INT J COMPUT COMMUN, ISSN 1841-9836
Vol.7 (2012), No. 4 (November), pp. 617-631

Nondeterministic Algorithm for Breaking Diffie-Hellman Key Exchange using Self-Assembly of DNA Tiles

Z. Cheng

Zheng Cheng
Zhejiang University of Technology
288 Liuhe Road, Hangzhou, P.R. China
chengzhenhust@gmail.com

Abstract: The computation based on DNA tile self-assembly has been demonstrated to be scalable, which is consider as a promising technique for computation. In this work, I first show how the tile self-assembly process can be used for computing the modular multiplication by mainly constructing three small systems including addition system, subtraction system and comparing system which can also be parallely implemented the discrete logarithm problem in the finite field $GF(p)$. Then the nondeterministic algorithm is successfully performed to break Diffie-Hellman key exchange with the computation time complexity of $\Theta(p)$, and the probability of finding the successful solutions among many parallel executions is proved to be arbitrarily close to 1.
Keywords: Modular multiplication; Discrete logarithm; Nondeterministic; Diffie-Hellman; Key exchange; Self-assembly; DNA tiles

1 Introduction

Nature is a rich source for computing devices design. In recent years, computing models and algorithms inspired by biological systems are deeply investigated, e.g., membrane computing inspired by cells and DNA computing inspired by DNA molecules. Most of such computing models inspired by cells (or DNA molecules inside cells) are proved to be universal and computationally efficient [1, 2]. This work focuses on computing systems based on DNA molecules, especially the self-assembly of DNA tiles. Since Adleman demonstrated that the DNA recombination techniques can be used to solve combinational search problem [3], the field of DNA-based computing has developed very fast. DNA computing potentially provides a degree of parallelism and high density storage far beyond that of conventional silicon-based computers [4].

DNA tile self-assembly is a crucial process, by which the small objects can autonomously assemble into big complexes [5]. Seeman proposed DNA nanotechnology to make self-assembled nanostructures from DNA molecules [6]. Based on this landmark work, Winfree utilized a kind of DNA nanostructure called DX (double crossover) tile to realize a patterned lattice and the complex algorithmic pattern [7]. Winfree et al. demonstrated that two dimensional DNA self-assembly can be capable of Turing-universal computation [8]. Winfree and Eng proved that self-assembly of linear, hairpin and branched DNA molecules can be generated regular, bilinear and context-free languages respectively [9]. DNA tile algorithmic self-assembly can also be used to create crystals with patterns of binary counters [10, 11] and Sierpinski triangles [12] to implement arbitrary circuit [13]. However, those crystals are deterministic. Mao experimentally implemented the first algorithmic DNA tile self-assembly [14], where a logical computation (cumulative XOR) is performed. Brun proposed the application of DNA tile self-assembly in arithmetic [15]. DNA self-assembly is also used to cope with combinational NP-complete problems, such as solving the satisfiability problem [16] by using 2D DNA self-assembly tiles, nondeterministically factoring numbers [17], deciding a system of subset sum problem [18]. DNA self-assembly has potential applications in the cryptography. XOR computation on pairs of bits

can be used to execute a one-time pad cryptosystem, which provides theoretically unbreakable security [19].

To provide modern security features, the algorithms for public-key cryptosystems such as RSA [20], Diffie-Hellman key agreement [21], the digital signature algorithm [22] and systems based on elliptic curve cryptography [23] are widely used. All these algorithms have one thing in common: they all need operate the modular multiplication and exponent multiplication [24, 25]. In 1976, Diffie and Hellman [26] proposed the public-key distribution scheme based on the discrete logarithm problem in a finite field $GF(p)$. Chen [27] gave deterministic algorithm to break Diffie-Hellman key exchange and constructed the basic tiles in which the complex modular multiplication operation in decimal is contained, so it can't be easily executed based on the experiment of simple binary arithmetic and logical operations. Here I mainly propose the nondeterministic algorithm to solve this problem by the addition, subtraction and comparing operations for binary numbers, which can be performed easily in experiments. The computation time complexity of our algorithm is $\Theta(p)$, and the probability of finding the successful solutions among the many parallel executions is proved to be arbitrarily close to 1.

The rest of this paper is structured as follows: Section 2 will describe the tile self-assembly model. Section 3 gives the method of computing modular multiplication using DNA tile self-assembly. Section 4 shows the process of breaking Diffie-Hellman key exchange based on modular multiplication by self-assembling. The conclusion will summarize the contribution of our work.

2 Algorithmic DNA tile self-assembly

The abstract Tile Assembly Model [28] is a formal model of crystal growth which is designed to model self-assembly of molecules such as DNA. Rothemund and Winfree [29] defined the abstract tile assembly model, which provides a rigorous framework for analyzing algorithmic self-assembly. The tile assembly model extends the theory of Wang tilings [30] of the plane by adding a natural mechanism for growth.

For the tile self-assembly model, the assembly complexes take place by starting with the seed tile denoted as the basic tile type set, which can be produced the seed configuration S. For each tile, it can be represented by the binding domains $\sum$ which is a 4-tuple $\{\sigma_N, \sigma_E, \sigma_S, \sigma_W\} \in \sum^4$. Here, N, E, S, W is labeled as the direction of north, east, south and west respectively. The set of directions is a set of four functions from positions to positions denoted as $D = \{N, E, S, W\}$, i.e. $\mathbb{Z}^2$ to $\mathbb{Z}^2$ such that all positions (x, y), $N(x, y) = (x, y+1)$, $E(x, y) = (x+1, y)$, $S(x, y) = (x, y-1)$, $W(x, y) = (x-1, y)$. For a tile t, for $d \in D$, $bd_d(t)$ is used to denoted as the binding domain of tile t on d's side.

Given a tile system $\mathbb{S} = \{T, g, \tau\}$ which is designed preparedly, here the parameter T is a function $\mathbb{Z} \times \mathbb{Z} \to T$, which is a configuration of the tile self-assembly model. g is a strength function $\sum \times \sum \to \mathbb{R}$, which denotes the strength of the binding domains, and $\tau \in \mathbb{N}$ is the temperature. When the growth of process terminates, these can be produced a unique final configuration $\mathbb{S}$ based on the seed configuration S. S is also a function $\mathbb{Z}^2 \to \Gamma$, here Γ is a set of tiles which can be used to design the configuration of the tile self-assembly model. Here, I mainly use the abstract tile assembly model to break Diffie-Hellman key exchange which is shown in Figure 1. Intuitively, the model has tiles or squares that stick or don't stick together based on various binding domain including $\sigma_N, \sigma_E, \sigma_S, \sigma_W$ on their four sides.

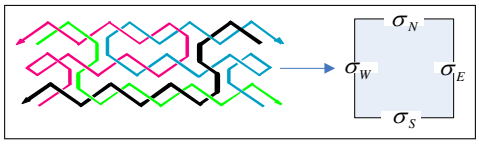

Figure 1: The structure of basic tile unit.

3 The algorithm for computing modular multiplication based on DNA tile self-assembly

Various techniques for speeding up modular multiplication have been reported in literature. Among them, two major approaches stand out: one is based on the interleaved modular multiplication algorithm where the multiplier is processed from the most significant position [31]. The other one is based on the Montgomery algorithm where the multiplier is processed from the least significant position [32]. The key that enables the linking of these two approaches is a new representation of residue classes modulo. The two methods account for most of the complexity in terms of time and resources needed. So this gives reason to search for dedicated solutions which compute the modular multiplications efficiently with minimum resources. Here, I use the method of interleaved modular multiplication based on DNA tile self-assembly. It takes advantage of these techniques and the ones that may eventually be devised to further boost speed.

For the integers X, Y, M with $0 \leq X, Y < M$, in order to get the result of $X*Y \mod M$, I use the algorithm which gives a pseudo code implementation of interleaved modular multiplication as follows:

(1)$P = 0$;
(2) for $(i = n-1;\ i \geq 0;\ i = i-1\ )\{$
(3) $P = 2 * P$;
(4) $P = P + x_i * Y$;
(5) if $(P \geq M)\ P = P - M$;
(6) if $(P \geq M)\ P = P - M;\}$

Here, n is the number of bits of X. x_i is denoted as the i-th bit of X. The idea of interleaved modular multiplication is very simple: the first operand is multiplied with the second operand bitwise and added to the intermediate result. The intermediate result is reduced with respect to the modulus. For this purpose at most two subtractions per iteration are required. The process is to interleave multiplication and reduction such that the intermediate results are kept as short as possible. For every $x_i (0 \leq i \leq n-2)$, there is a left shift, a partial product computation, an addition, and at most two subtractions. The partial product computation and left shift operations are easily performed by using an array of AND gates and wiring respectively. In each iteration of the loop, the estimation of the previous iteration can be added to the intermediate result.

In this section, I will introduce the algorithm for performing modular multiplication through constructing three small systems which are addition system, subtraction system and comparing system. Examples can be given to indicate how the tile assembly model performs the computations.

3.1 Addition system

This system will do addition operations between two positive integers. When the comparison result at the previous step is "<" which means the value of P is smaller than the modulus M, then

this system will make addition operations between two integers P and M. If $x_i = 0(0 \leq i \leq n-2)$, the result is the value of P which should be shifted one bit from right to left, intuitively the sum is the value of $2 * P$. If $x_i = 1(0 \leq i \leq n-2)$, the addition between P and Y is done, here P should be shifted one bit from right to left and Y does not need to shift one bit.

Theorem 3.1. Let the addition system be denoted as $\sum_+$={ab, abcd, a/bcd, a*bcd, a*bc, a*b, a, a*, #, a*bcd/, a/bc; a, b, c, d $\in \{0,1\}$}, and T_+ be the set of tiles as described in Fig.2(c). Let $g_+ = 1$, $\tau_+ = 2$, and S_+ be a seed configuration. Let n_P and n_Y be the sizes of P and Y in bits respectively. Let $n = \max(n_P, n_Y)$. If $n_P < n$, the number needs to be padded to be n bits long with extra 0 in the P's high bit, and if $n_Y < n$, the number needs to be padded to be n bits long with extra 0 in the Y's high bit. Then, there exists some $(x_0, y_0) \in \mathbb{Z}^2$, such that $S_+(x_0+1, y_0-1) = S0$, $S_+(x_0-n, y_0-1) = E0$, $S_+(x_0+1, y_0-0) = Add$; for all $i \in \{0, 1, \cdots, n\}$, $bd_N(S_+(x_0-(i-1)), y_0-1) = x_i p_i m_i y_i$, for all other positions (x, y),$(x, y) \notin S_+$. Then the tile system $\mathbb{S}_+$ produces a unique final configuration based on S_+ and can compute the sum of two input numbers by using $\Theta(1)$ distinct tiles in linear assembly time.

Proof. Consider this tile system $\sum_+$. Let Γ_+ be composed of the tiles $\{0*, \#, \#, null\}$, $\{a<, null, null, null\}$ with the label $S0$, $\{\#, null, null, null\}$ denoted as the tile $E0$, $\{a<, a, *, null\}$ represented as the tile Add, and the basic tile set T_+, here $a \in \{0, 1\}$. Let the seed configuration $S_+ : \mathbb{Z}^2 \to \Gamma_+$ be such that

$$\begin{cases} S_+(1,-1) = S0; \\ S_+(-n,-1) = E0, \ \ and \ \ S_+(1,0) = Add; \\ \forall i \in \{0, 1, \cdots, n\}, \ \ S_+(-i,-1) = x_i p_i m_i y_i; \\ For\ all\ other\ (x,y) \in \mathbb{Z}^2, \ \ S_+(x,y) = empty. \end{cases}$$

It is clear that there is only a single position where a tile may attach to S_+. And after that tile attaches there will only be a single position where a tile may attach. By induction, because $\forall t \in T_+$, the triplet $< bd_S(t), bd_E(t) >$ is unique, and because $\tau_+ = 2$, it follows that this tile system $\mathbb{S}_+$ produces a unique final configuration on S_+.

Fig.2(a) gives a sample seed configuration for adding two n-bit input numbers. Fig.2(b) shows the final configuration of the example for adding two integers $P = (001110)_2$ and $Y = (001101)_2$ with the result $(011011)_2$. The set of tiles which is described in Fig.2(c) shows the functions of the addition system has three functions.

The addition system has three functions. First, the value of P, M and Y are arranged in the seed configuration from the lowest bit to the highest bit, and the value of X is set from the highest to the lowest bit. Here, a, b, c, d, e, f, g, k $\in \{0, 1\}$. The initial value of P is set as 0. Of course, the highest bit of X denoted as x_{n-1} is 1, so the first addition operation only needs to pass the value of Y to P for the corresponding bits. The tile types with the red color can be seen in Fig.2(c). The value "a0cd" at the bottom of the tiles are denoted as X, P, M, Y respectively. The number 1 as the input, and output of the tile is the value of x_{n-1} which is assigned to 1. The numbers of bits of P and M are more than X and Y, so "bc" in the first tile of the tile types are represented as the higher bits of P and M respectively. More importantly, I use "a/// to label the highest bit of X which should be passed to the lowest bit, then $x_i = 0$ or 1 $(0 \leq i \leq n-2)$ can be passed to the addition system to determine whether the value of Y should be added with P together. "e/// is the value passed from the highest bit, so the label of "a// will be passed to the addition system at next step.

Second, for the next addition, if $x_i = 0(0 \leq i \leq n-2)$, the lowest bit of P is 0, and the value of P should be shifted one bit from right to left, then the result of the addition operation is $2 * P$. At the same time, $x_i = 0(0 \leq i \leq n-2)$ should be passed to the higher bit of Y, and the bits of Y and M are passed to the upper layer. So here, there is no carry bit for the addition at this step. If $x_i = 1(0 \leq i \leq n-2)$, the lowest bit of P is the corresponding lowest bit of Y,

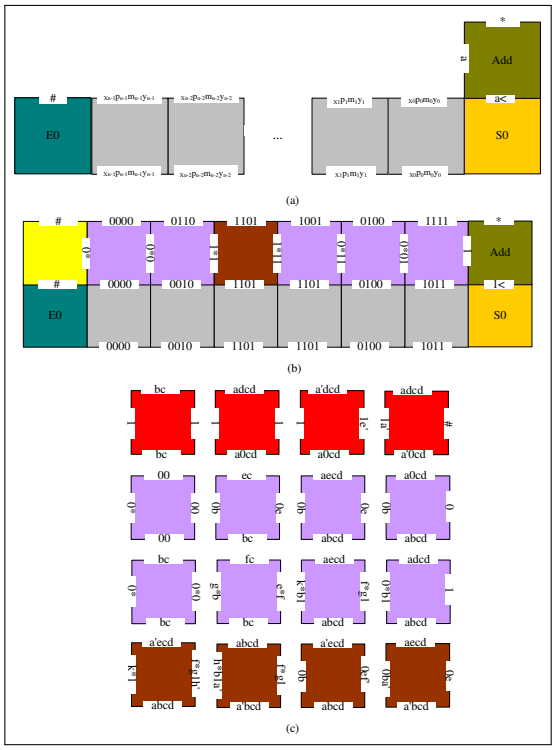

Figure 2: (a) A sample seed configuration for adding two n-bit input numbers. (b) The final configuration of the example for adding two integers $P = (001110)_2$ and $Y = (001101)_2$ with the result $(011011)_2$. (c) The basic tile types of the addition system.

then the addition is done between P and Y, here P also should be shifted one bit from right to left. So if the carry bit from the lower bit is "f*", and the shifted bit from P is "g", thus the addition result "e〃 and the carry bit "k*" can be generated at this step, synchronously the bit of P and Y labeled as "b" and 1 respectively should be passed to the higher bit. The tile types with lilac are shown this function of the addition system.

Third, this system can determine whether and what the bit of X is passed to the next addition system. If the lowest bit of X has been completed the addition operation, then it only should be passed to the final result. Otherwise, the value of x_i should be passed to the lower bit x_{i-1} which will be done the next round addition. So if the input on the right of the tiles includes the label "f〃〃, it can determine the value of x_{i-1} denoted as "a〃 will be passed to the next addition system.

3.2 Subtraction system

In this section, I will describe a system that can make subtraction between two positive integers. When the comparison result at the previous step is ">" or "=" which means the value of P is bigger than or equal to the modulus M, then this system will make the subtraction between the two integers P and M.

Theorem 3.2. Let the subtraction system $\sum_{-}$={ab, abcd, a*, *; a, b, c, d $\in \{0,1\}$ }, and T_{-} be the set of tiles as described in Fig.3(c). Let $g_{-} = 1$, $\tau_{-} = 2$, and S_{-} be a seed configuration. Let n_P and n_M be the sizes of P and M in bits respectively. Let $n = \max(n_P, n_M)$. If $n_P < n$, the number needs to be padded to be n bits long with extra 0 in the P's high bit, and if $n_M < n$, the number needs to be padded to be n bits long with extra 0 in the M's high bit. Then, there exists some $(x_0, y_0) \in \mathbb{Z}^2$, such that $S_{-}(x_0 + 1, y_0 - 1) = S0$, $S_{-}(x_0 - n, y_0 - 1) = E0$,

$S_-(x_0+1, y_0-0) = Sub$; for all $i \in \{0, 1, \cdots, n\}$, $bd_N(S_-(x_0-(i-1)), y_0-1) = x_i p_i m_i y_i$, for all other positions (x, y),$(x, y) \notin S_-$. Then the tile system $\mathbb{S}_-$ produces a unique final configuration based on S_- and can compute the difference of two input numbers with $\Theta(1)$ distinct tiles in linear time.

Proof. Consider the tile system $\sum_-$. Let Γ_- contain the tiles $\{0^*, \#, \#, \text{null}\}$, $\{>, null, null, null\}$ denoted as $S0$, $\{\#, null, null, null\}$ represented as the tile $E0$, $\{*, *, >, null\}\}$ labeled as the tile Sub, and the basic tile set T_-. Let the seed configuration $S_- : \mathbb{Z}^2 \to \Gamma_-$ be such that

$$\begin{cases} S_-(1, -1) = S0; \\ S_-(-n, -1) = E0, \ \ and \ \ S_-(1, 0) = Sub; \\ \forall i \in \{0, 1, \cdots, n\}, \ \ S_-(-i, -1) = x_i p_i m_i y_i; \\ For\ all\ other\ (x, y) \in \mathbb{Z}^2, \ \ S_-(x, y) = empty. \end{cases}$$

It is obvious that there is only a single position where a tile may attach to S_-. And after that tile attaches there will only be a single position where a tile may attach. By induction, because $\forall t \in T_-$, the triplet $< bd_S(t), bd_E(t) >$ is unique, and because $\tau_- = 2$, it follows that this tile system $\mathbb{S}_-$ produces a unique final configuration on S_-.

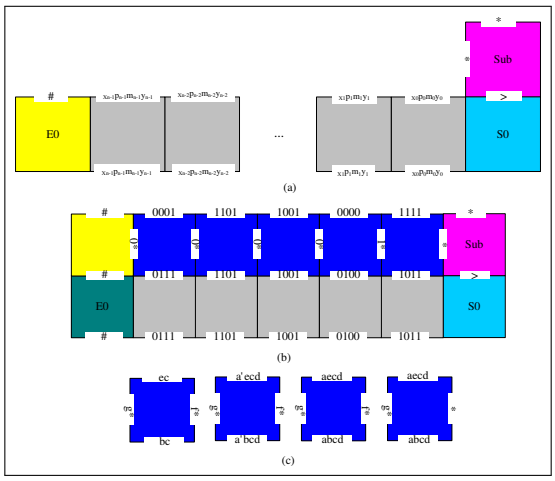

Figure 3: (a) A sample seed configuration for subtraction operation for two n-bit input numbers. (b) The final configuration for the example of subtracting two integers $P = (11010)_2$ and $M = (10001)_2$ with the result $(01001)_2$. (c) The basic tile types of this system.

Here, the value of "a, b, c, d" at the bottom of the tile is denoted as X, P, M, Y respectively, and a, b, c, d, e, f, g $\in \{0, 1\}$. The subtraction operation begins from the lowest bit to the highest bit with the label "*". "a′" is the value of X with the label which is passed to the addition system. The representation "f*" is the borrow bit from the lower bit and "g*" is the borrow bit which is generated at this step. The value of "e" is the result which can be computed by the sum of "b" and "c" minus "f". Fig.3(a) is a sample seed configuration for subtraction operation for two n-bit input numbers. Fig.3(b) shows the final configuration for the example of subtracting two integers $P = (11010)_2$ and $M = (10001)_2$ with the result $(01001)_2$, and (c) gives the basic tile types of the subtraction system.

3.3 Comparing system

This system is to check the relationship for given input string of binary bits which are represented as the addition or subtraction result P at each step and the modulus M respectively.

At the same time, the relationship can determine the next operation should be made addition or subtraction. The system compares two input numbers bit-by-bit from the highest bit to the lowest bit until the relationship which is less than (<), greater than (>) or equal (=) is determined. Here, I will describe the comparing system which uses $\Theta(1)$ distinct tiles in linear time to compare the size of two input numbers.

Theorem 3.3. Let the comparing system $\sum_R$={ab, abcd, a/bcd, >, <, =, a>, a<, a=, #; a, b, c, d$\in \{0,1\}$}, and T be the set of tiles as described in Fig.4(c). Let $g_R = 1$, $\tau_R = 2$, and S_R be a seed configuration. Let n_P and n_M be the sizes of P and M in bits respectively. Let $n = \max(n_P, n_M)$. If $n_P < n$, the number needs to be padded to be n bits long with extra 0 in the P's high bit, and if $n_M < n$, the number needs to be padded to be n bits long with extra 0 in the M's high bit. Then, there exists some $(x_0, y_0) \in \mathbb{Z}^2$, such that $S_R(x_0 - n, y_0 - 1) = S0$, $S_R(x_0 + 1, y_0 - 1) = E0$, $S_R(x_0 - n, y_0 - 0) = Com$; for all $i \in \{0, 1, \cdots, n\}$, $bd_N(S_R(x_0 - (i-1)), y_0 - 1) = x_i p_i m_i y_i$, for all other positions (x, y),$(x, y) \notin S_R$. Then the tile system $\mathbb{S}_R$ produces a unique final configuration based on S_R and can give the relationship of two input numbers.

Proof. The proof is referred to Theorem 3.1 and Theorem 3.2, so I don't restate it here.

This system mainly makes the comparison between two non-negative integers which are used in the binary form. First, for two given integers, the numbers of bits are the same. Second, the comparison begins from the most significant bit to the rightmost bit. When the $(i+1)$-th bit of the addition or subtraction result P is smaller then the modulus M, then the tile is labeled as "<", which also should be considered with the result of the i-th bit which is "<", ">" or "=", then the result which is "<" can be obtained and should be passed to the $(i-1)$-th bit. On the contrary, the final result is ">" also should be passed to the $(i-1)$-th bit no matter what the relationship of the i-th bit is. If the $(i+1)$-th bit of the addition or subtraction result P is equal to the modulus M, then the tile is labeled as "=", which also should be considered with the result of the i-th bit which are "<", ">" or "=", then the result which is "<", ">" or "=" respectively can be obtained and should be passed to the $(i-1)$-th bit. Furthermore, if the comparison result of the two integers is "<", the next operation would turn to the addition computation with $x_i = 0$ or 1. If the comparison result is ">" or "=", the next operation would turn to the subtraction computation between P and M. When the comparison result is "<" with the label of the position for the lowest bit of X, the modular multiplication is completed at this step.

Fig.4(a) is a sample seed configuration for two n-bit input numbers. Fig.4(b) shows the final configuration for the example of comparing two integers $P = (1010)_2$ and $M = (0101)_2$ with the result ">". The tiles used in this system are as follows in Fig.4(c). In each tile, the value of "a, b, c, d" is denoted as X, P, M, Y respectively.

3.4 An example

Now I take an example to verify the validity of the algorithm based on DNA tile self-assembly model introduced above. Here, for the integers X, Y, M with $0 \le X, Y < M$, I suppose $X = 11$, $Y = 13$, $M = 17$. The modular multiplication $X * Y \mod M$ is computed as the following steps:

First, X, Y, M can be represented as $X = 11 = (1011)_2$, $Y = 13 = (1101)_2$, $M = 17 = (10001)_2$ respectively. The initial value of P is set as 0 and arranged from the lowest bit to the highest bit together with Y, M. On the contrary, the bits of X are from the highest bit to the lowest bit.

Second, according to the method introduced above, I need construct the basic tile types in each of the three small systems and they are the same as the tiles described above. When all the tiles and the seed configuration are prepared, they are put together into the reaction buffer.

Figure 4: (a) A sample seed configuration for two n-bit input numbers. (b) The final configuration for the example of comparing two integers $P = (1010)_2$ and $M = (0101)_2$ with the result ">". (c) The basic tile types of the comparing system.

According to the mechanism of algorithmic DNA tile self-assembly through Watson-Crick base pairing, the self-assemble process starts at the same time with the connector tiles, so the final stage can be seen in Fig.5.

The process of the three small systems performing is shown. Here, because the highest bit of X which is denoted as $x_3 = 1$, the value of P is equal to Y, so the comparing system can check that P is smaller than M. The next addition operation begins with $x_2 = 0$, so the value of P at the previous step shifted one bit from right to left can be assigned to P at this step, which can be represented as $P = (011010)_2$, then the comparing system determines it is more than M, so the next operation is the subtraction operation between P and M, and the subtraction result should be assigned to P which is $(001001)_2$. When $x_1 = 0$, the addition system can make the sum between Y and P, here P should be shifted one bit from right to left. So the addition result also should be assigned to P which is $(011111)_2$, and it is bigger than M by the comparing system, then the subtraction result is $(001110)_2$. The computation stops until x_0 attaches to the addition system with the result of P which is smaller than M. The final result of P is $(000111)_2$ which can be obtained by repeated computations using the addition, subtraction and comparing system.

4 The nondeterministic algorithm for breaking Diffie-Hellman key exchange using self-assembly of DNA tiles

On basis of the algorithm for computing modular multiplication, I will first give the method of performing the discrete logarithm problem in the finite field $GF(p)$, then the nondeterministic algorithm for breaking Diffie-Hellman key exchange is successfully proposed.

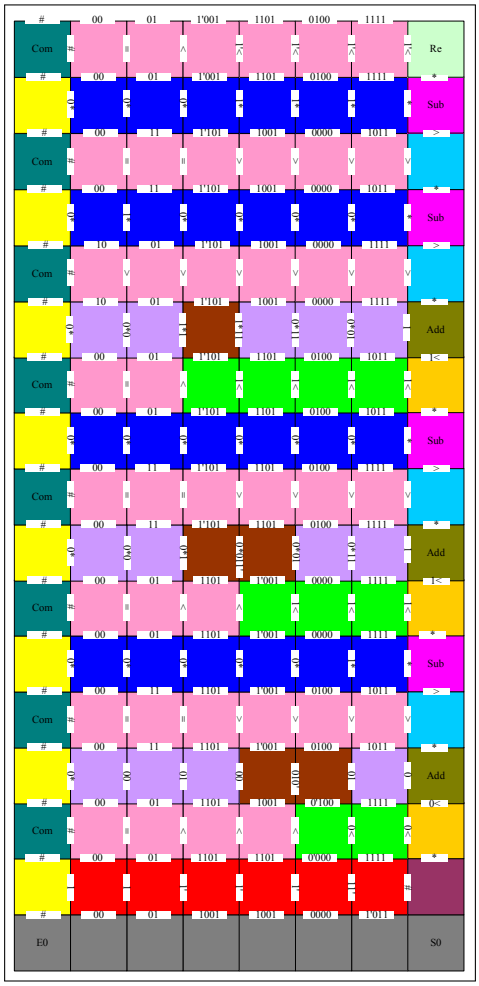

Figure 5: An example of computing modular multiplication $1011 * 1101 \bmod (10001)$.

4.1 The nondeterministic algorithm for solving the discrete logarithm problem

A group G is cyclic if there is an element $\alpha \in G$, such that for each $b \in G$, there is an integer i with $b = \alpha^i$. Such that an element α is called a generator of G. Now given a cyclic group G of order p, a primitive-root g and the element y of the group, the problem is to find an integer x such that $y \equiv g^x (\mathrm{mod}\ p)$ with $1 \leq x \leq p-1$, and this problem is called discrete logarithm problem in the finite field $GF(p)$. Considering the equation $y = g^x (\mathrm{mod}\ p)$, for given y, g and p, it is very difficult to compute the value of x. So here I solve the discrete logarithm problem by implementing the algorithm for computing the modular multiplication based on DNA tile self-assembly model.

In the process of implementing the tile self-assembly systems, many assemblies happen in parallel by creating billions of billions of copies of the participating DNA tiles, so this is simulated by an exponential number of DNA assemblies which can be converted into the space occupied by the DNA molecules, thus I expect that the procedure of computing y will run in parallel on all possible value of x under the condition $1 \leq x \leq p-1$. Then I can compare the computation result with the given value of y, finally the result of x can be read by the biological operations described in Section 3.

So first, I give the algorithm for computing modular exponentiation based on the modular multiplication introduced above. There are two differences between the two computations. Firstly, the modular exponentiation is actually a series of modular multiplications, so in the process of designing the seed configuration, it only needs to give different labels of the lowest bit of X which can be used to distinguish the value of x. Here, I consider one of the value of X as Y. Secondly, given the value of i, after the computation of $g^i (\mathrm{mod}\ p)$ is completed, there are some tiles that can convert the result of $g^i (\mathrm{mod}\ p)$ into the new value of X, and the value of Y doesn't need to change, then the assembly complexes can be used to implement the computation of $g^{i+1} (\mathrm{mod}\ p)$. Therefore I don't give too much explanation for the process of computing modular exponentiation.

On this basis, I use the nondeterministic algorithm to solve the discrete logarithm problem in the finite field $GF(p)$. It can nondeterministically guess the value of x so that a parallel implementation can be executed. I need to construct different labels to denote the value of x, which can determine the computation of modular exponentiation. Here, an example in $GF(11)$ is taken. Suppose the equation $y = 7^x (\mathrm{mod}\ 11)$, for given $y = 2$, I can get the value of x which is 3 as following steps and the final stage can be shown in Fig.6.

Step 1: The binary forms of g, p is obtained respectively, here I consider $Y = X$. The initial value of the integer P is set as 0 and arranged from the lowest bit to the highest bit together with Y, p. On the contrary, the bits of X are from the highest bit to the lowest bit. Then the basic tile types and the seed configuration are constructed and put together into the reaction buffer. According to the nondeterministic algorithm, there are many choices at the first position of the seed configuration, it can nondeterministically guess the value of x under the condition $1 \leq x \leq p-1$. In this example, the tile containing the label "11" which is denoted as the value of $x = 3$ attaches the seed configuration. So the modular exponentiation $y = 7^3 (\mathrm{mod}\ 11)$ can be parallely performed by the three small systems including addition system, subtraction system and comparing system, the assembly complexes can grow, therefore I can obtain the solutions of the problem.

Step 2: Once the self-assembly has occurred, it is necessary to extract the answer. An estimate of the length of the reporter strands can be obtained by annealing the reporter strands with the component strands, so I can extract the result strands of different lengths representing the output tiles which run through the value of y.

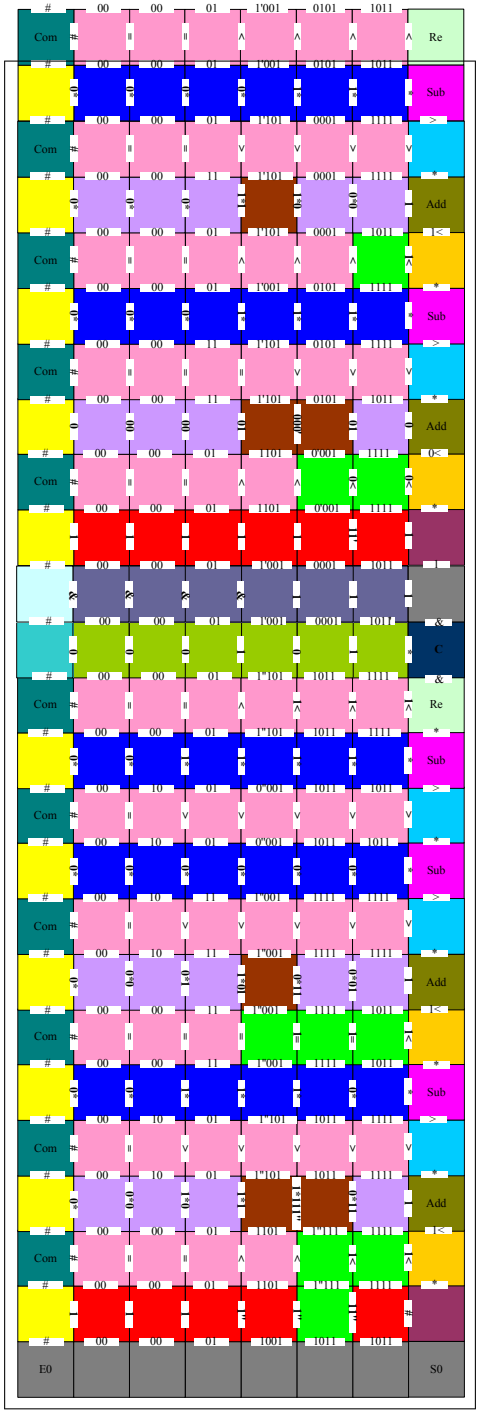

Figure 6: An example of computing modular exponentiation $y = 7^3 (\mathrm{mod}\ 11)$.

Step 3: By comparing the computation result of y for every feasible value of x with given $y = 2$, so I can obtain the solution of the discrete logarithm problem.

4.2 The nondeterministic algorithm for breaking Diffie-Hellman key exchange

Diffie-Hellman method is used for symmetric key exchange between entities. The steps of the algorithm are as follows:

(1) Key generation

(a) Generate a large random prime integer p and a generator α of the multiplicative group U_p. The set U_n is the multiplicative group of Z_n with order $\varphi(n)$, which is defined as follows:

$U_n = \{[a] \in Z_n : GCD(a, n) = 1\}$

(b) Select two random integers a and b, $1 \leq a,\ b \leq p-1$ for entity A and B, then compute $x = \alpha^a (\mathrm{mod}\ p)$ and $y = \alpha^b (\mathrm{mod}\ p)$.

(c) Broadcast the public key of A and B where A's public key is (α, x) and B's public key is (α, y).

(2) A computes the symmetric key $k_a = y^a (\mathrm{mod}\, p)$ and B computes the symmetric key $k_b = x^b (\mathrm{mod}\ p)$. A and B will use the Diffie-Hellman method to exchange symmetric keys.

In order to get the symmetric key K which is equal to k_a or k_b between the users A and B, I can give the nondeterministic algorithm for breaking the Diffie-Hellman key exchange as follows:

First, I can get value of the secret key of user A denoted as a according to the public key (α, x) and the primer p. As a matter of fact, this process can be performed by the method of solving the discrete logarithm problem in the finite field $GF(p)$ which is introduced above. I can design the all needed tiles in the computation, including the basic types of tiles, boundary tiles and the seed configuration by the binary form of the integer x, α and p. When all kinds of the tiles and the seed configuration are prepared, I puts them together into the reaction buffer, there are many choices at the first position of the seed configuration, it can nondeterministic guess the value of a under the condition $1 \leq a \leq p-1$. So the modular exponentiation $\alpha^a (\mathrm{mod}\, p)$ can be parallely performed by the three small systems, the assembly complexes can grow, therefore I can obtain the solutions of the problem. Then I can read the result of the process of the DNA tile growth using a combinational of PCR and gel electrophoresis.

Second, I can make the modular multiplication $K = y^a (\mathrm{mod}\ p)$ together with the value of a obtained at the first step and the public key (α, y) of the user B.

For example, suppose the multiplicative group U_{11} and the generator $\alpha = 7$. The user A selects one random integers a as his own secret key, at the same time, the users A and B broadcast their public key $x = 2$ and $y = 3$ respectively.

Considering the equation $x = 7^a (\mathrm{mod}\ 11)$, I can get the solution of the discrete logarithm problem in the finite field $GF(11)$ which is the secret key of the user A by the method described above, and the result of a is equal to 3. So the common key can be computed as $K = y^a (\mathrm{mod}\ p) = 3^3 (\mathrm{mod} 11) = 5$.

4.3 Complexity and probability analysis

Considering the nondeterministic algorithm for breaking Diffie-Hellman key exchange based on self-assembly of DNA tiles, the complexity of this algorithm can be computed in terms of computation time and the number of distinct tiles required. Generally, for the equation $y = g^x (\mathrm{mod}\ p)$, here, $1 \leq x \leq p-1$, suppose the number of the bits of g be k, and for the given problem, the value of k is a constant.

For a round modular multiplication, there is one addition operation and at most two subtraction operations followed by the comparison operation respectively, so the upper bound of the computation time T which is the number of assembled steps is $x(6k+2)+2 = \Theta(p)$.

Finally, these distinct tile types include the input boundary tiles and computation boundary tiles. As the basic tiles described in Section 3, the tile types are all $\Theta(1)$, so I can obtain the basic tile types are $\Theta(1)$ for a round modular multiplication, then I also consider the labels to distinguish the number of the rounds which can be determined by the size of $x(1 \leq x \leq p-1)$, therefore the tile types needed for this algorithm is $\Theta(p)$.

Now, I give the probability analysis for this algorithm.

Theorem 4.1. Let each tile that may attach to a configuration at a certain position attach there with a uniform probability distribution. For all integers $x(1 \leq x \leq p-1)$, n_p is denoted as the number of the bits of the primer p, for given integers g and y, then the probability of assembling a particular final configuration which attaches the value of x such that $y = g^x (\mod p)$ is at least $(\frac{1}{p-1})^{n_p}$.

Proof. Now I calculate the probabilities of each tile attaching in its proper position and then multiply those probabilities together to get the overall probability of a final configuration.

On one hand, there are $(p-1)$ possible tiles that may attach to the positions which represent the bits of the primer P in the seed configuration, and only one is correct, so the probability of it attaching is $P_i = \frac{1}{p-1}(1 \leq i \leq n_p)$. On the other hand, for the next two positions, there is only one tile that may attach, so the probability of the last two tiles are both $P_{n_p+1} = P_{n_p+2} = 1$.

The overall probability of a specific final configuration can be computed as the following formulas:

$\prod_{i=1}^{n_p+2} P_i = (\frac{1}{p-1})^{n_p}$

This is the probability of a nondeterministic assembly successfully identifying the solution. I use the nondeterministic computation to perform the function $y = g^x (\mod p)$, and construct three small systems to define the whole tile systems that use $\Theta(p)$ input tile types, and assemble in $\Theta(p)$ steps with probability of each assembly finding the solution at least $(\frac{1}{p-1})^{n_p}$. Therefore a parallel implementation of this tile system such as the execution of the DNA tile self-assembly model in [12], with $(p-1)^{n_p}$ seeds has at least a $(1-e^{-1})$ chance of finding the secret key of the user A or B, and one with $100(p-1)^{n_p}$ seeds has at least a $(1-e^{-100})$ chance. The probability of finding the successful solutions among the many parallel executions can be proved to be arbitrarily close to 1.

5 Summary and Conclusions

DNA tile self-assembly is looked forward to many applications in different fields. In this paper, I show how the DNA self-assembly process can be used for breaking Diffie-Hellman key exchange based on the discrete logarithm problem in the finite field $GF(p)$ by computing the modular multiplication. The advantage of our method is that once the initial strands are constructed, each operation can compute fast parallelly through the process of DNA self-assembly without any participation of manpower, thus the algorithm is proposed which can be successfully solved the modular multiplication, and then can be used to break Diffie-Hellman key exchange with the computation time complexity of $\Theta(p)$, and the probability of finding the successful solutions among the many parallel executions is proved to be arbitrarily close to 1.

Acknowledgments

This work was supported by the Research Project of Department of Education of Zhejiang Province (Y201120124) and the National Natural Science Foundation of China (Grant Nos. 60703047, 60803113, 60903105).

Bibliography

[1] L. Pan and C. Martin-Vide, Solving multidimensional 0−1 knapsack problem by P systems with input and active membranes, *Journal of Parallel and Distributed Computing*, Vol. 65, pp. 1578-1584, 2005.

[2] L. Pan and M. J. P¨Śrez-Jim¨Śnez, Computational complexity of tissue-like P systems, *Journal of Complexity*, Vol. 26, pp. 296-315, 2010.

[3] L.M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, Vol. 266ŁŹ pp. 1021-1024, 1994.

[4] L.Q. Pan, G.W. Liu , J. Xu, Solid phase based DNA solution of the coloring problem, *Progress in Natural Science*. vol. 14, pp. 104-107, 2004.

[5] A. Carbone, N.C. Seeman, Molecular tiling and DNA Self-assembly, *Springer-Verlag Berlin Heidelberg*, Vol. 2950, pp. 61-83, 2004.

[6] N.C. Seeman, DNA nanotechnology: novel DNA constructions, Annu. Rev. Biophy. *Biomol. Struct.*, Vol. 27, pp. 225-248, 1998.

[7] C. Mao, W. Sun, N.C. Seeman, Designed two dimensional DNA Holliday junction arrays visualized by atomic force microscopy, *J. Am. Chem. Soc.*, Vol. 121, pp. 5437-5443, 1999.

[8] E. Winfree, On the computational power of DNA annealing and ligation, *DNA Based Computers*, pp. 99-221, 1996.

[9] T. Eng, Linear self-assembly with hairpins generates the equivalent of linear context-free grammars, *3rd DIMACS Meeting on DNA Based Computers*, Univ. of Penn., 1997.

[10] R. Barish, P.W. Rothemund, E. Winfree, Two computational primitives for algorithmic self-assembly: copying and counting, *Nano Letters*, Vol. 12, pp. 2586-2592, 2005.

[11] P. M. Espane/s, A. Goel, Toward minimum size self-assembled counters, *Springer Science Business Media B.V.*, 2008.

[12] P.W. Rothemund, N. Papadakis, E. Winfree, Algorithmic self-assembly of DNA Sierpinski triangles, *PLoS Biology*, Vol. 12, pp. 2041-2053, 2004.

[13] M. Cook, P.W. Rothemund, E. Winfree, Self assembled circuit patterns, *DNA*, pp. 91-107, 2004.

[14] C. Mao, T.H. LaBean, J.H. Reif, Logical computation using algorithmic self-assembly of DNA triple-crossover molecules, *Nature*, Vol. 407, pp. 493-496, 2000.

[15] Y. Brun, Arithmetic computation in the tile assembly model: addition and multiplication, *Theoretical Computer Science*, Vol. 378, pp. 17-31, 2006.

[16] G.L. Michail, T.H. LaBean, 2D DNA self-assembly for satisfiability, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 44, pp. 139-152, 1999.

[17] Y. Brun, Nondeterministic polynomial time factoring in the tile assembly model, *Theoretical Computer Science*, Vol. 395, pp. 3-23, 2008.

[18] Y. Brun, Solving NP-complete problems in the tile assembly model, *Theoretical Computer Science*, Vol. 395, pp. 31-36, 2008.

[19] A. Gehani, T.H. LaBean, J.H. Reif, In DNA Based Computers: Proceedings of a DIMACS Workshop, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1999.

[20] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Commun. ACM*, Vol. 21, pp. 120-126, 1978.

[21] I.R. Jeong, J.O. Kwon, D.H. Lee, A Diffie-Hellman key exchange protocol without random oracles, *Springer-Verlag Berlin Heidelberg*, Vol. 4301, pp. 37-54, 2006.

[22] ANSI X9.30. Public Key Cryptography for the Financial Services Industry: Part 1: The Digital Signature Algorithm (DSA), *American National Standard Institute*, American Bankers Association, 1997.

[23] N. Koblitz. Elliptic curve cryptosystem, *Math. Comp.*, Vol. 48, pp. 203-209, 1987.

[24] G. Blakley, A computer algorithm for calculating the product $A * B \mod M$, *IEEE Transactions on Computers*, Vol. C-32, pp. 497-500, 1983.

[25] T. Acar, B.S. Kaliski, C. Koc, Analyzing and computing Montgomery multiplication algorithms, *IEEE micro*, Vol. 16, pp. 26-33, 1996.

[26] W. Diffie, M.E. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory*, Vol. 22, pp. 644-654, 1976.

[27] Z.H. Chen, Breaking the Diffie-Hellman key exchange algorithm in the tile assembly model, *Chinese Journal of Computers*, Vol. 31, pp. 2116-2122, 2008.

[28] E. Winfree, Algorithmic self-assembly of DNA, *Ph.D. Thesis*, Caltech, Pasadena, CA, June 1998.

[29] P.W. Rothemund, E. Winfree, The program-size complexity of self-assembled squares, *ACM Symposium on Theory of Computing (STOC)*, pp. 459-468, 2001.

[30] H. Wang, Proving theorems by pattern recognition, *I. Bell System Technical Journal*, Vol. 40, pp. 1-42, 1961.

[31] E. Marcelo Kaihara, T. Naofumi, A hardware algorithm for modular multiplication/division, *IEEE Transactions on Computers*, Vol. 54, pp. 12-21, 2005.

[32] P.L. Montgomery, Modular multiplication without trial division, *Mathematics of Computation*, Vol. 44, pp. 519-521, 1985.