A well defined factorization

N Vijayarangan, S Kasilingam, Nitin Agarwal

Abstract - In this paper, a factorization method is properly defined based on square root technique for cryptographic applications. It is applied and tested to any n-bits integer factorization, particularly on RSA algorithm. For product of many primes, RSA algorithm is modified (called extended RSA). Further, security issues on RSA and extended RSA are discussed.

Index Terms - Factorization, Multi - precision integer, RSA algorithm, Aurifeuillian identity

1. INTRODUCTION

Factorization is a long pending problem in number theory, theoretical computer science and cryptography. Methods such as number field sieve [3,11], Pollard (p-1) [6], Pollard rho [7], Selfridege [1], Elliptic curve multiplication [3,9] etc., could factorize integers into product of powers of primes, but each method has its own computational difficulty and weakness (*see Table 1*). To overcome this situation, we have studied and introduced a suitable method and an efficient algorithm to factor large numbers, which is an extension of trial division. We have described the factorization technique into two parts: 32-bits and multi precision integers and for each set up, pseudo code and examples are given. Using the proposed factorization methods, our interest is shown towards cracking of RSA algorithm. Further, we extend RSA algorithm for product of many primes. Then comparison on security issues is made between RSA and extended RSA. Also, an Aurifeuillian factorization [9] is discussed.

2. FACTORIZATION ALGORITHM I

The following algorithm gives factorization for 32-bits number n. Given an integer n (n > 1), the following procedure gives the factors of n.

The authors are with the PKI Group, Core R&D, ITI Limited, Bangalore, India.
E-mail: {vijayrangan_2005, kasilingam_s, nitagl} @yahoo.com

We require a list called factors to store the factorization of n in the form of a set consisting of elements {p,e}, where p^e is the exact power of p appearing in the factorization of n.

1. Let factors ={ }, e = 0.
2. Compute s = int(√n) (integral part of √n).
3. Find a suitable k_s such that 6 k_s + ε = s or 6 k_s - ε = s, where ε > 0.
4. Construct a sequence S = {2,3,6k±1} where k = 1 to k_s.
5. Choose a number p ∈ S such that q = n/p. Then n = q and e = e+1. Repeat the process 5 until p divides n.
6. Print {p,e}.
7. Compare p and q.
   7.1 If p > √q, then q is a factor of n.
   7.2 If p < √q, then repeat the process 5.

In this process, we can factorize any number of 32-bit length. Two important steps of this algorithm are square root of a number and the sequence S. Step 7.1 concludes that the number q is prime so that q is also a factor of n. This algorithm can be used for primality test. (If n is prime, then the above algorithm returns n only). Compared to Rabin probabilistic primality test [8], this method is relatively slow but not probabilistic. Being tested in Pentium III 700 MHz, the computational time analysis on algorithm I shows difference for composite number and prime number. Factorization on composite and prime, it takes 90 μ sec and 11 sec (optimum time) respectively.

Example: Let us take a 32-bit number n = 99990001. Using Algorithm I, we can find that n is prime. This n is expressed as $10^4 - 10^2 + 1$, but not factored.

We have given a sample `C' code comprising important functions for 32-bit integer factorization

/* Trial division operation on a number using the pre-stored contiguous sequence of primes, first 2048 primes are stored in a file. See Ref. [2]

The following function trialdiv ( ) returns prime divisors of a number n */

```
unsigned long trialdiv(unsigned long n)
{
        extern unsigned long primes[2048];
        unsigned long t;
        int i;
        if(n==0)
                return ((unsigned long)0);
        if(n==1)
                return ((unsigned long)1);

  for(i=0;i<2048;i++)
        {
                t= n % primes[i];
                if(t==0)
                {
                        return (primes[i]);
                }
        }
           return((unsigned long)0);
}

/* The function  factor () is to find the factors for a given
number lying in the sequence {2,3, 6k±1} where k = 1 to int(√n)
*/

unsigned long factor(unsigned long n)
{
        unsigned long k,t=1;
        k=((unsigned long)sqrt(n) -1)/6;
        k++;
        while(t<n)
        {
                t=6*k + 1;
                if(!(n % t))
                        return t;
                k++;
        }
        return 0;
}
```

Similarly, we can extend the above algorithm to multi-precision integers (MPI). The following algorithm for MPI factorization of an integer n (n > 32-bit) is given below:

3. FACTORIZATION ALGORITHM II

1. Compute s = int(√n).
2. Find $k_s = (s + \varepsilon)/6$ or $k_s = (s - \varepsilon)/6$ where $\varepsilon > 0$.
3. Construct a sequence S = {2,3,6k±1} where k = 1 to k_s.
4. Divide S into intervals such as $S_1, S_2, \ldots, S_t$.
5. For each S_j, perform steps 5,6,7 of Algorithm I in parallel processing
6. Find {p,e} of each S_j.

In this process, any MPI of n can be factored as product of powers of primes. This method is particularly efficient for two prime factors. Ultimately, we can apply this method to break RSA-algorithm. Before executing this algorithm, we should have parallel processing set up.

Example: Let us take n = 2320869986411928544793. It is a 71-bit number. Compute s = int(√n) = 48175408523 (36-bit number). Since $k_s = (s + 1)/6$, we obtain k_s = 8029234754. Construct a sequence S = {2,3, 6k±1} where k = 1 to k_s. Divide S into intervals $S_1, S_2, \ldots, S_9$, where $S_1 = [1, 2^4\text{-}1]$, $S_2 = [2^4, 2^8\text{-}1], \ldots, S_9 = [2^{32}, k_s]$. Executing each interval S_j, we get p = 67777783 (one of the factor of n) lying in S_6. Further, another factor of n is q = 34242341423471 so that n = pq.

Example: Let us take n = 80964686104403 (47-bit). Compute s = int(√n) = 8998037 (24-bit). Since $k_s = (s + 1)/6$ = 1499673, we obtain a sequence S = {2,3, 6k±1} where k = 1 to k_s. Divide S into intervals $S_1, S_2, \ldots, S_6$, where $S_1 = [1, 2^4\text{-}1]$, $S_2 = [2^4, 2^8\text{-}1], \ldots, S_6 = [2^{20}, k_s]$. Executing each interval S_j, we get p = 8996717 (one of the factor of n) lying in S_6. Further, another factor of n is q = 8999359 so that n = pq.

Factorization methods	**For all numbers**	**Technique used**
Elliptic Curve Multiplication	Can Factorize	Elliptic curves
Pollard (p-1)	Can not factorize	$(a^{k!} - 1) \bmod n$
Pollard Rho	Can not facotorize	Periodic sequences
Number field Sieve	Can factorize	Congruent squares
Selfridge	Can not factorize	$b^n \pm 1$ form
Factorization Algorithm I or II	Can factorize	Square root and division of intervals

Table1: Comparison of some factorization methods

4. ALGORITHM III

Let us consider the extended RSA-algorithm as follows:

1. Find large composite numbers $p_1, p_2, \ldots, p_r$ and $q_1, q_2, \ldots, q_m$ and define by $n = p_1 p_2 \ldots p_r q_1 q_2 \ldots q_m$.
2. Find a large random integer d that is relatively prime to $\phi(n) = (p_1 - 1)\ldots(p_r - 1)(q_1 - 1) \ldots (q_m - 1)$ where $\phi(n)$ is Euler's phi value of n.

3. Compute the unique integer e in the range $1 < e < \phi(n)$ from the formula $ed \equiv 1 \pmod{\phi(n)}$.
4. Make known the public key, which consists of the pair of integer (e,n)
5. Represent M, the message to be transmitted as an integer in the range {1,2,…,n} and gcd (M,n)=1.
6. Encrypt M into a cryptosystem C by the rule, $C \equiv M^e \pmod{n}$.
7. Decrypt by using the private key d and the formula $D \equiv C^d \pmod{n}$.

Example: Let us take n = product of three prime numbers = 39*59*71 = 163371. Then $\phi(n) = 154280$. Take e= 113 and d = 96937 so that $ed \equiv 1 \pmod{\phi(n)}$. Choose a message M = 200 such that gcd (M,n)=1. The message M is encrypted using the public key e, we get $C \equiv M^e \pmod{n} = 40760$. Using the private key d, we can get M.

5. SECURITY ISSUES

It is clear that Algorithm III works for encryption and decryption on large bit numbers. Factorization on extended RSA is computed easily compared to RSA. Since n is shared by more than 2 primes. If we apply factorization algorithm I or II on extended RSA, we can obtain factors of n in a collection of intervals S_1, S_2 , …, S_t. Using these algorithms, we can prove that breaking time on RSA –1024 bits (where $n= p_1 p_2$, $p_1 = p_2$ =512-bits) takes more than extended RSA-1024 bits (where $n= p_1 p_2 p_3 p_4$, $p_1 = p_2 = p_3 = p_4$ = 256-bits). The reason is that more division of intervals and computational processes are involved in between 500 and 512 bits compared to 250 and 256 bits. So it is (almost) equivalent to choose n=2048-bits (where $n = p_1 p_2 p_3 p_4$, $p_1 = p_2 = p_3 = p_4$ = 512-bits) in extended RSA, while comparing with RSA-1024 bits (where $n= p_1 p_2$, $p_1 = p_2$ = 512-bits). The weakness lies in RSA when one of the factors is broken.

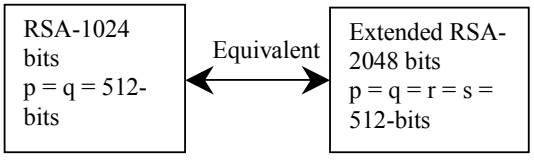

Fig1: Relation between RSA and Extended RSA

On the other hand, there are some numbers behaving like an Aurifeuillian Identity $2^{4n+2} + 1 = (2^{2n+1} - 2^{n+1} + 1)(2^{2n+1} + 2^{n+1} + 1)$. Factors of this identity are either primes or composites or both (*see Table 2*).

n	$2^{2n+1} - 2^{n+1} + 1$	$2^{2n+1} + 2^{n+1} + 1$	$2^{4n+2} + 1$
1	5(prime)	13(prime)	65
2	25(composite)	41(prime)	1025
3	113(prime)	145(composite)	16385
9	523265 (composite)	525313 (prime)	2748779069 45
16	8589803521 (composite)	8590065665 (composite)	7378697629 4838206465

Table2: Examples of Aurifeuillian identity

In the extended RSA-algorithm, n can be factored easily if n falls in the Aurifeuillian identity. Selection of n should not be arbitrary and it should not be equal to $2^{4n+2} + 1$. Therefore, it is always good to choose n = 1024/2048/4096-bits, which will be away from Aurifeuillian identity.

In RSA, the relationship between n and $\phi(n)$ is given by $p^2 - p(n+1-\phi(n)) + n = 0$ or $q^2 - q(n+1-\phi(n)) + n = 0$, whereas in the extended RSA no such relationship can be found.

6. CONCLUSION

For e-commerce applications, our choice is to take either RSA or extended RSA algorithm. But, one should be very circumspect in setting the value of n as well as cracking time of the resulting algorithm chosen. Fig.2 gives a clear view on RSA algorithms.

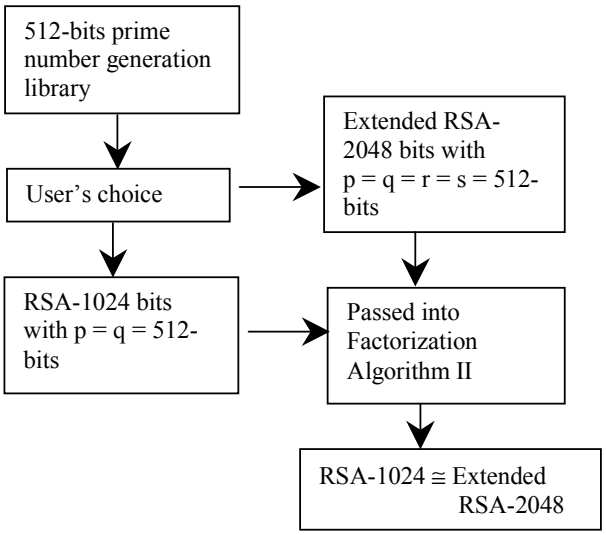

Fig2: View on RSA algorithms

7. REFERENCES

[1] J. Brillhart, D. Lehmer, J. Selfridge, B. Tuckerman, and S. Wagstaff Jr., "Factorizations of $b^n \pm 1$, b=2,3,5,6,7,10,11,12 upto high powers", *Second edition, vol. 22 of Contemporary Mathematics*, Amer. Math. Soc., 1988.

[2] C. Caldwell, *Website for prime numbers*, 1999. http://www.utm.edu/research/primes.

[3] R. Crandall and C. Pomerance, "Prime Numbers a computational perspective", *Springer-Verlag,* 2001.

[4] R. Crandall, "Parallelization of Pollard-rho factorization", 1999. http://www.perfsci.com.

[5] R. Lehman, "Factoring large integers", *Math.Comp.*, Vol. 28, pp.637-647, 1974.

[6] J. Pollard, "Theorems on factorization and primality testing", *Proc. Cambridge Philos. Soc*., Vol. 76, pp.521-528, 1974.

[7] J. Pollard, "Monte Carlo methods for index computation (mod p)",*Math. Comp.*, Vol. 32, pp.918-924, 1978.

[8] M. Rabin, "Probabilistic algorithms for testing primality", *J. Number Theory*, Vol. 12, pp.128-138, 1980.

[9] Ramanujachary Kumanduri and C. Romero, "Number Theory with Computer Applications", *Prentice Hall*, New Jersey, 1998.

[10] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Comm. ACM*, 21:120-126, 1978.

[11] H. Williams and J. Shallit, "Factoring integers before computers", In W. Gautschi, editor, *Mathematics of Computation*.