Universidade Federal de São João del-Rei

CRIPTOGRAFIA RSA E ALGORITMO AKS

José Luís Patrício da Silva Júnior

São João del-Rei - MG

2015

José Luís Patrício da Silva Júnior

Criptografia RSA e Algoritmo AKS

Trabalho de Conclusão de Curso apresentado à Coordenadoria do Curso de Matemática, da Universidade Federal de São João del Rei, como requisito parcial à obtenção do título de Licenciado em Matemática.

São João del-Rei, 13 de novembro de 2015

Banca Examinadora

_____
Orientador: Prof. Dr. Ronaldo Ribeiro Alves

_____
Prof. Ma. Carolina Fernandes Molina Sanches

_____
Prof. Esp. Marco Antônio Claret de Castro

AGRADECIMENTOS

Gostaria de agradecer ao professor Ronaldo que foi um anjo ao me acolher para a orientação deste trabalho.

Também ao professor Fábio que me introduziu este assunto ao qual gosto tanto.

A minha mulher, Jéssica, que mesmo nos momentos de fraqueza nunca me abandonou.

Finalmente, aos meus pais e irmã por simplesmente fazerem parte da minha vida.

RESUMO

Criptografia RSA e Algoritmo AKS

A criptografia é um assunto antigo, sendo esta, tema de filmes e conhecida por sua utilidade em guerras. Dentre as criptografias conhecidas, iremos destacar uma que utiliza método matemático, a Criptografia RSA, este método utiliza uma série de propriedades de aritmética modular e números primos com mais de 450 casas decimais, dependendo da necessidade. Iremos também tratar do Algoritmo AKS, o primeiro teste de primalidade de tempo polinomial e determinístico. Este trabalho é uma revisão bibliográfica sobre métodos matemáticos e computacionais envolvendo teoria dos números. No trabalho apresentado expomos o funcionamento da Criptografia RSA e do Algoritmo AKS, assim como fragilidades e características dos métodos, fazendo o leitor refletir sobre os avanços tecnológicos na área.

Palavras-chave: Algoritmo AKS, Criptografia RSA, Números primos.

Abstract

RSA Encryption and AKS Algorithm

Encryption is an old issue, which is the subject of films and known for its use in wars. Among the known encryptions, we will highlight one that uses mathematical method, RSA encryption, this method utilizes a series of modular arithmetic properties and prime numbers with more than 450 decimal, depending on the need. We will also deal with the AKS Algorithm, the first test of primality deterministic and polynomial time. This work is a bibliographic review of mathematical and computational methods involving number theory. In the work presented we expose the operation of the RSA encryption and AKS algorithm, as well as weaknesses and characteristics of the methods, making the reader think about the technological advances in the area.

Key-words: AKS Algorithm, RSA Encryption, Prime numbers.

SUMÁRIO

1 INTRODUÇÃO

A criptografia é um assunto antigo, sendo esta, tema de filmes e conhecida por sua utilidade em guerras. Devido a fatores comercias e o surgimento do computador, a criptografia e o estudo dos números primos deixaram de serem assuntos secundários dentro da matemática e passaram a ser estudados com mais afinco.

Dentre as criptografias conhecidas iremos destacar uma que utiliza um método matemático, a Criptografia RSA[1], esta criptografia utiliza uma série de propriedades de aritmética modular e números primos com mais de 450 casas decimais, dependendo da necessidade. A Criptografia RSA é conhecida por ser uma criptografia assimétrica.

Como se faz a necessidade de números primos tão grandes, iremos destacar alguns testes de primalidade, dentre eles o teste de Miller-Rabin e o Algoritmo AKS[2], este último sendo menos eficiente na busca por números primos, porém, diferente do Miller-Rabin que é probabilístico, o AKS é Determinístico, conforme Júnior (2010) destaca:

> A criptografia de chave pública é um método de encriptação muito utilizado hoje em dia, e estudos profundos foram feitos para o aperfeiçoamento da segurança desse sistema. Entretanto, essa ciência precisa ser constantemente aprimorada devido ao avanço da velocidade dos computadores, os quais estão se tornando cada vez mais capazes de fazer uma criptoanálise eficiente em um tempo relativamente curto. Um fator importante para garantir a segurança de um cifrário é o tamanho das chaves, que são números primos. Atualmente dispõe-se de algoritmos probabilísticos que acusam se um número é primo com baixíssimo percentual de erro, em tempo polinomial. O AKS é o primeiro algoritmo determinístico a executar este teste em tempo polinomial.

Temos por objetivo expor o conhecimento na área, proveniente de uma revisão bibliográfica concomitantemente aos testes realizados computacionalmente. Durante a leitura deste trabalho o leitor deve se questionar o seguinte: **“**Seria o algoritmo AKS um problema para a segurança computacional, visto que este encontra números primos em tempo polinomial?” (JUNIOR, 2010, p. 2)

---

[1] Criptografia RSA: É um método matemático de criptografia assimétrica, as letras RSA equivalem as iniciais de R. L. Rivest, A. Shamir e L. Adleman, inventores do código.

[2] Algoritmo AKS: É um algoritmo de tempo polinomial e determinístico quanto a primalidade de um número, similar ao RSA, as letras AKS equivalem as inicias de Manindra Agrawal, Neeraj Kayal, Nitin Saxena, inventores do algoritmo.

A justificativa deste trabalho, e resposta a questão anterior, é que, mesmo com grandes avanços na área computacional e a crescente necessidade de segurança digital, um teste computacional de tempo polinomial leva demasiado tempo e possui um custo computacional elevado, conforme veremos.

2 METODOLOGIA

Foi realizada uma vasta pesquisa em materiais já publicados para agregar conceitos teóricos e compreender a base matemática na qual são fundamentados os métodos descritos neste trabalho. Dentre os materiais pesquisados continham artigos, monografias e livros. Neste período também foram realizados testes computacionais sendo estes realizados em um computador pessoal com as seguintes características:

Processador: Intel(R) Core(TM) i3-4130 CPU @ 3.40GHz 3.40 GHz

Memória instalada (RAM): 4,00 GB

Tipo de sistema: Sistema Operacional de 64 bits, processador com base em x64.

Podemos destacar as seguintes etapas realizadas durante o trabalho realizado:

- Revisão bibliográfica: Aritmética modular e relações pertinentes;
- Revisão bibliográfica: Criptografia RSA;
- Revisão bibliográfica: Algoritmo AKS;
- Implementação e testes computacionais: Criptografia RSA e Algoritmo AKS;
- Escrita do material final.

Para implementar os softwares descritos neste trabalho utilizamos a linguagem computacional C/C++.

3 REFERENCIAL TEÓRICO

Para compreender melhor os cálculos aqui envolvidos, iremos introduzir alguns conceitos, dentre eles o conceito de número primo, a aritmética modular e o Pequeno Teorema de Fermat.

3.1 Números primos

Dentre todos os números conhecidos, alguns merecem destaque, sendo eles um mistério desde a Grécia antiga. Os números primos, por definição, possuem apenas dois divisores. Isso se dá, pois $\forall \mathrm{p}$ primo:

$$\forall \mathrm{n} \in \mathbb{N} | \mathrm{n} < p, mdc(n, p) = 1$$

Os números que não são primos são chamados compostos, todos os números compostos podem ser decompostos de forma única em fatores primos. Logo, todo número é primo ou é composto.

Apesar deste destaque entre os números, os números primos só receberam a devida atenção após da década de 60, com o surgimento dos computadores. Ainda assim, o aprofundamento em seus estudos tem uma característica mais comercial do que uma característica puramente acadêmica, conforme podemos destacar em Binder (2008, p. 1):

> A aplicação mais conhecida para os números primos é a criptografia de chave pública, que permite transações bancárias seguras e encriptação de emails ou de dados num disco rígido. Tudo para que terceiros não tenham acesso a informações que devem ser protegidas. Um dos algoritmos mais conhecidos é o RSA, que envolve a multiplicação de dois números primos p e q.

Existem alguns fatos interessantes sobre números primos que não iremos provar. O primeiro fato é que, qualquer número, primo ou composto, possui uma única fatoração de termos primos. O segundo fato é que existem infinitos números primos, logo ainda existem infinitos números primos a serem descobertos.

3.2 Módulo

A aritmética modular usada nos cálculos descritos neste trabalho envolve, principalmente, a relação módulo ou modulo o'clock. Ela diz que se A é congruente a B módulo N, então $A - B$ é divisível por N. Podemos também dizer que A e B deixam o mesmo resto quando divididos por N. A simbologia utilizada pra expressar isto é a seguinte:

$$A \equiv B \ (\mathrm{Mod}\ N)$$

3.3 Pequeno Teorema de Fermat

O Pequeno Teorema de Fermat é a base para o funcionamento dos testes de primalidade mais importantes utilizados atualmente, assim como na Criptografia RSA, conforme descreve "O Pequeno Teorema de Fermat é um teorema relacionado aos números primos e tem sido o passo inicial de vários algoritmos de descoberta de primalidade como o Monte-Carlo e o próprio AKS." (JUNIOR, 2010, p. 18)

Pequeno teorema de Fermat: Se p é um primo, então $a^{p-1} - 1$ é divisível por p, sempre que a for um inteiro que não é divisível por p.

Ou seja, se $\forall a$ tal que m.d.c.$(a,\ p) = 1$ e p for primo, então, $a^{p-1} \equiv 1 \ (\mathrm{Mod}\ p)$.

3.4 Função ϕ

Segundo Coutinho(2000) a função $\phi(n)$, lê-se phi de n, é a função que define quantos números menores do que n são primos com n, ela também pode ser conhecida como a função totiente. Esta função é importante nos dois métodos que trataremos, tanto para a Criptografia RSA, quanto para fornecer uma cota superior para um dos testes do Algoritmo AKS. A função $\phi(n)$ é representada da seguinte forma:

$$\phi(n) = (p-1).p^{a-1}.(q-1).q^{b-1}.(r-1).r^{c-1}.(s-1)s^{d-1}...$$

Onde $p, q, r, s, ... \in N$ são os fatores primos de n e os índices $a, b, c, d, ... \in N$ os respectivos índices dos fatores primos.

A função $\phi(n)$ é utilizada na Criptografia RSA com o propósito de gerar chaves de criptografia e decriptografia. Ela é utilizada na seguinte equação Diofantina:

$$c.d + \beta.\phi(n) = 1$$

Outra forma de representar esta equação é dizer que:

$$c.d \equiv 1 \,(\mathrm{Mod}\, \phi(n))$$

3.5 Teorema de Euler

Também para este trabalho, faz sentido citar o Teorema de Euler, ou também conhecido por Teorema de Fermat-Euler. Este teorema garante que:

$$(a)^{\Phi(n)} \equiv 1(Mod\ n)$$

Esta relação é utilizada na Criptografia RSA na etapa de decriptografia.

3.6 Função Piso e Função Teto

Para o Algoritmo AKS utilizamos duas funções que iremos definir na sequência. Para estas funções iremos utilizar exemplos numéricos.

Função Piso: $\lfloor a \rfloor$

$\lfloor a \rfloor \leq a < \lfloor a+1 \rfloor$.

$\lfloor 3{,}01 \rfloor = 3$

$\lfloor 2{,}99... \rfloor = 2$

$\lfloor 5 \rfloor = 5$

Função Teto: $\lceil b \rceil$

$b \leq \lceil b \rceil < b+1$

$\lceil 3{,}01 \rceil = 4$

⌈ 2,99... ⌉ = 3

⌈ 5 ⌉ = 5

Em resumo, estas funções são funções de inteiro menor ou igual (⌊ a ⌋) e inteiro maior ou igual (⌈ b ⌉).

3.2 Introdução a Criptografia RSA

A Criptografia RSA, que é um método matemático com chave pública, encaminha sua mensagem juntamente com sua chave de codificação. Para o leitor isto pode parecer um absurdo ou uma ousadia.

Imagine enviar uma mensagem criptografada e a chave de codificação utilizada para criptografar esta mensagem. A Criptografia RSA zomba dos “hackers” que tentam decodificar este método, principalmente por ser possível fazê-lo. O RSA é seguro por diversos motivos, mas o principal desses motivos é a dificuldade de se fatorar n, que é um número composto de dois primos. Devido a esta dificuldade de se fatorar n, a chave de criptografia não é o suficiente para se encontrar a chave de decriptografia. Porém, ainda assim, os ataques são constantes, conforme revela Barbosa (2003, p. 16):

> A segurança desse método se baseia na dificuldade da fatoração de números inteiros extensos. Em 1977, os criadores do RSA achavam que uma chave de 200 bits requereriam 10^{15} anos, porém chaves com 155 bits foram atacadas em menos de 8 meses. A saída é que na medida que os algoritmos se tornem melhores e os computadores se tornem mais velozes, maiores serão as chaves. Atualmente chaves com 300 dígitos (1000 bits) nos dão uma tranquilidade por algum tempo. Em níveis críticos, chaves com 2000 bits começam a ser usadas.

4 CRIPTOGRAFIA RSA

O método de criptografia RSA é um método matemático de criptografia assimétrica[3] com chave pública, isto significa que a mensagem, já criptografada, é enviada junto à chave de codificação. Neste método apenas quem possui a chave de decodificação poderá ter acesso a mensagem original, esta chave deve ser secreta, caso não seja, o processo estará comprometido para estas chaves.

A chave pública consiste de dois números "c" e "n" ou $(c,\ n)$. O número n é um número composto de dois números primos p e q, estes primos são cruciais para a segurança do método. O número c é um número que deve ser primo com $\phi(n)$, valor da função totiente de Euler, isso é necessário para que exista um número "d" que faz parte da chave de decodificação (d, n).

A Criptografia RSA é composta de três etapas, a pré-codificação e codificação, responsável pelo remetente, e a decodificação, responsável pelo destinatário.

A pré-codificação quer dizer substituir os símbolos da mensagem por números, esta transformação segue uma tabela comum às duas partes, podendo ser secreta ou não. Em seguida a mensagem numérica é fatorada em blocos numéricos menores que n, essa fatoração não é única.

A codificação consiste em pegar os blocos da etapa anterior e realizar o seguinte processo:

$$(b_i)^c \equiv C_i (\mathrm{Mod}\ n)$$

Todo bloco pré-codificado b_i, com $i \in N$, elevado a chave c, será congruente a um único bloco C_i módulo n.

Estes blocos C_1, C_2,..., C_{m-1} e C_m não podem ser tirados de ordem ou reunidos de modo a formar um longo número, caso isso aconteça será impossível decodificar esta mensagem, pois a fatoração destes blocos não é única conforme descrito anteriormente.

[3] Criptografia Assimétrica: Diferente das criptografias simétricas onde existe uma inversa da função geradora da mensagem criptografada, a criptografia assimétrica exige duas chaves, uma de criptografia e uma de decriptografia, sem esta segunda o processo de decriptografia se torna praticamente inviável.

A última etapa, a de decodificação, consiste em aplicar nos blocos da etapa anterior e realizar o seguinte processo:

$$(C_i)^d \equiv D_i \,(\mathrm{Mod}\ n)$$

Posteriormente mostraremos que estes blocos D_i são iguais aos b_i iniciais, ou seja, os blocos iniciais.

4.1 Compreendendo o funcionamento do RSA

Para se exemplificar melhor o funcionamento da Criptografia RSA o leitor deve acompanhar o método que explicitaremos abaixo passo a passo. Utilizaremos um exemplo numérico para facilitar a compreensão.

Neste momento, devemos ter uma mensagem a ser criptografada, uma tabela de conversão numérica comum às duas partes, uma chave de criptografia (pública) e uma chave de decriptografia (Oculta).

Vamos supor que queremos enviar a pequena mensagem "*TCC de Criptografia RSA e Algoritmo AKS 2015*" com a chave pública (199, 19549).

Com isso iniciamos a pré-criptografia. Para isto vamos usar a seguinte tabela:

A=10	B=11	C=13	D=14	E=15	F=16	G=18	H=19	I=20	J=21
K=22	L=23	M=24	N=25	O=26	P=27	Q=28	R=29	S=30	T=31
U=32	V=33	W=34	X=35	Y=36	Z=37	0=38	1= 39	2= 40	3= 41
4= 42	5= 43	6= 44	7= 45	8= 46	9= 47	Espaço = 99			

Com essa tabela vamos ter o seguinte:

"*TCC de Criptografia RSA e Algoritmo AKS 2015*" → "3113139914159913292027312618291016201099293010991023182629203124269 91022309940383943"

O leitor deve estar se questionando o motivo pelo qual cada símbolo equivale a um número de dois algarismos. Neste caso, o motivo é para se evitar ambiguidades. Suponha que nossa tabela comece pelo A=1, B=2, C=3, etc., com isso teremos M=13, porém as letras em sequência AC é igual a 13, assim como M. Os números de dois algarismos ou mais resolvem este tipo de problema, pois evitam ambiguidades.

Separamos o número obtido em blocos menores que nosso n, que é 19549:

3113 – 13991 – 4159 – 9132 – 9202 – 7312 – 6182 – 9101 – 620 – 10992 – 930 – 1099 – 10231 – 8262 – 9203 – 12426 – 9910 – 2230 – 9940 – 3839 – 43

Recordamos ao leitor que esta fatoração não é única e caso o leitor queira, poderá ter outros blocos ou até mais blocos que o descrito aqui. Lembre-se também que os blocos não devem se iniciar por zero, afinal, zero a esquerda não é considerado. A partir deste momento, os blocos não podem ser mudados de posição ou reunidos, caso isso aconteça a mensagem estará comprometida.

Neste ponto, terminamos a pré-codificação e iniciaremos a codificação.

Sabemos que $(b_i)^c \equiv C_i(\text{Mod } n)$, ou seja $(b_i)^{199} \equiv C_i(\text{Mod } 19549)$, logo para os blocos criados teremos:

$b_1 \rightarrow 3113^{199} \equiv \mathbf{16261} \ (\text{Mod } 19549) \rightarrow C_1$

$b_2 \rightarrow 13991^{199} \equiv \mathbf{640} \ (\text{Mod } 19549) \rightarrow C_2$

$b_3 \rightarrow 4159^{199} \equiv \mathbf{13360} \ (\text{Mod } 19549) \rightarrow C_3$

$b_4 \rightarrow 9132^{199} \equiv \mathbf{2561} \ (\text{Mod } 19549) \rightarrow C_4$

$b_5 \rightarrow 9202^{199} \equiv \mathbf{6344} \ (\text{Mod } 19549) \rightarrow C_5$

$b_6 \rightarrow 7312^{199} \equiv \mathbf{18577} \ (\text{Mod } 19549) \rightarrow C_6$

$b_7 \rightarrow 6182^{199} \equiv \mathbf{11006} \ (\text{Mod } 19549) \rightarrow C_7$

$b_8 \rightarrow 9101^{199} \equiv \mathbf{6816} \ (\text{Mod } 19549) \rightarrow C_8$

$b_9 \rightarrow 620^{199} \equiv \mathbf{7365} \ (\text{Mod } 19549) \rightarrow C_9$

$b_{10} \rightarrow 10992^{199} \equiv \mathbf{945} \ (\text{Mod } 19549) \rightarrow C_{10}$

$b_{11} \rightarrow 930^{199} \equiv \mathbf{4120} \ (\text{Mod } 19549) \rightarrow C_{11}$

$b_{12} \rightarrow 1099^{199} \equiv \mathbf{16909} \ (\text{Mod } 19549) \rightarrow C_{12}$

$b_{13} \rightarrow 10231^{199} \equiv \mathbf{14726} \ (\text{Mod } 19549) \rightarrow C_{13}$

$b_{14} \rightarrow 8262^{199} \equiv \mathbf{7521} \ (\text{Mod } 19549) \rightarrow C_{14}$

$b_{15} \rightarrow 9203^{199} \equiv \mathbf{13990} \ (\text{Mod } 19549) \rightarrow C_{15}$

$b_{16} \rightarrow 12426^{199} \equiv \mathbf{7620} \ (\text{Mod } 19549) \rightarrow C_{16}$

$b_{17} \rightarrow 9910^{199} \equiv \mathbf{16572} \ (\text{Mod } 19549) \rightarrow C_{17}$

$b_{18} \rightarrow 2230^{199} \equiv \mathbf{8256} \ (\text{Mod } 19549) \rightarrow C_{18}$

$b_{19} \rightarrow 9940^{199} \equiv \mathbf{11688} \ (\text{Mod } 19549) \rightarrow C_{19}$

$b_{20} \rightarrow 3839^{199} \equiv \mathbf{10669} \ (\text{Mod } 19549) \rightarrow C_{20}$

$b_{21} \rightarrow 43^{199} \equiv \mathbf{788} \ (\text{Mod } 19549) \rightarrow C_{21}$

Após isto, temos a seguinte mensagem codificada:

(199, 19549) 16261 – 640 – 13360 – 2561 – 6344 – 18577 – 11006 – 6816 – 7365 – 945 – 4120 – 16909 – 14726 – 7521 – 13990 – 7620 – 16572 – 8256 – 11688 – 10669 – 788

Para decodificar uma mensagem é necessária uma chave de decodificação, essa chave é única e muito similar à chave de codificação, ela é composta de dois números, um número d relativamente primo com $\phi(n)$ e atendendo a seguinte propriedade:

$$c.d + \beta.\phi(n) = 1$$

Ou no caso descrito:

$$199.d + \beta.\phi(19549) = 1$$

Para este n pequeno conseguimos fatorá-lo rapidamente em 113 e 173, logo temos que $\phi(19549) = (113 - 1).(173 - 1) = 19264$. Então:

$$199.d + \beta.19264 = 1$$

Portanto:

$$199.14327 + (-148).19264 = 1$$

Temos então que d=14327, com isso a chave de decodificação é (14327, 19549).

Logo, para qualquer bloco C_i, temos que:

$$(C_i)^d \equiv D_i(Mod\ n)$$

Agora codificamos e decodificamos uma mensagem, precisamos então mostrar que $b_i = D_i$. Durante o processo fizemos o seguinte:

$$((b_i)^c)^d \equiv (C_i)^d \equiv D_i(Mod\ n)$$

E sabemos que:

$$b_i{}^{c.d} \equiv b_i{}^{1-\beta.\phi(n)} \equiv D_i(Mod\ n)$$

Por fim:

$$b_i.((b_i)^{\phi(n)})^{-\beta} \equiv D_i(Mod\ n)$$

Pelo teorema de Euler $(b_i)^{\phi(n)} \equiv 1(Mod\ n)$, concluímos que:

$$b_i.(1)^{-\beta} \equiv b_i = D_i(Mod\ n)$$

Conforme nosso caso numérico, temos, para o bloco b_1, que:

$$((3113)^{199})^{14327} \equiv (16261)^{14327} \equiv 3113(Mod\ 19549)$$

O leitor deve notar que apenas conseguimos criar uma chave de decodificação por conhecermos $\phi(n)$ ou conhecemos os fatores primos de n, caso contrário, não conseguiríamos tão facilmente ter acesso à chave de decodificação (d, n).

4.2 Algumas considerações sobre segurança no RSA

Devido aos diversos usos da Criptografia RSA, seja em transações comerciais ou troca de mensagens secretas, este método tem sido alvo de diversos ataques. Iremos citar algumas vulnerabilidades desta criptografia, caso o usuário do RSA se atente a algumas medidas o método permanecerá seguro. As vulnerabilidades a seguir são exemplos das que foram colhidas nas bibliografias utilizadas e algumas percebidas nos testes realizados.

1. Fatoração de números grandes;

Conforme citamos anteriormente, são utilizados números primos grandes para a obtenção de um n seguro para o método. Segundo Coutinho(2000), para uso pessoal é recomendado números de 768 bits, que equivale a um n com 231 casas decimais, já outros recomendam números com 1024 a 2048 bits para se garantir a segurança por mais tempo.

Outra preocupação em se escolher primos grandes para formar n é que p e q devem ser distantes um do outro, afinal pode-se utilizar o método de fatoração de Fermat para decompor o número n em fatores $(x+y)$ e $(x–y)$ tais que:

$$n = (x + y).(x - y) = x^2 - y^2 \Rightarrow x = \sqrt{n + y^2}$$

Como a distância entre p e q será pequena, então y^2 será pequeno o suficiente para se encontrar e fatorar p e q.

2. Uso de c e d como chave de codificação;

Esta falha em segurança apenas ocorre caso uma chave seja utilizada para dois usuários. Obviamente não será disponibilizado o mesmo (c, n) para os dois usuários, porém as chaves(c, n) e (d, n) são feitas em pares. Um usuário com a chave (c_1, n) poderá testar sua chave em outros usuários com o mesmo n e encontrar um (c_2, n) tal que $c_1 c_2 \equiv 1 \ (\mathrm{Mod}\ n)$, logo $c_2 = d_1$ e $d_2 = c_1$, possuindo a chave de decodificação de outro usuário.

3. Ataques temporais;

Os ataques que se encaixam nesta vulnerabilidade são ataques que visam erros ou falhas na implementação do RSA. Neste caso existem motivos envolvidos na segurança do RSA que não são apenas matemáticos.

Este tipo de ataque visa geralmente sistemas de segurança que possuem confirmação de usuários.

4. Ataques anônimos.

Os ataques deste tipo de vulnerabilidade são aqueles em que, por não se conter uma assinatura ou confirmação de usuário e a chave de codificação ser pública, qualquer pessoa pode codificar uma mensagem e encaminhar ao destinatário, afinal a chave (c, n) é pública.

4.3 Assinaturas no RSA

As assinaturas digitais do RSA possuem as mesmas características da criptografia e decriptografia de mensagens. Apesar de possuir uma fragilidade caso seja mal implementado, aplicar esta ferramenta garante segurança aos usuários e ao próprio receptor, afinal um usuário não poderá negar que assinou a mensagem.

Suponha que seja necessária a Criptografia RSA para realizar uma importante transação comercial entre uma empresa e um banco. A empresa e o banco precisam constantemente confirmar se o emissor é realmente quem afirma ser, pois haveria um grande prejuízo no caso de falha na segurança. Para isto a empresa possui suas chaves C_e e D_e, de criptografia e decriptografia respectivamente, e o banco as chaves do cliente C_b e D_b.

Nestas condições, para se utilizar a assinatura, a empresa que quer enviar um bloco “a” qualquer, e utiliza sua chave de decodificação, que é oculta, no bloco a, tendo com isso um bloco $D_e(a)$, em seguida aplica a chave de codificação do banco, que é pública, no bloco $D_e(a)$, resultando no bloco $C_b(D_e(a))$.

Este bloco final ao ser encaminhado para o banco poderá decodificar a mensagem com sua chave de decodificação, que é oculta, resultando em outro bloco codificado $D_b(C_b(D_e(a))) = D_e(a)$. Após esta operação o banco utiliza a chave de codificação da empresa, que é pública, e obtém o bloco a original a $C_e(D_e(a)) = a$.

Observe que apenas a empresa poderia assinar o bloco “a”, afinal apenas ela possui acesso a sua chave de decodificação. Observe também que, se a

mensagem recebida e operada pelo banco resultar em uma mensagem incompreensível, significa que o emissor não é a empresa, evitando com isso a fraude.

5 TESTES DE PRIMALIDADE

Os números primos sempre foram presentes e os métodos para encontrá-los existem desde a Grécia antiga. Estes métodos eram pouco elaborados ou falhos. Apesar de existir essa questão na matemática há tanto tempo, este foi sempre um problema secundário.

Recentemente, na década de 60, a situação mudou. Devido ao surgimento dos computadores, que possuem capacidades de processamento cada vez mais elevadas, e com a Criptografia RSA, que demanda de uma grande quantidade de números primos extremamente grandes, a busca por números primos se tornou ferrenha.

Com isso, foram criados testes de primalidade, responsáveis por testar números a fim de verificar sua primalidade. Até a criação do Algoritmo AKS eram separados em duas classes com características disjuntas. São elas:

- Não-polinomial(NP) e Determinístico

Os testes que se encaixam nesta classe são aqueles que podem definir com 100% de certeza se um número é primo, porém a um custo, seja de tempo ou computacional, muito elevado. Um exemplo de teste Não-polinomial e Determinístico é o Crivo de Eratóstenes, que será descrito na seção 5.1.

- Polinomial e Não-determinístico

Os testes que se encaixam nesta classe são aqueles que possuem um tempo de resposta pequeno, mesmo com números muito grandes, porém não é uma certeza quanto a primalidade do número testado. Um exemplo de teste Polinomial e Não-determinístico é o teste de Miller-Rabin, que será descrito na seção 5.2.

5.1 Crivo de Eratóstenes

O Crivo de Eratóstenes é o teste de primalidade mais antigo que se tem conhecimento. Ele consiste em passar todos os números menores que n em um crivo(peneira), filtrando estes números e restando apenas os números primos.

Após anos desde a criação deste método, percebeu-se que é necessário buscar primos até $\sqrt{n}$, pois os outros números maiores que $\sqrt{n}$ até n ou já foram marcados como compostos ou são primos.

Este teste é um teste determinístico, porém seu tempo é não-polinomial, isto é, conforme o valor de n cresce, mais iterações são necessárias. Além disso, o Crivo de Eratóstenes necessita que os números sejam dispostos de forma crescente para serem verificados. Logo, descrever todos os números de 2 a 10^{100} pode ser ainda mais trabalhoso que cortá-los da lista. Por outro lado os números obtidos são todos primos.

5.2 Miller-Rabin

O teste de Miller-Rabin, provavelmente, é o teste de primalidade mais utilizado para verificar candidatos a números primos, possuindo tempo polinomial. Nos artigos onde são comparados os tempos de resposta entre o Algoritmo AKS e o teste de Miller-Rabin verifica-se que “Em todos os casos testados verificou-se a eficiência na questão tempo de processamento e memória no algoritmo do Miller-Rabin é superior ao Algoritmo do AKS.”(FARIAS, 2007, p. 8)

O algoritmo AKS em termos práticos, apesar de ser determinístico, perde em termos de tempo de resposta e praticidade, porém não existe uma chance de falha. Por não existir essa chance de falha, em termos de estudo, o AKS é mais interessante, conforme ressalta Farias(2007, p. 8) “O AKS tem sua relevância do ponto de vista matemático, mas em termos práticos o Rabin tem-se mostrado mais eficiente.”

Citaremos agora o funcionamento do teste de Miller-Rabin, porém sua implementação, com mais detalhes, pode ser encontrada em Farias (2007):

Teste de Miller-Rabin: Seja p ímpar e $p\text{-}1 = 2^k q$, com q ímpar e $k \geq 1$. Se p é primo e $a \in Z^*_p$, então $a^q \equiv 1 \pmod{n}$ ou existe um $i \in \{0, 1, \ldots, k\text{-}1\}$ tal que:

$$a^{2^i p} \equiv 1 \ (mod\ p)$$

Dado um “a” aleatório, calcula-se sucessivamente $a_0 \equiv a^p \pmod p$, $a_1 \equiv a_0^2 \pmod p$, ..., $a_k \equiv a_{k-1}^2 \pmod p$ até que $k = t$ ou $a_k \not\equiv 1 \pmod p$.

Se $k = t$ e $a_k \not\equiv 1 \pmod p$, então p é composto.

Se $k = 0$, então p é primo.

Se $a_{k-1} \not\equiv -1 \pmod p$ então p é composto.

Caso contrário p é primo.

Para um “a” aleatório e o teste retorne que p é primo existe a chance de 75% de p ser primo, o teste pode ser feito para outro “a” aleatório, aplicando 75% de chance sobre os 25% de o número não ser primo, aumentando as chances de o número ser primo para 93%, podendo ser realizado este teste sucessivas vezes incrementando cada vez mais a chance do número ser primo. Por exemplo: (Probabilístico) $0{,}75 + 0{,}75.(0{,}25) + 0{,}75.(0{,}25)^2 + \ldots \leq 100\%$ (Determinístico)

Sobre a geração de números primos temos Farias (2007, p. 7):

> Se os primos se destinarem para o uso "industrial" (por exemplo, para a encriptação RSA), geralmente não é necessário provar sua primalidade. É suficiente saber que a probabilidade do número ser composto é menor do que 10^{-24}%. Neste caso, podem-se utilizar os testes (fortes) de primalidade provável.

6 TESTE DE PRIMALIDADE: ALGORITMO AKS

Finalmente citaremos o teste de primalidade com tempo polinomial e determinístico. O teste de primalidade conhecido como Algoritmo AKS foi o primeiro a unir as características polinomial e determinística, antes características disjuntas, conforme é descrito em Farias (2007, p. 3):

> O algoritmo AKS ganhou destaque por ser o primeiro algoritmo publicado que é simultaneamente polinomial, determinístico, e incondicional. O que isto significa, o tempo máximo de processamento do algoritmo pode ser expresso como um polinômio em relação ao número de dígitos do número analisado. Isto permite classificar o número informado como primo ou composto ao invés de retornar um resultado probabilístico.

Além de identificar os números primos, o tempo computacional deve ser levado em consideração, afinal, já conseguimos encontrar números primos com o Crivo de Eratóstenes. Quando citamos o custo computacional, neste caso, queremos enfatizar quantas iterações por segundo podem ser realizadas, afinal não podemos esperar oito meses ou mais para verificar se um número é primo. Em concordância ao descrito, citamos Júnior (2010, p. 15) “Tratando-se de algoritmos, é sempre interessante buscar o mais eficiente, isto é, mais rápido e com custo computacional mais baixo”.

O Algoritmo AKS pode ser implementado a partir de um pseudo-algoritmo, este pseudo-algoritmo pode ser encontrado em Agrawal, Kayal e Saxena(2004, p. 3):

Input: integer n > 1.
1. If ($n = a^b$ for $a \in \mathbb{N}$ and $b > 1$), output COMPOSITE.
2. Find the smallest r such that $o_r(n) > \log^2 n$.
3. If $1 < mdc(a, n) < n$ for some $a \leq r$, output COMPOSITE.
4. If $n \leq r$, output PRIME.
5. For a = 1 to $\lfloor \sqrt{(\phi(r).\log n} \rfloor$ do
    if $((X + a)^n \not\equiv X^n + a \ (\text{Mod } X^r - 1, n))$, output COMPOSITE;
6. Output PRIME;
Algorithm for PrimalityTesting

Obviamente o pseudo-algoritmo não irá funcionar apenas com estas informações em qualquer software. O Algoritmo AKS é composto de pequenas operações de tempo polinomial que servem para o propósito de realizar pré-testes e

encontrar um r[4] "confortável" para realizar outros testes de tempo polinomial conforme iremos expor na sequência.

6.1 Conhecendo o funcionamento do Algoritmo AKS

O Algoritmo AKS é composto de pequenos testes de tempo polinomial que realizam pré-testes quanto à primalidade do número e encontram um r "confortável" para realizarmos o teste AKS propriamente dito.

Neste momento iremos revelar quais são estes pré-testes, o r "confortável" e o último teste. Este conjunto de testes é o Algoritmo AKS.

Caso o leitor queira implementar o método, informamos que todas as funções $\log\ n$ utilizadas no Algoritmo AKS estão na base 2, assim como em Agrawal, Kayal, Saxena (2004), deixaremos indicados apenas como $\log n$.

1. If $(n = a^b \text{ for } a \in \mathbb{N} \text{ and } b > 1)$, output COMPOSITE.

Este primeiro pré-teste consiste em verificar se n é uma potência própria de algum número. Por definição podemos procurar um b menor ou igual a $\log_{10} n$.

2. Find the smallest r such that $o_r(n) > \log^2 n$.

Encontrar o r "conveniente". Este r é o menor inteiro positivo que não divide $n^{\lceil log^5 n \rceil}.\prod_{i=1}^{\lfloor \log^2 n \rfloor}(n^i - 1)$.

Exemplos: Para 1993, $r = 5$. Para 31, $r = 29$.

3. If $1 < mdc(a, n) < n$ for some $a \leq r$, output COMPOSITE.

Calcular o máximo divisor comum de a e n, para a natural menor que r, se for maior do que 1 então n é composto; ou seja, se o MDC for maior que 1 então existe algum valor maior que um que divide algum a e n.

4. If $n \leq r$, output PRIME.

Segundo Agrawal, Kayal, Saxena (2004, p 3) "[…] that $r \leq \lceil \log^5 n \rceil$, so Step 4 is relevant only when $n \leq 5690034$", isto se dá pois, 5690034 é raiz da equação: $\lceil \log^5 n \rceil - n = 0$, logo para todo n maior que 5690034 então r nunca será maior que n.

[4] Detalhes sobre esse r "confortável" será descrito na sequência. Para mais informações indicamos Agrawal, Kayal e Saxena.

5. For $a = 1$ to $\lfloor \sqrt{(\phi(r)}.\log n \rfloor$ do

   If$((X + a)^n \not\equiv X^n + a \ (\text{Mod } X^r - 1, n))$, output COMPOSITE;

Verifica-se a congruência de$(X - a)^n$ com $(X^n - a)$ módulo $X^r - 1$ e módulo n, se para alguma tal que $0 < a \leq \lfloor \sqrt{(\phi(r)}.\log n \rfloor$ não satisfazer a condição então n é composto.

O coração do Algoritmo AKS é a relação em que se o número n a ser testado for primo, então $\forall a$, tal que $0 < a \leq \lfloor \sqrt{(\phi(r)}.\log n \rfloor$, tem-se:

$$(X + a)^n \equiv X^n + a^n \equiv X^n + a \ (\text{Mod } n)$$

A demonstração pode ser encontrada em Farias(2007, p. 3) e é equivalente ao Lema 2.1 em Agrawal, Kayal, Saxena (2004).

Por Agrawal, Kayal, Saxena (2004, p. 3) temos o Teorema 4.1 e o Lema 4.2:

> **Theorem 4.1.** The algorithm above returns PRIME if and only if n is prime. In the remainder of the section, we establish this theorem through a sequence of lemmas. The following is trivial:
> **Lemma 4.2.** If n is prime, the algorithm returns PRIME.
> **Proof.** If n is prime then steps 1 and 3 can never return COMPOSITE. By Lemma 2.1, the for loop also cannot return COMPOSITE. Therefore the algorithm will identify n as PRIME either in step 4 or in step 6.

Ou seja, o algoritmo retornará primo apenas se n for primo. Isto pode parecer pretensão, porém, pela prova os testes de número 1 e 3 nunca irão retornar composto se n for primo. Pelo Lema 2.1 o teste de número 5 garante que se n for primo então a congruência é verdadeira.

7 CONCLUSÕES

Para o funcionamento da Criptografia RSA e outras criptografias assimétricas, são necessários números primos grandes e isto é vital para a segurança deste método. A criação do Algoritmo AKS, ou outros testes de primalidade, não afeta diretamente a segurança deste tipo de criptografia. Concluímos que testes de primalidade para números grandes apenas fornecem mais números primos para este tipo de segurança computacional.

Com a implementação do Algoritmo AKS, pudemos perceber que nosso arquivo criado com 10^{11} caracteres atingiu um tamanho de disco muito superior ao esperado. Logo, o tabelamento de todos os números primos é impraticável, conforme já era esperado. Isto revelou um custo computacional altíssimo, implicando na falha do pré-teste proposto no pré-projeto, sendo que o arquivo criado continha menos de 10^{100} algarismos, o que já citamos que é ultrapassado, mesmo para uso pessoal.

Para o uso comercial o teste de Miller-Rabin é mais utilizado devido a sua velocidade mesmo sendo probabilístico, porém o Algoritmo AKS, por suas características Determinística, Polinomial e Incondicional, apresenta um material extremamente interessante para a matemática.

No passado, os números primos não recebiam tanta atenção, porém, depois que uma utilidade para eles foi descoberta, foi dado um enfoque a esta área, resultando em aprimoramento de testes, novos métodos e uma busca compulsiva por números primos maiores e melhores para satisfazer esta utilidade comercial. Isto nos levou a questionar sobre outras áreas da matemática que poderiam ser exploradas, mesmo que sem fins lucrativos.

Ainda se tratando de números primos, em determinado momento de nossa revisão bibliográfica, eles são chamados de “números mágicos” e realmente eles possuem certa mágica. Estes números mágicos podem possuir diversas utilidades ainda não descobertas, que podem revolucionar as mais diversas áreas da matemática, devido a isto novos estudos são necessários.

BIBLIOGRAFIA

AGRAWAL, M.; KAYAL, N.; SAXENA, N. **PRIMES is in P**, In: Annals of Mathematics, 160, (2004), no. 2, pp. 781–793;

BARBOSA, L. A. M. et al. **RSA:**Criptografia Assimétrica e Assinatura Digital. 2003. 50 f. Monografia (Especialização) – Universidade Estadual de Campinas. Campinas – SP. Disponível em: <www.braghetto.eti.br/files/Trabalho%20Oficial%20Final%20RSA.pdf>. Acesso em: 17 setembro 2015.

COUTINHO, S.C. **Números inteiros e criptografia RSA**. Segunda edição. IMPA, 2000.

BINDER, I. **Algoritmo AKS:** Teoria e implementação. 2008. 25 f. Monografia (Especialização) – Universidade Federal do Paraná. Curitiba – PR. Disponível em: <www.inf.ufpr.br/andre/files/Isis2008.pdf>. Acesso em: 13 outubro 2015.

FARIAS, F. **Uma análise comparativa entre os testes de primalidade AKS e Miller-Rabin**. 2007. 12 f. Trabalho de Conclusão de Curso (Graduação) – Universidade Católica de Brasília. Taguatinga – DF. Disponível em: <www.ucb.br/sites/100/103/TCC/22007/FernandodeFarias.pdf>. Acesso em: 27 setembro 2015.

JUNIOR, C. J. B. **O algoritmo AKS para verificar primalidade em tempo polinomial**. 2010. 49 f. Monografia (Especialização) – Universidade Federal de Lavras. Lavras – MG. Disponível em: <www.repositorio.ufla.br/bitstream/1/5237/1/MONOGRAFIA_O_algoritmo_AKS_para_verificar_primalidade_em_tempo_polinomial.pdf>. Acesso em: 11 outubro 2015.