

Fast Variants of RSA

Dan Boneh
dabo@cs.stanford.edu

Hovav Shacham
hovav@cs.stanford.edu

Abstract

We survey three variants of RSA designed to speed up RSA decryption. These variants are backwards compatible in the sense that a system using one of these variants can interoperate with a system using standard RSA.

1 Introduction

RSA [11] is the most widely deployed public key cryptosystem. It is used for securing web traffic, e-mail, and some wireless devices. Since RSA is based on arithmetic modulo large numbers it can be slow in constrained environments. For example, 1024-bit RSA decryption on a small handheld device such as the Palm III can take as long as 40 seconds. Similarly, on a heavily loaded web server, RSA decryption significantly reduces the number of SSL requests per second that the server can handle. Typically, one improves RSA's performance using special-purpose hardware. Current RSA coprocessors can perform as many as 10,000 RSA decryptions per second (using a 1024-bit modulus) and even faster processors are coming out.

In this paper we survey four simple variants of RSA that are designed to speed up RSA decryption in software. Throughout the paper we focus on a 1024-bit RSA modulus. We emphasize backwards compatibility: A system using one of these variants for fast RSA decryption should be able to interoperate with systems that are built for standard RSA; moreover, existing Certificate Authorities must be able to respond to a certificate request for a variant-RSA public key.

The security of these variants is an open problem. We cannot show that an attack on any of these variants would imply an attack on the standardized version of RSA (as described, e.g., in ANSI X9.31). Therefore, when using these variants, one can only rely on the fact that so far none of them has been shown to be weak. In other words, Use at your own risk.

We begin the paper with a brief review of RSA. We then describe the following variants for speeding up RSA decryption:

- Batch RSA [8]: do a number of RSA decryptions for approximately the cost of one.
- Multi-factor RSA [6, 14]: use a modulus of the form $N = pqr$ or $N = p^2q$.
- Rebalanced RSA [17]: speed up RSA decryption by shifting most of the work to the encrypter.

The RSA trapdoor permutation is used for both public key encryption and digital signatures. Since the exact application of RSA is orthogonal to discussion in this paper we use terminology consistent with the application to public key encryption. All the RSA variants we discuss apply equally well to digital signatures.

1.1 Review of the basic RSA system

We review the basic RSA public key system; refer to [10] for more information. We describe three constituent algorithms: key generation, encryption, and decryption.

Key generation: The key generation algorithm takes a security parameter n as input. We use $n = 1024$ as the standard security parameter. One generates two $(n/2)$ -bit primes, p and q , and sets $N \leftarrow pq$. Next, one picks some small value e that is relatively prime to $\varphi(N) = (p-1)(q-1)$. The value e is called the encryption exponent, and is usually chosen as $e = 3$ or $e = 65537$. The RSA public key consists of the two integers $\langle N, e \rangle$. The RSA private key is an integer d satisfying $e \cdot d = 1 \pmod{\varphi(N)}$. Typically, one sends the public key $\langle N, e \rangle$ to a Certificate Authority (CA) to obtain a certificate for it.

Encryption: To encrypt a message X using an RSA public key $\langle N, e \rangle$, one first formats the bit-string X to obtain an integer M in $\mathbb{Z}_N = \{0, \dots, N-1\}$. This formatting is often done using the PKCS #1 standard [1, 9]. The ciphertext is then computed as $C \leftarrow M^e \pmod{N}$. (Other methods for formatting X prior to encryption are described elsewhere in this issue.)

Decryption: To decrypt a ciphertext C the decrypter uses its private key d to compute M , the e th root of C in \mathbb{Z}_N , given by $C^d \pmod{N}$. Since both d and N are large numbers (each 1024 bits long) this is a lengthy computation for the decrypter. The formatting operation from the encryption algorithm is then reversed to obtain the original bit-string X from M . Unless d is taken as a large number (on the order of N), the RSA system is insecure [3, 17].

It is standard practice to employ the Chinese Remainder Theorem (CRT) for RSA decryption. Rather than compute $M \leftarrow C^d \pmod{N}$, one evaluates:

$$M_p \leftarrow C_p^{d_p} \pmod{p} \qquad M_q \leftarrow C_q^{d_q} \pmod{q}$$

Here $d_p = d \pmod{p-1}$ and $d_q = d \pmod{q-1}$. Then one uses the CRT to calculate M from M_p and M_q . This is approximately four times as fast as evaluating $C^d \pmod{N}$ directly [10, p. 613].

2 Batch RSA

Fiat [8] observed that, when using small public exponents e_1 and e_2 , it is possible to decrypt two ciphertexts for approximately the price of one. Suppose C_1 is a ciphertext obtained by encrypting some M_1 using the public key $\langle N, 3 \rangle$, and C_2 is a ciphertext for some M_2 using $\langle N, 5 \rangle$. To decrypt, we must compute $C_1^{1/3}$ and $C_2^{1/5} \pmod{N}$. Fiat observed that by setting $A = (C_1^5 \cdot C_2^3)^{1/15}$ we obtain:

$$C_1^{1/3} = \frac{A^{10}}{C_1^3 \cdot C_2^2} \quad \text{and} \quad C_2^{1/5} = \frac{A^6}{C_1^2 \cdot C_2} \tag{1}$$

At the cost of computing a single 15th root and some additional arithmetic, we are able to decrypt both C_1 and C_2 . Computing a 15th root takes the same time as a single RSA decryption.

This batching technique is only worthwhile when the public exponents e_1 and e_2 are small (e.g., 3 and 5). Otherwise, the extra arithmetic required is too expensive. Also, one can only batch-decrypt ciphertexts encrypted using the same modulus and *distinct public exponents*. This

is essential. In fact, it is known [12, Appendix A] that one cannot apply such algebraic techniques to batch the decryption of two ciphertexts encrypted with the same key (e.g., of $C_1^{1/3}$ and $C_2^{1/3}$).

Fiat generalized the above observation to the decryption of a batch of b RSA ciphertexts. We have b distinct and pairwise relatively prime public keys e_1, \dots, e_b , all sharing a common modulus N . Furthermore, we have b encrypted messages C_1, \dots, C_b , where C_i is encrypted using the exponent e_i . We wish to compute $M_i = C_i^{1/e_i}$ for $i = 1, \dots, b$.

Fiat describes this b -batch process using a binary tree. For small values of b ($b \leq 8$), one can use a direct generalization of (1). One sets $e \leftarrow \prod_i e_i$, and $A_0 \leftarrow \prod_i C_i^{e/e_i}$ (where the indices range over $1, \dots, b$). Then one calculates $A \leftarrow A_0^{1/e} = \prod_{i=1}^b C_i^{1/e_i}$. For each i , one uses the CRT to find a number x_i satisfying $x_i \equiv 1 \pmod{p_i}$ and $x_i \equiv 0 \pmod{p_j}$ (for $j \neq i$). Then

$$M_i = C_i^{1/e_i} = \frac{A^{x_i}}{C_i^{(x_i-1)/p_i} \cdot \prod_{j \neq i} C_j^{x_i/p_j}} \quad (2)$$

This b -batch requires b modular inversions; Fiat's tree based method requires $2b$ modular inversions, but fewer auxiliary multiplications.

2.1 Improving the performance of batch RSA

In [12] the authors show how to use batch RSA within the Apache web server to improve the performance of the SSL handshake. This requires changing the web server architecture. They also describe several natural improvements to batch RSA. We mention a few of these improvements here.

Batch division: Modular inversion is much slower than modular multiplication. We use a trick due to Montgomery to compute all b inversions in the batch algorithm for the cost of a single inversion with a few more multiplications. The idea is: To invert x and y we compute $\alpha \leftarrow (xy)^{-1}$ and then set $x^{-1} \leftarrow y \cdot \alpha$ and $y^{-1} \leftarrow x \cdot \alpha$, obtaining inverses of both numbers at the cost of a single modular inverse and some additional multiplications. More generally, we use the following fact [5, p. 481]:

Fact. *Let x_1, \dots, x_n be elements of \mathbb{Z}_N . All n inverses $x_1^{-1}, \dots, x_n^{-1}$ can be obtained at the cost of one inversion and $3n - 3$ multiplications.*

Consequently, only a single modular inversion is required for the entire batching procedure.

Global Chinese Remainder: In Section 1.1 we mentioned that RSA decryption uses the CRT to speed up the computation of $C^d \pmod{N}$. This idea extends naturally to batch decryption. We run the batching algorithm modulo p , and again modulo q , then use the CRT on each of the b pairs $\langle C_i^{1/e_i} \pmod{p}, C_i^{1/e_i} \pmod{q} \rangle$ to obtain the b decryptions $M_i = C_i^{1/e_i} \pmod{N}$.

Simultaneous Multiple Exponentiation: Simultaneous multiple exponentiation [10, §14.6] is a method for calculating $a^u \cdot b^v \pmod{m}$ without first evaluating a^u and b^v . It requires approximately as many multiplications as does a single exponentiation with the larger of u or v as exponent. Such products of exponents are a large part of the batching algorithm. Simultaneous multiple exponentiation cuts the time required to perform them by close to 50%.

2.2 Performance of batch RSA

Table 1 lists the running time for standalone batch-RSA decryption, using OpenSSL 0.9.5 on a machine with a 750 MHz Pentium III and 256 MB RAM, running Debian “Potato.” In all experiments, the smallest possible values for the encryption exponents e_i were used.

batch size	key size		
	768	1024	2048
(unbatched)	4.67	8.38	52.96
2	3.09	5.27	29.43
4	1.93	3.18	16.41
8	1.55	2.42	10.81

Table 1: RSA decryption time, in milliseconds, as a function of batch and key size

With standard 1024-bit keys, batching improves performance significantly. With $b = 4$, RSA decryption is accelerated by a factor of 2.6; with $b = 8$, by a factor of almost 3.5. Note that a batch size of more than eight is probably not useful for common applications, as waiting for many decryption requests to be queued can significantly increase latency.

batch size	Server load		
	16	32	48
(unbatched)	105	98	98
2	149	141	134
4	218	201	187
8	274	248	227

Table 2: SSL handshakes per second as a function of batch size. 1024 bit keys.

We also consider the batch-RSA performance as a component of a larger system—a web server handling SSL traffic. An architecture for such a system is described in [12]; the challenge is to choose, from amongst the queued requests, the batch to perform. Table 2 gives the number of SSL handshakes per second that the batch-RSA web server can handle, when bombarded with concurrent HTTP HEAD requests by a test client. Here “server load” is the number of simultaneous connections the client makes to the server. Under heavy load, batch RSA can improve the number of SSL handshakes per second by a factor of approximately 2.5.

2.3 The Downside of Batch RSA

Batch RSA can lead to a significant improvement in RSA decryption time. Nevertheless, there are a few difficulties with using the batching technique:

- When using batch RSA, the decryption server must maintain at least as many RSA certificates as there are distinct keys in a batch. Unfortunately, current CAs charge per certificate regardless of the public key in the certificate.
- For optimal performance, batching requires RSA public keys with very small public exponents ($e = 3, 5, 7, 11, \dots$). There are no known attacks on the resulting system, but RSA as usually deployed uses a larger public exponent ($e = 65537$).

3 Multi-factor RSA

The second RSA variant is based on modifying the structure of the RSA modulus. Here there are two proposals. The first, patented by Compaq [6], uses a modulus of the form $N = pqr$. When N is 1024 bits, each prime is approximately 341 bits. We refer to this as multi-prime RSA. The second, proposed by Takagi [14] and patented by NTT [15], uses RSA moduli of the form $N = p^2q$ and leads to an even greater speedup.

We begin with multi-prime RSA. We describe key generation, encryption, and decryption. We then discuss the performance of the scheme and analyze its security.

Key generation: The key generation algorithm takes as input a security parameter n and an additional parameter b . It generates an RSA public/private key pair as follows:

Step 1: Generate b distinct primes p_1, \dots, p_b each $\lfloor n/b \rfloor$ -bits long. Set $N \leftarrow \prod_{i=1}^b p_i$. For a 1024-bit modulus we can use at most $b = 3$ (i.e., $N = pqr$), for security reasons discussed below.

Step 2: Pick the same e used in standard RSA public keys, namely $e = 65537$. Then compute $d = e^{-1} \bmod \varphi(N)$. As usual, we must ensure that e is relatively prime to $\varphi(N) = \prod_{i=1}^b (p_i - 1)$.

The public key is $\langle N, e \rangle$; the private key is d .

Encryption: Given a public key $\langle N, e \rangle$, the encrypter encrypts exactly as in standard RSA.

Decryption: Decryption is done using the Chinese Remainder Theorem (CRT). Let $d_i = d \bmod p_i - 1$. To decrypt a ciphertext C , one first computes $M_i = C^{d_i} \bmod p_i$ for each i , $1 \leq i \leq b$. One then combines the M_i 's using the CRT to obtain $M = C^d \bmod N$. The CRT step takes negligible time compared to the d exponentiations.

Performance We compare the decryption work using the above scheme to the work done when decrypting a normal RSA ciphertext. Recall that standard RSA decryption using CRT requires two full exponentiations modulo $n/2$ -bit numbers. In multi-prime RSA decryption requires b full exponentiations modulo n/b bit numbers. Using basic algorithms computing $x^d \bmod p$ takes time $O(\log d \log^2 p)$. When d is on the order of p the running time is $O(\log^3 p)$. Therefore, the speedup of multi-prime RSA over standard RSA is simply:

$$\frac{2 \cdot (n/2)^3}{b \cdot (n/b)^3} = b^2/4$$

For 1024-bit RSA, we can use at most $b = 3$ (i.e., $N = pqr$), which gives a speedup of approximately 2.25 over standard RSA.

Security The security of multi-factor RSA depends on the difficulty of factoring integers of the form $N = p_1 \cdots p_b$ for $b > 2$. The fastest known factoring algorithm (the number field sieve) does not take advantage of this special structure of N . However, one has to make sure that the prime factors of N do not fall within the capabilities of the Elliptic Curve Method (ECM), which is analyzed in SW93. Currently, 256-bit prime factors are considered within the bounds of ECM, since the work to find such factors is within range of the work needed for the RSA-512 factoring project. Consequently, for 1024-bit moduli one should not use more than three factors.

3.1 Multi-power RSA: $N = p^{b-1}q$

One can further speed up RSA decryption using moduli of the form $N = p^{b-1}q$ where p and q are n/b bits each [14]. When N is 1024-bits long we can use at most $b = 3$, i.e., $N = p^2q$. The two primes p, q are then each 341 bits long.

Key generation: The key generation algorithm takes as input a security parameter n and an additional parameter b . It generates an RSA public/private key pair as follows:

Step 1: Generate two distinct n -bit primes, p and q , and compute $N \leftarrow p^{b-1} \cdot q$.

Step 2: Use the same public exponent e used in standard RSA public keys, namely $e = 65537$. Compute $d \leftarrow e^{-1} \bmod (p-1)(q-1)$.

Step 3: Compute $r_1 \leftarrow d \bmod p-1$ and $r_2 \leftarrow d \bmod q-1$.

The public key is $\langle N, e \rangle$; the private key is $\langle p, q, r_1, r_2 \rangle$.

Encryption: Same as in standard RSA.

Decryption: To decrypt a ciphertext C using the private key $\langle p, q, r_1, r_2 \rangle$ one does:

Step 1: Compute $M_1 \leftarrow C^{r_1} \bmod p$ and $M_2 \leftarrow C^{r_2} \bmod q$; thus $M_1^e = C \bmod p$ and $M_2^e = C \bmod q$.

Step 2: Using Hensel lifting [5, p. 137] construct an M'_1 such that $(M'_1)^e = C \bmod p^{b-1}$. Hensel lifting is much faster than a full exponentiation modulo p^{b-1} .

Step 3: Using CRT, compute an $M \in \mathbb{Z}_N$ such that $M = M'_1 \bmod p^{b-1}$ and $M = M_2 \bmod q$. Then $M = C^d \bmod N$, a proper decryption of C .

Comment. *Hensel lifting in Step 2 required a modular inversion. However, some accelerator cards do not provide support for modular inversion. The API to these cards typically does modular inversion using an exponentiation: $x^{-1} = x^{p^d - p^{d-1} - 1} \pmod{p^d}$. Unfortunately, using an exponentiation to do Hensel lifting greatly diminished the gains of this method over the multi-prime approach.*

Performance We compare the work required to decrypt using multi-power RSA to that required for standard RSA. For multi-power RSA, decryption takes two full exponentiations modulo (n/b) -bit numbers, and $b-2$ Hensel liftings. Since the Hensel-lifting time is negligible, we focus on the time for the two exponentiations. As noted before, a full exponentiation is cubic in the size of the modulus, so the speedup of multi-prime RSA over standard RSA is simply:

$$\frac{2 \cdot (n/2)^3}{2 \cdot (n/b)^3} = b^3/8$$

For 1024-bit RSA, b should again be at most three (i.e., $N = p^2q$), giving a speedup of approximately 3.38 over standard RSA.

Security The security of multi-power RSA depends on the difficulty of factoring integers of the form $N = p^{b-1}q$. As for multi-prime RSA, one has to make sure that the prime factors of N do not fall within the capabilities of ECM. Consequently, for 1024-bit moduli one can use at most $b = 3$, i.e., $N = p^2q$. We note that, although the Lattice Factoring Method (LFM) of Boneh, Durfee, and Howgrave-Graham [4] is designed to factor integers for the form $N = p^u \cdot q$ for large u , it cannot factor integers of the form $N = p^2q$ when N is 1024 bits long.

4 Rebalanced RSA

In standard RSA, encryption and signature verification are much less processor-intensive than decryption and signature generation. In some applications, one would like to have the reverse behavior. For example, when a cell phone needs to generate an RSA signature that will be later verified on a server one would like signing to be easier than verifying. Similarly, for SSL, web browsers (doing encryption) typically have idle cycles to burn whereas web servers (doing decryption) are overloaded. In this section we describe a variant of RSA that enables us to rebalance the difficulty of encryption and decryption. It is based on a proposal by Wiener [17] (see also [2]). Note that we cannot simply speed up RSA decryption by using a small value of d since as soon as d is less than $N^{0.292}$ RSA is insecure [17, 3]. As before, we describe key generation, encryption, and decryption.

Key generation The key generation algorithm takes two security parameters n and k where $k \leq n/2$. It generates an RSA key as follows:

Step 1: Generate two distinct $(n/2)$ -bit primes p and q with $\gcd(p-1, q-1) = 2$. Compute $N \leftarrow pq$.

Step 2: Pick two random k -bit values r_1 and r_2 such that

$$\gcd(r_1, p-1) = 1 \quad \text{and} \quad \gcd(r_2, q-1) = 1 \quad \text{and} \quad r_1 = r_2 \pmod{2}$$

Step 3: Find a d such that $d = r_1 \pmod{p-1}$ and $d = r_2 \pmod{q-1}$.

Step 4: Compute $e \leftarrow d^{-1} \pmod{\varphi(N)}$. The public key is $\langle N, e \rangle$; the private key is $\langle p, q, r_1, r_2 \rangle$.

Steps 3 and 4 require some explanation. First, we explain how to find d in Step 3. One usually uses the Chinese Remainder Theorem (CRT). Unfortunately, $p-1$ and $q-1$ are not relatively prime (they are both even) and consequently the theorem does not apply. However, $(p-1)/2$ is relatively prime to $(q-1)/2$. Furthermore, $r_1 = r_2 \pmod{2}$. Let $a = r_1 \pmod{2}$. Then using CRT we can find an element d' such that

$$d' = \frac{r_1 - a}{2} \pmod{\frac{p-1}{2}} \quad \text{and} \quad d' = \frac{r_2 - a}{2} \pmod{\frac{q-1}{2}}$$

Now, observe that the required d in Step 3 is simply $d = 2d' + a$. Indeed, $d = r_1 \pmod{p-1}$ and $d = r_2 \pmod{q-1}$.

In Step 4, we must justify why d is invertible modulo $\varphi(N)$. Recall that $\gcd(r_1, p-1) = 1$ and $\gcd(r_2, q-1) = 1$. It follows that $\gcd(d, p-1) = 1$ and $\gcd(d, q-1) = 1$. Consequently $\gcd(d, (p-1)(q-1)) = 1$. Hence, d is invertible modulo $\varphi(N) = (p-1)(q-1)$.

Typically, we take $k = 160$, although other larger values are acceptable. Note that e is very large — on the order of N . This is unlike standard RSA, where e typically equals 65537. All CAs we tested were willing to generate certificates for such RSA public keys.

Encryption: Encryption using the public key $\langle N, e \rangle$ is identical to encryption in standard RSA. The only issue is that since e is much larger than in standard RSA, the encrypter must be willing to accept such public keys. At the time of this writing all browsers we tested were willing to accept such keys. Except Microsoft’s Internet Explorer (IE). IE allows a maximum of 32 bits for e .

Decryption: To decrypt a ciphertext C using the private key $\langle p, q, r_1, r_2 \rangle$ one does:

Step 1: Compute $M_1 \leftarrow C^{r_1} \bmod p$ and $M_2 \leftarrow C^{r_2} \bmod q$.

Step 2: Using the CRT compute an $M \in \mathbb{Z}_N$ such that $M = M_1 \bmod p$ and $M = M_2 \bmod q$. Note that $M = C^d \bmod N$. Hence, the resulting M is a proper decryption of C .

Performance We compare the work required to decrypt using the above scheme to that required using standard RSA. Recall that decryption time for standard RSA with CRT is dominated by two full exponentiations modulo $(n/2)$ -bit numbers. In the scheme presented above, the bulk of the decryption work is in the two exponentiations in Step 1, but in each of these the exponent is only k bits long. Since modular exponentiation takes time linear in the exponent’s bit-length, we get a speedup of $(n/2)/k$ over standard RSA. For a 1024-bit modulus and 160-bit exponent, this is a factor of 3.20.

Security It is an open problem whether RSA using values of d as above is secure. Since d is large, the usual small- d attacks [17, 3] do not apply. We present the best known attack on the scheme.

Lemma. *Let $\langle N, e' \rangle$ be an RSA public key with $N = pq$. Let $d \in \mathbb{Z}$ be the corresponding RSA private exponent satisfying $d = r_1 \bmod p - 1$ and $d = r_2 \bmod q - 1$ with $r_1 < r_2$. If r_1 is m bits long we assume that $r_1 \not\equiv r_2 \pmod{2^{m/2}}$. Then given $\langle N, e' \rangle$ an adversary can expose the private key d in time $O(\sqrt{r_1} \log r_1)$.*

Comment. *Proof.* We know that $e' = (r_1)^{-1} \bmod (p - 1)$. Suppose r_1 is m -bits long. Write $r_1 = A \cdot 2^{m/2} + B$ where A, B are in $[0, 2^{m/2}]$. Pick a random $g \in \mathbb{Z}_N$ and define the polynomial

$$G(x) = \prod_{i=0}^{2^{m/2}} (g^{e' \cdot 2^{m/2} \cdot i} \cdot x - g)$$

Note that this polynomial has degree $2^{m/2}$. Next, observe that $G(g^{e' \cdot B}) = 0 \bmod p$. This follows since one of the products above is

$$(g^{e' \cdot 2^{m/2} \cdot A} \cdot g^{e' \cdot B} - g) = g^{e' r_1} - g = 0 \pmod{p}$$

Since $r_1 \not\equiv r_2 \pmod{2^{m/2}}$ it follows that $G(g^{e' \cdot B}) \not\equiv 0 \pmod{q}$. Hence, $\gcd(N, G(g^{e' \cdot B}))$ gives a non-trivial factor of N . Hence, if we evaluate $G(x) \bmod N$ at $x = g^{e' \cdot j}$ for $j = 0, \dots, 2^{m/2}$ at least one of these values will expose the factorization of N . Evaluating a polynomial of degree $2^{m/2}$ at $2^{m/2}$

values can be done in time $2^{m/2} \cdot m/2$ using FFT methods [16]. This algorithm requires $\tilde{O}(2^{m/2})$ space. Hence, in time at most $O(\sqrt{r_1} \log r_1)$ we can factor N . \square

The above attack shows that, to obtain security of 2^{80} , we must make both r_1 and r_2 be at least 160 bits long. This explains our choice of parameter sizes for r_1 and r_2 .

5 Conclusions

We surveyed four variants of RSA designed to speed up RSA decryption and be backwards-compatible with standard RSA. Table 3 gives the speedup factors for each of these variants using a 1024-bit RSA modulus. Batch RSA is fully backwards-compatible, but requires the decrypter to obtain and manage multiple public keys and certificates. The two multi-factor RSA techniques are promising in that they are fully backwards compatible. The rebalanced RSA method gives a large speedup, but only works with peer applications that properly implement standard RSA, and so are willing to accept RSA certificates with a large encryption-exponent e . Currently, IE rejects all RSA certificates where e is more than 32 bits long. Multi-factor RSA and rebalanced RSA can be used together to give an additional speedup. Finally, all these techniques are orthogonal to work in improving the performance of the fundamental number-theoretic algorithms (e.g., modular multiplication and exponentiation) on which RSA is built.

Method	Speedup	Comment
Batch RSA	2.64	Requires multiple certificates
Multi-prime	2.25	
Multi-power	3.38	
Rebalanced	3.20	Incompatible with Internet Explorer

Table 3: Comparison of RSA variants

Acknowledgments

The authors thank Ari Juels for his comments on preliminary versions of this paper.

References

- [1] M. Bellare and P. Rogaway. “Optimal Asymmetric Encryption.” In A. De Santis, ed., *Proceedings of Eurocrypt 1994*, vol. 950 of *LNCS*, pp. 92–111. Springer-Verlag, May 1994.
- [2] D. Boneh. “Twenty Years of Attacks on the RSA Cryptosystem.” *Notices of the American Mathematical Society*, 46(2):203–213, Feb. 1999.
- [3] D. Boneh and G. Durfee. “Cryptanalysis of RSA with Private Key d Less than $n^{0.292}$.” *IEEE Trans. Information Theory*, 46(4):1339–1349, Jul. 2000.
- [4] D. Boneh, G. Durfee, and N. Howgrave-Graham. “Factoring $N = p^r q$ for Large r .” In M. Weiner, ed., *Proceedings of Crypto ’99*, vol. 1666 of *LNCS*, pp. 326–337. Springer-Verlag, Aug. 1999.

- [5] H. Cohen. *A Course in Computational Algebraic Number Theory*, vol 138 of *Graduate Texts in Mathematics*. Springer-Verlag, 1996
- [6] T. Collins, D. Hopkins, S. Langford, and M. Sabin. *Public Key Cryptographic Apparatus and Method*. US Patent #5,848,159. Jan. 1997.
- [7] T. Dierks and C. Allen. *RFC 2246: The TLS Protocol, Version 1*. Jan. 1999.
- [8] A. Fiat. “Batch RSA.” In G. Brassard, ed., *Proceedings of Crypto 1989*, vol. 435 of *LNCS*, pp. 175–185. Springer-Verlag, Aug. 1989.
- [9] RSA Labs. *Public Key Cryptography Standards (PKCS), Number 1*.
- [10] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
- [11] R. Rivest, A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public Key Cryptosystems.” *Commun. ACM*, 21(2):120–126. Feb. 1978.
- [12] H. Shacham and D. Boneh. “Improving SSL Handshake Performance via Batching.” In D. Naccache, ed., *Proceedings of RSA 2001*, vol. 2020 of *LNCS*, pp. 28–43. Springer-Verlag, Apr. 2001.
- [13] R. Silverman and S. Wagstaff Jr. “A Practical Analysis of the Elliptic Curve Factoring Algorithm.” *Math. Comp.* 61(203):445–462. Jul. 1993.
- [14] T. Takagi. “Fast RSA-type Cryptosystem Modulo p^kq .” In H. Krawczyk, ed., *Proceedings of Crypto 1998*, vol. 1462 of *LNCS*, pp. 318–326. Springer-Verlag, Aug. 1998.
- [15] T. Takagi and S. Naito. *Scheme for fast realization of encryption, decryption and authentication*. US Patent #6,396,926. Mar. 1999.
- [16] J. Turk. “Fast Arithmetic Operations on Numbers and Polynomials.” In H. Lenstra, Jr. and R. Tijdeman, eds., *Computational Methods in Number Theory, Part I*, vol. 154 of *Mathematical Centre Tracts*. Mathematisch Centrum, Amsterdam, 1982.
- [17] M. Wiener. “Cryptanalysis of Short RSA Secret Exponents.” *IEEE Trans. Information Theory* 36(3):553–558. May 1990.