2006

A parallel key generation algorithm for efficient diffie-hellman key agreement

Yun Chen
Chengdu University of Information Technology

Xin Chen
Zhengzhou University of Light Industry

Yi Mu
University of Wollongong, ymu@uow.edu.au

A parallel key generation algorithm for efficient diffie-hellman key agreement

Abstract

The kernel function of the Diffie-Hellman (DH) protocol is a modular exponentiation over finite field with high computational complexity. In this paper, we propose a novel key generation algorithm for DH agreement that derives computational efficiency from constructing a parallel architecture. Compared to the serial structure for traditional binary representation (BR) method, our algorithm is significantly more efficiency on key generation and suitable for hardware implementation in an ephemeral-static mode for DH agreement which is thought to be more secure (Rosorla, 1999)

Disciplines

Physical Sciences and Mathematics

Publication Details

Chen, Y., Chen, X. & Mu, Y. (2006). A parallel key generation algorithm for efficient diffie-hellman key agreement. In Y. Cheung, Y. Wang & H. Liu (Eds.), International Conference on Computational Intelligence and Security (pp. 1393-1395). Hong Kong: IEEE.

A Parallel Key Generation Algorithm for Efficient Diffie-Hellman Key Agreement

Yun Chen
Chengdu University of Information Technology,
No.24, Block 1, Xuefu Road,
Chengdu, China
chy@cuit.edu.cn

Xin Chen
Zhengzhou University of Light Industry
No.5, Dongfeng Road,
Zhengzhou, China
chenxin@zzuli.edu.cn

Yi Mu
University of Wollongong
Northfields Avenue,
Wollongong, NSW 2522,
Australia
ymu@uow.edu.au

Abstract

The kernel function of the Diffie-Hellman (DH) protocol is a modular exponentiation over finite field with high computational complexity. In this paper, we propose a novel key generation algorithm for DH agreement that derives computational efficiency from constructing a parallel architecture. Compared to the serial structure for traditional Binary Representation (BR) method, our algorithm is significantly more efficiency on key generation and suitable for hardware implementation in an ephemeral-static mode for DH agreement which is thought to be more secure.[2]

1. Introduction

The DH key exchange protocol is a popular public key agreement algorithm used by two or multiple parties to generate a shared secret against eavesdroppers. Each participant possesses their own key pair generated as follows:

$$y_i = g^{x_i} \pmod{p}, i = 1,2,\cdots,N. \quad (1)$$

where N is the total number of participants; x_i is ith participant's secret key which is randomly chosen in (1, p); y_i is ith participant's public key; modulus p is usually a large prime with 1024 bits in length; and g is a generator over finite field GF(p). Both g and p are common group parameters in a system.

Formula (1) has a high computational complexity. Although many efficient algorithms for DH key agreement have been proposed [6,1,8,7,5], most of them focused on multi-participant key establishment problems or restrict finite field parameters.

To enhance security, a system usually requires a DH key agreement working in an ephemeral-static mode where the recipient has a static key pair, but the sender fleshly generates an ephemeral key pair for each message. Therefore, computational efficiency becomes critical.

In this paper, a new algorithm for DH key generation is presented. The computational efficiency of the algorithm is improved significantly in virtue of of the parallel structure.

2. Parallel scheme for DH key generation

Traditionally, y_i is generated with BR algorithm [3] which is an iteration technique by representing x_i in binary form and then decomposing modular exponentiation into a series of modular multiplications over a finite field. The total number of iterations L relies on the number of bits n and the hanmming weight $\omega(x_i)$ of the index x_i, [3] *i.e.*

$$L = n + \omega(x_i) - 2,\ 1 \le \omega(x_i) \le n \quad (2)$$

In some extent, L dominates the operational speed of BR algorithm since a participant has to calculate step by step in a series structure. Therefore, shortening L will reduce computational overhead.

Recall that there are two constant common parameters g and p in a DH agreement. By utilizing the property, we can construct a parallel computational structure. Write x_i as follows:

$$\begin{aligned} x_i &= b_{s-1}^{(i)} 2^{(s-1)k} + \cdots + b_j^{(i)} 2^{jk} + \cdots + b_0^i, \\ k &= 1,\cdots,n, j = 0,\cdots,s-1, b_j^i \in GF(2^k) \end{aligned} \quad (3)$$

where $s = \lfloor \frac{n}{k} \rfloor$ denotes the integer part of $\frac{n}{k}$.

Let $t = 2^k, r_j = g^{t^j} \pmod{p}, y_{ij} = r_j^{b_j^{(i)}} \pmod{p}$ and put Eq. (3) into Eq. (1), then

$$y_i = g^{x_i} = \prod_{j=0}^{s-1} y_{ij} \pmod{p} \quad (4)$$

Because $b_j^{(i)}, j = 0,\cdots,s-1$ are independent of one another, the s items in Eq. (4) can be calculated

1-4244-0605-6/06/$20.00 ©2006 IEEE.

simultaneously. Hence, we can firstly create a residue table by pre-computing residues $r_j, j = 0,\cdots,s-1$ and then use the table to compute y_{ij} in parallel.

Main procedures for the parallel DH key generation method are below:

1. The ith participant randomly picks a private key x_i in $(1,p)$, $i = 1,\cdots,N$.
2. Express x_i in the form of base t.
3. Represent $b_j^{(i)}$ in binary form.
4. Compute $y_{ij}, j = 0,\cdots,s-1$ in parallel with BR algorithm.
5. Calculate the public key y_i in Eq. (4).

At this case, the number of iterations L_j $j = 0,\cdots,s-1$ for y_{ij} is

$$L_j = k + \omega(b_j^{(i)}) - 2,\ 1 \le \omega(b_j^{(i)}) \le k \qquad (5)$$

Comparing Eq. (5) with Eq. (2), we know that $L_j << L$ when $k << n$. Therefore, the number of iterations could be cut down by choosing suitable k.

3. Analysis on the parallel algorithm

The main idea of the parallel algorithm is to separate modular exponentiation over $GF(p)$ into s portions where each portion is bounded with k. Generally, the algorithm consists of three stages: a) Create a residue table in advance. b) Compute s portions simultaneously. c) Evaluate the product of all s portions to get output y_i. The features of the output are mainly affected by k & n. Some technical features of the parallel algorithm respect to k and n are listed out in table 1. S_r and S_m denote the size of residue table in bytes for each portion and for all s portions respectively. L_m refers to the average number of iterations per portion. T_m and T_p stand for time consumption on a 512×512-bit modular

Table 1. Technical features versus k & n

k	8		16		32		64	
n	1024	2048	1024	2048	1024	2048	1024	2048
S_r	16k	64k	8k	32k	4k	16k	2k	8k
S_m	32k	128k	16k	64k	8k	32k	4k	16k
L_m	10	10	22	22	46	46	94	94
T_m	70	370	154	814	322	1702	658	3478
T_p	49	296	42	259	35	222	28	185
U_m	128	256	64	128	32	64	16	32
U_p	256	2048	128	1024	64	512	32	256

multiplication in one portion and in Eq. (4) separately. U_m and U_p are operational units to carry out a 512×512-bit modular multiplication on s portions and in Eq. (4) respectively.

It is shown in Table 1 that the size of residue table, either in one portion or in s portions, will increase along with the increase of n and decrease along with the increase of k.

We also know from Table 1 that the average number of iterations per portion L_m is irrelative to n. But it will increase with the increase of k.

We can deduce from Table 1 that T_m will increase along with the increase of n or k. While T_p increases versus the increase of n but decreases versus the increase of k.

Table 1 reads that operational unit U_m at stage b) or U_p at stage c) varies with k and n. They will increase when k is fixed and n is risen up, but decrease when n is fixed and k is risen up.

The data in Table 1 are constructed under the assumption of parallel computation.

4. Comparing our algorithm with BR

There have been hundreds of efficient algorithms on modular exponentiation over a finite field published so far. Almost all of them are based on BR. We, here, make a comparison on time and memory consumption between the BR and our algorithm. The data are given in table 2. U_Σ and T_Σ in the table express the total memory and time consumption on key generation respectively. U_Σ takes maximum number between U_m and U_p because U_m and U_p are similar operational units and we can reuse the operational units at stage c) before we finish the computation at stage b). Therefore, the total time consumption is the sum of T_m and T_p in Table 2.

Table 2. Comparison between the BR and the parallel algorithm.

k	BR(k=1)		8		32	
n	1024	2048	1024	2048	1024	2048
S_r	-	-	16k	64k	4k	16k
S_m	0.5k	2k	32k	128k	8k	32k
L_m	-	-	10	10	46	46
T_m	-	-	70	370	322	1702
T_p	0	0	49	296	35	222
U_m	4	16	128	256	32	64
U_p	0	0	256	2048	64	512
U_Σ	4	16	256	2048	64	512
T_Σ	10738	95170	119	666	357	1924

From the data in Table 2, we find:

- The BR algorithm has a series structure in a binary system. Hence, it is irrelevant to residue table.
- The BR algorithm is irrelevant to k, because it is a series algorithm while k is a key parameter that affects the parallel extent in our algorithm.
- n is used in both BR and the parallel algorithm. With

the increase of n, the operational scale expands. It makes an inevitable increase of both time and memory consumption on the BR and our algorithm.
- In our new algorithm, the computational time on each portion decreases and the time on product increases with respect to the decrease of k.
- The number of operational units decreases along with the increase of k.
- By comparison, we find that our parallel algorithm outperforms the BR algorithm in efficiency.
- The improvement on computational speed in our parallel algorithm mainly relies on the choice of k. The smaller k is, the faster the algorithm is.

5. Conclusion

Our goal is to speed up the key generation for the DH key agreement. Now, we define a speed improvement factor as the ratio of computational workload (per-unit time) in our algorithm to that in the BR algorithm. Actually T_m is approximately equal to T_p. In virtue of this, we figure out the speed improvement factors with respect to n and k. They are listed in Table 3.

Table 3. Speed improvement factors of our algorithm versus the BR algorithm

k	8	16	32
n=1024	90.24	54.786	30.078
n=2048	142.898	88.695	49.465

It is clear that we have achieved a significant improvement in speed. The improvement is not measured by percentage but by the order of magnitude. The improvement given in Table 3 reaches an order 1 or 2 in magnitude. If k decreases further, we will get a better improvement in speed. However, it comes with the cost of memory consumption.

By choosing proper parameters, we can obtain a significant improvement in computational speed compared with the BR algorithm. The efficiency is achieved by constructing a parallel operation structure and creating a pre-computation residue table. Compared to the DH key agreement algorithm proposed in reference [1], where a 43% of speed improvement has been reported, our parallel algorithm clearly performs much better.

Since our algorithm shows a great efficiency, it is very suitable for the ephemeral-static mode of the DH agreement. In addition, it is applicable to those public key agreements that have a similar algorithmic structure such as the ElGamal encryption and DSA. Furthermore, our algorithm is scalable. The communication systems involved in a DH key agreement can easily update their operational structure by properly changing k. Our algorithm also shows its potential in hardware realization.

Our parallel algorithm requires more system memory than the BR algorithm. We believe that it should be acceptable to most of the users, since the cost of memory has been greatly reduced in recent years. Notice that by choosing proper parameters, we can optimize the performance of our algorithm, without overusing the system memory.

6. References

[1] G.Gong and A. Hasan, *etc.*, "An Efficient Algorithm for Exponentiation in DH Key Exchange and DSA in Cubic Extension Fields", Faculty of Mathematics, University of Waterloo, 2002, CORR 2002-27.

[2] E.Rosorla, Diffie-Hellman Key Agreement Method, RFC 2631, 1999.

[3] Y.Chen, and Y.H. Gong, An Improvement Algorithm for Recursive Sums of Residue, *Journal of UEST* (Chinese), 2000, 29(1): 1-4

[4] W. Diffie and M.E. Hellman, New Directions on Cryptography, *IEEE transactions on IT*, 1976, 22:644-654

[5] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, An Efficient Protocol for Authenticated Key Agreement, University of Waterloo, 1998, CORR 98-05.

[6] D.V. Bailey and C. Paar, Efficient Arithmetic in Finite Extensions with Application in Elliptic Curve Cryptography, *Journal of Cryptology*, 2001, 14(3):153-176

[7] G. Horng, An Efficient and Secure Protocol for Multi-party key Establishment, *The Computer Journal*, 1976, 44(5):463-470

[8] G. Gong, L. Harn, and H. Wu, The GH Public-key Cryptosystem, *The Proceeding of the Eighth Annual Workshop on Selected Areas in Cryptography*, Waterloo, 2001, CORR 2001-35