A Public-Key Cryptosystem Based On Stochastic Petri Net

Zuohua Ding[a], Hui Zhou[a], Hui Shen[a], Qi-wei Ge[b]
[a] Lab of ICSE, Zhejiang Sci-Tech University, Hangzhou, Zhejiang, 310018, China
Email: zouhuading@hotmail.com
[b] Faculty of Education, Yamaguchi University, 1677-1 Yoshida, Yamaguchi 753-8513, Japan
Email: gqw@yamaguchi-u.ac.jp

Abstract— **In this paper, we present a new method to build public-key Cryptosystem. The method is based on the state explosion problem occurred in the computing of average number of tokens in the places of Stochastic Petri Net (SPN). The reachable markings in the coverability tree of SPN are used as the encryption keys. Accordingly, multiple encryption keys can be generated, thus we can perform multiple encryption to get as strong security as we expect. The decryption is realized through solving a group of ordinary differential equations from Continuous Petri Net (CPN), which has the same underlying Petri net as that of SPN. The decipherment difficulty for attackers is in exponential order. The contribution of this paper is that we can use continuous mathematics to design cryptosystems besides discrete mathematics.**

Index Terms— **Public-key cryptosystem, Stochastic Petri net, Continuous Petri net, Multiple encryption.**

I. Introduction

The Internet has become so widespread that any one can obtain and provide information easily through the public network. To guarantee the safe communications, the utilization of cryptography becomes very important in order to avoid the leak of secret information or dishonest alternation of the information.

There are two types of cryptography: private-key and public-key cryptosystems. Private-key cryptosystem uses a common private key to encrypt and decrypt messages, and can process encryption and decryption very fast, but it faces a problem to distribute the private key through the public network without leaking of the secrecy [22] [23]. Public-key cryptosystem successfully solves this problem by preparing a pair of keys (public and private keys) in the way that it opens the public key to the public and keeps the private key in secret [19].

Currently, there are a few public-key cryptosystems, such as RSA [21], ElGamal [4], and the elliptic curve cryptography [18]. Since the security(encryption or decryption) of RSA is $O(exp(c(logn)(loglogn))^{1/2})$ (c is a constant, and n is a large factoring number), which is subexponential order rather than an exponential order, we may classify these crytosystems as subexponential cryptosystems.

This work is partially supported by the NSF under Grant No. 61170015 and No.61210004.

Recently, Ge and Okamoto [7] proposed a new public-key cryptography, namely PNPKC, and Ge et al. [8] proposed a Multiple-Encryption Public-Key Cryptography, namely MEPKC. Both Cryptography use elementary T-invariants of Petri nets as the public keys. The security is e^m, where m is the number of transitions of the Petri net. If the encryption is performed in k stages, then the security will be $(e^m)^k$. Such kind of crytosystems can be called exponential cryptosystems.

So far, the public-key cryptosystems are designed with discrete mathematics. In this paper we propose a new public-key cryptography by using continuous mathematics. The encryption part is based on the well known state explosion problem of stochastic Petri nets(SPN), i.e. the state space will increase exponentially as the number of places or the number of initial marking values increases, even through many reduction skills have been proposed to combat this problem such as [13] [26]. The decryption part is based on a group of ordinary differential equations, which can be easily solved by using Runge-Kutta algorithm [9].

The encryption key is composed of a reachable marking in the coverability tree of SPN and the average number of tokens in the places of SPN. A brief description is in the following: The sender randomly select a group of initial markings from a well designed range to generate the coverability tree of a SPN, and compute the average number of tokens of SPN by some software such as SPNP and GreatSPN. By combing this reachable marking and the average number of tokens together in some way, we will get the encryption key. This key is a number that has integral part and decimal fraction part.

The uniqueness of reachable markings guarantee that the encryption key is unique. The key is hidden in a knapsack problem, which is NP-complete. Multiple encryption keys can be generated, thus we can perform multiple encryption to get as strong security as we expect. Hence, we open a key generator to the public.

For the attacker to decrypt, he/she has three difficulties: 1) To compute all the average numbers of tokens of the Petri nets that have initial marking from the given range. If we carefully design the maximum value of the range, then it is hard for attackers to solve all the average number of tokens even using some software. 2) From these average numbers of tokens, the attacker needs to determine which one matches the decimal fraction, and thus to determine the initial marking value selected by the sender. 3) The attacker needs to generate the coverability tree based on this initial marking value, and to determine which reachable marking in the tree has been selected by the sender, in other words, to determine the integral part of of the key.

For the receiver to decrypt the ciphertext, he/she also needs to solve all the average number of tokens of the Petri nets that have initial marking values in the given range. However, the receiver can get these average numbers of tokens by solving a set of ordinary differential equations, which is simple. In the equations, the initial values correspond to the initial markings. The set of equations comes from Continuous Petri Net (CPN), of which the underlying Petri net is the same as that of SPN. CPN is defined recently by David and Alla [2] [3] attempting to deal with the state explosion problem of discrete Petri net by removing the integrality constraints. The solutions of the equations are called state measures in the places. In this paper, we have proved that the state measures in the places equal the average numbers of tokens in the places.

Analysis shows that the security for our method is $v^{2I}O(na^{P(k,n)}b^{Q(k,n)})$, $a > 1, b > 1$, where v is the length of the range of initial marking values, I is the number of places that can be set the initial markings from the range, n is the number of places of the Petri net, k is the maximum number of tokens in the initial markings, and $P(k,n), Q(k,n)$ are two polynomials of k, and n. If applying r-stage encryption, the security can reach $v^{2Ir}O(n^r a^{rP(k,n)} b^{rQ(k,n)})$.

We are not claiming that our public-key encryptosystem is superior to the existing encryptosystems. It is our attempt to design the public-key encryptosystem from a different angle, and give some suggestions in this field.

The rest of the paper is organized as the following: Section 2 briefly introduces the method to compute the average number of tokens of SPN and gives the computational complexity. Section 3 presents a method to compute the state measures of CPN and analyzes the computational complexity. Section 4 proves that the average numbers of tokens of SPN equal the state measures of CPN. Section 5 discusses the security of our SPN based public-key cryptograph. In Section 6, we use a concrete example to illustrate how to use our method to design a public-key cryptograph. Section 7 discusses a related work. The last section, section 8 concludes the paper by simply describing how to combine our system with other encryption system for digital signature, and indicating our future work.

II. The Method To Compute The Average Number of Tokens of SPN and Its Computational Complexity

A. Stochastic Petri Nets(SPN)

The following definition comes from [14].

Definition 2.1: A **continuous stochastic Petri net** is a tuple $SPN = (P, T, A, M_0, \lambda)$, where (P, T, A, M_0) is a underlying untimed Petri net: $P = \{p_1, p_2, \ldots, p_n\}$ is the set of places, $T = \{t_1, t_2, \ldots, t_m\}$ is the set of transitions, $A \subset (P \times T) \cup (T \times P)$ is the set of arcs, and M_0 is the initial marking. $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ is a set of average firing rates of transitions satisfying exponential distributions:

$$\forall t \in T,\ F_t(x) = P\{X_t \le x\} = 1 - e^{-\lambda_t x}$$

where x represents time, X_t is a continuous random variable representing the time delay for transition t, λ_t is the average firing rate associated with transition t.

A Markov process is a stochastic process that satisfies the *Markovian property*

$$P\{X(\tau) \le x | X(t), t \in [0, \theta]\} = P\{X(\tau) \le x | X(\theta) = y\}$$

for any $\tau > \theta$. Markov processes with a discrete state space are called Markov chains. If the parameter t is continuous, the process is a continuous-time Markov chain (CTMC). The time spent in states of a CTMC is a random variable with nonnegative exponential probability density function(pdf).

In practice, a CTMC is described through either a state transition rate diagram or a transition rate matrix, denoted by Q. The state transition rate diagram is a labelled directed graph whose vertices are labelled with the CTMC states, and whose arcs are labelled with the rate of the exponential distribution associated with the transition from a state to another.

If we have the reachable graph (or coverability tree) of SPN, then replacing the firing transition t associated with the arc by average firing rate λ_t (or marking related λ_t), we will get the CTMC that is isomorphic to SPN. Actually we have the following result [15].

Theorem 2.2: Any SPN with finite places and finite transitions is isomorphic to a Continuous Time Markov Chain.

This result enables us to compute average number of tokens in places of SPN.

B. Method To Compute The Average Number of Tokens In Places of SPN

Average number of tokens in places is an important parameter in system performance analysis. It can be solved through the following steps [15].

(1) Constructing transition rate matrix $Q = [q_{ij}]$

The transition rate matrix can be constructed from the coverability tree of SPN. For the elements a_{ij} outside the main diagonal: if there is an arc from state M_i to state M_j, then the value is the rate of the exponential distribution associated with the transition from M_i to M_j; if there is no arc connected, then this element is 0. For the element a_{ii} on the main diagonal, its value is the opposite of the sum of firing rates outputted from state M_i.

(2) Finding the steady state probability $X = (x_1, x_2, \ldots, x_l)$, where l is the number of all reachable markings in the coverability tree of SPN.

This can be obtained by solving the equation group

$$\begin{cases} XQ = 0, \\ \sum_i x_i = 1, \qquad 1 \le i \le l. \end{cases}$$

where Q is a l dimension square matrix.

(3) Constructing place state table

Based on the coverability tree of SPN, we list all the reachable markings. Every reachable marking is a n-dimension vector, which is a row in the table. Thus the table has l rows and n columns.

(4) Obtaining the token probability density function (pdf)

That is to find in the steady state the probability of the number of tokens contained in a place. For a $p \in P$, letting $P[M(p) = i]$ denote the probability of i tokens contained in place p, then the token pdf can be obtained as

$$P[M(p) = i] = \sum_j P[M_j],$$

where $M_j \in R(M_0)$ and $M_j(p) = i$.

(5) Finding the average number of tokens in places

For any $p \in P$, in the steady state, $u_i = \sum_j j \times P[M(p_i) = j]$ is the average number of tokens in place p for any reachable marking.

C. The complexity of finding the average number of tokens in the places

Currently, the best algorithm to solve the equation group

$$\begin{cases} XQ = 0, \\ \sum_i x_i = 1, \qquad 1 \le i \le l. \end{cases}$$

is from Harrow et al. [10] and the complexity is $O(l)$. l depends on n, k and can be estimated as the following. Let k be the maximum number of tokens in the initial markings. Since the size of the reachability graph increases exponentially with both the number of places and the number of tokens in the initial marking, we may assume that $l = a^{p(k,n)}$, $a > 1$, where $p(k,n)$ is a polynomial of k and n. Thus, the complexity to solve this equation group is $O(a^{p(k,n)})$.

From step (3) of Section 2.2, the complexity to find the average number of tokens in a single place is $O(a^{p(k,n)}) + O(a^{p(k,n)}) = O(a^{p(k,n)})$. Hence to find average number of tokens in n places, the complexity is $n \times (O(a^{p(k,n)}) + O(a^{p(k,n)})) = O(na^{p(k,n)})$.

III. The Method To Compute The State Measures of CPN and Its Computational Complexity

Since the size of reachable markings in the coverability tree increases exponentially with both the number of places and the number of tokens in the initial marking. As a result, we will not be able to compute the steady state probability for very large models. We will use some relaxation technique to overcome this difficulty. This relaxation leads to a continuous-time formalism: Continuous Petri Net (CPN).

A. Continuous Petri Net

Definition 3.1: A *Continuous Petri Net* is a tuple $CPN =< P, T, A, M_0, \lambda >$, where (P, T, A) is a underlying Petri net MP: $P = \{p_1, p_2, \ldots, p_n\}$ is the set of places, $T = \{t_1, t_2, \ldots, t_m\}$ is the set of transitions, $A \subset (P \times T) \cup (T \times P)$ is the set of arcs, and M_0 is the initial marking. $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ is a set of average firing rates of transitions.

In the definition, the average firing rate λ_t is actually the reciprocal of firing delay of transition t. In a CPN, the marking of a place is no longer an integer but a nonnegative real number.

Definition 3.2: Let $I = [0, \infty)$ be the time interval and let $m_i : I \to [0, \infty), i = 1, 2, \ldots, n$ be a set of mappings that associated with place p_i. A marking of a Continuous Petri Net $CPN =< P, T, A, v >$ is a mapping

$m : I \to [0, \infty)^n$,

$m(\tau) = (m_1(\tau), m_2(\tau), \ldots, m_n(\tau))$.

Definition 3.3: (State Measure) Given any time moment $t \in [0, \infty)$, the marking value in a place is called the State Measure of this place, denoted as $m(t)$. State measures take nonnegative real numbers as their values.

A transition is enabled if all the input places have nonzero markings. Only enabled transitions can be fired. So, if some marking is moved into a place, we say that the state measure in this place is increasing; if some marking is moved out from a place, we say that the state measure in this place is decreasing. The change rate of state measure can be calculated as the following.

Let p_1 and p_2 be the input places of a transition t and their markings are $m_1(\tau)$ and $m_2(\tau)$, respectively. Let λ_t be the firing rate associated with t, then following the definition of Continuous Petri net defined by David and Alla [2] [3], the marking moving out from p_1 and p_2 is defined by $\lambda_t \times min\{m_1(\tau), m_2(\tau)\}$. If t has only one input p_1, then marking $\lambda_t \times m_1(\tau)$ will be moved out from p_1.

We consider the following type of Petri net as the underlying Petri net of SPN and CPN. It is a subclass of normal Petri net net, namely Message Passing Petri net, or MP for short.

Definition 3.4: A Petri net P is MP if

- P has finitely many places;
- the places of P are partitioned into two disjoint partitions C and B;
- each place from C has one or two input transitions and one or two output transitions, but can not have two input transitions and two output transitions at the same time;
- each place from B has one input transition and one output transition;
- each transition has one input place from C and one output place from C; and
- each transition has either
    1) no input places from B and no output places from B;
    2) no input places from B and one output place from B;

3) one input place from B and no output places from B; or
4) one input place from B and one output place from B.

Here C stands for "Internal States", while B stands for "Buffer".

In a MP, the Internal States and the directly connected transitions form one or several place/transition cycles, namely process net. Hence a MP can also be described as process nets that interact to each other through Buffer B.

With such kind of Petri net structure, the state space still increases exponentially as the number of tokens in the Internal States or Buffer increases.

B. Solving State Measures From Ordinary Differential Equation Model

Based on the semantics defined above, the state measure at each place can be calculated from a differential equation. We have the following cases.

1) One place to one place. As Fig.1 shows, place p will get marking from place p_1. Let the marking at place p and p_1 be m and m_1, respectively. Assume that the firing rates at transition t_1 and t are d_1 and d, respectively. Then the state measure m can be represented as

$$m'(\tau) = d_1 \times m_1(\tau) - d \times m(\tau). \quad (1)$$

If at least one of t_1 and t is not enabled, then $m_1(\tau) = 0$ or/and $m(t) = 0$. Hence the above equation is still true in these situations.

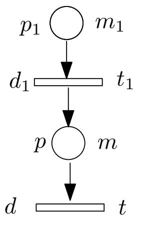

Figure 1. One place to one place model.

2) Two place to one place. As Fig. 2 shows, place p will get marking from place p_1 and p_2. Let the markings at place p_1, p_2 and p be m_1, m_2 and m, respectively. Assume that the firing rates at transition t_1 and t are d_1 and d, respectively. Then the state measure m can be represented as

$$m'(\tau) = d_1 \times min\{m_1(\tau), m_2(\tau)\} - d \times m(\tau). \quad (2)$$

If t_1 is not enabled, then $m_1(\tau) = 0$ or/and $m_2(\tau) = 0$. If t is not enabled, then $m(\tau) = 0$. Hence the above equation also covers these situations.

3) One place to two places. As Fig. 3 shows, place p will get marking from place p_1, but will send some marking out together with place p_2. Let the markings at place p_1, p_2 and p be m_1, m_2 and m, respectively. Assume that the firing rates at transition t_1 and t are d_1

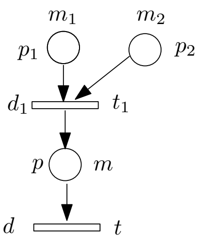

Figure 2. Two places to one place model.

and d, respectively. Then the state measure m can be represented as

$$m'(\tau) = d_1 \times m_1(\tau) - d \times min\{m(\tau), m_2(\tau)\}. \quad (3)$$

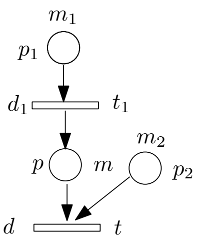

Figure 3. One place to two places model.

If t is not enabled, then either $m(\tau) = 0$ or $m_2(\tau) = 0$, or both. Hence the above equation is still true.

4) Two places to two places. As Fig. 4 shows, transition t_1 has two input places m_1 and m_2 and transition t has two input places m and m_3. Assume the firing rates at transition t_1 and t are d_1 and d, respectively. Then the marking m can be represented as

$$\begin{aligned} m'(\tau) = d_1 \times min\{m_1(\tau), m_2(\tau)\} \\ -d \times min\{m(\tau), m_3(\tau)\}. \end{aligned} \quad (4)$$

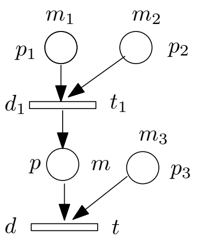

Figure 4. Two places to two places model.

From the differential equation model, we see that state measures are uniquely determined by the system structure and the firing rates.

C. The Computational Complexity

To get the state measures of all the places, we need to solve the initial value problem of ordinary differential equations.

Generally speaking, it is hard to find explicit analytic solutions for nonlinear ordinary differential equations, thus most of the time, we turn to find numerical solutions instead. We may use function $ode45$ in Matlab to solve our equations. Function $ode45$ is the implementation of combined fourth and fifth-order Runge-Kutta method. $ode45$ is designed to handle the following general problem

$$\frac{dx}{dt} = f(t,x),\ x(t_0) = x_0,$$

where t is the independent variable and x is a vector of dependent variables to be found.

Numerical solutions may give us computational errors due to the algorithm and the machine. In our situation, the computation error can come from two sources: truncation error (because a truncated Taylor Series is used in the computation), and rounding error (because a finite number of binary digits is used inside the machine).

For the truncation error, since the fourth-order Runge-Kutta method has local truncation error $O(h^5)$ and the fifth-order Runge-Kutta method has local truncation error $O(h^6)$, where h is the step size, thus the global truncation error of $ode45$ is $O(h^5)$ [9]. Noticing that the fifth-order Runge-Kutta method can automatically adjust the step size, thus $ode45$ can approximate to the given accuracy by setting $opts$ with command $odeset$. For the rounding error, since implicit Runge-Kutta method has stable area [9], and the algorithm is guaranteed to converge in the stable area. Thus the rounding error of the perturbation can not increase and will decrease to 0 in the iteration process [1].

For the complexity of Runge-Kutta method, if the accuracy is lower than 0.00001, then Runge-Kutta method is more efficient than Newton method. We know that the complexity for Newton method in general is $O(mn^3)$, where n is the number of variables and m is the iteration, which is usually $O(n)$ and never exceeds $O(n^2)$ [25]. Hence, the complexity for Runge-Kutta method in general is $O(n^4)$ and never exceeds $O(n^5)$.

Thus solving the state measure needs time $O(n^5)$, where n is the number of places.

IV. Average Numbers of Tokens of SPN = State Measures of CPN

SPN and CPN describe the events of the system, execution time of events, and the relations between events. In both models, the average execution time of events represent the firing rates of the corresponding transitions. The difference is located in: 1) In SPN, the average number of tokens are obtained by computing Markov chain, while in CPN, the state measures are obtained by solving ordinary differential equations; 2) The computational complexity for SPN is exponential while the computational complexity for CPN is polynomial.

The following result gives us the relation of SPN and CPN.

Theorem 4.1: If SPN and CPN are modeling the same system, then the average number of tokens in places of SPN equal the state measures in places of CPN.

Proof 4.2: Our proof is on the basis of [12]. We consider three situations.

(1) Net with single input

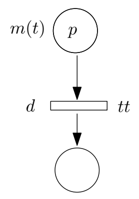

Figure 5. A transition with single input.

Fig. 5 shows a transition with a single input. In the figure, p is the input place of transition tt, $m(t)$ represents the state measure of p at time t, d is the average firing rate of tt, W is the time delay of tt. Assume that p has initial value k_1 at time t_0. In SPN, a transition needs time from enabled to firing, and this time is represented by a random variable W, which is subjected to an exponential distribution function: $F_{tt}(\Delta t) = P[W \leq \Delta t] = 1 - e^{-d\Delta t}$, $\Delta t > 0$. Assume that after Δt time, the average number of tokens of p is k_1', then

$$k_1' = E[m(p)] = k_1 P[W > \Delta t] = k_1 e^{-d\Delta t}.$$

In CPN, for this net, we have equation $x' = -dx$ and $x(t_0) = k_1$. Solving this equation, we get $m(t_0 + \Delta t) = k_1 e^{-d\Delta t}$.

Thus in this case, our result is correct.

(2) Net with two inputs

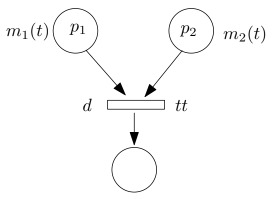

Figure 6. A transition with two inputs.

Fig. 6 shows a transition with two input places. In the figure, p_1 and p_2 are two input places of transition tt, $m_1(t)$ and $m_2(t)$ represent state measures of p_1 and p_2 at time t, respectively. d is the average firing rate of tt, W is the delay of tt. Assume that p_1 has initial value k_1 at time t_0 and p_2 has initial value k_2 at time t_0.

In SPN, let k_1' and k_2' be the new average numbers of tokens of p_1 and p_2 after time Δt, respectively. Then

$$\begin{aligned} k_1' &= E[m(p_1)] \\ &= k_1 - min\{k_1, k_2\} P[W \leq \Delta t] \\ &= k_1 - min\{k_1, k_2\}(1 - P[W > \Delta t]) \\ &= k_1 - min\{k_1, k_2\}(1 - e^{-d\Delta t}). \end{aligned}$$

In CPN, for this net, we have equations:

$$x_1' = -d \times min\{x_1, x_2\},\ x_2' = -d \times min\{x_1, x_2\}$$

with the initial values: $x_1(t_0) = k_1, x_2(t_0) = k_2$. Since $min\{x_1, x_2\}$ is nondeterministic at every time t, it is hard to give analytic solution. With numerical solution, we may partition Δt into several small intervals Δt_i, such that on each such interval, $min\{x_1, x_2\}$ will take a fixed value. Now we solve the equations on interval Δt_1 as the follows.

i) If $x_1(t_0) < x_2(t_0)$, or $k_1 < k_2$, then $x_1' = -dmin\{x_1, x_2\} = -dx_1$, and $k_1' = x_1(t_0 + \Delta t_1) = k_1 e^{-d\Delta t_1}$.

ii) If $x_1(t_0) \geq x_2(t_0)$, or $k_1 \geq k_2$, then $x_1' = -dx_2$, $x_2' = -dx_2$, then $x_2(t_0 + \Delta t_1) = k_2 e^{-d\Delta t_1}$. Since in the steady state, the output markings from p_1 and p_2 are equal, and the output marking from p_2 is $k_2 - k_2 e^{-d\Delta t_1} = k_2(1 - e^{-d\Delta t_1})$, so the output marking from p_1 should be $k_2(1 - e^{-d\Delta t_1})$. Thus $k_1' = k_1 - k_2(1 - e^{-d\Delta t_1})$.

Combining i) and ii), we get $k_1' = k_1 - min\{k_1, k_2\}(1 - e^{-d\Delta t_1})$, which is the same as the average marking of SPN. We will get similar results on other intervals. Hence in this case, our result is also correct.

(3) Now consider the general cases for Fig. 1, Fig.2, Fig.3, and Fig.4. In SPN, for $\forall t \in T$, its marking flow rate, i.e. average marking moving to the output place in unit time, is $R(t,s) = W(t,s) \times \sum_{M \in E} P(M) \times \lambda$, where E is the set of all reachable markings that make t enable, λ is the average firing rate of t, $W(t,s)$ is the weight attached to the arc from transition t to place s. From the proof of (1) and (2), we know that $R(t,s) = d \times m$, or $R(t,s) = d \times min\{m_1, m_2\}$, where m, m_1, m_2 are the input places of tt, and d is the average firing rate.

Hence for the place m in Fig. 1, Fig.2, Fig.3, and Fig.4, at time t, *the average number of markings of a place = average number of markings to this place - average number of markings out this place*, i.e., $\int_0^t (d_1 m_1 - dm)dt$ for Fig. 1; $\int_0^t (d_1 min\{m_1, m_2\} - dm)dt$ for Fig.2; $\int_0^t (d_1 m_1 - dmin\{m_2, m\})dt$ for Fig.3; $\int_0^t (d_1 min\{m_1, m_2\} - dmin\{m_3, m\})dt$ for Fig.4. These expressions are actually the solutions of Equation (1), Equation (2), Equation (3), and Equation (4) in Section 3.2, respectively. Hence, *the average number of tokens of m in SPN = state measure value of m in CPN.*

Hence, we complete the proof.

V. Public-Key Cryptosystem

A. Public Key and Private Key

1) Public Key: Let $\langle \sum, H \rangle$ be a public key, where $\sum = (P, T, A, M_0, \lambda)$ is a bounded Stochastic Petri Net that has an MP as the underlying Petri net. H is a Hash fucntion which will be defined in the following.

For any $p_i \in P$, let $M_0(p_i)$ represent the initial marking value for place p_i with the range $n_1 \leq M_0(p_i) \leq n_2$, $n_1, n_2 \in N^+$, $i = 1, \ldots, n$. Assume that the sender randomly selects the initial markings $M_0^+(p_i)$ from these range. Define

$$M_0'(p_i) = \begin{cases} H_1(M_0^+(p_i)), & M_0(p_i) \neq 0; \\ 0, & M_0(p_i) = 0. \end{cases}$$

Here H_1 is usually a linear function depending on the size of SPN, the maximum number of tokens in the initial markings, and the memory size. Let $CT(\sum(M_0'))$ be the coverability tree of the net. Define V by $V = H(M_0^+(p_i), J_i, U) = V_2 + V_3 = R_1 J_i^T + 10^{-p} R_2 U^T$, where

- J_i is the row vector of n dimension of $CT(\sum(M_0'))$; J_i^T is the transpose of J_i; R_1 is a n dimension vector with positive integers; $V_2 = H_2(J_i) = R_1 J_i^T$, where H_2 is 1-1 mapping; R_2 is a randomly selected n dimension vector with positive integers.
- $V_3 = H_3(U) = 10^{-p} R_2 U^T$, where $U = (u_1, u_2, \ldots, u_n)$, u_i is the average number of tokens in place p_i of SPN, that can be solved by the sender. $p(\in N)$ is an integer depending on the selection of R_2 and $M_0(p_i), n_1 \leq M_0(p_i) \leq n_2$, which can determine as the following. The receiver calculates the maximum value of all $R_2 U^T$ by solving the equation groups as discussed in Section 3.3. Let M be the maximum. Then take $p = min\{q \in N | 10^q \geq M\}$. This p can guarantee that the number $10^{-p} max\{R_2 U^T\}$ is a decimal fraction, thus the number $V_3 = H_3(U) = 10^{-p} R_2 U^T = g$ from the sender is also a decimal fraction. The receiver then computes all the V_3, and finds which V_3 is the most closest one to this g. The initial markings for this closest V_3 are exactly the initial markings selected by the sender. However, the attacker does not know how to generate p. In order to get p, the attacker has to has to try every possible initial marking, and calculate all possible V_3 from $CT(\sum(M_0'))$. The complexity for the attacker will be in multiple exponential.

2) Private Key: The private key is designed as $< E, \overline{H_2} >$, where E is a set of equations consisting of the following six types of ordinary differential equations generated from Petri net MP:

- Type 1 [Internal]. $m_i' = \tilde{d}_{i-1} m_{i-1} - \tilde{d}_i m_i$. Here m_i and m_{i-1} are the states of the same process net.
- Type 2 [Input-before]. $m_i' = \tilde{d}_{i-1} min\{m_{i-1}, x_k\} - \tilde{d}_i m_i$. Here m_i and m_{i-1} are the states of the same process net. m_k is the input to this process net from buffer.
- Type 3 [Input-after]. $m_i' = \tilde{d}_{i-1} m_{i-1} - \tilde{d}_i min\{m_i, m_k\}$. Here m_i and m_{i-1} are the states of the same process net. m_k is the input to this process net from buffer.
- Type 4 [Input-before-after]. $m_i' = \tilde{d}_{i-1} min\{m_{i-1}, m_k\} - \tilde{d}_i min\{m_i, m_l\}$. Here m_i and m_{i-1} are the states of the same process net. m_k and m_l are the inputs to this process net from buffer.
- Type 5 [Asynchronous]. $m_k' = \tilde{d}_i m_i - \tilde{d}_{i'} min\{m_{i'}, m_k\}$. Here m_i and $m_{i'}$ are the states of two different service nets respectively. m_k is the message between these two service nets.
- Type 6 [Synchronous]. $m_k' = \tilde{d}_i min\{m_i, m_l\} - \tilde{d}_{i'} min\{m_{i'}, m_k\}$. Here m_i and $m_{i'}$ are the states of two different service nets respectively. m_k and m_l are the messages between these two service nets, where m_l is usually indicates the request that can be calculated by Type 5 and m_k is the reply.

$\overline{H_2} : V_2 \to J_i$ is a mapping which can be determined based on the reachable marking set of coverability tree through the formula $V_2 = R_1 J_i^T$.

We claim that it is almost impossible to derive private key from public key. The reason is as the following. If we design the k(the maximum number of tokens in the initial markings) and the range of H_1 big enough, the attacker needs to try all possible initial markings to determine $M_0^+(p_i)$, and then further to determine $CT(\sum(M_0'))$ and all the J_i, and finally to determine the J_i in the private key. The whole process is multiple exponentially hard.

B. Encryption and Decryption

1) Steps of Encryption: There are five steps in the encryption.

(1) For a plaintext P, the sender generates cipertext C_1' using symmetric cryptosystems such as AES and 3DES.

(2) Randomly select the initial value $M_0^+(p_i)$ of $p_i, i = 1, 2, \ldots, I$ from the range $n_1 \le M_0(p_i) \le n_2$, $n_1, n_2 \in N^+$. Following the method to calculate the average number of tokens of SPN in the steady state and using some software such as SPNP or GreatSPN, we can compute $U = (u_1, u_2, \ldots, u_n)$.

(3) Compute $M_0'(p_i)$, and then get the coverability tree $CT(\sum(M_0'))$ of $SPN \sum(M_0')$. Randomly select a row vector J_i which corresponds to the node i in $CT(\sum(M_0'))$. Based on the function $V_2 = H_2(J_i) = R_1 J_i^T$ in the public key, we compute V_2 and thus obtain the value of $V = V_2 + V_3$, which is a number with integral part and decimal part.

(4) The cipertext $C_1 = (C_1', V)$ is obtained which can be regarded as the result of the first stage encryption.

(5) Multiple encryption: Regarding C_1 as the plaintext, randomly choose a row vector different from the first time vector from the coverability tree $CT(\sum(M_0'))$. Repeating the steps (1)-(4), eventually after r stages, we will get the cipertext $C_r = (C_r', V_r)$.

2) Steps of Decryption: There are four steps in the decryption.

(1) After receiving the cipertext $C_r = (C_r', V_r)$, based on the initial marking range $n_1 \le M_0(p_i) \le n_2$, $n_1, n_2 \in N^+$ and the semantics of CPN, we can build an ordinary differential equation group. Applying Runge-Kutta algorithm, we calculate many values of $V_3 = H_3(U) = 10^{-p} R_2 U^T$ at different initial marking values. Then we find the most closest V_3 to the decimal fraction from the sender. The initial markings $M_0(p_i)$ for this most closest V_3 are the initial markings selected by the sender.

(2) Based on the value of $M_0^+(p_i)$, and

$$M_0'(p_i) = \begin{cases} H_1(M_0^+(p_i)), & M_0(p_i) \ne 0; \\ 0, & M_0(p_i) = 0. \end{cases}$$

in the public key, we can determine the coverability tree $CT(\sum(M_0'))$ of $SPN \sum(M_0')$ from the sender.

(3) Building the mapping $\overline{H}_2 : V_2 \to J_i$, where $V_2 = H_2(J_i) = R_1 J_i^T$, and J_i is the row vector of $CT(\sum(M_0'))$. Find the V_2 that matches the integral part of V_r, thus from mapping $\overline{H}_2$, we can determine the row vector J_i selected by sender.

(4) Using J_i for the decryption to C_r', we will get the $r-1$ stage cipertext $C_{r-1} = (C_{r-1}', V_{r-1})$. Repeat the above steps until we get the plaintext P.

C. Security Measure

1) The Decryption Complexities For The Attacker and For The Receiver: In order to decipher the text, the attacker must know the initial marking values selected by the sender. Normally people will use the method in section 2.2 to compute average number of tokens for SPN. The time complexity to compute one time average number of tokens is $O(na^{P(k,n)})$, $a > 1$, here k is the maximum number of tokens in the initial markings, $P(k,n)$ is a polynomial of k and n. Since each of those places $p_i, i = 1, 2, \ldots, I$ that can be assigned the initial markings from the range $n_1 \le M_0(p_i) \le n_2$ has $n_2 - n_1 + 1$ possibilities to take the initial values, thus the computing complexity to get all possible v_3 is $(n_2 - n_1 + 1)^I O(na^{P(k,n)})$, $a > 1$. Since the number of states will increase exponentially as the number of places and the number of initial markings of places increase, we assume time to calculate the states is $O(b^{Q(k,n)})$, where $Q(k,n)$ is a polynomial of k and n, and $b > 1$. There are $(n_2 - n_1 + 1)^I$ possibilities for the sender to select the initial markings. Since every selected initial marking $M_0^+(p_i)$ corresponds a coverable tree $CT(\sum(M_0'))$, thus there are also $(n_2 - n_1 + 1)^I$ possibilities to select trees. Hence, for one time decryption, the complexity is $(n_2 - n_1 + 1)^I \times O(b^{Q(k,n)}) \times (n_2 - n_1 + 1)^I \times O(na^{P(k,n)}) = (n_2 - n_1 + 1)^{2I} \times O(na^{P(k,n)} b^{Q(k,n)})$, $a > 1, b > 1$.

However, the receiver only needs to solve a group of ordinary differential equation group and the complexity is $O(n^5)$. When the initial markings are changed, the initial values to the ordinary differential equation group will be changed, but the complexity to solve the equation group still stays the same. Thus to determine the initial marking of places selected by sender, the receiver at most needs to solve the equation group $(n_2 - n_1 + 1)^I$ times, so the complexity for the decryption is $(n_2 - n_1 + 1)^I O(n^5)$.

2) The Complexity for Multiple Encryption: In order to increase the decryption difficulty, we may adopt multi-stage encryption. The corresponding complexity is as the following.

(1) Since in every stage the selected row vectors J_i are different, then the complexity to decipher r-stage encryption plain text is $(n_2 - n_1 + 1)^{2rI}$ $O(n^r a^{rP(k,n)} b^{rQ(k,n)})$, $a > 1, b > 1$.

(2) We may increase the process nets in the Petri net, i.e. to increase the index I in the expression $(n_2 - n_1 + 1)^I O(na^{P(k,n)} b^{Q(k,n)})$, then as the number of process nets increase, the complexity will increase exponentially.

(3) We may increase the the maximum number k of tokens in the initial marking, accordingly, the state space will expand quickly and attackers can not compress this expanding state space caused by the increasing number of tokens.

In summary, the computing time for an attacker to compute the average number of tokens is exponential, thus it is hard to get the coverability tree selected by sender. On the other hand, in our public key, based on the knapsack problem (NP-complete problem [6]), we have designed a Hash function $V_2 = H_2(J_i) = R_1 J_i^T$. Thus the attacking to our encryption is also a NP-complete problem.

VI. An Example To Design A Public-Key Cryptosystem

A. Preparing Public Key and Private Key

(1) Choose a bounded Petri net as shown in Fig. 7. Fig. 8(a)(b) can be used for multiple encryption, where (a) is to increase the number of process nets and (b) is to increase the Internal States in the process net. In Fig. 7, $M_0(p_2) = M_0(p_4) = M_0(p_5) = M_0(p_6) = 0$, and the firing rates for the transitions t_1, t_2, t_3 and t_4 are 1. $M_0(p_1)$ and $M_0(p_3)$ are in the range: $1 \leq M_0(p_1) \leq M_0(p_3) \leq 10, M_0(p_1), M_0(p_3) \in N^+$.

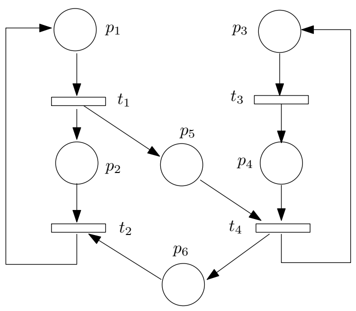

Figure 7. Petri net for encryption.

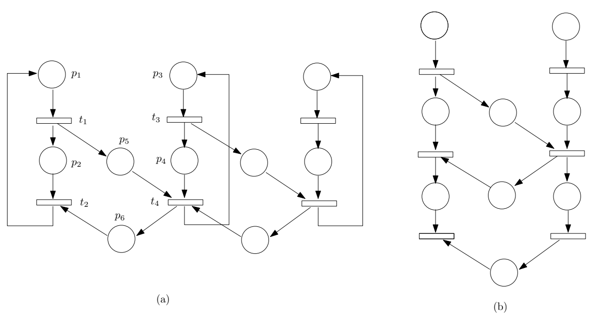

Figure 8. Extended Petri net for multiple encryption.

(2) Design Hash function H.

i) Define new initial markings:

$$M_0'(p_i) = V_1 = \begin{cases} H_1(M_0^+(p_i)), & M_0(p_i) \neq 0; \\ 0, & M_0(p_i) = 0. \end{cases}$$
$$= \begin{cases} 2(M_0(p_i)), & M_0(p_i) \neq 0; \\ 0, & M_0(p_i) = 0. \end{cases}$$

Note that here we design $H_1(M_0^+(p_i)) = 2(M_0(p_i))$ only for the convenience, and H_1 in the real time is designed based on the ability to decipher the text, memory and the calculation speed.

ii) Randomly select $R_1 = (1, 5, 13, 17, 7, 23)$ for $V_2 = H_2(J_i) = R_1 J_i^T$.

iii) Randomly select $R_2 = (1, 2, 3, 4, 5, 6)$. Based on the formula $V_3 = H_3(U) = 10^{-p} R_2 U^T$, we determine p by calculating the maximum value of $R_2 U^T$. Let $M_0(p_1) = M_0(p_3) = 10$, $M_0(p_2) = M_0(p_4) = M_0(p_5) = M_0(p_6) = 0$, we build the following set of equations:

$$\begin{cases} m_1' = min\{m_2, m_6\} - m_1, \\ m_2' = m_1 - min\{m_2, m_6\}, \\ m_3' = min\{m_4, m_5\} - m_3, \\ m_4' = m_3 - min\{m_4, m_5\}, \\ m_5' = m_1 - min\{m_4, m_5\}, \\ m_6' = min\{m_4, m_5\} - min\{m_2, m_6\}, \end{cases}$$

with the initial values: $m_1(0) = m_3(0) = 10$, $m_2(0) = m_4(0) = m_5(0) = m_6(0) = 0$. Using Matlab, we get $U = (3.3333, 6.6667, 3.3325, 6.6675, 3.3342, 3.3325)$. Thus, $R_2 U^T = 90.0002$. Since $p = min\{q \in N | 10^q \geq 90.0002\} = 2$, we get $V_3 = H_3(U) = 10^{-2} \times (1, 2, 3, 4, 5, 6) U^T$.

Thus the Hash function H would be

$$\begin{aligned} V &= H(M_0^+(p_i), J_i, U) = V_2 + V_3 \\ &= R_1 J_i^T + 10^{-p} R_2 U^T \\ &= (1, 5, 13, 17, 7, 23) J_i^T + 10^{-2} \times (1, 2, 3, 4, 5, 6) U^T. \end{aligned}$$

(3) Private key. First to determine E, which is a set of ordinary differential equations as the following:

$$\begin{cases} m_1' = min\{m_2, m_6\} - m_1, \\ m_2' = m_1 - min\{m_2, m_6\}, \\ m_3' = min\{m_4, m_5\} - m_3, \\ m_4' = m_3 - min\{m_4, m_5\}, \\ m_5' = m_1 - min\{m_4, m_5\}, \\ m_6' = min\{m_4, m_5\} - min\{m_2, m_6\}, \end{cases}$$

with the initial values: $1 \leq m_1(0), m_3(0) \leq 10$, $m_2(0) = m_4(0) = m_5(0) = m_6(0) = 0$. Next to determine $\bar{H}_2$, which is to determine J_i.

We need to solve the above differential equations covering all the cases that m_1 and m_3 take values from 1 to 10. Without loss of generality, we only calculate the solutions of the differential equation group for the range $1 \leq M_0(p_1), M_0(p_3) \leq 2$. If $M_0(p_1) = M_0(p_3) = 1$, $V_3 \approx 0.09$; If $M_0(p_1) = 1, M_0(p_3) = 2$, $V_3 \approx 0.13$; If $M_0(p_1) = 2, M_0(p_3) = 1$, $V_3 \approx 0.16$; If $M_0(p_1) = 2, M_0(p_3) = 2$, $V_3 \approx 0.18$. Since the decimal fraction of V_3 from sender is 0.092, by comparing with all the cases of V_3 here, we choose the most closest one $V_3 \approx 0.09$. From this value, we imply that the sender uses $M_0(p_1) = M_0(p_3) = 1$ as the initial markings.

Since

$$M_0'(s_i) = V_1 = \begin{cases} 2(M_0^+(s_i)), & M_0(s_i) \neq 0; \\ 0, & M_0(s_i) = 0. \end{cases}$$

we determine the coverability tree in the situation $M_0(s_1) = M_0(s_3) = 2$, which is shown in Fig. 9.

Since there are 18 reachable markings in the tree, we will have 18 row vectors. Based on the formula $V_2 =$

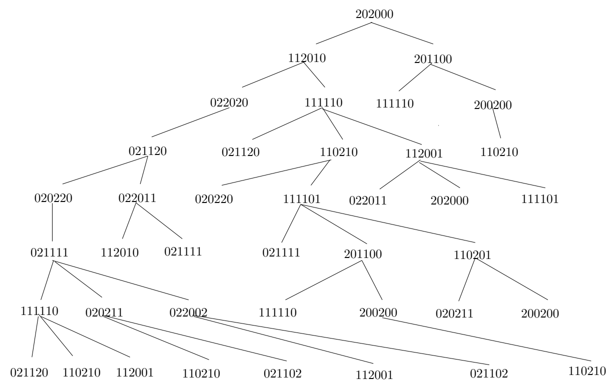

Figure 9. Part of the coverability tree.

$H_2(J_i) = R_1 J_i^T = (1, 5, 13, 17, 7, 23) J_i^T$, we define the mapping $\overline{H}_2 : V_2 \to J_i$ by Table I.

TABLE I.
INVERSE FUNCTION $\overline{H}_2 : V_2 \to J_i$

V_2	J_1	J_2	J_3	J_4	J_5	J_6
28	2	0	2	0	0	0
32	2	0	1	1	0	0
36	2	0	0	2	0	0
39	1	1	2	0	1	0
43	1	1	1	1	1	0
47	1	1	0	2	1	0
50	0	2	2	0	2	0
54	0	2	1	1	2	0
55	1	1	2	0	0	1
58	0	2	0	2	2	0
59	1	1	1	1	0	1
63	1	1	0	2	0	1
66	0	2	2	0	1	1
70	0	2	1	1	1	1
74	0	2	0	2	1	1
82	0	2	2	0	0	2
86	0	2	1	1	0	2
90	0	2	0	2	0	2

B. Encryption

The encryption contains the following steps:

(1) With AES, the sender transforms plaintext P to cipertext C_1'.

(2) Randomly select $M_0(p_1) = M_0(p_3) = 1$. By using the method in Section 2.2 and software SPNP or GreatSPN to calculate the average number of tokens of SPN, we get $U = (u_1, u_2, \ldots, u_m) = (0.3077, 0.6922, 0.3076, 0.6923, 0.3846, 0.3076)$. Thus $V_3 = H_3(U) = 10^{-2} \times (1, 2, 3, 4, 5, 6) U^T \approx 0.092$.

(3) Since

$$M_0'(p_i) = V_1 = \begin{cases} 2(M_0^+(p_i)), & M_0(p_i) \neq 0; \\ 0, & M_0(p_i) = 0. \end{cases}$$
$$= \begin{cases} 2, & M_0(p_i) \neq 0; \\ 0, & M_0(p_i) = 0. \end{cases}$$

we build the the coverability tree of $SPN \sum(M_0^+)$ as shown in Fig. 9. In the figure we only given part of the tree. In the tree, the sender randomly choose row vector $(0, 2, 2, 0, 1, 1)$ as the encryption vector J_i. Since $V_2 = H_2(J_i) = R_1 J_i^T = (1, 5, 13, 17, 7, 23)(0, 2, 2, 0, 1, 1)^T = 66$, we get $V = V_2 + V_3 \approx 66 + 0.092 = 66.092$.

(4) Finally, we get the cipertext $(C_1', V) = (C_1', 66.092)$.

Note: For the computing convenience, we only choose $M_0(p_1) = M_0(p_3) = 1$, and only perform one time encryption.

C. Decryption

After getting the cipertext $C_1 = (C_1', 66.092)$, the receiver knows that the integral part of V is 66. By checking the table of function $\overline{H}_2$, we find that the sender has chosen the vector $J_{13} = (0, 2, 2, 0, 1, 1)$ for encryption. Using J_{13} as the decryption key of AES, we will get the plaintext P.

VII. RELATED WORK

RSA [21] is the most extensively used public-key crytosystem, and its security relies on the difficulty of factoring the large integer problem. Its complexities of encryption, decryption and attacking are the same: $O(exp(c(logn)(loglogn))^{1/2})$, c is a constant and n is a large factoring number. This expression is subexponential, not exponential. In our public-key crytosystem, we first use private key cryptosystem such as AES or 3DES to encrypt the plaintext, which belongs to a NP-problem [22]. Then based on the knapsack problem, we design a Hash function: $V = H(M_0^+(p_i), J_i, U) = V_2 + V_3 = R_1 J_i^T + 10^{-p} R_2 U^T$, where the computing of J_i from V, R_1, and R_2 also belongs to a NP-complete problem [6]. Thus the security of our method is higher than that of RSA after one time encryption. If applying r-stage encryption, the security can reach $(n_2 - n_1 + 1)^{2rI} O(n^r a^{rP(k,n)} b^{rQ(k,n)})$.

PGP [5] is a protocol for email text encryption. Its core part is RSA. When encrypting, PGP first compresses plain text, and then encrypts the compressed plaintext with session key, finally encrypts the session key with RSA. Our technique is also based on session key. However, PGP is based on RSA, while ours is based on NP-complete problem, thus our security is higher than PGP. Also because PGP is based on RSA, the encryption is slow. Since our encryption requires computing average number of tokens in places of SPN, and decryption is to solve a group of differential equations, our encryption and decryption are comparably faster.

MEPKC [8] is designed based on elementary T-invariants of the Petri net. Petri nets are used as a key-generator and elementary T-invariants are used as the encryption keys. After r-stage encryption, the security is $(e^m)^r$, which is still an exponential expression. In MEPKC, the sender needs to construct a small Petri net such that the net contains as many as possible elementary T-invariants, where elementary T-invariants are used as the entryption keys. In our technique, we use the reachable markings of coverability tree to generate key, and the coverability tree is comparably easy to get from SPN, so our encryption is faster than MEPKC.

VIII. Conclusion

We have developed a new public-key cryptosystm based on the difficulty to solve average number of tokens in places of SPN for a given range of initial marking values, and used CPN to perform the decryption. The reachable markings in coverability tree of SPN are used as the encryption key, and the plaintext can be encrypted in multiple stages. Comparing with the traditional public-key cryptosystms such as RSA, PGP, our technique has higher security. Moreover, the encryption and the decryption are easier.

Usually both the public key and the private key in the public-key cryptosystem can be used to encrypt plaintext, such as RSA, ECC, etc. However, in our technique, only public key can be used for the encryption, so our technique can not be used for digital signature. To overcome this shortcoming, we may combine our system with those public-key cryptosystems that are qualified for digital signature. Now we use DSA as the example to illustrate the encryption and decryption process. Assume that the sender A(with DSA) sends the plaintext P to the receiver B(with our technique). The public key and private key of A are K_{eA} and K_{dA}, and public key and private key of B are K_{eB} and K_{dB}, respectively. A first uses its own private key K_{dA} to encrypt the plaintext P, and the result is $S = E(P, K_{dA})$; then A uses the public key K_{eB} of B to encrypt S, and the result is $C = E(S, K_{eB})$; finally A sends C to B. After B receives C, B uses its own private key K_{dB} to decrypt C, and obtain $S = D(C, K_{dB})$; then uses the public key K_{eA} of A to decrypt S, and obtain $P = D(S, K_{eA})$. Thus the secret and the reality are promised. In this way, our technique can also be used for digital signature.

In order to increase the attack difficulty, we may design more complicated hash function H in the public key. We may also combine our cryptosystem with other cryptosystems to increase the security. One may notice that while our technique increase the decryption difficulty for attackers, it also increase the computing work for the receivers. Two issues will be solved in the future: 1) How to store all reachable markings in the coverability tree of SPN; 2) How to estimate security if the key itself gets brute force attack.

References

[1] U. M. Ascher, L. R. Petzold, *Computer methods for ordinary differential equations and differential-algebraic equations*, Society for Industrial & Applied Mathematis, Philadelphia, PA, USA, 1998.

[2] R. David and H. Alla, Continuous Petri nets, *Proceedings of 8^{th} European Workshop on Application and Theory of Petri nets*, Zaragoza, Spain, pp.275-294, 1987.

[3] R. David and H. Alla, Autonomous and timed continuous Petri nets, *Proceedings of 11^{th} Intl Conference on Application and Theory of Petri nets*, Paris, France, pp.367-381, 1990.

[4] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions in Information Theroey*, vol.IT-31, no.4, pp.469-472, 1985.

[5] S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, 1994.

[6] M. R. Garey, D. S. Johson, *Computers and Intractability (A Guide to the Theory of NP-Completeness)*, W. H. Freeman and Company, New York,1991.

[7] Q. W. Ge, T. Okamoto, A Petri net based public-key cryptography: PNPKC, *IEICE. Trans. Fundamentals*, vol.E-84-A(6), pp.1532-1535, 2001.

[8] Q. Ge, C. Shigenaga, and R. Wu, A Petri net based new conception of publickey crytography, *Proceedings of ICFS'02*, pp.37-42, 2002.

[9] E. Hairer, S.P. Nϕrsett, G. Wanner, *Solving Ordinary Differential Equations(I)(II)*, Nonstiff Problems, Second Edition, Springer-Verlag, 1993.

[10] A. W. Harrow, A. Hassidim, S. Lloyd, Quantum algorithm for solving linear systems of equations, *Phys. Rev. Lett.*, vol.103, no. 15, pp.150502-150506, 2009.

[11] W. Henderson, D. Lucic, P. G. Taylor, A net level performance analysis of Stochastic Petri Nets, *J. Austral. Math. Soc. Ser. B*, vol.31, 176-187, 1989.

[12] K. Hiraishi, Performance evaluation of workflows using continuous Petri nets with interval firing speeds, *Petri Nets'08*, *LNCS*, vol.5062, pp.231-250, 2008.

[13] C. Lin, D. C. Marinescu, Stochastic high-level Petri nets and applications, *IEEE Transactions on Computers*, vol.37, no.7, pp.815-825, 1988.

[14] M. K. Molloy, Performance analysis using stochastic Petri nets, *IEEE Transactions on Computers*, vol. C-31, no.9, pp.913-917, 1982.

[15] M. K. Molloy, On the integration of delay and throughput measures in distributed processing models. Ph.D. dissertation, Univ. of California, Los Angeles, 1981.

[16] M. A. Marsan, A. Bobbio, S. Donatelli, Petri nets in performance analysis: An introduction, *Petri Nets'98*, *LNCS*, vol.1491, pp.211-256, 1998.

[17] B. B. Nich, S. E. Tavores, Modelling and analyzing cryptographic protocols using Petri nets, *Advance in Cryptology-LNCS*, vol.718, pp.275-295, 1992.

[18] T. Okamoto and S. Uchiyamaa, Recent topics of public-key cryptography: 1. On the security of elliptic curve cryptosystems, *IPSJ Magazine*, vol.39, no.12, pp.1252-1257, 1998.

[19] T. Okamoto, E. Fujisaki, and S. Uchiyamaa, Recent topics of public-key cryptography: Provably secure and practical public-key encryption, *IPSJ Magazine*, vol.40, no.2, pp.170-177, 1999.

[20] L. Recalde, S. Haddad, M. Silva, Continuous Petri nets: expressive power and decidability issues, *ATVA'07*, *LNCS*, vol.4762, pp.362-377, 2007.

[21] R. L. Rivest, A. Shamir, L. Adleman, A method of obtaining digital signatures and Public-Key cryptosystems, *Comm. of ACM*, vol.21, no.2, pp.120-126, 1978.

[22] A. Salomaa, *Public-Key Cryptograpghy*, Springer-Verlag, Berlin, Heidelberg, 1990.

[23] T. Shitayama, A survey of block ciper AES and a view of the future, *IPSJ Magazine*, vol. 40, no.2, pp.139-145, 1999.

[24] D. R. Stinson, *Cryptography: Theory and Practice*, CRC Press Inc., 1995.

[25] S. A. Teukolsky, W. H. Press, W. T. Vetterling, *Numerical recipes in C++ (2nd edition)*, Cambridge Univ Press, 1993.

[26] S. Tu, S. M. Shatz, and T. Murata, Applying Petri net reduction to support Ada tasking deadlock analysis, *Proceedings of the 11^{th} International Conference on Distributed Computing Systems*, pp.96-103, Paris, France, 1990.