Performance Evaluation of Several Efficient RSA Variants

Md. Ali-Al-Mamun[†], Mohammad Motaharul Islam^{††}, S.M. Mashihure Romman[†] and A.H. Salah Uddin Ahmad[†]

[†]Islamic University of Technology, Board Bazar, Gazipur-1704, Bangladesh ^{††}Training & Instrumentation Division, UGC, Agargaon, Dhaka-1207, Bangladesh

Summary

This paper includes survey on various research areas, such as efficient implementation of RSA which includes RSA key generation, its application, different RSA variants, faster RSA implementation and performance evaluation of those based on speed and memory consumption.

Key words:

Square & multiply method, public key, private key, CRT.

1. Introduction

The RSA Problem is now over two and half decades old. The elegant simplicity of the problem has led to numerous observations over the years, some trying to attack, others avoiding them. Public-key encryption schemes and digital signature schemes have been developed, whose strength is derived fully from the RSA Problem. The strength of RSA comes from the fact that factoring large numbers is difficult. The best-known factoring methods are still very slow.

2. RSA Cryptosystem

Before we present the evaluation result of several variants of RSA cryptosystem, we review the three basic algorithms that constitute the RSA, together with two frequently used optimization techniques. Compare to symmetric-key crypt-osystems, two key types are employed in public-key syste-ms: RSA public key (e, n)and RSA private key (d, n).

RSA - Key generation:

Algorithm 1: key generation for RSA public-key encryption

SUMMARY: each entity creates an RSA public key and a corresponding private key.
Step 1: Generate two large random (and distinct) primes p and q, each
roughly the same size.
Step 2: Compute $n = p \times q$ and $\phi = (p-1)(q-1)$.
Step 3: Select a random integer e, $l < e < \phi$, such that $gcd(e, \phi) = l$.
Step 4: Use the extended Euclidean algorithm to compute the unique
integer d, $l < d < \phi$, such that $ed \equiv l \mod \phi$.
Step 5: public key is (e, n); private key is (d, n).

The plaintext m can be encrypted with Algorithm 2:

Algorithm 2: Encryption for RSA public-key encryption

SUMMARY: encrypt a message *m* with public key *e* **Input**: *m*, *e*, *n* **Output**: *c* = *m^e* mod *n*

The cipher text c can be decrypted with Algorithm 3: Algorithm 3: Decryption for RSA public-key decryption

SUMMARY: decrypt a cipher text c with private key dInput: c, d, nOutput: $m = c^d \mod n$

This specification supports are called multi-prime RSA where the modulus may have more than two prime factors. The benefit of multi-prime RSA is lower computational cost for the decryption and signature primitives, provided that the CRT [1, 2] (Chinese Remainder Theorem) is used. Better performance can be achieved on single processor platforms, but to a greater extent on multiprocessor platforms, where the modular exponentiations involved can be done in parallel.

Another technique where secret exponent [6] *d* has binary notation $(d_{i.l}, d_{i.2}, \dots, d_l, d_0)_2$, and $d_{i-1} = 1$ denotes the most significant bit. Modular exponentiation is performed bit by bit by repeated modular multiplications. The algorithm is Square & Multiply method.

Manuscript received June 5, 2008 Manuscript revised June 20 2008 Algorithm 4: Square& multiply algorithm for exponentiation

Input: c, n, $d = (d_{i-1}, d_{i-2}, \cdots, d_{i}, d_{0})_{2}$ Output: $s=c^{d} \mod n$ Let s=c. If d = 0 then return (1). for j = i - 2 to 0 do $s = s^{2} \mod n$ if $d_{i} = = 1$ then Let $s = (s \cdot c) \mod n$. end if end for return (s)

We have used both the techniques in this paper to evaluate the performance of various known RSA variants [1, 3].

3. Results

All performance measurements were conducted on an AMD Athlon; Win XP and Linux platform, with 256 MB of RAM and using C language with GNU MP [9] (library GMP). For figure, charts we used Microsoft excel.

3.1 Speed comparison

What we can suggest after all these discussion is, we should not analyze cryptographic algorithms with a fixed key length; rather evaluate speed and memory requirements depending on the key length, so that our results won't be out of date if the recommended key length becomes larger in future. Below it is shown the speed up of different RSA implementation. First simple RSA and RSA with CRT [2]:

Speedup	Key Length		
	768	1024	2048
RSA without CRT	1.0	1.0	1.0
RSA with CRT	3.24	3.32	3.47

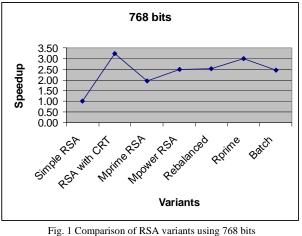
Now the comparison among other variants:

Speedup	Table 2: Major	RSA variants Key Length	
Variant	768	1024	2048
Mprime	1.95	1.89	1.97
Mpower	2.49	2.54	2.79

Rebalanced	2.52	3.02	5.98
Rprime	3.00	3.88	7.83
Batch	2.47	2.78	3.42

Practical speedup for Decryption process where b=4(no. of messages), k=3(no. of primes), s=160

For 768-bits moduli the variant that exhibits better performance would be Batch RSA, but for 1024 and 2048 bits moduli Rprime RSA presents the best performance. Notice that while the speedup of Batch, MPrime and MPower variants is fixed regardless of the size of the used moduli, speedup of the Rebalanced and the RPrime variants[11] significantly increases with larger moduli. This happens because the consideration s fixed and equal to 160 bits (remember that s is the size of the exponent used in decryption algorithm), while this exponent increases for all other variant. Here speedup comparisons are shown in the form of chart; first for 768 bits, next 1024 bits and next 2048 bits.



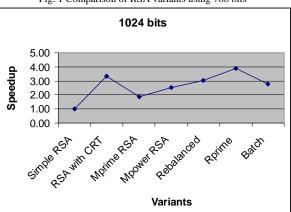


Fig. 2 Comparison of RSA variants using 1024 bits

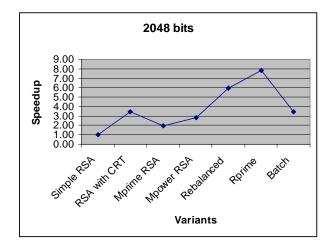


Fig. 3 Comparison of RSA variants using 2048 bits

For good encryption and decryption performance [2] and interoperability with systems that already implement the PKCS#1 [5]. We found out the use of MPrime RSA is the best. Although MPower and Batch RSA achieve better performance than MPrime and hence constitute better option when high speed is desired, they are not specified in PKCS#1. For the applications that prioritize the performa-nce the decryption and the signature generation, the best choice is RPrime RSA, which for 2048-bits moduli got a gain of 30% with relation to Rebalanced RSA and is there-fore about 27 times faster than original RSA. Besides, this variant can interoperate with systems that already use the PKCS #1 [5]. Another fact that favors this variation is that current systems that use MPrime RSA can easily be adap-ted to it, it is enough to modify the key generation algori-thm or create a hybrid key system.

3.2 Memory Comparison

We will show now the memory requirement of different RSA implementation [2, 7]. Below it is shown simple RSA and RSA with CRT implementation.

Table 3: With CRT and without CRT		
	Total Memory	
RSA without CRT	4n or 4096 bits	
RSA with CRT	8n or 8192 bits	
Here n=1024 bits		

Here memory requirement of other variants of RSA are shown in tabular as well as in the form of chart:

Table 4: Major RSA variants		
	Total Memory	
Mprime $(p r q)$	26n/3 bits=8875 bits	
Mpower (p^2q)	25n/3 bits=8534 bits	
Rebalanced (k=160)	7n+2k bits =7200 bits	
Batch (b=4)	67n/2+784 bits=35088 bits	
Here n=1024 bits		

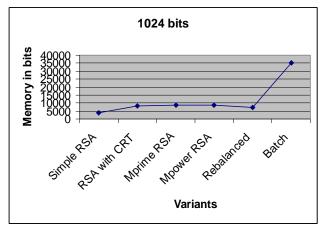


Fig. 4 Comparison of memory of RSA variants using 1024 bits

The idea of reducing the decryption time in detriment of the encryption, used by Rebalanced RSA and Rprime RSA [2], seems first sight not to present advantages in practical terms. However, there are applications where the balancing characteristic of these algorithms is desirable. Consider, for instance, a situation where the signature generation is executed much more often than verification. A bank, for example, can emit many digital signatures in a single day (in documents, receipts), while the user that receives this signature, has usually a much smaller burden. In this situation is reasonable to transfer the computational effort demanded for the signatures generation to the party verifying them.

Another example is provided by applications running on handheld devices (PDAs), which generally possess limited computational resources. In communications with servers (or even with notebooks or desktop computers), we could leave the task of decryption (fast) for the small device, and the encryption (slow) for the computers with more computational resources. A still better alternative would be to use an implementation of MPrime RSA with keys of the MPrime and RPrime RSA, with the use depending on the type of communication (desktop/desktop, or desktop/handheld), in other words, to use a scheme of hybrid keys.

4. Conclusion

The main security parameter of current cryptographic algorithms is the key length. We compare the different decryption schemes that have been described in terms of speed and memory. A big key will ensure a high level of security but the operations will also take a long time and consume a lot of memory. Currently, the recommended key length for the big standard cryptosystem RSA is more [2, 4, 7] than 512 bits. If 1024 bits or 2048 bits could be used that is safer. This key length is believed to be secure regarding the current computing abilities of our computers. But this length will probably be sooner or later too short.

References

- [1] D. Boneh and H. Shacham. Fast Variants of RSA. RSA Laboratories, 2002.
- [2] Cesar Alison," An efficient variant of the RSA cryptosystem" 2002.
- [3] M. Jason Hinek. Low Public Exponent Partial Key and Low private Exponent Attacks on Multi-prime RSA. Master Thesis, Waterloo, Ontario - Canada, 2002.
- [4] D. Boneh and G.Durfee. Cryptoanalysis of RSA with private key d less than n^{0.292}.IEEE Trans. on Info. Th., 46(4):1339– 1349, Jul. 2000.
- [5] RSA Laboratories. PKCS#1 v. 2.1: RSA cryptography standard. June 2002.
- [6] Camille Vuillaume. Efficiency Comparison of Several RSA Variants Master Thesis, Fachbereich Informatik der TU-Darmstadt, 2003.
- [7] T. Takagi. Fast RSA-type cryptosystem modulo p^{k} q. In Advances in cryptology, volume 1462, pages 318–326. Springer-Verlag, 1998.
- [8] R. L. Rivest, "RSA Problem" 2003
- [9] GNU MP. GMP. Version 4.1 -2002, available in http: //www. swox.com/gmp/.



Md. Ali-Al-Mamun received the B.Sc. in Computer Science and IT from Islamic University of Technology, Bangladesh in 2002 and Masters Degree in Computer Applications from University of Hyderabad, India in 2006. He now working as a Lecturer in Computer Science & IT department in

IUT, Bangladesh. In addition he is the member of Institute of Engineers Bangladesh (IEB).



Mohammad Motaharul Islam is senior lecturer and course coordinator in Computer Science and Engineering program at Training and Instrumentation Division of University Grants Commission of Bangladesh. He holds Bachelor degree in Computer Science and Information Technology from the Islamic

University of Technology (IUT), a subsidiary organ of Organization of Islamic Conference (OIC) in 2002. In addition he is the member of Institute of Engineers Bangladesh (IEB) and Bangladesh Computer Society (BCS). Mr. Motahar has around six years of experience in teaching and IT-related projects. Mr. Motahar supervises postgraduate and undergraduate students in high-speed networking, Open source operating system, information management and human computer interaction. He also leads and teaches modules at both BSc and postgraduate diploma levels in Computer Networking, operating system etc.



S. M. Mashihure Romman is pursuing B.Sc. in Computer Science and IT in Islamic University of Technology, Bangladesh. He is now in final semester of his final year. He has plan to continue his higher study in the relevant field of Computer Science.



A. H. Salah Uddin Ahmad is doing B.Sc. in Computer Science and IT in Islamic University of Technology, Bangladesh. He is now in final semester of his final year. He is very eager to continue his higher study in the relevant field of Computer Science.

Appendix A

RSA Key generation examples

1024-bits

e=9a6519976940da2189796fbc1fdc8611d0b74 233b2d784ab03c86dd89bbd8631b4731167e8ae d28fca657958c395eac8f7f5839f82def66bde4 36836f103c98b37dfd68e5d5aa5eee529327ada 13c12b764bdc10fc152105cf3e9d57cda97acf5 70386976bdee4d12b15d133f3a3e62b00d80248 bd066a7a74e82df2e7f972f

d=99b1313d4c61d22a73e7e81ef282f3e2c4141 4c1d0a7e8ab28e3ed18a986876e344170640091 a7e87c46112bd6bbdf6694db4d48a9b755ac02b 70c4331597cf2b12c804484ca3ca9d9b25cc8ea f8364f48d5c11ae51d2f94607056c363f44845e 34b3dd5fbd1ac7c532fa3fbc86b7be60ab99fed d7b5b562c4f20f506a3ed4f

n=c6f4db66d824d75d5ee8dbd0fa6911ad109c3 526ee5efbe5fa4d8dbdf1be1c8ccb5b28beecb6 870eef55913865b50952e99a56ca5826e82578a 95fbeecfa33a02465ee2598e56cf6c50ac20f0e 16c4080051d3e7dacf34273147ba39447a98fc5 7f3044da3b6dd4c784ac29374b61cffe4530cc6 48da5f870c29eb3258940ad

2048-bits

e=128b2c987d7b61a13cd19c5d2a11fff8a3d8e 4b0957b910761e0c45a24f6a447bb9e3ee5cd87 c84cb2750f5cd169ef2d1280b7dc976f4107f5f f328ac1d392884fadc657ea8868c709a1e5bd0d 5205798ca38be9c0fd6cdeef66cdd7acd65ec35 5a525eb9b98e5b5da569258b352cf4d309a231b de159ade344c1ed944e1f37e0e91b800f2ff339 85de7dfc7986a2a7044517bdda75125cc85c517 8ed51551fd702c36617cb71f05fbdfcf2a73f55 06aff720ddfd93ec6cbe203a76def6a7c54f1e6 d46660fccdf17b3d5210fb16ca9e9015746cc2d 1ec1ff6a2d9259ceb2d84c1902c275ec02c4ec6 443e6fa347c4aa874b46cb580817a58b941610b eae0e9b

d=b8f91effd76ad445519525597247d7a8511c6 4f3eb5d984462824ad2059971df9d8de4ba152e 82fdc810880d072e9d486960db1c9d78e5ef9c4 713f9807beb67d85294297a1e338c1510994422 62eb69d125d3fe8cf97b98daff899e659b77bad d7e7896642b82b9becef4eae85ff81f7ea9d792 66ecdbe97314e6694e21286317e8fd34821fb60 9126f069a58ec0e3910ac3c7a4677ba6dc523df f123200d2d761e40daa7269dba91ed02d87778f a50a27526aed40fb5b15b6c8e3d9ef2f66519d1 lac9c382496af817910cbdbab933740ebffb757 7f00f36efab487b9b1e82aa29093b8b103e5c5e 8ca5402f62d33fd683ecb279e1fdbfb53723723 f466d3

 $n=1957dbd9276d6168213b706b6aa9b1aa8cfd1\\65c7fb9092b282a0f11a520417b7b40a5f61404\\2d6e33bd3b4a95367d8d4c1d97e3057438f9a6d\\695ad2e2442260a444e58087c3b6a7b3e941cbb\\afb774a241df1dd93464cdaa99a3581ef0b1ef1\\2c9813a70b227da440951c99753507312702a86\\51788ff825d12bcc7d47831899377a4896f8dfd\\dfc790e68d0a9b5d67e0298b8274e44ed2a6f89\\2de7425ffa51bb6976a7418e25deaf45c1ce8a6\\fd38d104a63900d51c4fb496ad18a8647b09476\\1c672b720ae1d9e5798728071f30150decb2e49\\7b805b823e917a735ff1ade1b61cabf7062a1ed\\7d5f5d2eb1eb98869fcce3133bec02d40db827e\\5a8c605$