Evaluation of the Intensity of 1,024-bit RSA Cryptography Code

Project Representative
Hidehiko Hasegawa Faculty of Library, Information and Media Science, University of Tsukuba

Authors
Yasunori Ushiro Department of Mathematics, School of Education, Waseda University
Hidehiko Hasegawa Faculty of Library, Information and Media Science, University of Tsukuba

RSA cryptography is the key technology for safe Internet use, and currently 1,024-bit RSA code is the standard. To maintain safety when using RSA code, decryption should take more than 1 year even with the fastest supercomputer. Under 1,024-bit RSA cryptography, 1 to 10 years is the expected range.

Decryption processing consists of sieve processing, processing of linear equations in characteristic 2, and the computation of algebraic square roots. In order to allow the use the Earth Simulator 2 (ES2), we used the solution of a system of non-linear equations with multiple-precision numbers of more than 10^{11} digits and tuned the computation of algebraic square roots.

As a result, the decryption processing had a 98.6% vector operation ratio and 99.9% vectorization ratio over all integer operations. In addition, the resource requirement of the ES2 for the decryption of 1,024-bit RSA code (RSA-1024, 309 digits) was estimated to be 2000 node-years for the sieve processing, 6 years on 64 nodes for the processing of linear equations in characteristic 2, and 10 hours on 32 nodes for the computation of the algebraic square roots.

Keywords: Factorization of many-digit composite numbers, Sieve processing, Algebraic square roots, GNFS, Integer operations on the ES2

1. Introduction

The RSA cryptosystem, which was created by R. L. Rivest, A. Shamir, and L. M. Adleman in 1978, is the most important technology for using the Internet safely; however, the currently used 1,024-bit RSA code (RSA-1024, 309 digits) will not be safe in the near future. The RSA cryptosystem is based on the difficulty of the factorization of large composite numbers, and the decryption time of 1,024-bit RSA code is several years even on the fastest supercomputer. For an RSA code with some number of bits to be considered safe, the decryption time with the fastest algorithm and on the fastest supercomputer must be on the order of years. The safeness of this standard is based on a result that, for a given number n, the factorization of n into P and Q has a high computational complexity and consumes an enormous amount of computation time.

The present world record for RSA decryption for RSA-768 (768 bits, 232 digits)[10] is 1,677 CPU-years. This means that if one core in a CPU (AMD64 2.2 GHz) is used, then decryption takes 1,677 years. All reported RSA decryption world records were achieved on PC clusters and there has not yet been a report regarding vector supercomputers.

The aim of the our project was to obtain basic information on processing RSA decryption based on the general number field sieve (GNFS) method for the Earth Simulator 2 (ES2), which is a vector supercomputer. The decryption processing consists of three parts; the first step is sieve processing, the second step is the processing of linear equations in characteristic 2, and the third step is the computation of algebraic square roots. In 2012, the last year of our project, the authors tuned the computation of algebraic square roots for decryption software implemented on the ES2, and evaluated/discussed the safety of 1,024-bit RSA cryptography code.

2. Decryption of RSA cryptography code

The RSA algorithm uses two many-digit prime numbers P and Q, and another prime number e. First it computes $N = P \times Q$, $F = (P-1) \times (Q-1)$, and $D = e^{-1} \pmod F$. N and e are used as the encryption key, and D is used as the decryption key. The encryption key is available to the public, but the decryption key must be kept secure. This is possible because the decryption key is not necessary for encoding a secure message.

To decrypt an RSA encoded message, it is necessary to factor a composite number N into two prime numbers. The sieve method is the most effective factorization algorithm. It is said that multiple polynomial quadratic sieve (MPQS) factorization is the fastest method for the factorization of numbers with fewer than 100 digits and that GNFS is fastest for the factorization of numbers with more than 100 digits [6][7][9]. In the case of the

RSA-768 world record, the factorization was carried out using a linear function and a sixth-order polynomial in the GNFS method.

The procedure of GNFS is as follows:

(1) Exploration of polynomials f(x) and g(x):

Set

$f(x) = ax^6 + bx^5 + cx^4 + dx^3 + ex^2 + fx + g,$

$g(x) = Ax + B.$

Some integer M satisfies the following equations:

$f(M) \equiv 0 \pmod N$, $g(M) \equiv 0 \pmod N$.

Choose integer coefficients a, b, …, g, A, and B such that they are as small as possible in absolute value. The integer M is used instead of g(x) in the classical GNFS method [7]. In that case, the integer M should be as possible as small. As the computation time to explore for good polynomials is variable, we omitted this computation time from our evaluation/ discussion.

(2) Sieve method (choosing prime numbers and a prime ideal base)

For factoring primes in a base, a sieve method is used for fast computation instead of the modulo operation. In the sieve processing, the base used for factoring f(x) and g(x) is chosen. The base for g(x) consists of all prime numbers less than a specified value and -1, and the base of f(x) consists of almost half of the prime numbers less than a specified value, which satisfy

$f(x) = (\alpha x - \beta) \times v(x) \pmod P$

for some integers α and β, and for some linear function v(x). For the sieved data sθ+t, the factored polynomial norms for f(x) and g(x) are defined as follows:

$Nf(-t/s) = |at^6 - bst^5 + cs^2t^4 - ds^3t^3 + es^4t^2 - fs^5t + gs^6|,$

$Ng(-t/s) = At - Bs.$

The value s must be a positive integer and t must be a coprime integer; s and t are chosen in order to be able to factorize these equations with the prime numbers in the ideal base. The number of chosen data is greater than the sum of the number of elements in both bases.

(3) Processing of linear equations in characteristic 2

First we make a matrix H whose elements are 0 or 1 by modulo operation from the sieved results, which are derived from the factorization of N by the bases Nf(-t/s) and Ng(-t/s). For a polynomial f(x), some quadratic residues besides the already used sθ+t are added[7]. If the number of data is n and the sum of the number of elements in the bases and the number of quadratic residues is m, then n must be greater than m (n>m). Because a matrix H in characteristic 2 is m×n (m<n), the system of linear equations Hx=0 has non-zero (least square) solutions. The block Lanczos method is applied to obtain some of these solutions.

Table 1 Computational complexity of RSA-768 (768 bits, 232 digits).

	PC-year	Ratio(%)
Exploration of polynomials	20	1
Sieve Processing	1500	90
Processing of Linear equations	155	9
Algebraic square roots	1	0
Other	1	0
Total	1677	100

Note: One PC-year means that one core of an AMD64 (2.2 GHz) spends one year.

(4) Solution of the algebraic square roots module a polynomial f(x)

There exists the following polynomial H(θ) and integer K satisfying the following equations:

$H(\theta)^2 \equiv \Pi(s\theta+t) \pmod{f(\theta)}$: modulo a polynomial,

$K^2 \equiv \Pi(s\theta+t) \pmod{g(\theta)}$: modulo a linear function.

The s and t were computed from some of solutions of the system of linear equations in characteristic 2. Integer K, which is modulo a linear function, is directly computed; however, H(θ), the square of which is modulo a polynomial, exists but is not directly computed from the processing of linear equations in characteristic 2. We compute the algebraic square root H(θ) by computing the polynomial $\Pi(s\theta+t) \pmod{f(\theta)}$.

(5) Factorization of N by constructing $a^2-b^2=0 \pmod N$

Set a=K (mod N) and b=H(M) (mod N). If P equals GCD(a+b,N) and Q equals GCD(|a-b|,N), then N equals P×Q. This factors N; however, P and Q are 1 and N with probability 1/2[7]. Steps (4) and (5) are repeated until P and Q are not 1 and N.

The computational complexity of decrypting RSA-768 code using the GNFS method in PC-years of an AMD64 (2.2 GHz) is shown in Table 1. The size of matrix H is 192,795,550×192,796,550[10].

3. Computation of algebraic square roots by GNFS

For computing algebraic square roots, an application of the Chinese remainder theorem[7] and solving a system of non-linear equations with multiple-precision numbers are available. The Chinese remainder theorem is currently used for the decryption of RAS code[10]. This method requires relatively few bits but many conditional-branch operations. This is not suitable for a vector machine, such as the ES2. Therefore, we decided to use the solving of a system of non-linear equations. The procedure for finding algebraic square roots using non-linear equations by the GNFS method is as follows:

Set

$G(\theta) = \Pi(s\theta+t) \pmod{f(\theta)}.$

Find a polynomial H(θ) which satisfies

$H(\theta)^2 \equiv G(\theta) \pmod{f(\theta)}.$

If f(θ) is sixth degree, then G(θ) and H(θ) are fifth degree:

$G(\theta) = g_0\theta^5 + g_1\theta^4 + g_2\theta^3 + g_3\theta^2 + g_4\theta + g_5,$

$H(\theta) = x_0\theta^5 + x_1\theta^4 + x_2\theta^3 + x_3\theta^2 + x_4\theta + x_5$.

The square of $H(\theta)$ expressed module $f(\theta)$ is as follows:

$H(\theta)^2 \equiv h_0(x)\theta^5 + h_1(x)\theta^4 + h_2(x)\theta^3 + h_3(x)\theta^2 + h_4(x)\theta + h_5(x) \pmod{f(\theta)}$

where $x=(x_0,x_1,\ldots,x_5)^T$.

For $H(\theta)^2 \equiv G(\theta) \pmod{f(\theta)}$ to hold, $h_j(x)$, which is a second-degree polynomial of x, must satisfy the following conditions:

$h_j(x) - g_j = 0, \ j=0,1,\ldots,5.$

The integer solution of this system of non-linear equations is an algebraic square root $H(\theta)$. To solve this system, we used the following Newton iteration procedure:

$J(x^{(k)})\Delta x = h(x^{(k)}) - g,$

$x^{(k+1)} = x^{(k)} - \Delta x,$

where $h(x)=(h_0(x),h_1(x),\ldots,h_5(x))^T$ and $g=(g_0,g_1,\ldots,g_5)^T$. Δx is the increment of x. J(x) is the Jacobian matrix of h(x), expressed as follows:

$$J(x) = \begin{pmatrix} \frac{\partial h_0(x)}{\partial x_0} & \frac{\partial h_0(x)}{\partial x_1} & \cdots & \frac{\partial h_0(x)}{\partial x_5} \\ \frac{\partial h_1(x)}{\partial x_0} & \frac{\partial h_1(x)}{\partial x_1} & \cdots & \frac{\partial h_1(x)}{\partial x_5} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_5(x)}{\partial x_0} & \frac{\partial h_5(x)}{\partial x_1} & \cdots & \frac{\partial h_5(x)}{\partial x_5} \end{pmatrix}.$$

In this procedure, an acceleration of basic arithmetic operations for more than 10^{11} digits is necessary because the coefficients of the second-degree polynomial $h_j(x)$ and solution x_j have more than 10^{11} digits.

4. Vectorization for the ES2

To compute algebraic square roots, it is necessary to use basic arithmetic operations for multiple-precision numbers of more than 10^{11} digits, especially for the acceleration of the multiplication operation. On the ES2, we accelerated the multiplication operation for multiple-precision numbers by using vector processing.

(a) Each element stores a 32-bit number, and one number in the procedure is expressed by a large number of elements.

(b) All arithmetic operations are 64- or 32-bit integer arithmetic operations.

(c) The multiplication operations for multiple-precision numbers are constructed based on the integer fast module transformation (integer FMT)[3][4].

(d) To utilize bits more effectively, one convolution based on the Chinese remainder theorem is applied for each four multiplication operations.

(e) The computational result is carried for each element (32 bits).

The step for integer FMT has large computational complexity; however, this step is computed in a short time by vector processing on the ES2. The processing of "carry" has a data dependency in that the result of the lower part affects the computation of the higher part, making it difficult to apply vector processing. We applied the following method for vectorization on the ES2.

(e1) All elements n are divided into L groups (n=M×L) because of the enormous number of elements to be carried.

(e2) One additional element is added to each group, and thus a multiple-precision number is expressed by (M+1)×L elements.

(e3) The computation order for carrying was changed so as to be computed in parallel for each group.

(e4) The additional element for the final solution x is eliminated by the normalization.

The following program is the program used on conventional computers for multiple-precision (M×L×32 bits) addition with carrying. The lower part is stored in the array with the smaller index value in ascending order.

```
Cover = 0;
for (i=0; i<n i++) {
  Apval   =   A[i] + Cover;
  Cover   =   (Apval < A[i]);
  C[i]    =   Apval + B[i];
  Cover   +=  (C[i] < Apval);
}
```

The next program is that on the ES2.

```
  Cover[0] = 0;
#pragma cdir nodep(A,B,C,Cover) on_adb(Cover)
  for (i=0; i<L-1; i++)
    { Cover[i+1] = A[i][M] + B[i][M]; }
  for (j = 0; j<M; j++) {
#pragma cdir nodep(A,B,C,Cover) on_adb(Cover)
    for (i=0; i<L i++) {
      Apval     =   A[i][j] + Cover[i];
      Cover[i]  =   (Apval < A[i][j]);
      C[i][j]   =   Apval + B[i][j];
      Cover[i]  +=  (C[i][j] < Apval);
  } }
#pragma cdir nodep(Cover)
  for (i=0; i<L; i++)
    { C[i][M] = Cover[i]; }
```

The unsigned int type is used for the variables and arrays for the carrying of the final result of multiple-precision multiplications because results are stored as 32-bit integers. The unsigned long long int type, 64-bit integer, is used for other addition operations without carrying. The sign and exponent part are stored in other variables separately. This method may introduce discontinuous accessing of the arrays A, B, and C in the innermost loop, and thus adjustment of the array size is necessary to avoid accessing different memory banks. For only the final solution x, which is derived by the Newton iteration

process, a process to eliminate C[i][M] is added.

"#pragma" is the directive statement for vectorization for the ES2. "nodep" means that they have no data dependency and can be used in vector processing. "nodep" was necessary for array C. "on adb" means to use the special cache on the ES2 for vector processing. It gave about a 15% computation time reduction.

5. Size and features of RSA decryption on the ES2

A comparison of the decryption of RSA code using GNFS on PC clusters and a vector machine is shown in Table 2.

Sieve processing can be parallelized into millions of nodes because a problem is divided into many small ranges and a small amount of data is collected by a relatively large computation. However, as the base becomes large, performance decreases because the memory is accessed by a variable stride whose length equals a prime number. The processing of linear equations in characteristic 2 by the block Lanczos method requires many millions of iterations[2], and all data must be stored in memory.

The size of each step in the decryption of the RSA code by GNFS is shown in Table 3. In this computation, the combinations of the linear functions and the polynomials are used.

In the sieve processing for a 1,024-bit RSA code, more than 10^{10} sievings are necessary for one datum. In this process, mis-sieved and not-to-be-sieved results are acceptable because the sieved result is confirmed by simple computations. However, the processing of linear equations in characteristic 2 and the computation of algebraic square roots must be computed correctly. The size of matrix H is determined from the number of sieved data and the number of elements in the base. If the solution of a system of linear equations in characteristic 2, Hx=0, has non-zero solutions, then the number of sieved data should be 1000 more than the number of elements in the base.

6. Estimation of the resource requirement of RSA decryption

An estimation of the resource requirement of decryption of an RSA code is shown in Table 4. We use the resource requirement as a measure of the intensity of the RSA code. A modified method for computing the algebraic square roots is used for comparison/estimation.

The sieve processing can be performed on one node of the ES2 because the memory requirement is comparatively small. For the processing of linear equations in characteristic 2 and the computation of algebraic square roots, the minimal

Table 2 Comparison of PC clusters and a vector machine for RSA decryption.

	Sieve Processing	Linear Equation in Characteristic 2	Algebraic Square Roots
PC Clusters	In cache	Block Lanczos	Chinese remainder theorem
Vector Machine	In memory	Small change	Should be modified
Parallelization	A great degree of parallelization	Much communication	Medium communication
Property	Prime number stride access	Very sparse binary matrix	10^{11} digits
Operations	Integer	Integer	Integer

Table 3 Size of decryption of RSA code by GNFS.

		RSA-768 232 digits	RSA-896 270 digits	RSA-1024 309 digits
Sieve Processing	Degree of polynomials	6	6 or 7	7 or 8
	# in base	2×10^{8}	10^{9}	5×10^{9}
	# of sieved data	10^{17}	3×10^{18}	10^{20}
	# of collected data	2×10^{8}	10^{9}	5×10^{9}
Linear Equation in Characteristic 2	Matrix size	2×10^{8}	10^{9}	5×10^{9}
	# of non-zero elements/row	hundreds	hundreds	hundreds
Algebraic Square Roots	# of multiplications by$\Pi(s\theta+t)$	10^{8}	5×10^{8}	2.5×10^{9}
	Max # of digits of coefficients	5×10^{9}	3×10^{10}	2×10^{11}

Table 4 Estimation of the resource requirement of the RSA decryption.

		RSA-768	RSA-896	RSA-1024
Resource Requirement of the ES2	Sieve Processing	2 node-years	60 node-years	2000 node-years
	Linear Equation in Characteristic 2	800 hours on 4 nodes	8 months on 16 nodes	6 years on 64 nodes
	Algebraic Square Roots	3 hours on 1 node	5 hours on 8 nodes	10 hours on 32 nodes
Performance Ratio: One Node of ES2/PC (2.2 GHz)	Sieve Processing	700	750	800
	Linear Equation in Characteristic 2	450	500	550
	Algebraic Square Roots	600	650	700

number of nodes is determined by the memory requirements. In all three cases, RSA-768, RSA-896, and RSA-1024, the sieve processing is the most time-consuming part, and the computation of the algebraic square roots makes up is a very small part of the decryption. On the ES2, the sieve processing is the most accelerated part. The processing of linear equations in characteristic 2 has a relatively small acceleration ratio because it requires additional computations for vector processing[2]. The vector operation ratio (Number of vector operations / Number of all operations ×100) on the ES2 was measured as 98.6%, and the vectorization ratio (Time of the part where vector operations can be executed as normal scalar operations / Time of all parts executed as normal scalar operations ×100) was 99.9%. The vectorization ratio is an estimation based on that one vector operation is 50 times faster than one scalar operation and that the average vector length is 200. The vectorization ratios at the sieve processing, the processing of linear equations in characteristic 2, and the computation of algebraic square roots are almost identical, and almost all vector operations in these processes are integer operations.

7. Summary

On the ES2, for the decryption of the current RSA code, RSA-1024, it is estimated that the sieve processing will take 2000 node-years, the processing of linear equations in characteristic 2 will take 6 years on 64 nodes, and the computation of algebraic square roots will take 10 hours on 32 nodes. This estimate is based on GNFS, which was used for the decryption of RSA-768. The required numbers of nodes and computation times vary according to the used values, functions, and choice of base. In real use, even given two RSA codes having the same bit lengths, the decryption times may differ significantly.

The effective performance ratio of scalar supercomputers is slightly smaller than that of the ES2; however, the K computer will take about one year, and so a supercomputer 100 times faster would take only a couple of days for the decryption of RSA-1024. If a good method for decryption of the RSA code appears, much improvement of computation times will occur.

From these results, the current RSA-1024 (1,024 bits, 309 digits) should be replaced by RSA-2048 before the 2019 deadline [8] that was recommended for the 2010 problem.

Almost all computations of the decryption of RSA code are integer operations, and meeting decryption challenges was attempted by using scalar computers. To create software to use on the ES2 required much modification of the software for scalar machines. However, the resulting software worked effectively and the ES2 was shown to be suitable for the decryption of RSA code, which has no floating-point number operations.

Acknowledgement

The authors show the thanks to Dr. Yoshinari Fukui and Mr. Tadashi Kai of JAMSTEC who gave us precious opinions by speedup of the ES2 heartily.

References

[1] Y. Ushiro, “RSA Decryption using Earth Simulator”, Annual Report of Earth Simulator Center, 167-171, 2011.

[2] Y. Ushiro and H. Hasegawa, “Acceleration of the Processing of Linear Equations in Characteristic 2 for RSA Decryption”, Annual Report of Earth Simulator Center, 165-170, 2012.

[3] Y. Ushiro, “High-precision Multiplication by Fast Modulo Transformation”, IPSJ Journal, Vol. 44, No. 12, 3131-3138, Dec. 2003 (in Japanese).

[4] Y. Ushiro, “Studies on fast algorithms for high-precision computation”, Ph. D thesis at Waseda University, Mar. 2005 (in Japanese).

[5] Y. Ushiro, “A high speed sieve method for the RSA cipher using the vector computer”, 1733, 101-117, RISM kokyuroku, Mar. 2011 (in Japanese).

[6] Richard A. Mollin, *“RSA and PUBLIC-KEY CRYPTOGRAPHY”*, CRC Press, 2002.

[7] Yuji Kida, “Prime factoring by General Number Field Sieve”, 2003, http://www.rkmath.rikkyo.ac.jp/~kida/nfs_intro.pdf (in Japanese).

[8] Junichi Yamazaki and Souta Kamaike, “The 2010 problem of the Cryptography”, ASCII technologies, 75-93, Sep. 2010 (in Japanese).

[9] Neal Koblitz, translate by Kouichi Sakurai, *“A Course in Number Theory and Cryptography”*, Springer, 1997 (in Japanese).

[10] Kazumaro Aoki, “Advances in Integer Factoring Technique: The Way to Factor RSA-768”, IPSJ Magazine, 51(8), 1030-1038, Aug. 2010 (in Japanese).

1024 ビット RSA 暗号の強度推定

プロジェクト責任者
長谷川秀彦　　筑波大学　図書館情報メディア系
著者
後　　保範　　早稲田大学　教育学部　数学科
長谷川秀彦　　筑波大学　図書館情報メディア系

インターネットを安全に使ううえで欠かせない技術である RSA 暗号には桁数の多い合成数の因数分解の困難性が利用されており、その安全性はスーパーコンピュータを数年使用しても解読されないという仮定のもとで成り立っている。本プロジェクトでは、RSA 暗号の安全性の検証のため、ベクトル方式のスーパーコンピュータである地球シミュレータ (ES2) において RSA 暗号の解読実験を行い、現在使われている 1024 ビットの RSA 暗号（RSA-1024, 10 進 309 桁）の強度推定を行う。一般的な RSA 暗号解読は、ふるい処理、標数 2（0-1 データ）の線形計算、代数的平方根の計算の 3 段階からなり、初年度はふるい処理の高速化、2 年目は標数 2 の線形計算の高速化を行い、最終年度である 3 年目は「代数的平方根の計算」を並列ベクトル化し、ES2 において RSA-1024 を解読するのに必要な資源量（メモリサイズ＆演算時間）を推定した。

代数的平方根の計算には中国剰余定理 Chinese Remainder Theorem を応用した方法と多数桁の連立非線形方程式を解く方法がある。中国剰余定理を応用した方法は比較的短い桁数で計算できる反面、判定処理が多発するため、ベクトル計算機ではベクトル化が困難である。そこで、多数桁の数を係数とする連立非線形方程式を作成し、ニュートン法を用いて数値解を求める方式を採用した。必要な整数解をニュートン法で求めるには、異なる値に収束する初期値の選定と整数解以外の除去が必要になる。6 次式で係数が 10 進 50 億桁の場合、初期値の探索を 1 万桁で行い、整数解を求めるのに平均 3 個の解が必要である。1 個の整数解での暗号解読の成功率は 50% なので、平均で 6 回の解の計算が必要である。なお、初期値探索の時間は解の計算時間の 1% 以下である

ニュートン法に使用する多数桁の数は一要素（32 ビット）に 2 進 32 桁を詰めて表現した。ES2 は 64 ビット整数の剰余演算が高速であり、すべての計算は整数演算（int64,int32）を使用し、多数桁の数の乗算には整数 FMT（高速剰余変換）を使用してベクトル化した。乗算結果の有効桁数を増やすために、4 回の乗算結果を中国剰余定理で重ね合わせた。計算結果の桁上げ処理にはデータ依存性があるため、要素数 n を L 要素ごとに M 個のブロックに分割し（n=M × L)、各 M に 1 要素追加した (M+1) × L 要素として表現し、桁上げのデータ依存性を L 要素のブロック内で留める。複数のブロックで並列化することによって、代数的平方根の計算は ES2 で高速にベクトル処理できる。

RSA-768 の解読では、代数的平方根の計算は 50 億桁の係数を持つ 6 次多項式、RSA-1024 の解読では 2000 億桁の係数を持つ 7 ～ 8 次式の多項式と推定できる。代数的平方根の計算時間は、RSA-768 では ES2 の 1 ノードで 3 時間、RSA-1024 では 32 ノードで 10 時間と推定した。RSA 暗号の解読時間の推定は、RSA-768 ではふるい処理が 2 ノード・年、標数 2 の線形計算が 4 ノードで 800 時間、代数的平方根が 1 ノードで 3 時間である。RSA-1024 では、ふるい処理が 2000 ノード・年、標数 2 の線形計算が 64 ノードで 6 年、代数的平方根が 32 ノードで 10 時間である。ふるい処理は、1 ノードでも実行可能だが、数百万台規模の並列化も容易である。標数 2 の線形計算と代数的平方根の計算は規模により、実行可能な最低ノード数が決まる。

RSA-1024 は、ES2 より実行性能が少し劣るとしても、京で 1 年程度、100 倍高速なスーパーコンピュータなら数日で解読されると推定できる。より効率の良い解法が考案されれば危険性は増す。現在使用されている RSA-1024 は、最終期限 2019 年よりも前に 2048 ビットに変更した方が良いと考える。

RSA 暗号の解読計算（ふるい処理、標数 2 の線形計算、代数的平方根）はすべて整数演算である。ES2 で高速に計算させるためには工夫が必要だが、ベクトル演算率 98.6%、ベクトル化率 99.9% が達成できた。

キーワード : 多数桁数の因数分解 , 篩（ふるい）, 代数的平方根 , GNFS, 整数演算