UNICAMP – Universidade Estadual de Campinas
Especialização em Redes de Computadores

RSA

Criptografia Assimétrica e Assinatura Digital

Luis Alberto de Moraes Barbosa
Luis Fernando B Braghetto (RA 504339)
Marcelo Lotierso Brisqui
Sirlei Cristina da Silva

Campinas, Julho/2003

Índice

ÍNDICE **2**

ÍNDICE DE TABELAS 4
ÍNDICE DE FIGURAS 4

CAPÍTULO 1 - INTRODUÇÃO **5**

CAPITULO 2 – TEORIA DA CRIPTOGRAFIA DE CHAVES PÚBLICAS **6**

2.1 – O QUE É CRIPTOGRAFIA? 6
2.2 – TIPOS DE CRIPTOGRAFIA 6
2.2.1 – Introdução às Técnicas Simples de Criptografia *7*
2.2.1.1 - Cifras de Substituição 7
2.2.1.2 – Cifras de Transposição 8
2.2.2 – Técnicas Modernas de Criptografia: Criptografia Simétrica *9*
2.2.2.1 – DES 9
2.2.2.2 – AES 9
2.2.3 – Técnicas Modernas de Criptografia: Criptografia Assimétrica *9*
2.2.3.1 – Chaves Públicas e Chaves Privadas 10
2.3.3.2 – Diffie-Hellman 12
2.2.3.3 – RSA 13
2.2.3.4 - DSA 13
2.2.3.5 – Rápida Comparação das Utilidades dos Algoritmos 14
2.2.4 – Criptografia Simétrica vs. Criptografia Assimétrica *14*

CAPÍTULO 3 – O ALGORITMO RSA **16**

3.1 - HISTÓRICO RSA 16
3.2 – ALGORITMO RSA 16
3.3 – ASPECTOS COMPUTACIONAIS DO RSA 19
3.4 – INTRODUÇÃO SIMPLES À TEORIA DOS NÚMEROS 20
3.4.1 –Módulo: A mod B *20*
3.4.2 – Calculando $C = A^b \bmod n$ *21*
3.4.3 – Números Primos. *21*
3.4.3.2 – Primos entre si 23
3.4.3.2 – Descobrindo e provando primos 23

CAPÍTULO 4 – UTILIZAÇÃO PRATICA, FORMATAÇÃO E GERENCIAMENTO DE CHAVES NO RSA. **24**

4.1 – APLICAÇÕES QUE UTILIZAM O RSA 24
4.1.1 – Certificados de Segurança *24*
4.1.2 – Assinatura Digital *26*
4.1.3 – S/Mime e PGP *28*
4.1.4 – SSL/TLS *28*
4.1.5 – IPSEC *29*

4.2 – PADRÕES E FORMATOS DOS CERTIFICADOS DIGITAIS 29
4.2.1 – Recomendação X.509 .... *30*
4.2.2 – PKIX *30*
4.2.3 – ASN.1 .... *32*
4.2.3.1 – BER 32
4.2.3.2 – DER 33
4.2.2 – PKCS *33*
4.3 – GERENCIAMENTO DE CHAVES NO RSA 34
CAPÍTULO 5 – SEGURANÇA DO RSA 35
5.1 – INTRODUÇÃO 35
5.2 – ATACANDO O RSA 35
5.2.1 – Força Bruta *35*
5.2.2 – Ataques Matemáticos *36*
5.2.2.1 – Problema da Fatoração 36
5.2.2.2 – Calcular $\oint n$ sem fatorar n 38
5.2.2.3 – Expoente de Decriptação 38
5.2.2.4 – Calcular d sem possuir $\oint n$ 39
5.2.3 – Ataques Temporais *39*
5.2.4 – Outros tipos de Ataques *40*
5.2.4.1 – Expoente d pequeno para decriptografia 40
5.2.4.2 – Expoente e pequeno para criptografia 40
5.2.4.3 – Ataque de Módulo Comum 40
5.2.4.4 – Criptoanálise do RSA se d for menor que $n^{0,292}$ 40
5.2.4.5 – Ataque de Broadcast de Hastad 40
5.2.4.6 – Ataque de Exposição parcial da chave privada 41
5.2.4.7 – Ataque de Bleichenbacher no PKCS 1 41
CAPITULO 6 – COMPARATIVOS ENTRE ALGORITMOS 42
6.1 – SIMÉTRICA X ASSIMÉTRICA 42
6.2 – ALGORITMOS E TABELAS COMPARATIVAS 42
6.3 – PERFORMANCE 45
6.3.1 – RSA *45*
6.3.2 – DSA *45*
6.3.3 – ECC *45*
6.3.4 - RSA x Curvas Elípticas *46*
CONCLUSÃO 48
BIBLIOGRAFIA 49
LIVROS: 49
TESE: 49
PAPERS: 49
INTERNET SITES: 50

Índice de Tabelas

Tabela 1: Comparação das Utilidades dos Algoritmos..14
Tabela 2: Comparação Criptografia Simétrica vs. Assimétrica...15
Tabela 3: Teste de Mesa para algoritmo a^b mod n..20
Tabela 4: Tempo de Fatoração no RSA..37
Tabela 5: Comparação de Algoritmos Simétricos e Assimétricos 42
Tabela 6: Problemas Computacionais x Algoritmos 42
Tabela 7: Esquemas de Assinatura relacionados ao RSA 43
Tabela 8: Descrição dos Problemas, suporte a criptoanálise e patente no mundo 44

Índice de Figuras

Figura 1: Esquema de Criptografia Assimétrica para garantir sigilo 10
Figura 2: Esquema de Criptografia Assimétrica para assinatura digital..............................11
Figura 3: Esquema de Criptografia Assimétrica para para sigilo e assinatura digital..........12
Figura 4: Algoritmo para cálculo de a^b mod n..19
Figura 5: Esquema de Assinatura Digital no RSA 27
Figura 6: Formato do Padrão X509 31
Figura 7: Comparação do certificado X509 e CRL 32

Capítulo 1 - Introdução

“*A origem da criptografia, provavelmente, remonta aos princípios da existência humana, logo que as pessoas tenham tentado aprender a comunicar. Conseqüentemente, tiveram de encontrar meios para garantir a confidencialidade de parte das suas comunicações*.” - Pierre Loidreau, escreveu um artigo na primeira revista Linux da França

Esta frase expressa o significa inicial de criptografia e, aos dias de hoje, podemos notar que uma grande mudança nos métodos e maneiras de se criptografar.

Este documento irá prover informações sobre uma das mais bem sucedidas implementações de algoritmos de criptografia e assinatura até hoje, o RSA.

Conforme o leitor irá adiantando a leitura, irá saber sobre o que é e o que significa a criptografia, criptoanalista e afins; também entenderá métodos simples de criptografia até chegar ao RSA, que é o foco deste documento. Apesar disto, conceitos de chaves privadas, públicas, outros algoritmos, performance, etc não serão esquecidos.

A idéia é dar ao leitor leigo, uma noção geral sobre como funciona a criptografia hoje, assim como o RSA e explicando até mesmo em contas para detalhar tal algoritmo. Esperamos que ajude a todos interessados em criptografia.

O GRUPO

Capitulo 2 – Teoria da Criptografia de Chaves Públicas

2.1 – O que é criptografia?

Como já vimos, criptografar é baseado nas dificuldades em resolver problemas difíceis. Criptoanalistas são pessoas que estudam como comprometer (ou desvendar) os mecanismos de criptografia, e a criptologia são o resultado do estudo da criptografia pelos criptoanalistas.

Criptografia vem da palavra grega kryptos ("escondida") e graphia ("escrever"). Criptografar significa transformar uma mensagem em outra ("escondendo" a mensagem original), usando para isso funções matemáticas fazendo com que seja impossível (ou quase) que uma pessoa sem o conhecimento de como aquilo foi gerado consiga desvendar a mensagem e descobrir o texto original.

Criptografar é o ato de pegar uma mensagem e embaralhar usando uma senha especial (chave) os dados de forma que a saída não faça sentido aparente a um criptoanalista. Decriptografar é o ato de pegar a mensagem cifrada e com o uso de uma senha (chave) ela possa ser revertida para o texto original.

Criptografar e decriptografar exigem em algum momento algo que somente os interessados possam saber, que no nosso caso é chamado de chave. A chave pode ser um texto qualquer ou uma informação que não pode ser divulgada a ninguém.

Atualmente criptografia é mais do que isso, pois além de guardar informações de pessoas não autorizadas, ainda existe a Autenticação (para saber quem gerou esse documento).

Uma das definições formais para "Criptografia" é o estudo das técnicas matemáticas relacionadas para aspectos de segurança da informação, tais como confidencialidade, integridade dos dados, autenticação de entidades e verificação da origem. [*Handbook Applied Criptology*]

2.2 – Tipos de criptografia

Existem dois tipos de criptografia, cada uma com características, vantagens e desvantagens: Criptografia Simétrica e Criptografia Assimétrica

Criptografia simétrica é basicamente a criptografia onde é usada apenas 1 chave para cifrar e decifrar o texto. Criptografia Assimétrica é a criptografia onde é usada 1 chave para ciframento e uma outra para deciframento.

2.2.1 – Introdução às Técnicas Simples de Criptografia

Os métodos de criptografia têm sido divididos em duas categorias: as cifras de substituição e as de cifras de transposição. Estas são utilizadas na criptografia simétrica. A criptografia simétrica em alguns locais também é referida como "criptografia de chave secreta" (*secret-key criptography*).

2.2.1.1 - Cifras de Substituição

Este sem dúvida é o jeito mais fácil de cifrar (e também de decifrar). As cifras de substituição trocam uma letra por outra letra correspondente. Vejamos um exemplo onde nós vamos trocar cada uma das 26 letras do *abecedário* pelas letras na ordem do teclado.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Q	W	E	R	T	Y	U	I	O	P	A	S	D	F	G	H	J	K	L	Z	X	C	V	B	N	M

Este tipo de substituição é chamado de monoalfabética.

Para exemplificar vamos cifrar a palavra "palavra" com nossa cifra de substituição:

p	a	l	a	v	r	a
H	Q	S	Q	C	K	Q

Por fim, a palavra "palavra" seria escondida como "hqsqckq". Quem recebesse isso bastaria voltar atrás para descobrir a palavra original "palavra".

Qual a probabilidade de combinações possíveis para esta cifra? A resposta é 26! (fatorial de 26) que é 26 . 25 . 24 . 23 .... 3 . 2 . 1 que dá $4{,}032 \times 10^{26}$ diferentes dicionários... porém o ataque pode ser feito através de analise de freqüência.

Em qualquer língua, alguns sons são utilizados com mais freqüência do que outros. Isto significa que, na linguagem escrita, algumas letras também são mais utilizadas que outras. Determinar a freqüência com que ocorrem determinadas letras em determinada língua, ou seja, fazer uma análise da freqüência de ocorrência de letras, não é nenhuma novidade. O grande sábio árabe al-Kindi já teve esta idéia há mais de 1.000 anos atrás.

Na substituição monoalfabética, cada letra é trocada pela letra correspondente da chave. Isto significa que as características das letras originais são transferidas para as novas letras, inclusive suas características de freqüência de ocorrência. É o mesmo que trocar seis por meia dúzia... o caminho das pedras para quebrar a cifra! Uma análise de freqüência do alfabeto português e uma amostra razoável de texto cifrado mostra a facilidade de quebrar isso. (http://www.numaboa.com/criptologia/matematica/estatistica/freqPortBrasil.php).

Os algoritmos ROT13 e César usam isso. Depois algoritmos como Playfair e Hill adicionaram alguma dificuldade fazendo substituições através de grupos de letras, mas ainda é muito fácil ser quebrada.

2.2.1.2 – Cifras de Transposição

As cifras de transposição usam como técnica a mudança da ordem das letras. Vejamos um exemplo onde vamos aplicar uma chave a um texto para efetuar uma mudança na ordem do texto. O texto original será “a ponte de Londres está caindo” e a chave será “viagem”

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Vamos colocar a palavra “viagem” em ordem numérica. O resultado será:

Ordem original	1	2	3	4	5	6
Chave	V	I	A	G	E	M
Valor Numérico das letras	22	9	1	7	5	13
Nova Ordem	6	4	1	3	2	5

V	I	A	G	E	M
6	4	1	3	2	5
a		p	o	n	t
e		d	e		L
o	n	d	r	e	s
	e	s	t	a	
c	a	i	n	d	O

E o resultado será as colunas na ordem (1, depois 2, e assim por diante) e o texto cifrado será “pddsin ead oertn nea tLs Oaeo c”

O que se pode fazer é misturar transposição com substituição, porém ainda é ineficiente já que hoje se tem muito poder computacional.

2.2.2 – Técnicas Modernas de Criptografia: Criptografia Simétrica

2.2.2.1 – DES

O DES – Padrão de Ciframento de Dados (*Data Encription Standart*) foi o primeiro modelo de criptografia simétrica na época moderna (criado na década de 70).

O DES processa blocos de texto de 64 bits cada vez, usando uma chave de 56bits, produzindo um texto cifrado de 64bits. O DES para causar um efeito mais interessante faz este procedimento 16 vezes, cada uma usando uma porção diferente da chave.

Entretanto antigamente o DES era extremamente seguro, porém com o aumento significativo do poder computacional nas mãos dos criptoanalistas, o DES tornou-se inseguro. Recentemente o DES conseguiu ser quebrado em pouco menos de 1 dia com uma chave de 56bits.

Atualmente o DES possui uma versão mais fortalecida composto de três chaves de 56bits (168bits no total) e foi chamado de 3-DES.

2.2.2.2 – AES

O AES – Padrão Avançado de Ciframento (*Advanced Encryption Standart*) é um algoritmo simétrico que foi a resposta à requisição de um novo algoritmo de criptografia pela NIST – Instituto Nacional (Americano) de padrões e tecnologia (*U.S. National Institute of Standards and Technology*). O AES é um algoritmo simétrico que pode usar chaves de 128, 192 ou 256 bits com blocos de dados de 128 bits.

Em 2001, o AES virou um padrão reconhecido pelo NIST depois de vencer a batalha em cima de outros algoritmos (MARS (IBM), RC6 (RSA Labs), Rijndael (Hoan Daemen e Vicent Rijmen), Serpent (Ross Anderson, Eli Biham, Lars Knudsen) e Twofish (Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall e Niels Ferguson))

2.2.3 – Técnicas Modernas de Criptografia: Criptografia Assimétrica

Criptografia Assimétrica, como já foi dito, usa uma chave para ciframento e uma outra para deciframento. Desde o inicio, toda a criptografia era baseada em transposições e substituições, e a criptografia assimétrica foi uma verdadeira revolução para a criptografia moderna.

Algumas idéias devem ser levadas em conta antes de prosseguirmos. Criptografia assimétrica não é mais ou menos seguro que a criptografia simétrica. Cada uma tem utilidades diferentes. O que define a segurança a não permitir a criptoanálise é o tamanho da chave. Quanto maior a chave, mais difícil é criptoanalisar o dado.

O que todos devem estar se perguntando é se existe a criptografia de chaves públicas porque ainda usar a criptografia simétrica? A criptografia assimétrica exige muito mais processamento do que a criptografia simétrica. Na vida real a criptografia assimétrica é usada para combinar uma chave que será usada posteriormente por uma criptografia simétrica, ou no caso de assinaturas digitais é feito um *hash* da mensagem e a criptografia acontece no *hash* para diminuir o *overhead*.

2.2.3.1 – Chaves Públicas e Chaves Privadas

Um novo conceito é lançado baseado que existem duas chaves: Chave Pública e Chave Privada. Mas o que faz cada uma das chaves? Depende.

Uma única definição é certa e correta: a chave privada não pode sair da mão do dono do par de chaves. Somente a chave pública pode ser distribuída.

Se a finalidade do algoritmo assimétrico é ciframento dos dados, impedindo que outros não possam saber o que tem dentro da mensagem, somente o destinatário correto, a chave pública será uma chave que servirá para a criptografia dos dados e a chave privada será a chave para decriptografia (deciframento) da mensagem previamente cifrada.

Neste caso, vamos exemplificar que Alice quer enviar uma mensagem para Bob, mas somente Bob poderá lê-la. Alice irá até Bob através de um canal inseguro qualquer e requisitará a chave pública de Bob. Alice irá pegar a mensagem “M” e aplicar a chave pública (KU_p) de Bob usando um algoritmo conhecido de todos. Somente Bob tem a chave privada para decifrar a mensagem “C” (Mensagem C é o resultado da mensagem M após aplicar KU_p). Para tanto Bob pega o algoritmo conhecido e aplica a sua chave privada (KR_b) para obter a mensagem “M” novamente. Neste caso a chave pública faz o ciframento e a chave privada faz o deciframento.

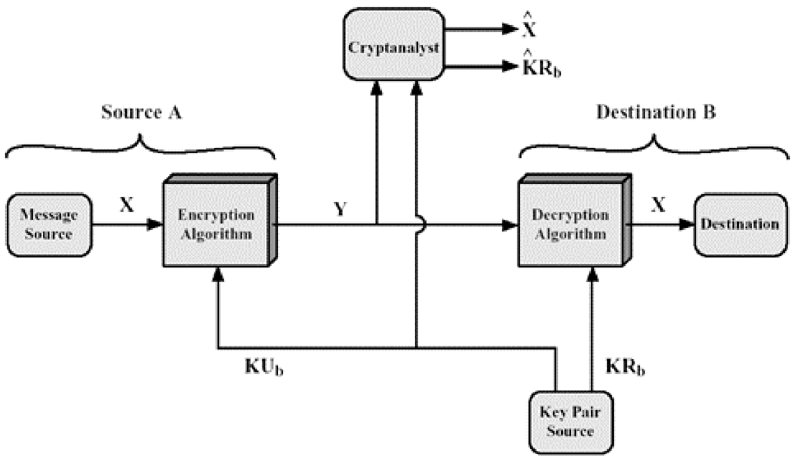


Figura 1: Esquema de Criptografia Assimétrica para garantir sigilo

Se a finalidade for assinatura digital, entende-se que somente a pessoa de posse da chave privada poderá criar a mensagem, impedindo o repúdio e garantindo a autoria e autenticação da mensagem e do autor.

Neste caso, vamos exemplificar que Bob pegará uma mensagem “M” (não cifrada, como por exemplo “vou pagar 1000 reais para Alice”) e aplica a sua chave privada (KR_b). A chave privada irá fazer a criptografia, ou seja, gerar a mensagem “C”. Sendo a chave pública de Bob (KU_b) conhecida de todos, qualquer um poderá decriptografar a mensagem. Se a chave pública de Bob foi capaz de gerar novamente a mensagem “M”, ele e somente ele (Bob) poderia ter gerado a mensagem, garantindo a autoria e autenticação do autor. Quando se distribui a mensagem deste jeito, não se garante a confidencialidade dos dados já que a chave KU_b é pública e qualquer um poderá ler isto. Isto é chamado de assinatura digital[1] (ver capítulo 4).

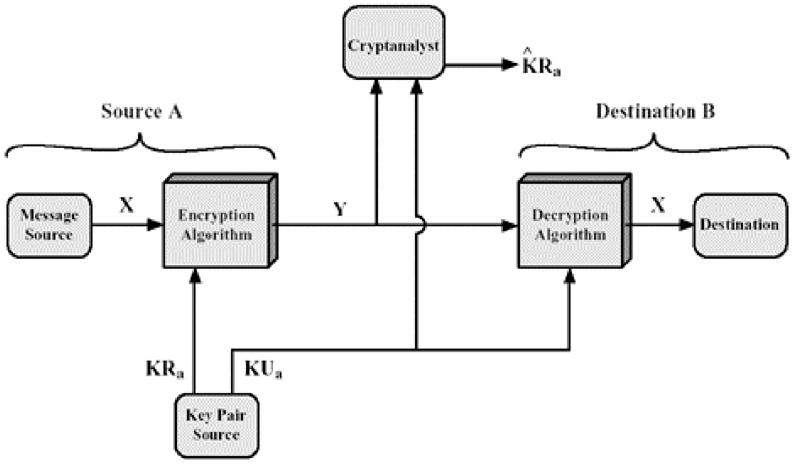


Figura 2: Esquema de Criptografia Assimétrica para assinatura digital

E se nós quiséssemos garantir que a mensagem veio de alguém e o sigilo fosse importante? Simples, basta usarmos dois pares de chaves públicas e privadas... lembrando que se distribuirá uma para criptografar (sigilo) e uma para decriptografar (assinatura digital). Lembrando que devemos cifrar primeiro e assinar depois, pois assim poderemos verificar a assinatura sem sabermos o conteúdo.

[1] Na assinatura digital é assinado um *hash* da mensagem ao invés da mensagem toda por questões de performance.

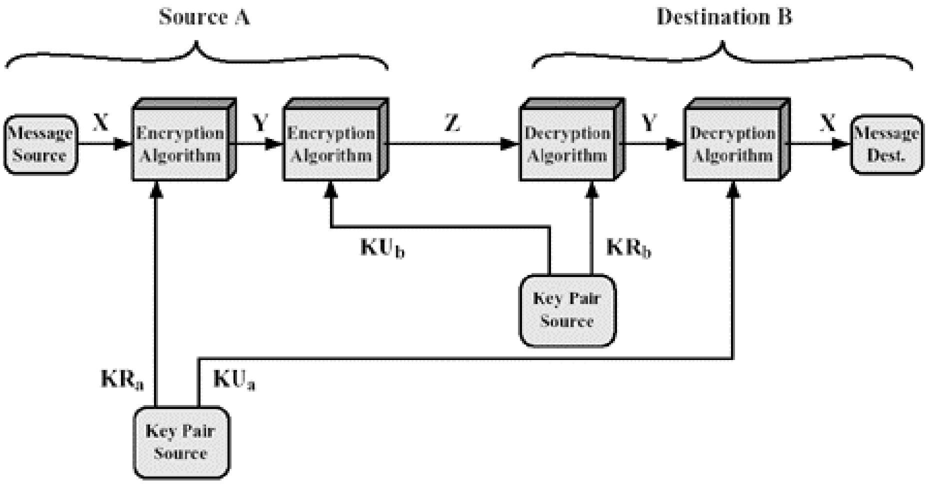


Figura 3: Esquema de Criptografia Assimétrica para para sigilo e assinatura digital

2.3.3.2 – Diffie-Hellman

Diffie-Hellman é um protocolo para troca de chaves usando meios inseguros, tais como a Internet, chamado também de acordo de chaves exponenciais (exponential key agreement) e foi desenvolvido por Whitfield Diffie e Martin Hellman em 1976, públicado em um documento chamado "As Novas Direções da Criptografia". O protocolo permite que dois usuários troquem chaves secretas em um meio inseguro.

O protocolo tem dois parâmetros: *p* e *g*. Ambos (*p* e *g*) podem ser públicos e ser usados por todos em uma rede. O parâmetro *p* é um primo qualquer e o parâmetro *g* (chamada de chave geradora) é um número inteiro menor que *p*, com o seguinte requisito: para cada número *n* entre 1 e *p* inclusive, existe um expoente k de g tais que $n = g^k \bmod p$.

Vamos supor que Bob e Alice querem concordar em uma nova chave usando Diffie-Hellman. Primeiro Alice gera um número aleatório *a*, e Bob gera um número aleatório *b*. Ambos a e b estão em um set de inteiros {1,...,p-2}. Então eles calculam *p* e *g* de suas chaves. O valor público de Alice é $g^a \bmod p$, enquanto o de Bob é $g^b \bmod p$. Agora eles trocam suas chaves públicas. Por fim Alice calcula $g^{ab} = (g^b)^a \bmod p$ e Bob calcula $g^{ba} = (g^a)^b \bmod p$. Já que $g^{ab} = g^{ba} = k$, então Alice e Bob tem sua chave secreta simétrica k.

Diffie-Hellman original é vulnerável ao ataque "*man-in-the-middle*"[2].

Em 1992, o protocolo STS foi desenvolvido por Diffie, Oorschot e Wiener para não sofrer mais o ataque "*man-in-the-middle*". A imunidade a este ataque foi alcançada possibilitando que duas partes autentiquem eles mesmos através de assinaturas digitais. A idéia é que antes de iniciar, A e B obtenham um par de chaves públicas e privadas. Durante o protocolo, A calcula a assinatura em algumas mensagens, escondendo o valor de $g^a \bmod p$.

[2] Neste tipo de ataque Pedro (P) negocia um valor com Alice (A) e um outro com Bob (B). Desta maneira ele traduz 100% das mensagens A – B, fazendo A – P / P – B e B – P / P – A.

2.2.3.3 – RSA

RSA foi criado por R. Rivest, A. Shamir e L. Adleman em 1977. Foi o primeiro algoritmo a usar a técnica Diffie-Hellman, usando criptografia assimétrica. O RSA é muito popular até hoje e sua segurança advém da dificuldade de fatorar números inteiros muito grandes.

Para garantir a segurança o RSA utiliza números primos p e q com média de 300 dígitos.

Detalhes sobre o algoritmo do RSA serão visto no capítulo 3.

2.2.3.4 - DSA

DSA é o acrônimo de Padrão de Assinatura Digital (*Digital Signature Standart*), criado pelo NIST, e especifica o DSA para assinatura digital e SHA-1 para *hash*ing.

O DSA é um algoritmo assimétrico e a chave privada opera sobre o *hash* da mensagem SHA-1. Para verificar a assinatura um pedaço do código calcula o *hash* e outro pedaço usa a chave pública para decifrar a assinatura, e por fim ambos comparam os resultados garantindo a autoria da mensagem.

O DSA trabalha com chaves de 512 à 1024 bits, porém ao contrário do RSA que é multipropósito, o DSA somente assina e não garante confidencialidade. Outro ponto contra o DSA é que a geração da assinatura é mais rápida do que o RSA, porém de 10 a 40 vezes mais lenta[3] para conferir a assinatura.

O algoritmo do DSA é o seguinte para geração das chaves

1) Selecionar um primo q tal que $2^{159} < q < 2^{160}$.

2) Selecionar t tal que $0 <= t <= 8$, e selecionar um primo p tal que $2^{511+64t} < p < 2^{512+64t}$ com a propriedade que q seja divisível por (p-1).

3) Selecionar um gerador a de um grupo de ordem q em Z^*_p, e selecionar um elemento g que pertença a Z^*_p, e calcular $a = g^{(p-1)/q} \bmod p$.

4) Selecionar um número aleatório x tal que $1 <= x <= q – 1$

5) Calcular $y = a^x \bmod p$.

6) Chave pública de A é (p,q,a,y). Chave Pública é x.

Para gerar assinatura o DSA usa os seguintes passos:

1) Seleciona um número aleatório secreto k, onde $0 < k < q$.

2) Calcula $r = a^k \bmod p$

3) Calcula $k^{-1} \bmod q$

4) Calcula $s = k^{-1}\{h(m) + xr\} \bmod q$. (h(m) é o *hash* da mensagem)

5) Assinatura de A para m são r e s.

[3] segundo *Applied Cryptography*, 2nd Ed, Bruce Schneier

Para verificar a assinatura de A (*r* e *s*), B deve usar os seguintes passos:

1) Obtem a chave pública de A (p,q,a,y)
2) Verifica que ($0 < r < q$) e ($0 < s < q$). Se não, rejeita assinatura.
3) Calcula $w = s^{-1} \bmod q$ e calcula h(m) (h(m) é o *hash* da mensagem)
4) Calcula $u_1 = w.h(m) \bmod q$ e $u_2 = rw \bmod q$
5) Calcula $v = (a^{u1}\ y^{u2} \bmod p) \bmod q$
6) Aceita assinatura se v = r.

2.2.3.5 – Rápida Comparação das Utilidades dos Algoritmos

Dentre a criptografia de chaves públicas, veremos abaixo algumas das finalidades destes algoritmos:

- Ciframento/Deciframento (Criptografia): O destinatário e o rementente desejam conversar com privacidade
- Assinatura Digital: Quando se precisa provar a autoria de uma mensagem.
- Troca de Chaves: Dois lados cooperam para trocar a chave que será usada em uma sessão de transmissão de dados.

Algoritmo	Criptografia	Assinatura Digital	Troca Chaves
RSA	✓	✓	✓
Diffie-Hellman			✓
DSA		✓	

Tabela 1: Comparação das Utilidades dos Algoritmos

Pode-se perceber que o RSA é o algoritmo que permite flexibilidade e por isso ele continua sendo o principal algoritmo de criptografia de chaves públicas há 20 anos.

2.2.4 – Criptografia Simétrica vs. Criptografia Assimétrica

A criptografia simétrica foi uma revolução da criptografia moderna. Todos devem pensar: Porque usar a criptografia simétrica diante das inúmeras vantagens da criptografia assimétrica?

Basicamente pela dificuldade de gerar as chaves de maneira segura e problemas relativos à performance. Estes problemas serão abordados com profundidade no capítulo 6.

Alguns itens também devem ser conhecidos para diferenciar a criptografia simétrica da assimétrica:

	Criptografia Simétrica	**Criptografia Assimétrica**
Funcionamento	O mesmo algoritmo é usado para criptografar e decriptografar a mensagem	O mesmo algoritmo é usado para criptografar e decriptografar a mensagem, porém usando duas chaves.
Requer	Que destino e origem saibam o algoritmo e a chave	A origem e o destino devem saber uma (somente uma) chave do par de chaves. Todos podem ter a chave pública, porém só 1 deve saber a chave privada.
Segurança	A chave deve ser mantida em segredo Mesmo sabendo o algoritmo e tendo exemplos dos textos criptografados deve impossibilitar a determinação da chave.	Apenas 1 das duas chaves deve ser mantida em segredo É impossível decifrar uma mensagem mesmo tendo acesso ao algoritmo, à chave pública e a exemplos dos textos cifrados. Tendo a chave pública deve ser impossível chegar na chave privada.
Utilidade	• Privacidade	• Identificação • Assinatura Digital • Privacidade • Troca de Chaves • (muitas utilidades)
Velocidade de Processamento	Muito Rápida	Lenta
Segurança	Alta	Alta
Chaves	Apenas 1	2 Chaves (Pública e Privada)

Tabela 2: Comparação Criptografia Simétrica vs. Assimétrica

Capítulo 3 – O Algoritmo RSA

3.1 - Histórico RSA

Em 1976, Whitfield Diffie e Martin Hellman escreveram um documento chamado "As Novas Direções da Criptografia" mostrando a idéia de usar a criptografia de chaves públicas.

Logo após, os criptologistas começaram a tentar desenvolver um algoritmo que pudesse atender as especificações propostas por Diffie-Hellman.

O RSA foi desenvolvido em 1978 em resposta a essa necessidade no MIT por Ron Rivest, Adi Shamir e Len Adleman (daí a sigla RSA). O algoritmo do RSA foi o primeiro algoritmo de chaves públicas é amplamente usado desde então.

3.2 – Algoritmo RSA

O RSA é basicamente o resultado de dois cálculos matemáticos. Um para cifrar e outro para decifrar. O RSA usa duas chaves criptográficas, uma chave pública e uma privada. No caso da criptografia assimétrica tradicional, a chave pública é usada para criptografar a mensagem e a chave privada é usada para decriptografar a mensagem.

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 tranqüilidade por algum tempo. Em níveis críticos, chaves com 2000 bits começam a ser usadas.

Para tanto vale lembrar que "M" é a mensagem que queremos cifrar (plaintext), "C" é a mensagem cifrada, "e" é a chave pública, "d" é a chave privada e "n" é um número que é calculado e que todos sabem (público).

Criptografar: $C = M^e \mod n$

Decriptografar: $M = C^d \mod n$

Para cada bloco a ser cifrado deve-se fazer o cálculo acima. Ambos devem saber o valor de "n". Portanto, a chave pública definida pela dupla "e" e "n", sendo $KU_a = \{e, n\}$. A chave privada é definida pela dupla "d" e "n", sendo $KR_b = \{d, n\}$.

Para gerar a chave precisamos de algumas coisas:

1. Selecionar dois números primos p e q grandes (geralmente maior que 10^{100}). (veja próxima seção para ver como achar números primos)
2. Calcule o valor de n = p . q
3. Calcule $fn = (p-1)\cdot(q-1)$
4. Selecione um inteiro "d" relativamente primo à fn.
5. Calculamos "e" de forma que (e . d) mod fn = 1

Vejamos um exemplo:

1. p = 3 e q = 11
2. n = 3 * 11, logo n = 33
3. $fn = (3-1)\cdot(11-1) = 2\cdot 10$, portanto fn=20
4. d é um inteiro relativamente primo à fn, e atende 1 < d < fn
   d = 7
   (9, 11, 13, 15, 17 e 19 seriam outras opções)
   (não seria possível 5 já que 5 vezes 4 = fn(20))
5. Calculamos "e" de forma que (e . 7) mod 20 = 1
   e = 1 => (1.7) = 7 mod 20 $\neq$ 1 => falso
   e = 2 => (2.7) = 14 mod 20 $\neq$ 1 => falso
   e = 3 => (3.7) = 21 mod 20 = 1 => verdadeiro
   (outros múltiplos de 3 seriam possíveis (6,9,12, etc)).

Portanto teríamos KU = {3, 33} e KR = {7, 33}. Lembrando que neste caso *e*, *d* e *n* tem menos de 2^6, então temos apenas 6 bits.

Se tivéssemos um texto com o número 20, uma mensagem cifrada seria:
C = M^e mod n
C = 20^3 mod 33
C = 8000 mod 33
C = 14

E para decifrar:
M = C^d mod n
M = 14^7 mod 33
M = 105.413.504 mod 33 (resposta = 3.194.348 x 33 + 20 = 105.413.504)
M = 20

No próximo exemplo, vamos tentar com outros números:

1. p = 7 e q = 17
2. n = 7 . 17 = 119
3. $fn = (7-1)\cdot(17-1) = 6 \cdot 16$, portanto fn =96
4. d = mmc(fn,d) = 1, tal que (1 < d < fn)
   (se máximo múltiplo comum é 1, então são primos entre si)
   d = 77
5. (e . d) mod fn = 1
   (e . 77) mod 96 = 1
   logo e = 5 => 77 x 5 = 385 = 4 x 96 + 1

Portanto
KU = {5,119} (publico)
KR = {77,119} (privado)

Criptografando o número (entrada não cifrada) "19":
M = 19
C = M^e mod n
C = 19^5 mod 119
C = 2.476.099 mod 119
C = 66 (porque 20.807 x 119 + 66 = 2.476.099)

Decriptografando
C = 66
M = C^d mod n
M = 66^{77} mod 119
M = 1,27316015 x 10^{140} mod 119
M = 19

E o último exemplo acontecerá com um número um pouco maior:

1. p = 2357 e q = 2551
2. n = p.q = 6012707
3. fn = (2357-1)(2551-1) = 6007800
4. d = mmc(fn,d) = 1, tal que (1 < d < fn), então d = 3674911
5. (e . d) mod fn = 1, então e = 422191

Então KU = {3674911,6012707} e KR = {422191,6012707}

Para criptografar a mensagem M = 5234673
C = M^e mod n
C = $5234673^{3674911}$ mod 6012707
C = 3650502

Para decriptografar a mensagem C = 3650502
$M = C^{d} \bmod n$
$M = 3650502^{422191} \bmod 6012707$
M = 5234673

Agora nossa chave pública (3.674.911) tem somente 23 bits ($2^{23} = 8388608$), imaginem com 512 bits ou 1024 bits o tamanho dos cálculos. Isto expressa o grande overhead do protocolo RSA.

Entende-se como um mínimo de segurança sendo *p* e *q* números com 512 bits e *n* tendo aproximadamente 1024 bits.

3.3 – Aspectos Computacionais do RSA

Criptografia e decriptografia do RSA envolvem muita exponenciação de inteiros e módulos de n. Porém exponenciação é algo que se leva tempo linear (x passos para um exp oente x)

$$x^{32} = x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x$$

Porém pode-se resolver rapidamente em tempo $\log_2(x)$. Para isso basta a cada multiplicação, faze-la por ela mesma, formando sucessivamente x^2, x^4, x^8, x^{16}, x^{32} (5 passos ao invés de 32). Se x for um valor não múltiplo de 2, então basta fazer até o maior múltiplo antes de x, e depois multiplicações sucessivas. Por exemplo x^{40}, então faríamos $x^{32} * x * x * x * x * x * x * x * x$, ou ainda $(x^{32} * x^8)$.

O algoritmo para calcular $a^b \bmod n$ é:

```
c ← 0;
d ← 1;
k ← len(bits(b))          ; tamanho de b em binário.
for i ← k downto 0 do
{
        c ← 2 * c
        d ← (d * d) mod n
        if bi = 1 then
        {
                c ← c + 1
                d ← (d * a) mod n
        }
}
return d
```

Figura 4: Algoritmo para cálculo de a^b mod n

i	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
d	7	49	157	526	160	241	298	166	67	1

Tabela 3: Teste de Mesa para algoritmo a^b mod n

Resultado para a^b mod n, onde a = 7, b = 560 (1000110000), n = 561

3.4 – Introdução Simples à Teoria dos Números

3.4.1 –Módulo: A mod B

Nós dizemos que C=A mod B, sendo C o resto da divisão entre A e B.
A função inversa é A=x*B + C, onde x é o resultado inteiro da divisão.

Por exemplo, para C=A mod B, onde A = 20 e B = 8:

20 / 8 = 2 e o resto é 4 (2*8=16+4=20), logo C = 20.

Lembrando que c=a em c=a mod b onde a > b, ou seja, se a > b então o resultado de a mod b é o valor de a. Exemplo: 5 mod 20 = 5.

Agora, são chamados de "*congruentes módulo n*" quando dois números inteiros a e b tem a seguinte expressão verdadeira (a mod n) = (b mod n). A notação para isso é:

$$a \equiv b \bmod n$$

Por exemplo:

a = 31 e b = 41, onde n = 5.
31 mod 5 = 1 e 41 mod 1 = 1

portanto 31 ≡ 41 mod 5.

Outro exemplo:
73 ≡ 4 mod 23 pois:

73 mod 23 = 4 (porque 3 * 23 = 69 + 4 = 73)
4 mod 23 = 4 (porque 0 * 23 = 0 + 4 = 4, e também a>b, então c=a)

3.4.2 – Calculando $C = A^b$ mod n

Como calcular facilmente um "$C = M^e \ mod \ n$" sem uso de ferramentas (talvez usando só uma calculadora simples)?

Vejam por exemplo para M = 30, e = 13 e n = 35:

Passo 0: $C = 30^{13}$ mod 35
Passo 1: $C = (30^{12}) * 30$ mod 35
Passo 2: $30^{12} = (30^6)^2$
Passo 3: $30^6 = (30^3)^2$
Passo 4: $30^3 = (30^2) * 30$
Passo 5: $C = (30^2 * 30)$ mod 35

Agora resolveremos de baixo para cima
Passo 5: $C = 30^2$ mod 35 = 900 mod 35 = 25
(já que 25 x 35 = 875+25 = 900)
Passo 4: C = 25 * 30 mod 35 = 750 mod 35 = 15
(antes era $30^2 * 30$ mod 35)
Passo 3: $C = 15^2$ mod 35 = 225 mod 35 = 15
(antes era $30^6 = ((30^2)*30^{(1)})^2$ mod 35)
Passo 2: $C = 15^2$ mod 35 = 225 mod 35 = 15
(antes era $30^{12} = (30^6)^2$ mod 35)
Passo 1: C = 15 * 30 mod 35 = 450 mod 35 = 20

Portanto $C = 30^{13}$ mod 35 = 20

3.4.3 – Números Primos.

Números primos são números (p) que somente são divisíveis por 1 ou por ele mesmo (na verdade ± 1 e $\pm$p). Portanto para um primo ele deve ter p mod z $\neq$ 0 onde 1 < z < p (não incluso 1 e p), para $z \in$ Z* (inteiros positivos).

Os números primos menores que 3000 são:

2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229
233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349
353	359	367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523	541
547	557	563	569	571	577	587	593	599	601
607	613	617	619	631	641	643	647	653	659
661	673	677	683	691	701	709	719	727	733
739	743	751	757	761	769	773	787	797	809
811	821	823	827	829	839	853	857	859	863
877	881	883	887	907	911	919	929	937	941
947	953	967	971	977	983	991	997	1009	1013
1019	1021	1031	1033	1039	1049	1051	1061	1063	1069

1087	1091	1093	1097	1103	1109	1117	1123	1129	1151
1153	1163	1171	1181	1187	1193	1201	1213	1217	1223
1229	1231	1237	1249	1259	1277	1279	1283	1289	1291
1297	1301	1303	1307	1319	1321	1327	1361	1367	1373
1381	1399	1409	1423	1427	1429	1433	1439	1447	1451
1453	1459	1471	1481	1483	1487	1489	1493	1499	1511
1523	1531	1543	1549	1553	1559	1567	1571	1579	1583
1597	1601	1607	1609	1613	1619	1621	1627	1637	1657
1663	1667	1669	1693	1697	1699	1709	1721	1723	1733
1741	1747	1753	1759	1777	1783	1787	1789	1801	1811
1823	1831	1847	1861	1867	1871	1873	1877	1879	1889
1901	1907	1913	1931	1933	1949	1951	1973	1979	1987
1993	1997	1999	2003	2011	2017	2027	2029	2039	2053
2063	2069	2081	2083	2087	2089	2099	2111	2113	2129
2131	2137	2141	2143	2153	2161	2179	2203	2207	2213
2221	2237	2239	2243	2251	2267	2269	2273	2281	2287
2293	2297	2309	2311	2333	2339	2341	2347	2351	2357
2371	2377	2381	2383	2389	2393	2399	2411	2417	2423
2437	2441	2447	2459	2467	2473	2477	2503	2521	2531
2539	2543	2549	2551	2557	2579	2591	2593	2609	2617
2621	2633	2647	2657	2659	2663	2671	2677	2683	2687
2689	2693	2699	2707	2711	2713	2719	2729	2731	2741
2749	2753	2767	2777	2789	2791	2797	2801	2803	2819
2833	2837	2843	2851	2857	2861	2879	2887	2897	2903
2909	2917	2927	2939	2953	2957	2963	2969	2971	2999

Um tipo específico de primos chama-se números “Primos Mersenos” (Mersenne Primes), já que satisfazem $2^x - 1$ é primo.

Como provar se um número é primo? Um dos teoremas simples é o Teste de Lucas-Lehmer que é: para um número *p* ímpar, o número merseno 2^p-1 é primo se e apenas se 2^p-1 divide S(p-1) onde s(n+1) = $S(n)^2$-2 e s(1) = 4.

Um dos algoritmos para esse é:

```
Lucas_Lehmer_Test(p):
  s := 4;
  for i from 3 to p do s := s2-2 mod 2p-1;
  if s == 0 then
    2p-1 is prime
  else
    2p-1 is composite;
```

Vamos provar que 2^7-1 é primo:

$2^7 - 1 = 128 - 1 = 127$
S0 = 4 (inicia-se com 4)
S1 = (4 * 4 - 2) mod 127 = 14 mod 127 = 14
S2 = (14 * 14 - 2) mod 127 = (196-2) mod 127 = 67
S3 = (67 * 67 - 2) mod 127 = (4489-2) mod 127 = 42
S4 = (42 * 42 - 2) mod 127 = (1767-2) mod 127 = 111
S5 = (111 * 111 - 2) mod 127 = (12321 – 2) mod 127 **= 0**

Se o resto = 0 então 2^7-1 é primo.

Atualmente (Jun/2003) o maior número merseno é $2^{13.466.917}$-1 que é um número com exatos 4.053.946 dígitos e foi encontrado em 2001. A descoberta foi feita em um AMD 800MHz T-Bird e levou 42 dias para provar que este número era primo. Este e o trigésimo nono numero merseno encontrado em 2000 anos.

3.4.3.2 – Primos entre si

Dizemos que a e b são primos entre si quando o mdc(a,b) = 1. Ou seja, somente 1 divide a e b com mod 0.

Por exemplo, 8 e 15 são primos entre si, já que os divisores de 8 são 8, 4, 2 ou 1 e os divisores de 15 são 15, 5, 3 e 1, ou seja, somente 1 é comum a ambos.

3.4.3.2 – Descobrindo e provando primos

Descobrir números primos é uma tarefa de tentativa e erro. Atualmente não existe uma técnica para dizer precisamente se um número é primo. O jeito é escolher um número ímpar e testar para saber se ele é primo. Se não for, escolher outro número até encontrar.

Também dizer se o número escolhido é primo com certeza não é uma tarefa tão simples. Para isso usamos testes de primaridades e roda-se testes para que a probabilidade de um número *p* ser primo ser próximo de 1,0. Um dos algoritmos mais populares é Miller-Rabin.

Estatisticamente, os números primos estão espaçados um a cada ln(p). Porém na média todos os números pares podem ser rejeitados no início, portanto precisamos testar ln(p)/2 números. Por exemplo, para um primo da ordem de 2^{200}, precisamos de no máximo $\ln(2^{200})/2$ = aproxidamente 69 tentativas antes de achar um primo.

O método Miller-Rabin diz que supondo que n>1 é um inteiro impar, escreve-se n-1 2^k com *m* impar e escolhe-se um inteiro a tal que (1< a <n-1).

Se $a^m \equiv \pm 1 \bmod n$ ou $a^{2^r \cdot m} \equiv -1 \bmod n$ para pelo menos um *r*, tal que (1< r < k-1), então *n* é declarado provável primo. Se isso for verdade, a chance de *n* ser primo é boa, porém deve-se repetir o teste com outro valor de *a* para melhorar a probabilidade (geralmente 10 vezes). De acordo com a hipótese de *Riemann*, se todos os valores de a até $2 \cdot (\ln(n))^2$ forem testados *n* é declarado é garantidamente primo.

Capítulo 4 – Utilização Pratica, Formatação e Gerenciamento de Chaves no RSA.

O RSA, embora já existam no mercado novos e sofisticados métodos e algoritmos como o AES – Padrão Avançado de Ciframento (*Advanced Encryption Standart*), algoritmos de curvas elípticas até de algoritmos quânticos, o RSA é um dos mais utilizados hoje em dia, principalmente em dados enviados pela Internet. A incapacidade de se fatorar números muito grandes utilizando os sistemas computacionais atuais torna este algoritmo muito forte. Ele é o único capaz de implementar assinatura digital e troca de chaves, entre os outros algoritmos mais comuns. O mesmo vem sendo largamente utilizado em várias aplicações de segurança envolvendo desde as camadas mais altas como certificados digitais, assinaturas digitais, S/Mime e até as camadas mais inferiores como o SSL (*Secure Socket Layer*) e TLS na camada de transporte e IPSEC na camada de rede

Nos itens seguintes estaremos mostrando brevemente o que cada uma das aplicações de segurança se utilizam do RSA nas diversas camadas de rede. Estaremos mostrando também o formato do algoritmo bem como é realizado o gerenciamento das chaves no RSA.

4.1 – Aplicações que utilizam o RSA

São vários os protocolos e aplicações de segurança que utilizam como base os algoritmos de criptografia RSA, entre elas: Certificados de Segurança, Assinaturas Digitais, S/Mime e PGP; protocolo SSL, TLS e IPSec.

4.1.1 – Certificados de Segurança

No funcionamento de troca de chaves públicas, os usuários da tecnologia RSA normalmente anexam a chave exclusiva a um documento enviado, de modo que o destinatário não precise procurá-la em um repositório de chaves públicas. Porém, como pode o destinatário ter certeza de que esta chave pública realmente pertence à pessoa indicada ou se um invasor não entrou na rede e esta se passando pelo remetente, fazendo-se passar por um usuário legítimo, logo pode ficar observando enquanto as pessoas enviam documentos confidenciais para uma conta falsa, criada por ele com essa finalidade.

Para contornar este problema existe o Certificado Digital, um tipo de passaporte ou credencial digital. O certificado digital é a chave pública do usuário que foi "assinada digitalmente" por alguém qualificado para isso, como o diretor de segurança de uma rede, os funcionários de um sistema de gerenciamento de informação (MIS) ou uma Autoridade Certificadora (CA).

Ao enviar uma mensagem, seu certificado digital segue em anexo. O destinatário da mensagem utiliza o certificado digital inicialmente para verificar se a chave pública do autor é autêntica e/ou então para ler a própria mensagem. Dessa forma, apenas uma chave pública (a da autoridade certificadora) deve ser armazenada ou públicada, pois a partir daí todos os outros podem simplesmente transmitir suas respectivas chaves públicas e certificados digitais válidos juntamente com suas mensagens.

Através da utilização dos certificados digitais, uma cadeia de autenticação que corresponda a uma hierarquia organizacional pode ser estabelecida, permitindo o registro e certificação adequados de chaves públicas em um ambiente distribuído.

Depois de adquirido o certificado digital, ele pode ser utilizado para uma grande variedade de aplicações, desde a troca de e-mails entre escritórios, empresas até a transferência eletrônica de fundos em todo o mundo. Para a utilização de certificados digitais, deve existir um elevado grau de confiança quanto ao vínculo entre um certificado digital (solicitado pelo usuário) e o usuário ou organização a ele associado (*certificate authority*).

Esta confiança é estabelecida através da construção de hierarquias de certificados digitais, onde todos os membros da hierarquia devem aderir ao mesmo conjunto de políticas. Os certificados digitais somente serão emitidos a pessoas ou entidades, como membros em potencial da hierarquia e após estabelecida uma comprovação de identidade. O estabelecimento da comprovação da identidade ou de como os certificados são emitidos, pode ser de diferentes formas de acordo com as políticas de cada hierarquia.

Um exemplo de empresa geradora de certificados digitais é a VeriSign que opera várias hierarquias de certificados digitais. Uma Autoridade Certificadora comercial tem um elevado grau de segurança quanto ao vínculo entre o certificado digital do usuário final e o próprio usuário final. Os membros de uma CA comercial RSA/VeriSign podem contar com um alto nível de certeza, em virtude da adesão às políticas, quanto às pessoas/empresas com quem se comunicam. É muito importante analisar a entidade certificadora, pois sem a devida segurança associada à uma hierarquia de certificação adequadamente gerenciada, o uso de certificados digitais tem valor restrito.

Seria mais simples se a empresa geradora de certificados se a data de validade de um certificado não precisasse ser alterada, ou seja só perdesse a validade conforme o prazo estipulado no seu próprio certificado.

Caso a chave pública de algum usuário foi comprometida ou ainda caso a CA não deseja certificar um determinado usuário devido alguma negligência as normas, pode-se recorrer à **Revogação de Certificados,** mencionado na RFC1422**.** Desta forma toda CA deve manter uma lista de certificados revogados (CRL – Certificate Revocation List), ao requerer um certificado a uma CA o usuário deve verificar a CRL e possivelmente manter um cache dos certificados revogados. No sistema de gerenciamento de certificados, onde é distribuído pode causar atraso na atualização dos certificados revogados, esta é uma das principais vulnerabilidades no gerenciamento de chaves baseadas em certificados públicas.

4.1.2 – Assinatura Digital

Para assinar uma mensagem a ser enviada de modo que comprove a integridade e autenticidade da mesma, uma função *Message Digest* (MD – resumo da mensagem ou mensagem digerida) é usada para processar o documento, produzindo um pequeno pedaço de dados, chamado de *hash*. Uma MD é uma função matemática que refina toda a informação de um arquivo em um único pedaço de dados de tamanho fixo.

Para garantir uma assinatura digital tem que verificar as seguintes propriedades:

- Um usuário não pode forjar a assinatura de outro usuário e as assinaturas digitais devem ser únicas para cada usuário;
- O emissor de uma mensagem não pode invalidar a assinatura de uma mensagem. Ou seja, não pode negar o envio de uma mensagem com sua assinatura;
- O receptor da mensagem não pode modificar a assinatura contida na mensagem;
- Um usuário não pode ser capaz de retirar a assinatura de uma mensagem e colocar em outra

Funções MD são mais parecidas com checksums quanto a não receber uma chave como parte de sua entrada. Na verdade, entra-se com os dados a serem "digeridos" e o algoritmo MD gera um *hash* de por exemplo 128 e 160 bits (dependendo do algoritmo, são exemplos: MD4, MD5 e Snefru). Uma vez computada uma *message digest*, criptografa-se o *hash* gerado com uma chave privada. O resultado de todo este procedimento é chamado de assinatura digital da informação. A assinatura digital é uma garantia que o documento é uma cópia verdadeira e correta do original, e garantindo também a autoria da mensagem.

O motivo para se usar funções *message digest* está diretamente ligado ao tamanho do bloco de dados a ser criptografado para se obter a assinatura. De fato, criptografar mensagens longas pode durar muito tempo, enquanto que criptografar *hashs*, que são blocos de dados pequenos e de tamanho fixo, gerados pela MD torna o processamento mais eficiente.

As assinaturas digitais, como outras convencionais, podem ser forjadas. A diferença é que a assinatura digital pode ser matematicamente verificada. Dado um documento e sua assinatura digital, pode-se facilmente verificar sua integridade e autenticidade. Primeiro, executa-se a função MD (usando o mesmo algoritmo MD que foi aplicado ao documento na origem), obtendo assim um *hash* para aquele documento, e posteriormente, decifra-se a assinatura digital com a chave pública do remetente. A assinatura digital decifrada deve produzir o mesmo *hash* gerado pela função MD executada anteriormente. Se estes valores são iguais é determinado que o documento não foi modificado após a assinatura do mesmo, caso contrário o documento ou a assinatura, ou ambos foram alterados. Infelizmente, a assinatura digital pode dizer apenas que o documento foi modificado, mas não o que foi modificado e o quanto foi modificado. Todo o processo de geração e verificação de assinatura digital pode ser visto na figura abaixo, utilizando o algoritmo de criptografia de chave pública RSA. Para ser possível que um documento ou uma assinatura adulterada não seja detectada, o atacante deve ter acesso a chave privada de quem assinou esse documento.

As assinaturas digitais são possíveis de serem verificadas usando chaves públicas. A assinatura digital também é valiosa, pois pode-se assinar informações em um sistema de computador e depois provar sua autenticidade sem se preocupar com a segurança do sistema que as armazena. Abaixo segue um fluxograma do funcionamento do trafego de uma mensagem com assinatura digital.

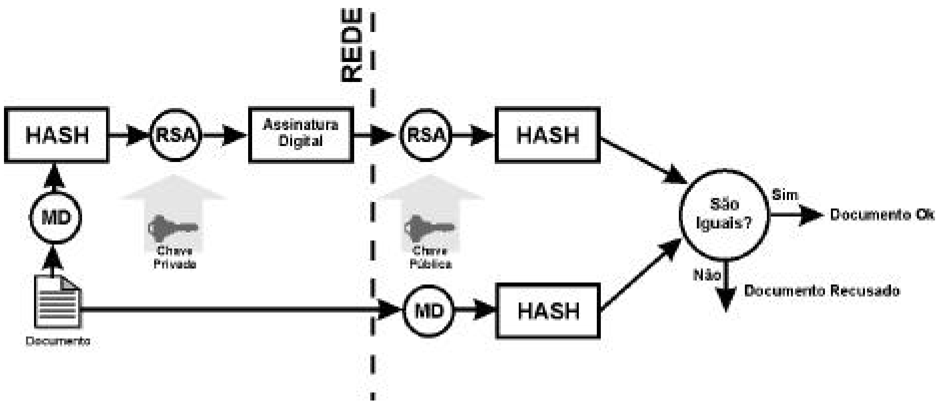


Figura 5: Esquema de Assinatura Digital no RSA

Exemplificando, cada membro de uma comunicação possui uma chave pública (nI , cI) e uma chave privada (nI , dI) do sistema de cifragem RSA. Suponhamos que Ana pretende enviar ao Beto uma mensagem M, assinada e cifrada.

• Primeiramente, Ana vai assinar a mensagem M aplicando a sua chave privada

(nA , dA), obtendo:

S ≡ MdA (mod nA).

• Para cifrar a mensagem, o Ana usa a chave pública do Beto (nB , cB):

X ≡ ScB (mod nB).

Existe um projeto lei do Comitê Executivo do comercio eletrônico que regulamenta a utilização da assinatura digital no comercio, através da Lei "PROJETO DE LEI N.º 4.906, em 26/09/2001 Dispõe sobre o valor probante do documento eletrônico e da assinatura digital, regula a certificação digital, institui normas para as transações de comércio eletrônico e dá outras providências". No entanto muitas instituições dividem-se nas diferenças entre uma assinatura manual e uma digital, pois para alguém assinar em nome de outra pessoa tem que ser reconhecido em cartório com as burocracias legais, já para transferência de assinatura digital isso não é possível.

4.1.3 – S/Mime e PGP

Para implementações de segurança em correio eletrônico, existem duas aplicações que são mais utilizadas que são: S/Mime e o PGP *(Pretty Good Privacy)*

O S/MIME é um mecanismo mais recente proposto pela RSA Inc. em conjunto com várias empresas, dentre elas a Microsoft e a Netscape. A versão 3.0 está em forma de drafts e prove a utilização de DH/DSS ao invés do RSA.

O PGP é um dos mecanismos mais utilizados. A partir da versão 5.0 também começou a utilizar algoritmos DH/DSS que são livres de patente, o incentivo a padronizações dado pela IETF interessada nos mecanismos que utilizam algoritmos proprietários (como é o caso do RSA).

4.1.4 – SSL/TLS

SSL (*Secure Socket Layer*) RFC 3207 é uma camada do protocolo de rede, situada abaixo da camada de aplicação, com a responsabilidade de gerenciar um canal de comunicação seguro entre o cliente e o servidor. O SSL foi desenvolvido pela Netscape Communications Corporation . Atualmente é implementado na maioria dos browsers. A palavra-chave ***https://*** é usualmente empregada para designar uma conexão segura.

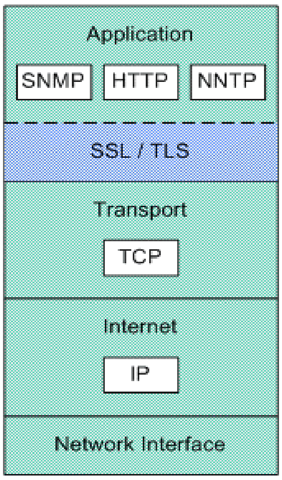
O SSL preenche os seguintes critérios que o fazem aceitável para o uso nas transmissões das mais sensíveis informações, como dados pessoais e números do cartão de crédito:

- **Autenticidade:** De modo a garantir a autenticidade de A no caso anterior, um sistema de códigos um pouco mais complexo é necessário. A mensagem de A para B é primeiramente criptografada com a chave privada de A e posteriormente com a chave pública de B. Para decodificar a mensagem B usa primeiro sua chave privada e depois a chave pública de A. Agora B pode ter certeza de que A é realmente quem diz ser, pois ninguém mais poderia criptografar a mensagem usando a chave privada de A. Isso é desenvolvido pelo SSL com o uso de certificados.
- **Privacidade:** Digamos que uma mensagem é transmitida de A para B. Neste caso A usa a chave pública de B para criptografar a mensagem, tornando B a única pessoa que pode decodificar a mensagem, usando a sua chave privada. Nós não podemos entretanto ter certeza quanto a identidade de A.
- **Integridade:** a integridade é garantida pelo uso do MAC (Message Authentication Code) com as necessárias funções da tabela *hash*. Na geração de uma mensagem, o MAC é obtido por aplicação das funções da tabela *hash* e é codificado junto com a mensagem. Após a mensagem ser recebida sua validade pode ser checada comparando-se o MAC com o resultado obtido pelas funções *hash*. Isto previne mensagens alteradas por terceiros durante a transmissão.
- **Não repudiação:** Um sistema totalmente seguro deve ser capaz de detectar impostores ou, ainda melhor, se prevenir contra a duplicação das chaves. Isto é executado por um hardware baseado em ficha única. O SSL não

suporta sozinho esta implementação, mas a realiza em conjunto com Fortezza.

O protocolo SSL suporta uma variedade de diferentes algoritmos criptográficos, para uso em operações de autenticação do servidor e do cliente, transmissão de certificados, e estabelecimento de sessões. Clientes e servidores podem suportar um algoritmo diferente, ou um conjunto de algoritmos, dependendo de fatores como: qual versão de SSL eles (cliente e servidor) suportam, políticas da companhia, e restrições do software SSL. Além de outras funções, é o protocolo de Handshake do SSL que determina como cliente e servidor negociam quais conjuntos de algoritmos irão usar para se autenticar um para o outro, para transmitir certificados, e para estabelecer sessões. Algoritmos de troca de chaves como KEA e *RSA Key Exchange* governam a maneira pela qual clientes e servidores determinam as chaves simétricas que usarão durante uma sessão com SSL. O mais comumente usado é o *RSA Key Exchange*.

SSL in the IP Architecture



O protocolo TLS *(Transport Layer Security)* é padronizado pela IETF e é baseado na SSLv3. Este protocolo não aprimorou tanto a os mecanismos de checksum, no entanto possui mensagens extras de prevenção de ataques. Apesar disso a versão é a mais largamente desenvolvida e a SSLv2. Atualmente poucos comerciantes utilizam somente o TLS.

4.1.5 – IPSEC

É um padrão de protocolos criptográficos desenvolvidos para o IPv6. Realiza também o tunelamento de IP sobre IP. É composto de três mecanismos criptográficos: Authentication Header (define a função *hash*ing para assinatura digital), *Encapsulation Security Payload* (define o algoritmo simétrico para ciframento) e ISAKMP (define o algoritmo assimétrico para gerência e troca de chaves de criptografia). Criptografia e tunelamento são independentes. Permite *Virtual Private Network* fim-a-fim.

4.2 – Padrões e formatos dos certificados digitais

Com o objetivo de estabelecer uma estrutura de certificados, ciclo de vida, disponibilização de informações estruturadas em larga escala levou o órgão ITU-T aos padrões de hierarquia dos protocolos da família X.500. As recomendações da série X.500 fornecem a base para o protocolo ITU-T Recommendation X.501, aprovado em 1988 e em 1993, onde seria possível localizar, recuperar, inserir e remover qualquer tipo de informação de forma estruturada e distribuída, conhecido como serviço de diretórios.

4.2.1 – Recomendação X.509

O padrão X.509 definido nas RFCs 3280 e 3279, trata do relacionamento entre as autoridades de certificadoras. Busca criar uma estrutura mundial, de raiz única, comparando aos diretórios de um sistema de arquivos, onde é possível de forma fácil e rápida localizar e recuperar informações, bem como organiza-las. Cada nó da árvore teria um identificador único, chamado OID (*Object Identifier*) e seria responsável por emitir e gerenciar certificados para garantir acesso as informações pertencentes ao seu nível e possivelmente aos níveis superiores.

O X.509 (ITU-T Recomendation X.509) descreve dois formatos de autorização para o protocolo X.500: um chamado de mais fraco que utiliza somente conta e senha; e outro forte, usando certificados digitais. O padrão dito forte fornece uma autenticação para a estrutura de acesso e define o sujeito existente no certificado de acordo com a estrutura de nomes implementada para o acesso a dados. Este padrão tem sofrido mudanças devido nas abordagens de Konfelder, sendo base para a atual implementação do padrão PKIX, um dos mais importantes padrões de certificados.

4.2.2 – PKIX

Os padrões PKIX foram desenvolvidos para prover segurança ao padrão X.500. E as versões do mesmo foram se evoluindo de acordo com a constatação de novas exigências para prover mais segurança.

Estes padrões possuem dois formatos de certificados:

- certificados: gerar a associação da entidade-fim (no original, End Entity) com a respectiva chave pública;
- certificados de revogação: é bem semelhante à entidade fim e serve para divulgar uma lista dos certificados que foram revogados.

O X.509 passou por 3 versões, abaixo seguem as recomendações adicionadas/alteradas e as justificativas de vulnerabilidades encontradas a cada versão dos mesmos:

- ***X.509v1***: tinha um número restrito limitado de campos nesta forma de utilização; além disso, alguns problemas de segurança foram identificados no padrão. Foi o adotado pelo padrão PEM (*Privacy Enhanced Mail*) em 1993.

- ***X.509v2:*** Após a revisão da primeira versão, nesta versão foram adicionados novos campos de acordo com a figura abaixo com o objetivo de possibilitar a reutilização de nomes iguais em diferentes certificados digitais.

- ***X.509v3:*** Foi implementado devido a ineficiência encontrada nas versões anteriores quanto a implementação do PEM. Desta forma na atual versão, de junho de 1997, foram adicionados campos de extensão, o que torna o certificado mais flexível e com uma expansão na utilização muito maior.

Embora existam várias estruturas de certificados em uso na Internet, sem dúvida a descrita na recomendação X.509 é a mais aceita. Segue o formato do padrão X.509 e suas alterações no decorrer das versões:

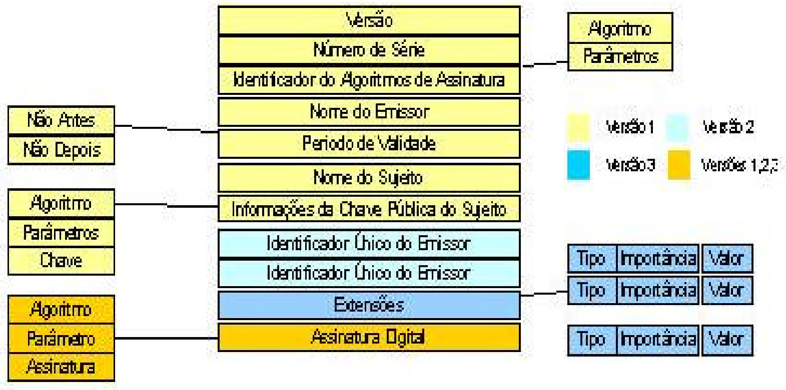


Figura 6: Formato do Padrão X509

- *Versão:* Identifica a versão do formato do certificado;
- *Numero de serie:* numero seqüencial único para cada certificado emitido por uma CA.
- *Identificador:* algoritmo usado para assinar o certificado (Ex.RSA)
- *Nome do emissor:* nome da CA
- *Período de validade:* data de inicio e data de termino da validade
- *Nome do sujeito:* Nome da entidade cuja a chave pública foi assinada
- *Informações da chave pública:* algoritmo, parâmetros e atributos da chave pública propriamente dita.
- *Assinatura Digital:* Assinatura gerada usando a chave pública da CA sobre as informações acima.

Abaixo segue uma comparação do formato de um certificado X.509 versão3 e um formato de um certificado revogado CRL (Certificate Revocation List).

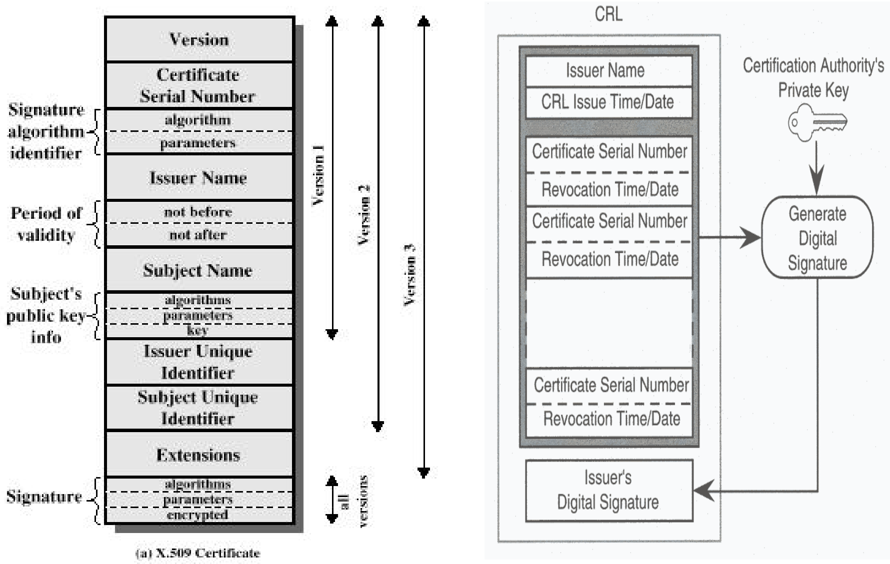


Figura 7: Comparação do certificado X509 e CRL

4.2.3 – ASN.1

A Abstract Syntax Notation One (ASN.1) encontra-se especificada nas normas X.208, embora tenha sido criada para especificar tipos de dados e valores de dados nas normas OSI, é hoje utilizada sistematicamente para especificar objetos nos mais variados standards, incluindo os criptográficos. As recomendações da linha X.500 definem que todas as informações a serem armazenadas são primeiramente descritas segundo uma sintaxe padronizada chamada Abstract Syntax Notation One (ASN1). O ASN1 que introduz a definição de Object Identifieres (OIDs) e torna possível a comunicação da semântica da informação entre sistemas distintos. As formas mais usuais de codificação do ASN.1 são:

- **DER** (Distinguished Encoding Rules)
- **BER** (Basic Encoding Rules).

4.2.3.1 – BER

É um mecanismo de codificação definido no standard X.209 e que, por permitir obter várias codificações para o mesmo valor, não é conveniente quando é necessária uma codificação sem ambigüidades.

4.2.3.2 – DER

É um subconjunto do BER definido no standard X.509 e que, introduzindo restrições adicionais à codificação, garante uma codificação única para cada valor ASN.1. O DER é geralmente utilizado quando se pretende garantir a compatibilidade da codificação ASN.1 com implementações heterogêneas. É utilizado após a descrição do certificado, no momento do transporte, os dados são codificados de acordo com o DER de modo que os mesmos podem ser armazenados e transferidos independentemente da plataforma de hardware e software.

4.2.2 – PKCS

PKCS *(Public Key Cryptography Standards)* é o conjunto de especificações criadas para padronizar os formatos e operações de criptografia. Existem vários padrões para os diversos algoritmos, dentre estes citados abaixo, o PKCS1 versão 2.0 especifica o algoritmo RSA:

- **PKCS#1** *RSA Encryption Standard* - Especificação de padrão de dados para o protocolo RSA, incluindo o padrão para criptografia e assinatura digital RSA e padrão para estocagem de chaves públicas e privadas.
- **PKCS# 3** *Diffie-Hellman Key Agreement Standard* – Este padrão descreve o método para implementação de chaves no Diffie-Hellman, cujo objetivo deste padrão é fornecer protocolos para estabelecimento de conexões seguras.
- **PKCS#5** *Password-Based Encryption Standard* - Especificação de um padrão para proteção de dados para ser usar a criptografia baseada em senha com o DES.
- **PKCS #6** *Extended-Certificate Syntax Standard*
- **PKCS#7** *Cryptographic Message Syntax Standard* – Este padrão descreve um sintaxe geral para os dados serem criptografados e aplicados em assinaturas digitais e mensagens digitais. Tomando como base a RFC 2630. È utilizado para prover mensagens seguras em S/MIME.
- **PKCS#8** *Private-Key Information Syntax Standard* - Especificação de um padrão para estocagem de chaves privadas, incluindo a vantagem de criptografá-las com PKCS#5.
- **PKCS#10** *Certification Request Syntax Standard* - Especificação de um padrão para codificar requisições de certificados, incluindo o nome da pessoa que requisita o certificado e sua chave pública.
- **PKCS #11** *Cryptographic Token Interface Standard* – Este padrão descreve a interface de programação chamada “Cryptoki" utilizada para operações criptográficas em hardwares: tokens, smart cards.É quão popular utilizar o PKCS#11 para prover o suporte aos tokens como o Netscape as aplicações de SSL e S/MIME.

- **PKCS #12** *Personal Information Exchange Syntax Standard* - Este padrão define o formato para armazenamento e transporte de chaves privadas, certificados, entre outros
- **PKCS #13** *Elliptic Curve Cryptography Standard* – Padronização de algoritmos de criptografia baseado em curvas elípticas incluindo o formato, a geração de validação de chaves, assinaturas digitais, etc.
- **PKCS #15** *Cryptographic Token Information Format Standard* – É o padrão que define o uso da tecnologia de criptografia baseada em tokens.

O PKCS#1 versão 2.1 (RFC 3447) estabelece algumas normas na base de cálculos do RSA já vista anteriormente, o método de cifragem e decifragem, assinaturas digitais, e padrões para gerenciamento e armazenamento de chaves.

4.3 – Gerenciamento de chaves no RSA

O ponto mais vulnerável da criptografia assimétrica é a certificação das chaves públicas, ou seja, provar se quem tem em mãos a chave pública de quem realmente você quer trocar informações e não uma pessoa mal intencionada. Uma das formas de estabelecer a autenticidade de uma chave do modo convencional é empregá-las em mãos. Uma outra forma é utilizar a assinatura digital já discutida anteriormente.

A infra-estrutura para lidar com o gerenciamento de chaves públicas é definida pelo padrão *Public Key Infrastructure* (PKI) que define onde os certificados digitais serão armazenados e recuperados, de que forma estão armazenados, como um certificado é revogado, entre outras informações, já discutidas anteriormente.

Existem vários pontos fracos do PKI, dentre eles: espaço para armazenamento manutenção do diretório de chaves públicas; a infra estrutura do padrão PKI é bastante complexa; há um custo computacional para verificação de um certificado antes de utilizar uma chave pública; o tempo decorrido entre a revogação de um certificado e a atualização CRL gera falhas de segurança; impossibilidade de recuperação de chaves privadas acarretando perdas de informações.

Capítulo 5 – Segurança do RSA

5.1 – Introdução

O RSA se inicia com dois grandes números primos, *p* e *q*. Dados esses números, encontra-se seu produto, *n=pq*, denominado **módulo**. Um terceiro número *e*, menor que *n* e relativamente primo com *(p-1)(q-1)* é escolhido e seu inverso *mod (p-1)(q-1), d*, calculado.

Como já vimos, o algoritmo do RSA é:

Para Criptografar: $C = M^e \bmod n$

Para Decriptografar: $M = C^d \bmod n$

A segurança do RSA é baseada na dificuldade de fatorar um número *n* quando *n* é um número muito grande. Suponhamos que perdemos os números p e q, mas conhecemos o produto n = pq. Como recuperar?

Existem vários métodos de ataque ao RSA, mas todos os caminhos são pelo menos tão difíceis quanto fatorar *n*.

5.2 – Atacando o RSA

Mostraremos algumas das técnicas de ataques ao RSA. O ataque mais grave ao RSA é descobrir a chave privada correspondente a uma chave pública. Isto permitiria ao hacker ler as mensagens criptografadas e forjar assinaturas. O caminho óbvio é tentar fatorar o número *n* e descobrir os fatores *p* e *q*. Se o RSA for equivalente ao problema da fatoração, esse é o único ataque possível, porém isso ainda não foi provado e é razoável admitir que pode existir outro ataque ao RSA que não tente uma fatoração direta de *n*.

Uma das características do RSA é que ele um sistema **multiplicativo**. Uma função é dita multiplicativa quando dado *f(x)* e *f(y)*, calcular *f(xy)* é fácil. Isso significa que o RSA é suscetível a ataques por texto em claro escolhido.

Os ataques são Força Bruta, Ataques Matemáticos e Ataques Temporais.

Claro que se o administrador de rede armazenar a chave privada de maneira insegura ou com uma criptografia baixa (normalmente se criptografa a chave privada por segurança) é possível atacar o servidor e roubar o arquivo contendo a chave privada. Isto não é exatamente um ataque ao RSA, porém deve-se tomar muito cuidado onde guardar a chave privada.

5.2.1 – Força Bruta

O ataque de força bruta significa tentar todas as combinações de chaves possíveis.

Os ataques de busca (ou força bruta) são os que não empregam nenhum esforço intelectual para descobrir a chave que decifra os dados. De posse do algoritmo, o atacante simplesmente tenta todas as combinações possíveis até quebrar a criptografia. Em muitos casos, especialmente quando se trata de chaves de comprimento longo, pode ser necessário muito poder de computação para que a chave seja descoberta em tempo hábil.

A defesa do RSA contra esse tipo de ataque baseia-se na mesma dos outros algoritmos: usar um número gigantesco de possíveis chaves. Por exemplo, com uma chave de 128bits existem 2^{128} chaves possíveis, ou seja, $3{,}403 \times 10^{38}$ chaves possíveis. E além da lentidão no processo de criação de chaves longas pode fazer com que o sistema de ataque usando força bruta se torne muito difícil.

5.2.2 – Ataques Matemáticos

Três são os ataques matemáticos:

- Fatorar *n* em dois números primos. Isto possibilita calcular $f\,n$ = (p-1)(q-1), e com isso determinar o valor de $d = e^{-1}(mod\ f\ n)$
- Determinar f n diretamente, sem determinar *p* e *q*, permitindo novamente determinar o valor de $d = e^{-1}(mod\ f\ n)$
- Determinar d sem descobrir $f\ n$.

Um jeito de atacar o RSA é achar uma técnica que calculo a enésima raiz mod *n*. Já que C = M^e mod *n*, a enésima (*eth*) raiz de c mod n é a mensagem M. Entretanto não existe nenhum método conhecido para calcular isso, sem fatorar *n*.

5.2.2.1 – Problema da Fatoração

Para um *n* grande usando fatores primos, tentar fatorar *n* é um sério problema.

O mais rápido algoritmo até o momento é *General Number Field Sieve* que é feito em duas etapas, criar muitas relações independentes, tais como $x_1\ x_2\ x_3 \ldots\ldots x_k = y_1\ y_2\ y_3 \ldots\ldots y_r$ mod n, usando processamento distribuído, tais como peer-to-peer. O segundo passo é resolver um sistema linear a fim de encontrar $x^2 = y^2$ (mod n), usando um computador com muita memória. O tempo para rodar este algoritmo em um inteiro de n bits é $\exp\left((c+o(1))n^{\frac{1}{3}} \cdot \log^{2/3} n\right)$

Em situações práticas, em 1976, estimou-se que o RSA-129 (412 bits) usado pelos autores do RSA para cifrar a primeira mensagem com método de chaves públicas demoraria milhões de anos para fatorar. Entretanto em 1994, cerca de 1600 computadores fatoraram o número em menos de 8 meses. O RSA129 fatorou-se em dois números inteiros primos de 64 e 65 digitos usando o algoritmo *Quadratic Sieve* gerando uma matrix de 188346^2 itens mostrando o texto criptografado "*The magic words are squeamish ossifrage*".

Um projeto da distributed.net em 22 de agosto de 1999 anunciou a decifragem de uma mensagem criptografada, um desafio da RSA Labs. Nesta ocasião, um conjunto de computadores permitiu a fatoração de um número com 512bits (155 dígitos decimais) em dois primos de 78 dígitos cada. O algoritmo foi o *Number Field Sieve* em uma rede com 300 mil computadores. Seriam necessários 8000 Anos-MIPS para fazer isso. A filtragem (passo 2) foi feito em um supercomputador Cray C916 no Centro de Computação Acadêmica de Amsterdã e a implementação do código foi feito por Peter Mongomery consumiu 3,2GB de RAM, 224 horas de CPU para processar a matriz tridimensional com 6.699.191 linhas, 6.711.336 colunas e 417.131.631 linhas em profundidade. O tempo de fatoração total foi de 5,2 meses, além de 2,2 meses para selecionar os polinômios. A RSA Labs mantém vários desafios em seu site para números de 576 à 2048 bits. Para ter uma idéia da dimensão, a RSA Labs estima que o RSA1024 requer 1 milhão de vezes mais esforço computacional do que o RSA512.

Abaixo o tempo estimado nos processos em fatoração:

Números de Dígitos Decimais	Número aprox. de Bits	Dados conseguidos	Anos MIPS	Algorítimo
100	332	Abril 1991	7	Quadrático
110	365	Abril 1992	75	Quadrático
120	398	Junho 1993	830	Quadrático
129	428	Abril 1994	5000	Quadrático
130	431	Abril 1996	500	Campo númerico generalizado

Tabela 4: Tempo de Fatoração no RSA

Os ataques em RSA-130 usaram um algoritmo mais novo, de campo de número generalizada (GNFS), e pôde fatorar um número maior que RSA-129 a só 10% do esforço de computação.

O aumento continua em poder de computação, e continua o refinamento de fatoração do algoritmo. Vimos que o movimento para um algoritmo diferente resultado em uma tremenda velocidade. Nós podemos esperar refinamentos adicionais no GNFS, e o uso no algoritmo até melhor. Na realidade, um algoritmo relacionado, o campo de número especial (SNFS), pode fatorar números com uma forma especializada consideravelmente mais rápido que o campo de número generalizado. É razoável esperar uma inovação que habilitaria um desempenho fatorando todo dentro do mesmo tempo como SNFS, ou até melhor.

Atualmente chaves de 1024 bits são o mínimo para ser considerado seguro, porém precisamos ter cuidado para escolher um tamanho fundamental para RSA. Para o próximo futuro, um tamanho fundamental na gama de 1024 a 2048 bits parece razoável, já que a cada dia a tecnologia avança e existe mais poder computacional para fatorar *n* e melhores algoritmos de fatoração. No entanto, se algum dia alguém descobrir um método eficaz de fatoração de inteiros, será fácil quebrar todas as cifras RSA. Se um dia isso acontecer, criptosistemas como curvas elípticas seria a solução, já que utilizam grupos e polinômios mais complexos.

5.2.2.2 – Calcular ∮*n* sem fatorar *n*

Se descobrirmos $f\,n$ poderemos facilmente obter d calculando o inverso de c mod $f\,n$. Se conhecemos o produto P = $pq = n$ e $f\,n = (p\text{-}1)(q\text{-}1)$ então é igual a n-(p+1)+1. É fácil obter a soma S = p+q calculando S= p + q = n - f n +1. Logo p e q são raízes do polinômio x – (n- f n +1)x+n.

Outra maneira semelhante de calcular p e q, primeiro calculamos p + q = n - f n + 1 e depois calcularemos:

$$p - q = \sqrt[2]{(p+q)^2 - 4n}$$

$$q = \frac{(p+q)-(p-q)}{2}.$$

Estes dois processos permitem facilmente fatorar n conhecendo $f\,n$, porém conhecer $f\,n$ é tão difícil quanto fatorar n.

Suponha que o criptoanalista aprendeu que n = 84773093 e $f(n)$ = 84754668. Esta informação dá origem à equação quadrática seguinte:

$$p^2 - 18426p + 84773093 = 0.$$

Isto pode ser resolvido pela fórmula quadrática, enquanto os dois roots 9539 e 8887. São os dois fatores de n.

5.2.2.3 – Expoente de Decriptação

O resultado é muito interessante para nós, agora que sabemos fatores de n, qualquer algoritmo que compute o exponente de decriptação a ser usada como uma subrotina de dados em um algoritmo de probabilidades fator n pode resolver o problema.

O valor de n chega a um acordo. Se isto acontecer, não é suficiente para Bob escolher um expoente de nova encriptação; ele também tem que escolher um novo módulo n. O algoritmo irá descrever uma probabilidade do algoritmo do tipo de Las Vegas.

Definição:

Suponha $0 \leq e < 1$ é um número real. Um algoritmo de Las Vegas é um algoritmo de probabilidade, que não pode dar uma resposta com um pouco de probabilidade (por exemplo, pode terminar com a mensagem "nenhuma resposta"). Porém, se o algoritmo devolver uma resposta, então a resposta, deve estar correta.

Observe que um algoritmo de Las Vegas pode não dar uma resposta, mas qualquer resposta que mostrar estará correta.

Em Monte Carlo o algoritmo sempre mostra para uma resposta, mas a resposta pode estar incorreta.

Se nós tivermos um algoritmo de Las Vegas para resolver um problema, então nós rodamos o algoritmo simplesmente inúmeras vezes até que ache uma resposta. A probabilidade que o algoritmo vá devolver "nenhuma resposta" m vezes em seqüência é de e^m. A média (esperada) de números de vezes que o algoritmo deverá rodar para obter uma resposta na realidade é $1/(1 - e)$.

Suponha que ***A*** é um algoritmo hipotético que compute o expoente de decriptação a de b e n. Descreveremos um algoritmo de Las Vegas que usa ***A*** como um guia. Este algoritmo irá fatorar n pelo menos com probabilidade 1/2. Conseqüentemente, se o algoritmo é rodado m vezes, então n será pelo menos fatores com probabilidade $1 - 1/2^m$.

5.2.2.4 – Calcular d sem possuir ∮*n*

Calcular d sem possuir ∮n não é mais fácil do que fatorar n. Provou-se que n pode ser fatorado usando um múltiplo de ∮n, já que $cd \equiv 1 (mod$ ∮$n)$ é equivalente a $cd = 1 + k$ ∮(n) para algum k contido em Inteiros.

Outro jeito de fazer isso é encontrar M (mod n) a partir de M^r mod n, ou seja, calcular a enésima raiz do módulo n sem fatorar n. Atualmente não se sabe uma maneira rápida de descobrir isso, mas com certeza é um dos problemas difíceis.

5.2.3 – Ataques Temporais

Ataques temporais são baseados no tempo em que um sistema demora para processar uma mensagem. Prova-se que ataques temporais são possíveis baseados no tempo que o computador leva para decifrar as mensagens, já que o ataque leva em conta que os algoritmos normalmente não rodam em tempo fixo. Por exemplo, no algoritmo para calcular a^b mod n, o hacker pode saber o primeiro bit para ter um padrão e para o algoritmo mais usado (mostrado no capítulo 3) o cálculo quando b_k é 1 é mais lento do que quando é 0.

O ataque procede bit por bit que começa com o bit mais a esquerda de b_k. Suponha que o primeiro bit de j é conhecido. Para um determinado texto cifrado, o ataque pode completar as primeiras repetições da volta de j. Se o bit for $b_j = 1$ então $d = (d \times a) \pmod{n}$ será executado. Para alguns valores de a e d, a multiplicação modular estará extremamente lenta, e o hacker saberá que são estes os valores. Se o tempo observado para executar o algoritmo de decriptação quando esta repetição em particular estiver lenta com um bit, então é assumido que este bit é 1. Se vários tempos de execução observados para o algoritmo inteiro forem rápidos, então é assumido que este bit é 0.

Para proteger-se o RSA recomenda o uso de tempo constante de exponenciação (certificando-se que demore o mesmo tempo para calcular qualquer informação), espera randômica (inclusão de delays aleatórios ou cálculos inúteis no meio do código) ou *blinding* (“cegamento”).

Para *bliding* a RSA gera um número r aleatório entre 0 e n-1, calcula $C^r = C^{re}$ mod n, onde e é a chave pública, $M' = (C^r)^d$ e calcula $M^r = M'r^{-1}$ mod n. Isto causa uma perda de 2 à 10% na performance geral.

Uma outra técnica para evitar este tipo de ataques é a utilização de tabelas, tal como foi feito na implementação da rotina *ByteSub* do RSA. Na maioria dos compiladores C, a utilização de tabelas, leva a um código que é executado sempre com o mesmo tempo, evitando este tipo de ataque. No entanto, a memória ocupada para o alojamento das tabelas é uma desvantagem que é necessária ponderar.

5.2.4 – Outros tipos de Ataques

5.2.4.1 – Expoente d pequeno para decriptografia

Uma vez um expoente e, pode ser desejável escolher um expoente d um pouco menor para melhorar a eficiência da decriptografia. Entretanto se mdc(p-1;q-1) é pequeno, em um caso típico se e e d tem aproximadamente um quarto do tamanho de n, existe um algoritmo eficiente para calcular d baseado em n e e. Entretanto para evitar este ataque d deve ser aproximadamente do tamanho de n.

5.2.4.2 – Expoente e pequeno para criptografia

Para reduzir o tempo de criptografia ou de verificação da assinatura, é possível usar um e pequeno para o expoente público. O menor valor para e é 3, porém para defender-se de certos ataques um valor de $e = 2^{16} + 1$ (65537) é recomendado. Quando $2^{16}+1$ é usado, a verificação da assinatura leva 17 ciclos de multiplicação, ao contrário de possíveis 1000 ciclos quando $3 < e < \oint n$.

5.2.4.3 – Ataque de Módulo Comum

Como se sabe a chave pública é d e n. Entretanto se em uma rede que implementa uma autoridade local e distribui pares de chaves para várias máquinas na rede, e se ele começa a selecionar um módulo n para um grupo de máquinas, é possível se uma máquina enviar dados para duas outras e usarmos a técnica de captura de pacotes na rede (*eavesdropper*) é mais fácil conseguir $\{e,n\}$ baseado na informação públicamente disponível.

5.2.4.4 – Criptoanálise do RSA se *d* for menor que $n^{0,292}$

Um recente papel de Dan Boneh e Glenn Durfee, mostra que é possível atacar a chave privada d se o valor de $d < n^{0,292}$. Entretanto requer um alto grau de conhecimento matemático. Para maiores informações consulte:
http://crypto.stanford.edu/~dabo/papers/lowRSAexp.ps

5.2.4.5 – Ataque de Broadcast de Hastad

Supondo que Bob (B) quer enviar uma mensagem M para um número k de computadores P, tais que seriam $P_1,P_2,....,P_k$. Cada P possui sua própria chave pública. Assumimos que M é menor que todos os n da rede. Para B enviar M, ele tem que cifrar com a chave pública de cada P e enviar o *i*-nésimo texto cifrado para P_i. Um hacker poderia capturar esses pacotes enviados por B.

Vamos imaginar que todos os expoentes públicos são 3. O hacker pode refazer M se k > 3.

$C_1 = M^3 \bmod n_1$ $C_2 = M^3 \bmod n_2$ $C_3 = M^3 \bmod n_3$

Aplicando um teorema chamado "Teorema do Resto Chinês" acharíamos C pertence a todos Inteiros satisfazendo $C' = M^3 \bmod n_1 n_2 n_3$. Já que M é menor que todos os *n*'s, então é possível saber que $C' = M^3$ tem todos os inteiros, portanto basta achar a raiz cúbica de M. Generalizando, se os expoentes públicos são iguais a *e*, podemos saber M em menos de k $\geq$ e. Porém só é possível este ataque se *e* é pequeno.

5.2.4.6 – Ataque de Exposição parcial da chave privada

Se {d,n} são a chave privada RSA, e imaginemos que uma fração de bits de *d* foi exposta, então é possível reconstruir *n* se $e < \sqrt{n}$

5.2.4.7 – Ataque de Bleichenbacher no PKCS 1

Que N seja o enésimo bit do módulo RSA e M seja um bit *m* da mensagem com $m<n$. Antes de aplicar o ciframento RSA é natural "encher" a mensagem de M com *n* bits adicionando bits aleatórios nele. Na versão 1 do PKCS1 usa esse esquema e antes do "*padding*" a mensagem parece com

02	Aleatório	00	M

O resultado da mensagem é *n* bits e é criptografado usando RSA. O bloco inicial "02" é um inteiro longo de 16 bits e indica que um valor de bits aleatório foi incluído na mensagem.

Quando PKCS1 é recebida, a aplicação decifra a mensagem, checando o bloco inicial e removendo a parte aleatória. Entretanto algumas aplicações vêem "02" e se ele não estiver presente retorna um erro com "*texto cifrado inválido*". Com a resposta

Suponha que interceptemos *C* que era para Bob decifrar. Para montar o ataque o computador X (atacante) escolhe um número aleatório *r* e calcula $C' = rC \bmod N$. A aplicação rodando na máquina B recebe C' e responde com um erro. Porém X aprende se os 16 bits mais significantes da decriptografia de C' é igual a "02". Agora X tem quem teste para ele isso, para qualquer *r* que ele escolha. Bleichenbacher mostrou que isso é suficiente para ajudar a decriptografar C.

Capitulo 6 – Comparativos entre Algoritmos

6.1 – Simétrica x Assimétrica

Uma breve comparação entre criptosistema de chave simétrica e assimétrica está abaixo. Ela compara de forma rápida as características principais, confrontando-as.

	Criptosistema de Chave Simétrica	**Criptosistema de Chave Assimétrica**
Velocidade	Alta	Baixa
Confiabilidade	Boa	Muito Boa
Nível de Segurança	Alto	Alto
Requer uma Terceira Parte Confiável	Algumas vezes	Sempre
Quantidade de Chaves Usadas	Uma	Duas

Tabela 5: Comparação de Algoritmos Simétricos e Assimétricos

6.2 – Algoritmos e tabelas comparativas

Como já sabemos, quem introduziu a idéia de chave pública foram Diffie e Hellman em 1976. A primeira realização de sistema de chave pública veio em 1977, com Rivest, Shamir e Adleman, o qual denominou-se RSA. Desde então muitos sistemas de chaves publicas foram propostos. Nem todos destinam-se a cifragem de mensagens, alguns orientam-se mais à distribuição de chaves, outros somente à assinatura digital. Dentre estes, podemos citar: RSA, Merkle-Hellman Knapsack, McEliece, ElGamal, Chor-Rivest, Curvas Elípticas, Feige-Fiat-Shamir, Guillou-Quisquater etc…

Algoritmo	**Baseia-se em**	**Uso**	**Ano**
Diffie-Hellman	problema de logaritmos discretos	Distribuição de chaves	1976
Merkle-Hellman	problema de somar sub-conjuntos	Cifragem	1978
RSA - Rivest-Shamir-Adleman	baseado em fatoração de inteiros	Cifragem + Assinatura	1978

Tabela 6: Problemas Computacionais x Algoritmos

Mais algoritmos...

Algoritmo	Ano	Aspecto Positivo	Aspecto Negativo
Feige-Fiat-Shamir	1988	Tempo de assinatura melhorado	- Tempo de verificação degradado - Tamanho de chave grande
Guillou-Quisquater	1988	Tempo de assinatura melhorado	- Tempo de verificação degradado

Tabela 7: Esquemas de Assinatura relacionados ao RSA

A tabela abaixo mostra as características, em geral, dos algoritmos assimétricos, com sua descrição e criptoanálise:

Algoritmo	Descrição	Criptoanálise	Patente
KNAPSACK Hellman e Merkle, 1978	Baseado no problema NP-completo.	Todas as variantes deste algoritmos são inseguras.	EUA e Europa.
MCELICE Robert McElice, 1978	Baseado na teoria de codificação algébrica. De duas a três vezes mais rápido do que o RSA. Chave pública muito grande: 2^{19} bits de comprimento. O texto cifrado é duas vezes maior que o texto claro original.	Princípio semelhante ao algoritmo KNAPSACK. Contudo, nenhum ataque bem sucedido contra este algoritmo. Pouco divulgado.	Não patenteado.
ELGAMAL Elgamal, 1985	Pode ser usado tanto para gerar assinaturas digitais como para cifrar dados. Baseado no problema do logaritmo discreto. Mesmo princípio do promeiro algoritmo assimétrico proposto por Diffie-Hellman.	Intratabilidade do problema do logaritmo discreto em um corpo finito.	Não patenteado.

continuação

Algoritmo	Descrição	Criptoanálise	Patente
DSA Nist, 1991 (proposto)	Algoritmo para gerar assinaturas digitais proposto pelo governo dos EUA. Esquema de assinaturas baseado no problema do logaritmo discreto. Semelhante ao algoritmo de ELGAMAL e SCHNORR.	Intratabilidade do problema do logaritmo discreto em um corpo finito.	Patenteado nos EUA; licença livre. Pendência em relação às patentes: Diffie-Hellman, Merkle-Helman e Schnorr.
ECC Criptosistema de Curvas Elípticas	Sistema de criptografia assimétrica definidos sobre corpos compostos de pontos de uma curva elíptica.	Intratabilidade do problema do logaritmo discreto em grupos aritméticos definidos sobre os pontos de uma curva elíptica. Oferecem o mesmo nível de segurança que os sistemas baseados em corpos de inteiros, mas com tamanho de chave menor. Um ECC de 160 bits equivale a um sistema RSA de 1024 bits.	Algumas patentes para cálculos específicos e otimizações.

Tabela 8: Descrição dos Problemas, suporte a criptoanálise e patente no mundo

6.3 – Performance

Vamos comparar a performance do RSA com o DSA e ECDSA. Vamos assumir que:

1024-bits RSA ≡ 1024-bits DSA com subgrupos 160-bits ≡ 160-bits ECC sobre GF(p)

6.3.1 – RSA

Considere que t seja o tempo para a multiplicação de 1024-bits e que verificação $e = 2^{16} + 1$ requer tempo de 17t.

Então teremos que o tempos de assinatura em:

- regular:

$(1024 * 3/2) * t = 1536\ t$

- com CRT (Chinese Remainder Theorem):

$2 * (3/2 * 512) * (512/1024)^2 * t = 384\ t$

6.3.2 – DSA

Assinatura: $160 * 3/2 * t = 240\ t$

Já para a verificação, computar $2 * 160 * 3/2 * t = 480\ t$

6.3.3 – ECC

Antes de compararmos RSA com curvas elípticas (ECC), vamos comparar multiplicações 1024 e 160-bits.

$(1024 / 160)^2 \equiv 40$, portanto uma multiplicação 1024 bits ≡ 40 x multiplicação 160-bits.

Assumindo que uma adição de curvas elípticas ≡ 10 multiplicações, teremos:

Assinatura:

$160 * 3/2 * 10 * 1/40 * t = 60t$

Verificação:

$2 * 160 * 3/2 * 10 * 1/40 * t = 120\ t$

	Curvas Elípticas	RSA com 1024-bit n, $e=2^{16}+1$, e CRT	Sistemas de Logaritmos discretos com primo de 1024-bit	Diffie-Hellman
Encriptar	120	17	480	480
Decriptar	60	384	240	480
Assinar	60	384	240	480
Verificar	120	17	480	480

* os valores da tabela são medidas em unidades de tempo necessárias para completar uma determinada operação se nós assumirmos que uma multiplicação modular de 1024 bits requer uma unidade de tempo.
Nesta tabela podemos verificar que o RSA é mais rápidos que os demais para: encriptar e verificar, mas perde na assinatura e decriptação. (somente ficando a frente de Diffie-Hellman)
* Via Internet, consulte http://www.rsasecurity.com/rsalabs/technotes/elliptic_curve.html.

Devido à grande importância das curvas elípticas em competição ao RSA, vamos destacar um pouco mais este comparativo, como abaixo...

6.3.4 - RSA x Curvas Elípticas

Comparando o RSA com as curvas elípticas, a grande diferença seria que as curvas elípticas podem prover o mesmo nível de segurança com chaves menores. Por exemplo: uma chave de 160-bits de curvas elípticas oferece a mesma segurança que uma chave de 1024 bits do sistema RSA ou logaritmos discretos. Portanto, o comprimento da chave publica e privada é muito menor em criptosistemas de curvas elípticas.

Curvas elípticas são mais rápidas que os sistemas de logaritmo discreto. São mais rápidas que o RSA na assinatura e decriptamento, mas são mais lentos para encriptar e verificar assinaturas.

A tabela abaixo mostra os eventos da evolução do cripto-sistema RSA e das curvas elípticas.

Data	**RSA**	**ECC**
1977	RSA proposto	
1985	Algoritmo de fatoramento quadrático torna-se cada vez mais prático.	Proposto o primeiro uso de curvas elípticas
1991		Redução do problema do algoritmo dos logaritmos discretos das curvas elípticas para um algoritmo sub-exponencial para algumas curvas.
1993	Melhoria no algoritmo de fatoramento sub-exponencial.	
1994		Algoritmo sub-exponencial sobre curvas hiper-elípticas

Para o RSA, o tamanho do problema é o tamanho do módulo que deve ser fatorado. Já nas curvas elípticas, o tamanho do problema é o número de pontos N no grupo em que se está trabalhando.

Conclusão

Vimos que existem dois métodos de criptografia: Criptografia Simétrica (que usa apenas uma chave) e a Criptografia Assimétrica (chaves-públicas e privadas).

Criptografia serve para vários propósitos, desde a privacidade nas comunicações até a troca de chaves e assinaturas digitais.

Entre os algoritmos simétricos temos o DES, ElGamal, AES, etc, algoritmos para assinatura digital como RSA, DSA, ElGamal, etc e algoritmos de chaves públicas tais ECC (Curvas Elípticas), RSA, etc

Aqui mostramos que para cifrar uma mensagem usamos o algoritmo $C = M^e \bmod n$ e para decriptografia $M = C^d \bmod n$.

Entre os ataques ao RSA estão os ataques temporais, ataques matemáticos e ataques de força bruta, mas nenhum deles até o momento encontrou um método eficaz em tempo hábil para quebrar o RSA.

Concluindo, há mais de 20 anos, muitos tentaram inverter o RSA, porém nenhum ataque importante foi encontrado além da força bruta que na maioria das vezes não vale a pena devido à necessidade de gigantesco processamento computacional para faze-lo, tornando muitas vezes o sistema que fará o ataque custa muito mais do que os dados dentro do texto cifrado, desencorajando os invasores.

Bibliografia

Livros:

Cryptography and Network Security –
Principles and Practice –
Second Edition – William Stallings – Cap. 6

Handbook of Applied Cryptography
A Menezes, P van Oorschot and S. Vanstone
CRC Press, 1996 - Cap. 1, 3, 8, 9, 11 e 13

Tese:

Uso de Cifragem para Proteção em Canais Abertos
Jorge Manuel Lopes Santos
Faculdade de Ciências da Universidade do Porto
http://www.fc.up.pt/cmup/monograph/jmlsantos_mestrado.pdf

Papers:

Cryptanalysis of RSA with Private Key d Less than $N^{0,292}$
Dan Boneh and Glen Durfee

Factoring $N = p^r q$ for large r
Dan Boneh, Glen Durfee and Nich Howgrave Graham

RSA FAQ - May 2000
RSA Laboratories
http://www.rsasecurity.com/rsalabs/

History of RSA
Jonathan Bannet, Justin Brickell, Chad Burwick, Carol Chen, Joe Montgomery, Tim Oberg
http://www.owlnet.rice.edu/~jbannet/finalpresentation.ppt

AES Proposal: Rijndael
Joan Daemen and Vicent Rijmen
http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf

Twenty Years of Attacks on the RSA Cryptosystem
Dan Boneh
http://crypto.stanford.edu/~dabo/abstracts/RSAattack-survey.html

Internet Sites:

http://www.isg.rhul.ac.uk/msc/teaching/opt8/week10.pdf
http://www.rsasecurity.com/rsalabs/technotes/elliptic_curve.html
http://www.rsasecurity.com/rsalabs/pkcs/index.html
http://www.rsasecurity.com/rsalabs/
http://www.redepegasus.com.br/abuse/
http://www.wedgetail.com/technology/pkcs.html
http://ce.mdic.gov.br/
http://www.di.ufpe.br/~flash/ais98/cripto/criptografia.htm
http://www.digitrust.com.br/AssinaturaDigital.doc
http://www.modulo.com.br
http://ece.gmu.edu/courses/ECE636/viewgraphs/s02_lecture7_pub_survey_3.pdf
http://www.vectorsite.net/ttcode1.html
http://www.numaboa.com/criptologia/matematica/estatistica/freqPortBrasil.php
http://home.pacbell.net/tpanero/crypto/dsa.html
http://www.wikipedia.org/wiki/Miller-Rabin_primality_test