_N & \text{se } [x]_N < N/2 \\ N - [x]_N & \text{caso contrário} \end{cases}$$

Sendo assim, teremos $[dx]_N$ pequeno e na adição $rx + dx$ a probabilidade de redução modular ocorrer também será pequena ($\frac{\varepsilon}{2}$). Note que conforme a função BKGCD evolui, os valores $[ax]_N$ e $[bx]_N$ intermediários continuam pequenos.

Algoritmo 3.1: Inversor ACGS para o RSA.

Entrada: $y = \mathrm{RSA}(x)$, módulo N e o expoente de encriptação e.
Saída: a mensagem original x.

início
  `// Parte 1: Aleatorização`
  Escolha de maneira uniforme e independente $a, b \in \mathbb{Z}_N$;
  `// Parte 2: Brent-Kung GCD de` $[ax]_N$ `e` $[bx]_N$
  $\alpha \leftarrow n$;
  $\beta \leftarrow n$;
  limiteGCD $\leftarrow 0$;
  enquanto $\mathrm{PAR}(a, y) = 0$ **faça**
    $a \leftarrow [a/2]_N$;
    $\alpha \leftarrow \alpha - 1$;
    limiteGCD $\leftarrow$ limiteGCD $+ 1$;
    se limiteGCD $> 6n + 3$ **então** `VolteParaLinha(2);`
  repita
    enquanto $\mathrm{PAR}(b, y) = 0$ **faça**
      $b \leftarrow [b/2]_N$;
      $\beta \leftarrow \beta - 1$;
      limiteGCD $\leftarrow$ limiteGCD $+ 1$;
      se limiteGCD $> 6n + 3$ **então** `VolteParaLinha(2);`
    se $\beta \leq \alpha$ **então**
      `Troca(a,b);`
      `Troca(`α`,`β`);`
    se $\mathrm{PAR}(\frac{a+b}{2}, y) = 0$ **então**
      $b \leftarrow [(a + b)/2]_N$;
    senão
      $b \leftarrow [(a - b)/2]_N$;
    limiteGCD $\leftarrow$ limiteGCD $+ 1$;
    se limiteGCD $> 6n + 3$ **então** `VolteParaLinha(2);`
  até $b = 0$;
  `// Parte 3: Inversão`
  se $\mathrm{RSA}(ax) \neq \pm 1$ **então** `VolteParaLinha(2);`
  $c \leftarrow [\pm a^{-1}]_N$;
  retorna c ; `/* c é a mensagem original` x `*/`
fim

Algoritmo 3.2: Função de paridade PAR.

Entrada: $y = \mathrm{RSA}(x)$, o multiplicador d.
Saída: o $\mathrm{LSB}(\mathrm{abs}_N(dx))$.

```
1  início
2      contador_0 ← 0;
3      contador_1 ← 0;
4      para i ← 1 até m faça
5          escolha de maneira aleatória r_i ∈ Z_N;
6          se O_N(RSA(r_i x)) = O_N(RSA(r_i x + dx)) então
7              contador_0 ← contador_0 + 1;
8          senão
9              contador_1 ← contador_1 + 1;
10     se contador_0 > contador_1 então
11             retorna 0;
12     senão retorna 1;
13 fim
```

Com esse esquema de votos, conseguimos construir uma função de paridade quase perfeita, desde que a quantidade de medidas m seja grande o suficiente. Isso é garantido pela lei dos grandes números.

Na Linha 6 do Algoritmo 3.2 são feitas duas consultas ao oráculos. Isso gera um problema conhecido como **duplicação de erro** [Ben-Or et al. 1983]. Para contorná-lo, o ACGS usa a função PAR de modo que o oráculo só é consultado do lado direito da igualdade ($rx + dx$). Quanto ao lado esquerdo (rx), o LSB de rx já é conhecido. Isso é possível dividindo o conjunto dos inteiros módulo N em intervalos e aplicado uma técnica de suposição sobre eles.

Note que das suposições feitas em [Alexi et al. 1988], temos ao todo $2 \cdot 4\varepsilon^{-1} \cdot 2 \cdot 4m\varepsilon^{-1} = 2^6 m\varepsilon^{-2}$ possibilidades. Rodamos o ACGS com cada uma dessas alternativas. Em apenas uma delas estaremos computando corretamente $\mathrm{LSB}(r_i x)$ e chamaremos essa instância de **alternativa correta**. Então, na alternativa correta o ACGS recupera a mensagem original.

Os autores em [Alexi et al. 1988] calcularam utilizando Desigualdade de Chebyshev a probabilidade de erro da função PAR igual a $\Pr(\mathrm{PAR\ errar}) = \frac{4}{m\varepsilon^2}$. Então, é preciso escolher uma amostragem m suficientemente grande para se ter uma função de paridade quase perfeita. Foi escolhida $m \stackrel{def}{=} 64n\varepsilon^{-2}$, assim $\Pr(\mathrm{PAR\ errar}) \leq \frac{1}{16n}$.

Com probabilidade de erro $\frac{1}{16n}$, a função PAR, em média, consegue passar sem erro em todas as requisições da função BKGCD (no máximo $limiteGCD = 6n + 3$).

De acordo com a nossa discussão sobre a Desigualdade de Chebyshev na Seção 2, acreditamos que a probabilidade de erro da função PAR é superestimada aqui. Veremos também, empiricamente, que na média o limitante do BKGCD pode ser menor do que $6n + 3$, pelo menos metade.

O tempo de execução do ACGS corresponde a ε^{-2} tentativas para se obter ax e

bx pequenos, mais duas tentativas para que sejam coprimos $\left(\frac{\pi^2}{6}\right)$, mais a execução das $2^6 m\varepsilon^{-2}$ alternativas, $6n+3$ chamadas à função PAR e m medidas-dx. Escrevendo tudo:

$$\varepsilon^{-2}{\cdot}2{\cdot}2^6 m\varepsilon^{-2}{\cdot}(6n{+}3){\cdot}m \approx 3{\cdot}2^8\varepsilon^{-4}nm^2 \approx 3{\cdot}2^8\varepsilon^{-4}n\left(\frac{64n}{\varepsilon^2}\right)^2 \approx 3{\cdot}2^{20}\varepsilon^{-8}n^3 = O(\varepsilon^{-8}n^3)$$

3.2. Algoritmo para o DHCE

O algoritmo BS faz uma adaptação do algoritmo ACGS para uso no DHCE. Considere um grupo cíclico de curva prima $E(\mathbb{F}_p)$ com um gerador G de ordem prima q e as chaves privadas de Alice e Beto $a, b \in [1, q-1]$, definimos o segredo da função Diffie-Hellman $DH_{E,G}(aG, bG) = abG$ como a coordenada x do ponto abG. Nessa seção, supomos a existência de um oráculo $\mathcal{O}_p$ para $[abG]_x$, tal que $\mathrm{Adv}^X_{E,G}(\mathcal{O}_p) = \varepsilon$.

Para um $\lambda \in \mathbb{F}_p^*$, $\phi_\lambda(E)$ define a curva elíptica $Y^2 = X^3 + A\lambda^4 X + B\lambda^6$, denominada *twisted elliptic curve*. É fácil identificar que esses *twists* são na verdade isomorfismos. Assim, para pontos $Q, R, S, \in E$, com $Q = (x, y)$, temos $(Q)_\lambda = Q_\lambda = (\lambda^2 x, \lambda^3 y) \in \phi_\lambda(E)$ e para $Q + R = S$, temos $Q_\lambda + R_\lambda = S_\lambda$ (propriedade homomórfica). Ainda, com tais *twists*, definimos para uma curva inicial E_0, uma família de curvas isomorfas $\phi_\lambda(E_0)_{\lambda \in \mathbb{F}_p^*}$.

Seja K_{ab} o ponto combinado no DHCE sobre a curva E_0 com as chaves públicas PK_a e PK_b. A coordenada x de K_{ab} é a chave secreta. Da mesma forma como no ACGS, sorteamos a' e b' (notamos a/b linha para não confundir com as chaves privadas a/b) esperando que $[a'x]_p$ e $[b'x]_p$ sejam pequenos. Similarmente ao BKGCD, a' e b' geram valores d, aqui chamamos λ, e uma consulta para a paridade é feita com a chamada de $\mathrm{PAR}(\lambda, PK_a, PK_b)$. Porém, a diferença fica por conta da requisição ao LSB. A função PAR é perguntada pelo LSB de $\lambda^2 x$, pois $(K_{ab})_\lambda = (\lambda^2 x, \lambda^3 y)$. Então, a requisição é feita para um ponto que fica em outra curva, diferente de E_0.

Os autores em [Boneh and Shparlinski 2001] consideram um oráculo $\mathcal{O}_p$ com vantagem ε em predizer o LSB na curva E_0. Isso quer dizer que $\mathcal{O}_p$ não tem vantagem em todas as curvas na família $\phi_\lambda(E_0)_{\lambda \in \mathbb{F}_p^*}$. Por definição, é considerado que $\mathcal{O}_p$ mantém essa vantagem para pelo menos uma fração δ das curvas na família de isomorfismo.

O problema é que o algoritmo BS precisa consultar a paridade para vários valores de λ. Então, é construído um novo oráculo $\mathcal{B}_p$ cuja probabilidade de sucesso dentro ou fora da fração δ das curvas seja conhecida.

A seguir, resumidamente apresentamos os passos do algoritmo BS:

1. com entrada $\langle E, G, PK_a, PK_b \rangle$, onde $PK_a = aG$ e $PK_b = bG$ e G de ordem prima q, queremos calcular o ponto $C = abG$;
2. como a e b são fixos e desconhecidos, aleatorizamos o processo definindo $PK_{ra} = a_r aG$ e $PK_{rb} = b_r bG$, para $a_r, b_r \in [1, q-1]$, e esperamos que os valores $a_r a$ e $b_r b$ levem a um caso onde o oráculo $\mathcal{B}_p$ tenha vantagem não negligenciável;
3. sendo $DH_{E,G}(PK_{ra}, PK_{rb}) = D$, basta calcular D para obtermos C, pois $C = c_r \cdot D$, onde $c_r \equiv (a_r b_r)^{-1} \bmod q$;
4. agora, executamos um algoritmo similar ao ACGS com o oráculo $\mathcal{B}_p$;
5. a fim de assegurar que encontremos os valores $a_r a$ e $b_r b$ desejados pode ser necessário repetir todo o processo $\frac{8}{\varepsilon\delta}$ vezes.

Ao final do algoritmo BS, teremos uma lista de candidatos C para o ponto abG. Porém, ao contrário do ACGS, não temos condições de identificar automaticamente a alternativa correta, o que nos obriga a executar todas as alternativas e apenas ao final tentar identificar a resposta correta. A identificação é possível com um algoritmo devido a Shoup [Shoup 1997, Theo.7].

3.3. Biblioteca criptográfica Relic

Para implementar os algoritmos ACGS e BS, utilizamos a linguagem de programação C e a biblioteca criptográfica ***Relic Toolkit*** versão 0.3.0 [Aranha and Gouvêa].

Utilizamos as curvas SECG_P160 e NIST_P224 da biblioteca para o DHCE. Para o RSA, utilizamos módulos de 1024, 2048 e 5000 bits, além de um módulo teórico de 128 bits para registrar tempos mais demorados. Escolhemos 1024 e 5000 bits devido a [Fischlin and Schnorr 2000] e 2048 bits por ser a recomendação atual do NIST[1]. E por ter equivalência no nível de segurança a 1024 e 2048 bits do RSA, escolhemos 160 e 224 bits para curvas elípticas.

A respeito da implementação dos oráculos, a resposta para a consulta no DHCE é imediata, com a execução de apenas alguns poucos cálculos. Já no RSA, é feita uma decriptação para cada consulta, resultando em um tempo para a resposta bem maior.

O gerador pseudoaleatório construído dentro da Relic é o FIPS 186-2 baseado em SHA1. A Relic já possui ferramentas de cronômetro (*benchmark*) para registrar os tempos de execução, nós utilizamos a versão mais precisa desses cronômetros (HPROC). Para simulação dos algoritmos e coleta dos tempos de execução, utilizamos um computador com processador Intel Core 2 Duo T5450 de 1,66 Ghz com 2 GB de memória RAM.

Os códigos fontes das implementações estão disponíveis no site do Laboratório de Segurança de Dados do IME-USP[2].

4. Resultados

Nesta seção apresentamos os resultados obtidos das implementações. Apresentamos os resultados para o RSA com o ACGS e os resultados para o DHCE com o BS.

4.1. Resultados para o algoritmo ACGS

O sucesso da execução do algoritmo ACGS dependia que da escolha de a e b, resultasse em ax e bx pequenos ($abs_N(ax) < \frac{\varepsilon N}{2}$), e além disso, que fossem primos entre si. Por agilidade e a fim de dar mais regularidade aos tempos apresentados, implementamos o algoritmo de modo a sempre garantir essas condições. Depois, para a inferência de tempos, adicionamos de volta o tempo médio necessário para satisfazê-las. Esse método é consistente, porque que as probabilidades envolvidas são bem conhecidas.

Então, tendo essas condições iniciais satisfeitas e rodando o algoritmo sempre na alternativa correta, calculamos os tempos do algoritmo ACGS. Entretanto, mesmo com esses cortes, rodar e observar o tempo do algoritmo para um RSA de 1024 bits ainda era muito demorado, ainda mais para vantagens pequenas, $\varepsilon < 0{,}05$. Decidimos então coletar os dados de um RSA teórico de 128 bits para calcular esse tempo na alternativa correta.

[1] `http://csrc.nist.gov/groups/ST/toolkit/key_management.html`
[2] `http://lsd.ime.usp.br/`

Apresentamos a Figura 1 com os tempos para um RSA de 128 bits em relação ao número m de amostras. Observe que de acordo com o tamanho das amostras, a cada vez que o tamanho da amostra dobra, o tempo consumido de processamento também dobra.

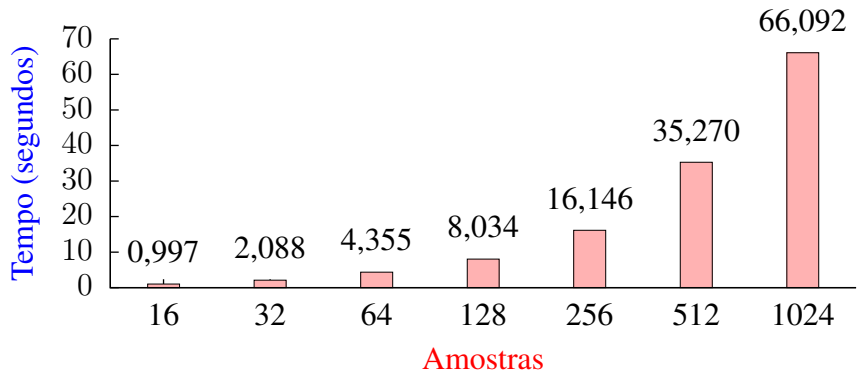

Figura 1. Tempo consumido em relação ao número de amostras.

As operações aritméticas realizadas dentro do algoritmo têm tempo variável em relação ao número de bits dos números calculados. O tempo da exponenciação com números de 256 bits é maior do que com números de 128 bits, o tempo da exponenciação com números de 512 bits é maior do que com números de 256 bits, e assim por diante. Na Figura 2 temos o tempo gasto a cada chamada à função PAR em relação ao tamanho dos operandos; a função PAR faz 16 consultas ao oráculo. Note que o tempo para 5000 bits foi apenas indicado por uma flecha para não prejudicar as proporções do gráfico.

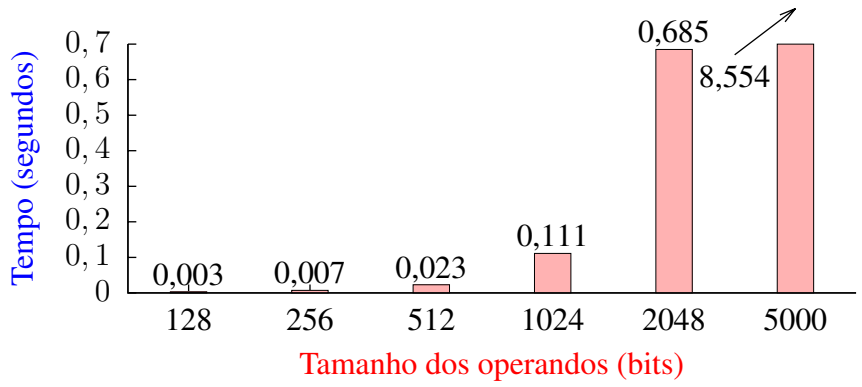

Figura 2. Tempo da função PAR no ACGS em relação aos operandos ($m = 16$).

O número m de amostragens é um parâmetro crítico do sistema. No modelo original do ACGS ele é por definição $m \stackrel{def}{=} \frac{64n}{\varepsilon^2}$. Dessa forma, o valor de m depende apenas do tamanho do módulo N em bits e da vantagem ε do oráculo. Na Tabela 1 apresentamos alguns exemplos de amostragem em relação ao par n, ε.

Procuramos tentar reduzir a amostragem. Percebemos que para valores menores do que a do modelo original, o algoritmo ACGS ainda obtinha sucesso em alguns casos. Após tentativas, obtivemos empiricamente a seguinte definição do parâmetro $m \stackrel{def}{=} \frac{4}{\varepsilon^2}$.

Na Figura 3, apresentamos exemplos para o novo padrão de amostragem. Note que no gráfico incluímos apenas a vantagem, já que no novo padrão o tamanho do módulo

Tabela 1. Amostragem em relação aos bits e à vantagem do oráculo.

	128 bits	1024 bits	5000 bits
0,4	51.200	409.600	2.000.000
0,3	91.023	728.178	3.555.556
0,2	204.800	1.638.400	8.000.000
0,1	819.200	6.553.600	32.000.000
0,05	3.276.800	26.214.400	128.000.000
0,01	81.920.000	655.360.000	3.200.000.000

não influencia o número de amostras. Faça uma comparação com os valores da Tabela 1.

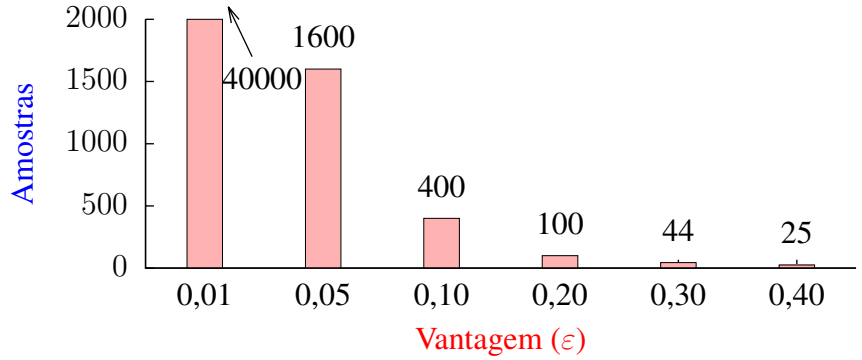

Figura 3. Número de amostras em relação à vantagem do oráculo.

Utilizando então essa nova metodologia, executamos simulações do ACGS para um RSA de 128 bits. O algoritmo obteve sucesso em sucessivas execuções até parar na primeira vez que falhou e não conseguiu recuperar a mensagem original. Observamos os seguintes resultados:

- $\varepsilon = 0{,}4$: rodou 66 vezes e parou;
- $\varepsilon = 0{,}3$: rodou 30 vezes e parou;
- $\varepsilon = 0{,}2$: rodou 53 vezes e parou;
- $\varepsilon = 0{,}1$: rodou 44 vezes e parou;
- $\varepsilon = 0{,}05$: rodou 29 vezes e parou;
- $\varepsilon = 0{,}01$: rodou 2 vezes e continuou.

Para o caso de $\varepsilon = 0{,}01$, ele rodou 2 vezes com sucesso, mas tivemos que abortar já que cada teste levou cerca de 40 minutos para ser executado. Para $\varepsilon = 0{,}2$ e $\varepsilon = 0{,}05$, com $m = 100$ e $m = 1600$ respectivamente, tivemos alguns problemas, os testes falhavam muito cedo. Porém, com apenas a adição de uma unidade na amostragem ($m = 101$ e $m = 1601$) tivemos resultados mais consistentes.

Observamos um fato particular ao rodar o ACGS na alternativa correta. Para um RSA de 128 bits, dentro de 200 testes, a função BKGCD fez de 324 a 368 chamadas à função PAR, ou seja, algo entre $2{,}53n$ e $2{,}88n$. Não conseguimos observar uma execução na alternativa correta com mais de $3n$ chamadas à função PAR.

O limitante da função BKGCD é definido por $limiteGCD = 6n + 3$, mas de acordo com nossas observações esse valor poderia ser alterado para $limiteGCD = 3n$.

Note que essa redução afeta não apenas a alternativa correta, mas também todas as demais alternativas, oferecendo uma redução pela metade no tempo do ACGS.

Assim, o novo tempo total inclui os tempos da condição inicial ($\varepsilon^{-2} \cdot 2$), o tempo de cada uma das alternativas ($2^6 m\varepsilon^{-2}$), o tempo da função BKGCD ($3n$), o tempo da função PAR (m) e o tempo do oráculo (1, consideramos constante): $(\varepsilon^{-2})(2)(2^6 m\varepsilon^{-2})(3n)(m)(1) = 3 \cdot 2^{11}\varepsilon^{-8}n$.

A Tabela 2 traz essa nova estimativa de complexidade junto com a original. Adicionamos também o tempo do algoritmo de solução para o PFI ([Buhler et al. 1993]) como o utilizado em [Fischlin and Schnorr 2000]. Perceba que agora o ACGS é mais rápido que o PFI.

Tabela 2. Comparação entre as complexidades existentes.

	Estimativa	1024 bits	2048 bits	5000 bits
PFI	$\exp(1{,}9(\ln N)^{\frac{1}{3}}(\ln \ln N)^{\frac{2}{3}})$	$6{,}409 \cdot 10^{25}$	$5{,}817 \cdot 10^{34}$	$3{,}755 \cdot 10^{50}$
ACGS orig.	$3 \cdot 2^{20}\varepsilon^{-8}n^3$	$3{,}378 \cdot 10^{31}$	$2{,}702 \cdot 10^{32}$	$3{,}932 \cdot 10^{33}$
ACGS novo	$3 \cdot 2^{11}\varepsilon^{-8}n$	$6{,}292 \cdot 10^{22}$	$1{,}258 \cdot 10^{23}$	$3{,}072 \cdot 10^{23}$

Agora podemos utilizar os tempos da função PAR na Figura 2 para obter estimativas de tempo. Nessa figura, temos o uso de apenas 16 amostras, mas como vamos utilizar $\varepsilon = 0{,}1$, teremos $m = 400$. Assim, seja **tPAR** o tempo da função PAR na Figura 2, o cálculo proporcional se dá pela fórmula $\frac{400}{16}\text{tPAR} \cdot 3n \cdot 2^6 m\varepsilon^{-2} \cdot \varepsilon^{-2} \cdot 2 = 3{,}84 \cdot 10^{10} \cdot \text{tPAR} \cdot n$.

A Tabela 3 traz os tempos estimados para se tentar inverter o RSA. Temos uma coluna para o tempo na alternativa correta e uma coluna para o tempo total do algoritmo, além do tempo **tPAR** da Figura 2.

Tabela 3. Tempos estimados para execução do algoritmo ACGS.

n (bits)	tPAR (s)	Alt. correta	Tempo total
1024	0,111	19,7 dias	$138{,}4 \cdot 10^3$ anos
2048	0,685	243,5 dias	$1{,}7 \cdot 10^6$ anos
5000	8,554	20,3 anos	$52{,}1 \cdot 10^6$ anos

4.2. Resultados para o algoritmo BS

Vamos seguir a mesma linha de discussão do RSA: com a nova amostragem definida e com o limitante para o BKGCD igual a $3 \lg p$. Apenas uma execução do algoritmo BS é considerada, assim, não entra aqui as execuções adicionais requeridas pelo algoritmo de Shoup.

Quanto à função BKGCD, não há problemas com o limitante igual a $3 \lg p$. Entretanto, quanto à amostragem, deve-se levar em conta que vamos utilizar dois algoritmos, o oráculo $\mathcal{O}_p$ e o oráculo $\mathcal{B}_p$. O oráculo $\mathcal{B}_p$ é regido pelo parâmetro δ que define a proporção de curvas onde o oráculo $\mathcal{O}_p$ possui vantagem ε. O caso ideal é quando $\delta = 1$, o que define o oráculo $\mathcal{O}_p$ como aplicável a todas as curvas da família.

No caso onde $\delta = 1$ temos um algoritmo BS semelhante ao ACGS. Com base nesse valor podemos obter os tempos da função PAR para operandos de 160 bits e 224 bits em um DHCE conforme a Figura 4.

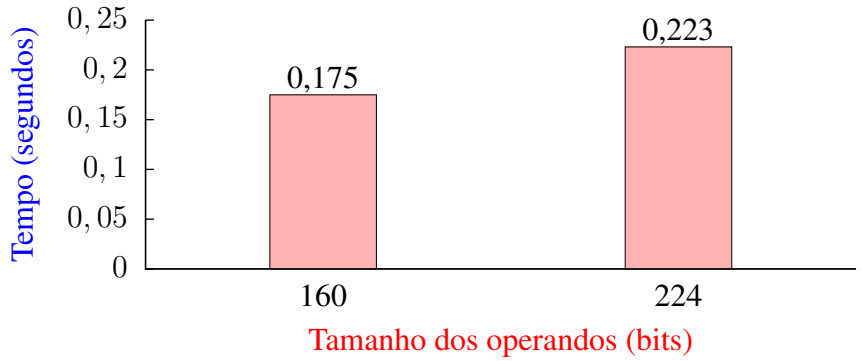

Figura 4. Tempo da função PAR no DHCE em relação aos operandos ($m = 40000$).

Escolhemos a amostragem $m \stackrel{def}{=} \frac{8}{\varepsilon^2}$ e executamos simulações do BS para um DHCE de 160 bits. O algoritmo obteve sucesso em sucessivas execuções até errar ou ser abortado pela extensão do tempo. Temos os seguintes resultados desses testes:

- $\varepsilon = 0{,}4$: rodou 368 vezes e parou;
- $\varepsilon = 0{,}3$: rodou 247 vezes e parou;
- $\varepsilon = 0{,}2$: rodou 183 vezes e parou;
- $\varepsilon = 0{,}1$: rodou 862 vezes e continuou;
- $\varepsilon = 0{,}05$: rodou 35 vezes e continuou;
- $\varepsilon = 0{,}01$: rodou 1 vez e continuou.

Agora vamos inferir os tempos para $\varepsilon = 0{,}1$ e $m = 800$, com os tempos **tPAR** da Figura 4. Temos o tempo total $\frac{800}{40000}\text{tPAR} \cdot 3 \lg p \cdot 2^6 m \varepsilon^{-2} \cdot \varepsilon^{-2} \cdot 2 = 6{,}144 \cdot 10^7 \cdot \text{tPAR} \cdot \lg p$. Dessa equação montamos a Tabela 4 com os tempos estimados para o BS.

Tabela 4. Tempos estimados para execução do algoritmo BS.

$\lg p$ (bits)	tPAR (s)	Alt. correta	Tempo total
160	0,175	5,6 minutos	54,6 anos
224	0,223	9,9 minutos	97,3 anos

Agora analisaremos o pior caso, onde $\delta < 1$ e próximo de zero. Fixamos $\delta = 0{,}1$, de modo que em apenas 10% das curvas o oráculo $\mathcal{O}_p$ tem vantagem dentro da família de isomorfismo. Então usamos o oráculo $\mathcal{B}_p$ que implica em uma nova amostragem $m2 \stackrel{def}{=} \frac{8}{\varepsilon\delta^2}$ relacionada com os acessos ao oráculo $\mathcal{O}_p$ pelo oráculo $\mathcal{B}_p$.

Essa nova amostragem é obtida utilizando a mesma metodologia anterior. Diretamente, apresentamos a Tabela 5 com os tempos estimados para se tentar descobrir a chave combinada pelo DHCE com uso do oráculo $\mathcal{B}_p$.

Tabela 5. Tempos estimados para execução do algoritmo BS com o oráculo $\mathcal{B}_p$.

$\lg p$ (bits)	Alt. correta	Tempo total
160	15,6 anos	$80 \cdot 10^6$ anos
224	44,7 anos	$229 \cdot 10^6$ anos

5. Conclusões

Com base nos algoritmos propostos, conseguimos construir implementações dos artigos com êxito na recuperação dos segredos. Alcançamos sucesso com um nível de convergência bem maior que o esperado. Nas implementações, fizemos o uso dos números pseudoaleatórios gerados pela Relic.

Nossos testes conseguiram, de maneira experimental, ajustar os parâmetros críticos do sistema para diminuir significativamente o tempo de execução. E com os tempos diminuídos, os métodos ainda alcançavam taxas de sucesso consideradas aceitáveis. Isto é garantido pela lei dos grandes números.

O resultados provêem medidas de quão rápido é inverter os criptossistemas citados a partir de um mínimo comprometimento do LSB. Em particular, no caso do ACGS com RSA-1024 tivemos um avanço significativo no tempo de inversão a ponto de ficar abaixo do tempo de solução para o PFI, anteriormente não alcançado.

Sugestões para trabalhos futuros. Em [Alexi et al. 1988] existe uma versão do ACGS para o algoritmo Rabin que poderia ser também implementada para base de comparação. Uma ideia para prova de conceito de nossos métodos é criar uma implementação falha (revelando o LSB através de um ataque de canal secundário) do RSA e então adaptá-la como oráculo para o ACGS. Um trabalho como esse é capaz de dar dimensões práticas da aplicabilidade do método.

Referências

[Alexi et al. 1988] Alexi, W., Chor, B., Goldreich, O., and Schnorr, C.-P. (1988). RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194–209.

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

[Ben-Or et al. 1983] Ben-Or, M., Chor, B., and Shamir, A. (1983). On the cryptographic security of single RSA bits. In *ACM Symposium on Theory of Computing (STOC '83)*, pages 421–430, Baltimore, USA. ACM Press.

[Blum et al. 1986] Blum, Blum, and Shub (1986). A simple unpredictable pseudo-random number generator. *SICOMP: SIAM Journal on Computing*, 15.

[Blum and Micali 1984] Blum, M. and Micali, S. (1984). How to generate cryptographically strong sequence of pseudo-random bits. *SIAM Journal Computing*, 13:850–864.

[Boneh and Shparlinski 2001] Boneh, D. and Shparlinski, I. (2001). On the unpredictability of bits of the elliptic curve Diffie-Hellman scheme. In Kilian, J., editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 201–212. Springer.

[Boneh and Venkatesan 1996] Boneh, D. and Venkatesan, R. (1996). Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *CRYPTO '96*, volume 1109 of *LNCS*, pages 129–142. IACR, Springer, Berlin.

[Brent and Kung 1984] Brent and Kung (1984). Systolic VLSI arrays for polynomial GCD computation. *IEEE Transactions on Computers*, 33.

[Buhler et al. 1993] Buhler, J. P., Lenstra, H. W., and Pomerance, C. (1993). Factoring integers with the number field sieve. In *The development of the number field sieve*, volume 1554 of *LNM*, pages 50–94. Springer-Verlag, Berlin.

[Chevalier et al. 2009] Chevalier, C., Fouque, P.-A., Pointcheval, D., and Zimmer, S. (2009). Optimal randomness extraction from a Diffie-Hellman element. In Joux, A., editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 572–589. Springer-Verlag.

[Diffie and Hellman 1976] Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654.

[Fischlin and Schnorr 2000] Fischlin, R. and Schnorr, C.-P. (2000). Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244.

[Goldwasser et al. 1982] Goldwasser, S., Micali, S., and Tong, P. (1982). Why and how to establish a private code on a public network (extended abstract). In *FOCS*, pages 134–144, Chicago, Illinois. IEEE.

[Hofheinz and Kiltz 2009] Hofheinz, D. and Kiltz, E. (2009). Practical chosen ciphertext secure encryption from factoring. In Joux, A., editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 313–332. Springer.

[Jetchev and Venkatesan 2008] Jetchev, D. and Venkatesan, R. (2008). Bits security of the elliptic curve Diffie-Hellman secret keys. In Wagner, D., editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 75–92. Springer.

[Knuth 1981] Knuth, D. E. (1981). *The Art of Computer Programming*, volume 2, Seminumerical Algorithms. Addison-Wesley, Reading, MA, 2 edition.

[Menezes et al. 1996] Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. (1996). *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, USA.

[Rabin 1979] Rabin, M. (1979). Digitalized signatures as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science.

[Rivest et al. 1978] Rivest, R. L., Shamir, A., and Adleman, L. M. (1978). A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126.

[Roh and Hahn 2010] Roh, D. and Hahn, S. G. (2010). On the bit security of the weak Diffie–Hellman problem. *Information Processing Letters*, 110:799–802.

[Ross 2006] Ross, S. M. (2006). *A First Course in Probability*. Prentice Hall, New Jersey, 7 edition.

[Shoup 1997] Shoup, V. (1997). Lower bounds for discrete logarithms and related problems. In Fumy, W., editor, *EUROCRYPT '97*, volume 1233 of *LNCS*, pages 256–266, Berlin Germany. Springer-Verlag.

[Stein 1967] Stein, J. (1967). Computational problems associated with Racah algebra. *Journal of Computational Physics*, 1(3):397 – 405.

