Journal of Computer and System Sciences 79 (2013) 518–529

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Security-aware optimization for ubiquitous computing systems with SEAT graph approach

Meikang Qiu[a], Lei Zhang[b], Zhong Ming[c,*], Zhi Chen[a], Xiao Qin[d], Laurence T. Yang[e]

[a] Dept. of Elec. and Comp. Engr., University of Kentucky, Lexington, KY 40506, USA
[b] Dept. of Comp. Sci., University of Texas at Dallas, Richardson, TX 75083, USA
[c] College of Comp. Sci. and Software Engr., Shenzhen University, Shenzhen, GD 518060, China
[d] Dept. of Comp. Sci. and Software Engr., Auburn University, Auburn, AL 36849, USA
[e] Dept. of Computer Science, St. Francis Xavier University, Antigonish, NS, B2G 2W5, Canada

ARTICLE INFO

Article history:
Received 20 March 2012
Received in revised form 5 October 2012
Accepted 8 November 2012
Available online 12 December 2012

Keywords:
Security
Real-time
Ubiquitous system
Cryptographic algorithm
Schedule

ABSTRACT

For ubiquitous computing systems, security has become a new metric that designers should consider throughout the design process, along with other metrics such as performance and energy consumption. A combination of selected cryptographic algorithms for required security services forms a security strategy for the application. In this paper, we propose methods to generate security strategies to achieve the maximal overall security strength while meeting the real-time constraint. In order to express security requirements of an application, we propose a novel graph model called *Security-Aware Task* (SEAT) graph model to represent real-time constraints and precedence relationships among tasks. Based on the SEAT graph approach, we propose an optimal algorithm, *Integer Linear Programming Security Optimization* (ILP-SOP). For the special structures such as simple path graph and tree, we propose two dynamic programming based algorithms (DPSOP-path/tree) to generate the optimal security strategy. Experiment results demonstrate the correctness and efficiency of our proposed method. The experimental results show that, by using our proposed techniques, the security strength can be improved by 44.3% on average.

1. Introduction

Ubiquitous computing systems have being widely deployed in various commercial and military real-time applications, such as the aircraft control, wireless sensor networks, and medical electronics etc. Security has become a new metric that designer should consider throughout the design process, along with other metrics such as performance and energy consumption. Compared to desktops and servers, applying security to ubiquitous systems, especially embedded systems, must take resources and timing constraints into account. For critical applications, it is necessary to protect the system against various security threats and attacks. Integrating cryptographic algorithms into ubiquitous applications is one of the most important methods for improving security strength of systems [1].

However, computational overhead incurred by cryptographic algorithms may easily overwhelm the processing capabilities of many ubiquitous systems [2]. It is well known that the correctness of a real-time system depends on not only the computational result, but also the time instant when the result becomes available. Therefore, security requirements have not been adequately considered in most existing real-time ubiquitous systems. Recently, security issues of real-time ubiquitous systems has received more attention [3–7].

* Corresponding author.
E-mail addresses: mqiu@engr.uky.edu (M. Qiu), doczhanglei@hotmail.com (L. Zhang), mingz@szu.edu.cn (Z. Ming), zch229@uky.edu (Z. Chen), xqin@auburn.edu (X. Qin), ltyang@gmail.com (L.T. Yang).

0022-0000/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jcss.2012.11.002

A real-time ubiquitous application generally consists of a set of distinct, often dependent tasks [8,9]. Hence, in the system design process, an application can be always specified as a *task graph*, in which nodes represent tasks and directed edges represent precedent relationships among tasks. However, conventional task graph models lack the capability to represent security requirements of each task. On one hand, a task may require more than one type of security service. For example, the data generated by one task need to be encrypted and have authentication information inserted. On the other hand, in the same application, different tasks have different security strength requirements for the same security service.

For the purpose of security optimization, we propose a novel model, named *Security-Aware Task* (SEAT) graph, to represent an application with security requirements. Ubiquitous systems often perform periodic applications to run control loops with real-time deadlines. The period of an application is equal to the completion time of the last task and cannot exceed the given deadline.

Security requirements of a task can be divide into several types of security services such as data integrity, encryption, and authentication etc. Each security service can be implemented by several cryptographic algorithms with different computational overheads and security strengths. For example, it is well known that IDEA is a stronger block cipher compared to RC5 [10]. However, in computation, IDEA is more costly than RC5. Therefore, application scopes of IDEA and RC5 are different. The former is always used to satisfy a high security requirement while the later is more suitable for systems with strict timing limits. For a specific algorithm such as AES [11], different parameters, such as the block size, the key size, and the number of rounds, lead to different security strengths and overheads. In order to optimize the security strength of a real-time ubiquitous application, constructing a quantitative model to measure security strengths and computational overheads of different security algorithms is a fundamental problem.

The computational overhead model proposed by Xie [12] is adopted in this paper. The security strength is the ability of a cryptographic algorithm to withhold cryptanalytic attacks. It is usually quantized as the *security level* [12,13]. We suggest using the probability of being risk free to express the security strength. For a given real-time ubiquitous application represented by a SEAT graph model, security optimization is to choose appropriate cryptographic algorithms to implement security requirements of each task while guaranteeing the schedulability of the system. The combination of all selected security algorithms forms a security strategy. Different security strategies have different security strength and overhead. It is clear that the security optimization is a combinational optimization problem, which is not only to satisfy the basic security requirement of each task, but also to improve the overall security strength.

In this paper, we propose three methods to generate the optimal security strategy for a real-time ubiquitous application. The key idea is to judiciously distribute the slack time for each task to maximize the security strength of their required security services while guaranteeing system schedulability.

First, we use *Integer Linear Program* (ILP) method to solve the security strategy generation problem. We call this method as ILP-SOP (*ILP Security Optimization*). With ILP-SOP method, we can achieve an optimal security strategy for any application. But the computation time is an issue. Hence, for a simple path or tree-structured SEAT graph, we propose two *dynamic programming* (DP) based algorithms (DP-path and DP-tree) to generate the optimal security strategy. The main advantage of these two algorithms is that they can achieve the optimal security strategy in pseudo-polynomial time, although the security strategy generation problem is NP-hard. Compared to the modified version of a heuristic algorithm MTPADPS [14], our experimental results show that the ILP-SOP method improves the security strength by 6.25% on average. Moreover, under all deadline constraints our proposed methods can achieve the optimal solution efficiently. The main contributions of our work compared to previous research are:

- We propose a novel graph model SEAT that can express security requirements of each task of the application. The real-time constraint and task precedence relationship can also be represented by this model. Any security service requirement can be integrated into the SEAT model.
- We suggest using the security probability to measure security strength of a cryptographic algorithm. Based on this quantitative security model, the optimal security strategy can be achieved by our proposed integer linear programming model while guaranteeing the system schedulability. We provide the complete source code of this ILP model that can be used directly.
- When a ubiquitous real-time application can be represented by a tree or path structured SEAT graph, two dynamic programming based algorithm can generate the optimal security strategy in polynomial time, although strictly speaking they are pseudo-polynomial time algorithms.

The rest of this paper is organized as follows. Related works are described in Section 2. Section 3 introduces basic concepts and related techniques. Section 4 describes the problem formally. The ILP-SOP method and two efficient optimal algorithms are proposed in Section 5 and Section 6, respectively. Experimental results and concluding remarks are provided in Section 7 and Section 8.

2. Related work

Much research has been done on guaranteeing schedulability of ubiquitous real-time applications. However, most existing real-time scheduling algorithms, such as *Rate Monotonic* (RM) algorithm [15], *Earliest Deadline First* (EDF) [16], and Spring scheduling algorithm [17], cannot be applied to security-aware real-time applications directly, because they cannot make full use of the slack time to incorporate security overhead into the process of scheduling tasks. Moreover, conventional scheduling algorithms lack the capability to generate the optimal security strategy.

Recently, integrating cryptographic algorithms into ubiquitous real-time applications has received more attention. In [3], Xie et al. proposed a *Security-Aware Scheduling Algorithm* (SASES) to improve the security of embedded system applications composed of independent periodic tasks. The SASES algorithm is a heuristic algorithm based on the dynamic scheduling algorithm EDF. It cannot guarantee predictability and optimality of the security strategy. In [14], Xie et al. proposed a task allocation scheme named TPADPS to maximize the security quality and schedulability for real-time applications with task dependence. In [4], Lin and Yang proposed an ILP model to solve the static schedulability driven security optimization problem. Their method can only be applied to applications with independent periodic tasks.

In order to achieve the maximal system security strength and to guarantee the real-time schedulability, two critical issues, i.e., quantitative security strength and computational overhead models, need to be solved. Researchers have worked on quantitative measurements of the security strength of various cryptographic algorithms. Irvine and Levin [13] proposed a security taxonomy and overhead framework. Xie et al. also proposed overhead models [12] that can approximately measure security overheads experienced by tasks with security requirements. They mainly considered the security overhead model for three types of basic and widely deployed security services: confidentiality, integrity, and authentication. The security strength of various algorithms is quantized as quantitative security levels.

Similar to the level-based security model, Yang et al. [4] proposed a group based security model. They assume that the system has different services with different quality. In [14] and [4], the overall system security strength is a weighted sum of security strength of selected security algorithms. Note that how to accurately express the overall security strength is still a challenge and open issue.

In this paper, we propose to use the risk-free probability to measure the security strength of each security algorithm. Thereby, the overall strength of a security strategy is the product of these probability values.

3. Basic models and definitions

In this section, first we introduce the basic models which will be used in the later sections. Then, we formally define the security optimization problem in this paper.

3.1. Security-aware task graph model

A large number of ubiquitous applications are implemented in a periodic manner, in which a set of tasks are executed periodically. Previous literature uses independent task models to study the real-time scheduling problem [3,4]. However, the precedence relationship among tasks must be taken into account in many real applications. For example, in a video monitor system, video signal capture, coding, compression, and transition are set of dependent tasks and executed in an identical period.

In this paper, we propose the SEAT graph model to specify a real-time ubiquitous application. The precedence relationship among tasks can be represented by directed edges of a SEAT graph. Security requirements and timing constraints can be regarded as attributes of task nodes. The formal definition of the SEAT graph model is stated as follows.

Definition 3.1 *(Security-aware task graph).* A SEAT $G = \langle V, E, L, a(v)\rangle$ is a directed graph, where V is a set of tasks, $E \subseteq V \times V$ is a set of edges that define the precedence relationships among task nodes in V, L represents the deadline of a single iteration of the SEAT, $a(v) = \{t(v), d(v), s_v\}$ is the attribute vector of task v, where $t(v)$ denotes the execution time of task v, $d(v)$ denotes the amount of data (measured in KB) that needs security protection, $s_v = \{s_v^1, s_v^2, \ldots, s_v^n\}$ is the security vector of task v, the value of s_v^i represents the security requirement of the ith security service for task v.

We assume that all tasks in a SEAT have the same rate, i.e. the period of each task is identical. Therefore, the real-time requirement can be guaranteed via deadline L. An edge $e(v_i \to v_j)$ in E denotes that task v_j cannot be executed until task v_i is completed. Note that in a SEAT the value of s_v^i of the security vector s_v is only the lower bound security requirement of the ith security service for task v. In the next section, we present a general model to formulate the security level of the application.

3.2. Security model

The security of a ubiquitous system is an ability of a system to withhold various attacks. The overall system security strategy is composed of diverse security services such as data integrity, confidentiality, and authentication. And each security service can be implemented by different cryptographic algorithms which lead to different security strength and computational overhead.

For example, data confidentiality can be implemented by RC4 and AES cryptographic algorithms. RC4 is a very fast algorithm since it uses only 7 CPU clock cycles per byte of output on a Pentium CPU architecture [18]. The memory space cost of RC4 is also very low, say only 256 bytes of RAM. Hence, it was one of the best encryption schemes of the past decade. However, Fluhrer et al. have discovered several vulnerabilities in the RC4 algorithm [19] that make it unsafe for

Table 1
Performance of optimized assembly language implementations of MD4-like hash functions on a 90 MHz Pentium using a 32-bit flat memory model (i.e., running in native protected mode).

Algorithm	MD4	MD5	RIPEMD	RIPEMD-128	SHA-1	RIPEMD-160
Clock cycles	241	337	480	592	837	1013
Mbit/s	191.2	136.7	96.0	77.8	55.1	45.5
Relative performance	1.00	0.72	0.50	0.41	0.29	0.24

any key size. On the contrary, AES encryption is considered to be very secure. It has been rigorously reviewed for security loopholes for more than two years before it was standardized by NIST in 2001. One downside of AES is that it has a larger overhead than RC4. To illustrate, the entire RC4 is often coded in 50 lines of code, whereas AES is about 350 lines.

The security strength of a specific cryptographic algorithm is usually determined by the key size and the number of operation rounds. The key size directly reflects the strength of the cryptographic algorithm against key search attacks. For example, AES allows different options of the key size, e.g., 128 bits, 192 bits, and 256 bits. The number of guesses it would take to crack AES protected data is 3.4×10^{38} for the 128-bit key, 6.2×10^{55} for the 192-bit key, and 1.1×10^{77} for the 256-bit key. As more operation rounds are used, the cipher tends to be more secure since it leaves no trails of the original data. The number of operation rounds is often used to determine the strength of a cipher against cryptanalysis attacks [20].

In order to optimize the security strength of an application, it is fundamental to quantitatively measure the strength and computational overhead of different cryptographic algorithm and different combinations of parameters of specific algorithms, such as key size and operational rounds. Basically, the security strength of a cryptographic algorithms is proportional to its computational overhead. In this paper, the algorithm used to implement a security service is defined as a *security instance*.

Definition 3.2 *(Security instance).* A security instance S_i^j represents the ith implementation method for the jth security service. The security strength of S_i^j is represented by $SS(S_i^j)$, which is a probability value between 0 to 1. The computational overhead of S_i^j is denoted by $CO(S_i^j)$.

The security strength and the computational overhead are two critical attributes of a security instance S_i^j. Given a security service, the strongest security instance is assigned with the value 1. Note that quantitatively modeling the security strength is still an open issue. The computational overhead of a security instance can be obtained by program profiling. For example, the quantitative estimation of six security instances for data integrity security service is shown in Table 1 [21]. We take the performance value in terms of Mbit/s as the computational overhead.

For a given SEAT $G = \langle V, E, L, a(v)\rangle$, if the ith security instance is selected to implement the jth security service for task k, the computational overhead for this security service is

$$CO(i,j,k) = CO\big(S_i^j\big) * d(k) \tag{1}$$

where $d(k)$ is the overhead for computing S_i^j on task k. Consequently, the security overhead of task k can be stated as

$$SO(k) = \sum_{j=1}^{N} CO(i,j,k) \tag{2}$$

where N is the number of security services.

Overall security strength and overhead. Given an application represented by SEAT G, one security service of a task can only be implemented by one security instance, and the same security service for different tasks can be implemented by different security instances. We use a binary variable $x_{i,k}^j$ to denote whether the ith security instance is selected to implement the jth security service of task k.

$$x_{i,k}^j = \begin{cases} 1, & \text{if the } i\text{th security instance is selected,} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Therefore, the overall security strength of the application $Sec(G)$ and the corresponding computational overhead $Overhead(G)$ are formulated as Eqs. (4) and (5), respectively.

$$Sec(G) = \prod_{k=1}^{|V|} \prod_{j=1}^{NS(k)} \prod_{i=1}^{NI(j)} \big[SS\big(S_i^j\big)\big]^{x_{i,k}^j}, \tag{4}$$

$$Overhead(G) = \sum_{k=1}^{|V|} \sum_{j=1}^{NS(k)} \sum_{i=1}^{NI(j)} CO(i,j,k) * x_{i,k}^j \tag{5}$$

where $NS(k)$ is the number of required security services of task k, $NI(j)$ is the number of security instances of security service j.

4. Problem statement

We know that one security service can be implemented by several security instances with different security strengths and computational overheads. Given a real-time application represented by SEAT $G = \langle V, E, L, a(v)\rangle$ and quantitative security models of security services, the combination of security instances for each security service of each task forms the system security strategy. The primary goal of this study is to find the optimal security strategy to maximize the system security while guaranteeing the SEAT is schedulable on the homogenous multiprocessor ubiquitous systems.

From the definition of system security (see Eq. (4)), we know the security optimization is a combinatorial optimization problem. Solving this problem includes two processes as follows:

1. **Security strategy determination.** Choosing security instances for each security service of each task. Two constraints must be satisfied: (1) The security strength of each selected security instance must be no less than the minimal security requirement given in SEAT. (2) For each task, each security service can only be implemented by one security instance.
2. **Schedulability test.** When a security strategy is determined, the real execution time of task k is equal to $t(k)$ plus the security overhead $SO(k)$. Our target architecture is a homogenous multiprocessor ubiquitous system $P = \{P_1, P_2, \ldots, P_M\}$, where M is the number of PEs. We schedule the tasks including security overheads to evaluate whether this security strategy is practicable or not. A legal schedule should obey task dependence and deadline constraints defined in SEAT.

Since the time slack for a security instance execution is a certain value, increasing the security strength of one security service of one task would degrade the security of other tasks, and even the system overall security. So the security instance determination for different tasks are interdependent by each other. It is hard to find an optimal security strategy for the application in a phase ordered manner. However, search-based techniques such as linear programming allow to model these interactive phases.

5. Our ILP security optimization method

In this section, we formulate our ILP method ILP-SOP (ILP Security Optimization) to solve the security optimization problem described in Section 4. We use binary decision variables with three indices: $X = \{x_{i,k}^j;\ i = 1, 2, \ldots, NI(j);\ j = 1, 2, \ldots, NS(j);\ k = 1, 2, \ldots, |V|\}$ to denote the security instance options. For detailed definition of $x_{i,k}^j$ refer to Eq. (3).

Objective function. According to the definition of overall system security strength $sec(G)$ (see Eq. (4)), the objective function of our ILP model is stated as

$$\max \prod_{k=1}^{|V|} \prod_{j=1}^{NS(k)} \prod_{i=1}^{NI(j)} \left[SS\left(S_i^j\right)\right]^{x_{i,k}^j}. \tag{6}$$

This objective function is nonlinear. However, an equivalent linear objective function can be obtained via a logarithmic operation. Note that the *equivalent objective function* is the one has the same optimal solution, although the function value may not be equal.

$$\max \sum_{k=1}^{|V|} \sum_{j=1}^{NS(k)} \sum_{i=1}^{NI(j)} x_{i,k}^j * \ln\left(SS\left(S_i^j\right)\right). \tag{7}$$

In the following, we will formulate the two subproblems defined in Section 4 with linear constraints.

Determining security strategy. Given a SEAT graph, the minimal security requirement of task k is given by the vector $s_k = \{s_k^1, s_k^2, \ldots, s_k^n\}$. Constraints in Eq. (8) are used to guarantee the minimal security requirements of each task.

$$\sum_{i=1}^{NI(j)} SS(S_i, j) * x_{i,k}^j \geqslant sr_k^1, \quad k = 1, 2, \ldots, |V|,\ j = 1, \ldots, NS(k). \tag{8}$$

It is clear that for each task one security service can be implemented by only one security instance.

$$\sum_{i=1}^{NI(j)} x_{i,k}^j \leqslant 1, \quad k = 1, 2, \ldots, |V|,\ j = 1, \ldots, NS(k). \tag{9}$$

Schedulability testing. After adding the security services to tasks, the computational time of each task is increased. To express the real computational time of each task we define a function $t'(k)$ in terms of decision variables X as follows.

$$t'(k) = \sum_{i=1,NI(j)} \sum_{j=1,NS(k)} t(k) + x_{i,k} * CO(i, j, k), \quad k = 1, \ldots, |V|. \tag{10}$$

To schedule the given SEAT on to the multiprocessor system $P = \{P_1, P_2, \dots, P_M\}$, we define binary decision variables with two indices: $Y = \{Y_{i,j};\ i = 1, 2, \dots, |v|;\ j = 1, 2, \dots, M\}$. A decision variable $y_{i,j}$ is 1 only when task i is mapped onto processor j. And a task can be mapped to exactly one processor.

$$\sum_{j=1,M} y_{i,j} = 1, \quad k = 1, 2, \dots, |V|. \tag{11}$$

We let nonnegative integer decision variables: $T = \{t_i;\ i = 1, 2, \dots, L\}$ to denote the start time of tasks. For a given SEAT G, $e(i, j) \in G$ indicates that task j cannot be executed until task i completed. The precedence relationship between tasks can be formulated as

$$t_j \geqslant t_i + t'(i), \quad \forall e(i, j) \in G. \tag{12}$$

The task precedence constraint effectively prevents two dependent tasks from competing for the same processor resource. We should also ensure that there is no execution time overlap between any two independent tasks mapped to the same processor. We use a function $p(i)$ to denote the processor on which the task i is mapped.

$$p(i) = \sum_{j=1}^{M} j * y_{i,j}, \quad \forall i = 1, 2, \dots, |V|. \tag{13}$$

We let binary variables: $W = \{w_{i,j};\ \forall i, j \in 1, 2, \dots, |V|\}$ as in Eq. (14) to denote whether two tasks are on the same processor.

$$w_{i,j} = \begin{cases} 1, & \text{if } p(i) \neq p(j), \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

Since Eq. (14) is a nonlinear, it needs to be transformed into equivalent linear conditions. Recall that $y_{i,k} = 1$ if task i is mapped on processor k otherwise $y_{i,k} = 0$. Therefore, the nonlinear condition can be replaced by the following two linear conditions.

$$2 - y_{i,k} - y_{j,k} \geqslant w_{i,j}, \quad \forall k = 1, 2, \dots, M, \tag{15}$$

$$w_{i,j} \geqslant y_{i,k} + y_{j,l} - 1, \quad \forall k, l = 1, 2, \dots, M,\ k \neq l. \tag{16}$$

When two independent tasks are mapped onto the same processor, we need to consider the execution sequence between them. We define another set of binary decision variables $Q = \{q_{i,j};\ \forall i, j \in 1, 2, \dots, |v|\}$ to express the order between any two tasks. Let $q_{i,j} = 0$ if task i and task j are mapped on the same processor and task i is executed after task j. Similarly, let $q_{j,i} = 0$ if tasks j and i are mapped to the same processor and task j is executed after task i. Then, we ensure disjoint lifetimes for the two tasks on the same processor via the following constraints.

$$q_{i,j} + q_{j,i} - w_{i,j} = 1, \tag{17}$$

$$t_i > t_j + t'(j) - \infty * q_{i,j}, \quad \forall i, j \in 1, 2, \dots, |V|,\ i \neq j, \tag{18}$$

$$t_j > t_i + t'(i) - \infty * q_{j,i}, \quad \forall i, j \in 1, 2, \dots, |V|,\ i \neq j. \tag{19}$$

6. Highly efficient algorithms based on dynamic programming

In this section, we propose two dynamic programming based algorithms to solve the security optimization problem for SEAT graphs with special structures. The first one is called the DPSOP-path algorithm, which is applied to SEAT graphs with a simple path structure. The second one is called the DPSOP-tree algorithm that is applied to tree-structured SEAT graphs. Even though strictly speaking they are pseudo-polynomial time algorithms, in practice, these two algorithms are more efficient than the ILP-based method.

6.1. DPSOP-path algorithm

First, we use a figure shown in Fig. 1 to illustrate the basic idea of this algorithm. Then, we formally describe the DPSOP-path algorithm and use an intuitive example to demonstrate its working process. In the end, we analyze the complexity of the DPSOP-path algorithm.

In Fig. 1 (a), the SEAT graph includes three sequential tasks. Fig. 1 (b) is a *phase graph* derived from the SEAT in Fig. 1 (a). We assume the deadline of the last task is 190. For simplicity, we also assume that the computation time for tasks 1, 2, and 3 are 50, 50, and 40 time units. The networks in the eclipses are composed of optional security instances for the required security services. For example, the network between nodes 1 and 2 denotes that task 1 requires two security services which can be implemented by two security instances. Each node of the phase graph has two attributes: security

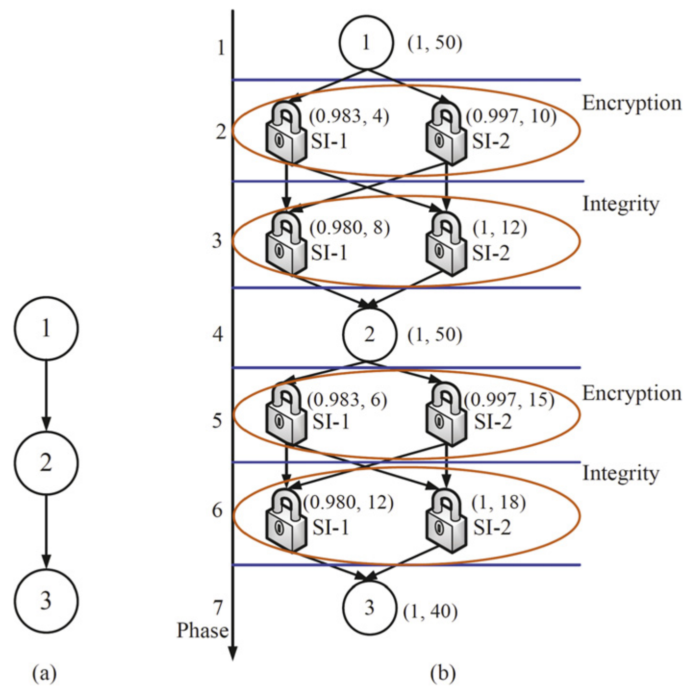

Fig. 1. (a) the original SEAT graph, (b) the phase graph derived from SEAT graph.

Algorithm 6.1 Dynamic programming based security optimization for single path SEAT graph.

```
Require: (1) simple path STAG: $G$, (2) security strength: $SS(S_i^j)$, (3) security overhead: $CO(S_i^j)$
Ensure: An optimal security strategy for the STAG
  Step 1. Associate an array $X_i[1,2,\ldots,L]$ to each phase and $X_i[j]$ store the maximum system security of the path from phase 1 to phase $i$ with total computational time $\leqslant j$. For $1 \leqslant j \leqslant L$, $X_0[j] = 0$;
  Step 2. $\mathcal{N}$ is the number of phases
  for $i = 1$ to $\mathcal{N}$ do
      $$X_i[j] = \begin{cases} \max_{1\leqslant k\leqslant F(i)}\{X_{(i-1)}[j - t_k(i)] * sec_k(i) \text{ if } j - t_k(i) \geqslant T_{min}(i-1)\}, \\ \text{no feasible solution, otherwise} \end{cases} \quad (20)$$
    where $T_{min}(i-1)$ is the minimum time needed to process the path from phase 1 to phase $i-1$ and $T_{min}(0) = 0$, $sec_k(i)$ is the security strength of the $k$th implementation for phase $i$
  end for
  Step 3. $X_{\mathcal{N}}[L]$ is the maximum system security and the optimal security strategy can be obtained by tracing how to reach $X_{\mathcal{N}}[L]$;
  Return optimal security strategy.
```

strength and computational time. These two attributes of security instances are in the form of $\langle SS(S_{i,j}), CO(i, j, k)\rangle$. If a task does not require any security service there is no security network follows it (e.g. node 3 in Fig. 1 (b)). If the last task node of SEAT graph has security service requirements we should add a dummy sink node to the phase graph. Since task nodes (e.g. nodes 1, 2, 3) and sink node have no security attribute, for the sake of modeling the problem defined in Eq. (6), the security value of all task nodes and sink node are assigned value 1. Therefore, the security optimization problem can be converted to the following problem: *finding a path* (*from the first phase to last phase*) *such that the product of security values of each node on the path is maximized while the total execution time of the path does not exceed the deadline.* The pseudocode of the DPSOP-path algorithm is shown in Algorithm 6.1.

Theorem 6.1. $X_i[j]$ *obtained by* Algorithm DPSOP-path *is the maximum system security of the path from phase* 1 *to phase* i *with total execution time* $\leqslant j$.

Proof. By induction. **Basic Step:** When $i = 1$, $X_0[j] = 0$ and $T_{min}(0) = 0$, so $X_1[j] = \max_{1\leqslant k\leqslant F(1)}\{Sec_k(1) \text{ if } j \geqslant t_k(1)\}$, where $F(1)$ is the number of implementations for the phase 1. When $i = 1$, $F(1) = 1$, thus Theorem 6.1 is true. **Induction Step:** We need to show that for $i \geqslant 1$, if $X_i[j]$ is the maximum system security of the path from phase 1 to phase i, then $X_{i+1}[j]$ is the maximum system security of the path from phase 1 to phase $i+1$. It is obviously true from Eq. (20). Thus, Theorem 6.1 is true for any phase. □

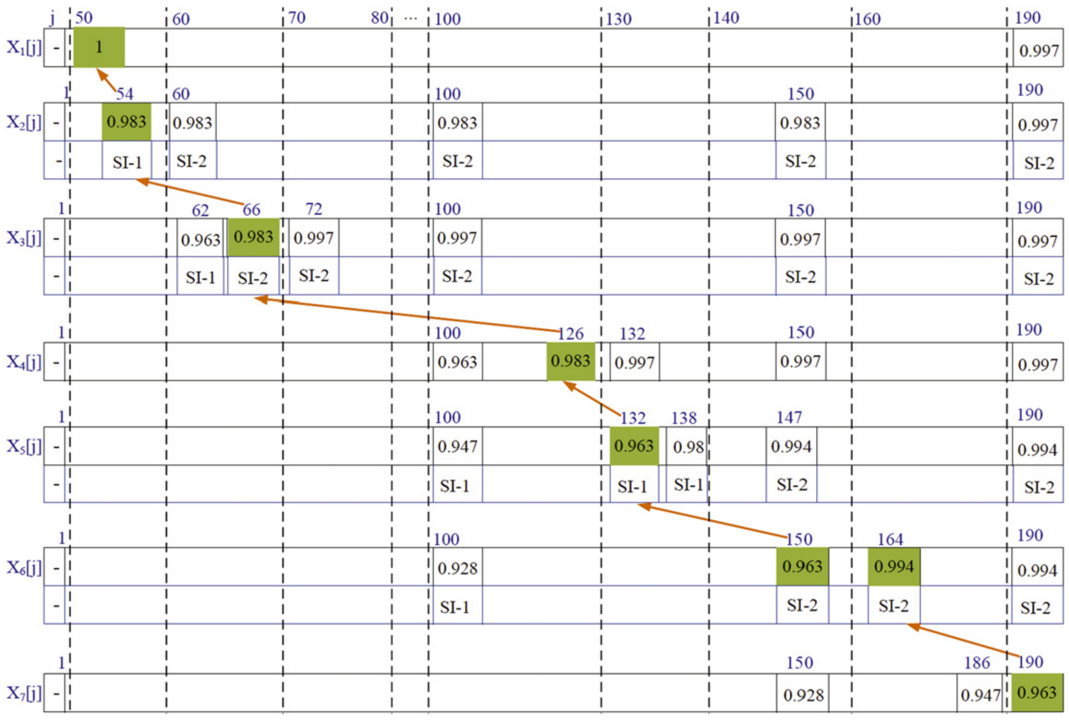

Fig. 2. The executing process of *DPSOP-path*.

From Theorem 6.1, we know $X_{\mathcal{N}}[L]$ records the maximum system security of the whole path within the time constraint L. We can record the corresponding implementation of each phase when computing the maximum system security in Step 2 in *Algorithm* 6.1. Using this information, we can get an optimal security strategy by tracing how to reach $X_{\mathcal{N}}[L]$.

We take the SEAT graph in Fig. 1 as an evaluation example to demonstrate the execution process of the *DPSOP-path* algorithm. The security strength and computational cost of each node have been labeled. In Fig. 2, array $X_i[j]$ is used to record the system security strength of phase i at step j. For security service phases (phases 2, 3, 5, 6), the selected security instance is recorded as well. For example, the label under $X_2[54]$ is “SI-1”, which means security instance “SI-1” can be completed in step 54, and the maximal security strength of phase 2 at this step is 0.983. In this algorithm, each security service phase will produce an optimal solution for each time constraint. Based on the optimal solutions of last phase, we can find a best security instance to maximize the security strength for the current tasks. Using the *DPSOP-path* algorithm we can get the optimal system security strength in $X_7[190]$. The optimal security strategy can be obtained by tracing how to reach $X_7[190]$. Starting from $X_7[190]$, we know the execution time of task 3 $t(3) = 40$ which is shown in Fig. 1 (b). Then, we can get the index j for $X_6[j]$ by substracting $t(3)$ from deadline: $190 - t(3) = 150$. Consequently, we know security instance SI-2 is selected as implementation of the integrity service for task 2, and the security strength is 0.963. Since the computation time for security instance SI-2 is 18, we can get the optimal computation time by encryption of task 2 is $150 - 18 = 132$. Therefore, the security instance SI-1 is used to implement the encryption instance of task 2. In this way, we can get the entire optimal security strategy. The tracking back process is illustrated by arrows in Fig. 2.

DPSOP-path takes $O(I)$ to compute one value of $X_i[j]$ where I is the number of implementation method for a phase. Thus, the complexity of *Algorithm DPSOP-path* is $O(\mathcal{N} * L * I)$, where $\mathcal{N}$ is the number of phases and L is the given timing constraint. Usually, the execution time of each phase is upper bounded by a constant. So L equals $O(|V|^k)$ (k is a constant). In this case, *DPSOP-path* is polynomial.

6.2. *DPSOP-tree algorithm*

In this section, we propose an efficient algorithm, *DPSOP-tree*, to produce the optimal security strategy for the *security optimization problem* when the given SEAT graph is a tree.

An example of tree-structured SEAT graph is shown in Fig. 3. Define a *root* node to be a node without any parent and a *leaf* node to be a node without any child. In Fig. 3, the “locks” represent security services and the other nodes represent the tasks. For example, “lock” S1-1 denotes the first security service of task 1. *A post-ordering* for a tree is a linear ordering of all its nodes such that if there is an edge $e(u \rightarrow v)$ in the tree, then v appears before u in the ordering. For example, both $\{S3-1, 3, S4-1, 4, 2, S1-2, S1-1, 1\}$ and $\{S4-1, 4, S3-1, 3, 2, S1-2, S1-1, 1\}$ are post-ordering for the given tree in Fig. 3. Here, sequences do not matter as long as post-ordering is followed, since post-ordering is used to guarantee

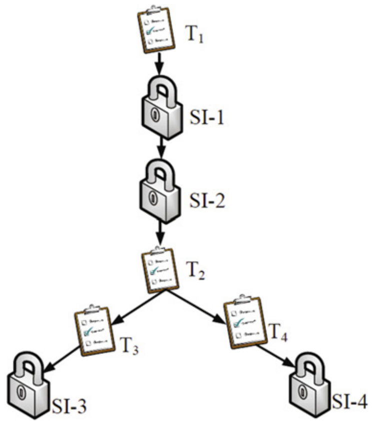

Fig. 3. Tree-structured SEAT graph with security services.

Algorithm 6.2 Dynamic programming based security optimization for tree-structured SEAT graphs.

```
Require: (1) tree-structured STAG: $G$, (2) security strength: $SS(S_i^j)$, (3) security overhead: $CO(S_i^j)$
Ensure: An optimal security strategy for the STAG
  Step 1. Post-order SEAT graph and let $V = \{v_1, v_2, \dots, v_{\mathcal{N}}\}$ be the post-ordering node set. Without loss of generality, for each security service node $v_i$, let $T(v_i) = \{t_1(i), t_2(i), \dots, t_l(i)\}$ where $t_j(i)$ is the computational time of node $v_i$ implemented by the $j$th security instance; and $Sec(v_i) = \{sec_1(i), sec_2(i), \dots, sec_l(i)\}$ where $sec_j(i)$ is the security probability of node $v_i$ implemented by the $j$th security instance;
  Step 2. Associate an array $X_i[1, 2, \dots, L]$ to each phase and $X_i[j]$ stores the maximum system security of the subtree rooted on $v_i$ with total computational time $\leqslant j$. For $1 \leqslant j \leqslant L$, $X_0[j] = 0$, $\forall j \leqslant L$;
  Step 3. $\mathcal{N}$ is the summation of task and security service nodes
  for $i = 1$ to $\mathcal{N}$ do
    compute $X_i[j]$ $(j = 1, 2, \dots, L)$ as follows:
      $$X_i[j] = \begin{cases} \min_{1\leqslant k\leqslant M}\{X_{i'}[j - t_k(i)] + sec_k(i) \text{ if } j - t_k(i) \geqslant T_{min}(v_i)\}, \\ \text{no feasible solution,} \quad \text{otherwise} \end{cases} \quad (21)$$
    where $T_{min}(v_i)$ is the minimum time needed to process the subtree rooted on $v_i$ except $v_i$. $v_i'$ is a pseudo node. Assume that $v_{i_1}, v_{i_2}, \dots, v_{i_R}$ are all child nodes of node $v_i$ and $R$ is the number of child nodes of node $u_i$, then $X_{i'}[j]$ $(j = 1, 2, \dots, L)$ is calculated as follows:
      $$X_{i'}[j] = \begin{cases} 0 & \text{if } R = 0, \\ X_{i_1}[j] & \text{if } R = 1, \\ \sum_{1\leqslant h\leqslant R} X_{i_h}[j] & \text{if } R \geqslant 1 \end{cases} \quad (22)$$
  end for
  Step 4. $X_{\mathcal{N}}[L]$ is the maximum system security for a SEAT graph and the optimal security strategy can be obtained by tracing how to reach $X_{\mathcal{N}}[L]$;
  Return optimal security strategy.
```

that, when we begin to process a node, the processing of all of its child nodes has already been finished in the algorithm. The pseudo-polynomial algorithm for trees, *DPSOP-tree*, is shown in Algorithm 6.2.

Following the post-ordering in Step 1, we can get $T_{min}(v_i)$ for each node $v_i \in V$ by setting the minimum execution time for each node and computing the longest path from any leaf node to v_i. In Eq. (21), basically, we select the maximum system cost from all possible system costs caused by adding v_i into the subtree. In the following, we prove *Algorithm DPSOP-tree* gives the optimal security strategy for tree-structured SEAT graphs.

Theorem 6.2. *$X_i[j]$ $(1 \leqslant i \leqslant N)$ obtained by* Algorithm DPSOP-tree *is the maximum system security of the subtree rooted on v_i with total execution time $\leqslant j$.*

Proof. By induction. **Basic Step:** When $i = 1$, because the computation of $X_i[j]$ follows the post-ordering, v_1 must be a leaf node. Thus, $X_{1'}[j] = 0$. $X_0[j] = 0$ and $T_{min}(v_1) = 0$, so $X_1[j] = \min_{1\leqslant k\leqslant I}\{sec_k(1) \text{ if } j \geqslant t_k(1)\}$. Thus, when $i = 1$, Theorem 6.2 is true. **Induction Step:** We need to show that for $i \geqslant 1$, if $X_i[j]$ is the maximum system security of the subtree rooted on v_i, then $X_{i+1}[j]$ is the maximum system security of the subtree rooted on v_{i+1}. According to the post-ordering, the computation of $X_i[j]$ for each child node of v_{i+1} has been finished before computing $X_{i+1}[j]$. From Eq. (22), $X_{(i+1)'}[j]$ gets the summation of the maximum system security of all child nodes of v_{i+1} because they can be allocated and executed simultaneously within time j. From Eq. (21), the maximum system security is selected from all possible system costs caused by adding v_{i+1} into the subtree rooted on v_{i+1}. So $X_{i+1}[j]$ is the maximum system security of the subtree rooted on v_{i+1}. Therefore, Theorem 6.2 is true for any node in the tree. □

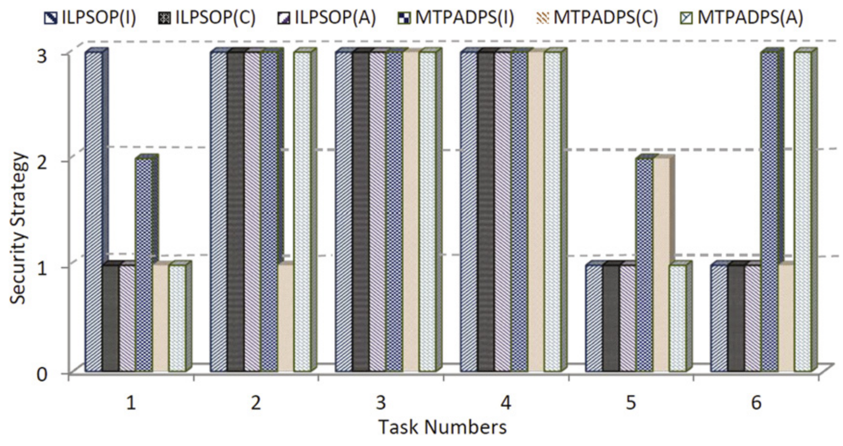

Fig. 4. The comparison of the optimal security strategy for ILP-SOP method and MTPADPS method.

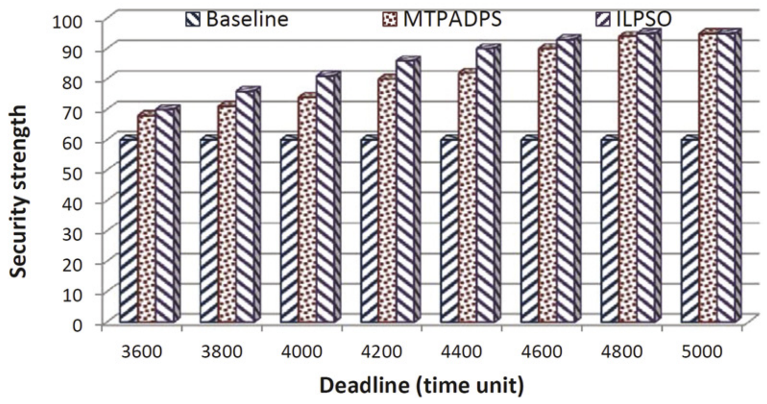

Fig. 5. Security strength comparison of baseline, MTPADPS, and our ILP-SOP method under different deadline constraints.

From Eq. (22), $X_{i'}[j] = 0$ if v_i has no child node and $X_{i'}[j] = X_{i_1}[j]$ if v_i has only one child node v_{i_1}. Thus, we do not really need to compute $X_{i'}[j]$ in these cases. When there are more than one root node in STAG, we need to add a pseudo root node u_{N+1} to construct a tree structure.

The complexity of *DPSOP-tree* is $O(|V| * L * I)$, where $|V|$ is the number of nodes, L is the given time constraint, and I is the number of implementations for a node. When L equals $O(|V|^k)$ (k is a constant) which is the general case, in practice, *DPSOP-tree* is polynomial.

7. Experiment

In this section, we conduct experiments with the ILP-SOP method on a set of synthetic SEAT graphs. We use a commercial ILP solver *Lingo* [22] to implement the ILP-SOP method.

In our experiment, each task requires three types of data security services: Integrity (I), Confidentiality (C), and Authentication (A). For each type of security service, there are three security instances. Quantitative security strength and computational overhead of these security instances are expressed by two matrix *sgain* and *scost*, respectively. We assume that the deadline of this SEAT graph is 24 000 time unit. Our goal is to achieve the optimal security strategy.

Solving this ILP model by Lingo 11.0 can be completed in only 5 seconds on a 2.0 GHz Pentium 4 laptop with 2 GB of memory. When the deadline is 4000 time units, one optimal security strategy generated by our ILP-SOP method is shown in columns under the “ILP-SOP” in Fig. 4. For example, the first row under “ILP-SOP” is $\{3, 1, 1\}$, which means for the first task the integrity service is implemented by the third security instance, and confidentiality and authentication service are both implemented by their first security instance. The columns under “MTPADPS” is the security strategy generated by modified TPADPS algorithm [14].

Fig. 5 shows the comparison of security strength derived from different techniques under different deadline constraints. In the baseline method, the security instances with the minimal security strength (large than or equal to the required security strength) are selected as the implementations for all security services of each task. According to computational time of each task and security overheads, we assume that the range of deadline is from 3600 to 5000 time units.

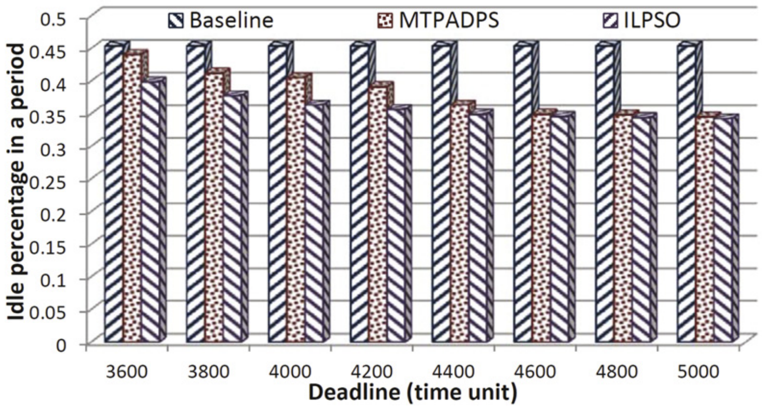

Fig. 6. Idle percentage comparison of baseline, MTPADPS, and our ILP-SOP method under different deadline constraints.

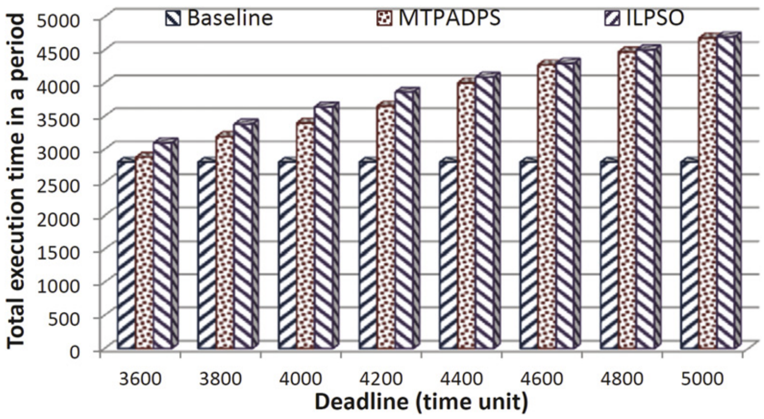

Fig. 7. Total execution time comparison of baseline, MTPADPS, and our ILP-SOP method under different deadline constraints.

In the set of a SEAT graph, the security strength is only 60.24% for an entire application, on average. Therefore, it is necessary to optimize the security strategy. By using our ILP-SOP technique, the security strength can be improved by 44.3% on average. The main reason for this improvement is that our ILP-SOP technique can make full use of the slack time to achieve higher security. As shown in Fig. 6, we compare the idle percentage in one period. Our ILP-SOP method has the lowest percentage of idle time in one period. Thus, using a larger percentage of the slack time contribute to the improvement of the security strength of an application. We also compare the total execution time of an application in one period by using different scheduling strategies, as shown in Fig. 7. Although our ILP-SOP has the longest total execution time in one period, it does not impact on the whole performance of the system. This is mainly because the speed of the system primarily depends on the period of the application, as long as the total execution time of an application does not exceed the deadline.

Another observation of Fig. 5 is that system security strengths generated by our ILP-SOP method (shown as "ILP" in the figure) are higher than that derived from the MTPADPS algorithm when the deadline constraint is between 3600 to 4400. The average security strength improvement is 6.25%. The main reason is that MTPADPS is only a local greedy algorithm that does not consider the interaction between different security instance selections. If all the security services are implemented by the security instances with the maximum security strength, we can get the optimal security strength when the deadline is 4715 time units. From Fig. 5 we can see when the given deadline is 4800 time units our ILP-SOP method can achieve the optimal security strength but the MTPADPS algorithm cannot. In addition, we compare the security overhead in Fig. 8, which shows that our ILP-SOP method results in the largest security overhead, aligning with the result shown in Fig. 5.

8. Conclusion

This paper addressed the security optimization problem for real-time ubiquitous application. We designed an ILP-based method ILP-SOP, which can solve the security strategy generation and schedulability test problem simultaneously. The main advantage of the ILP-SOP method is that it can make full use of the slack time to achieve the optimal security strategy under a given deadline constraint. Furthermore, we made special efforts to design two polynomial time algorithms, DPSOP-

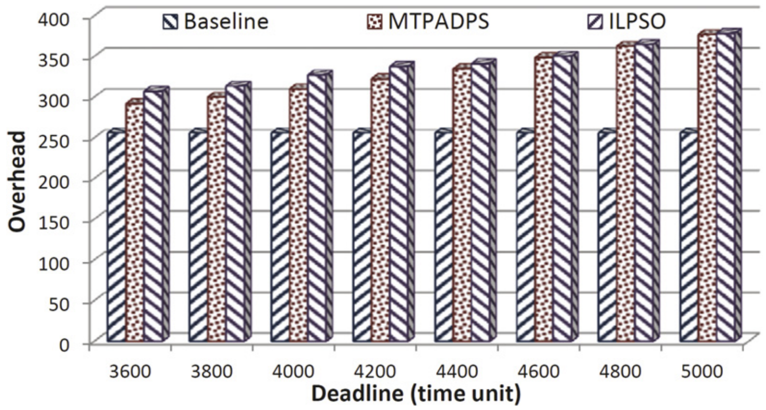

Fig. 8. Overhead comparison of our of baseline, MTPADPS, and our ILP-SOP method under different deadline constraints.

path and DPSOP-tree, to achieve the optimal security strategy for special structured STAG graphs. The experimental results validate our proposed methods.

As the future work, we plan to extend our ILP method to more complex architecture such as heterogenous multiprocessor systems. Improving qualities of quantitative security strength measurement is also worthy of further study.

Acknowledgments

This work was supported in part by the NSF CNS-1249223, NSFC 61071061, the Univ. of Kentucky Start Up Fund; NSFC 61170077, SZ-HK Innovation Circle project ZYB200907060012A, NSF GD:10351806001000000, S & T project of SZ JC200903120046A; the NSF CNS-0969013, CNS-0917021, and CNS-1018108; the NSF CNS-0915762 (CSR), CCF-08452578 (CAREER), CNS-0917137 (CSR).

References

[1] S. Son, R. Mukkamala, R. David, Integrating security and real-time requirements using covert channel capacity, IEEE Trans. Knowl. Data Eng. 12 (2000) 865–879.
[2] P. Kocher, R. Lee, G. Mcgraw, A. Raghunathan, S. Ravi, Security as a new dimension in embedded system design, in: Proceedings of ACM/IEEE DAC, 2004, pp. 753–760.
[3] T. Xie, X. Qin, Improving security for periodic tasks in embedded systems through scheduling, ACM Trans. Embed. Comput. Syst. 6 (3) (2007) 1–19, Article 20.
[4] M. Lin, L.T. Yang, X. Qin, N. Zheng, Z. Wu, M. Qiu, Static security optimization for real-time systems, IEEE Trans. Ind. Inform. 5 (1) (2009) 22–37.
[5] Z. Shao, C. Xue, Q. Zhuge, M. Qiu, B. Xiao, E.H.-M. Sha, Security protection and checking for embedded system integration against buffer overflow attacks via hardware/software, IEEE Trans. Comput. 55 (4) (2006) 443–453.
[6] Z. Shao, C. Xue, Q. Zhuge, E. Sha, B. Xiao, Security protection and checking in embedded system integration against buffer overflow attacks, in: IEEE ITCC, 2004, pp. 409–412.
[7] P. Kocher, Security in embedded systems: Design challenges, ACM Trans. Embed. Comput. Syst. 3 (3) (2004) 461–491.
[8] F. Balarin, L. Lavagno, P. Murthy, Scheduling for embedded real-time systems, IEEE Trans. Design Test Computers 15 (1) (1998) 71–82.
[9] M. Qiu, E.H.-M. Sha, Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems, ACM Transact. Des. Automat. Electron. Syst. (TODAES) 14 (2) (March 2009) 1–30, Article 25.
[10] A. Menezes, P. Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.
[11] J. Daemen, V. Rijmen, The Design of Rijndael: AES–The Advanced Encryption Standard, Springer, 2002.
[12] T. Xie, X. Qin, Scheduling security-critical real-time application on clusters, IEEE Trans. Comput. 55 (7) (2006) 864–879.
[13] C. Irvine, Quality of security service, in: Proc. ACM New Security Paradigms Workshop, 2000, pp. 91–99.
[14] T. Xie, X. Qin, A new allocation scheme for parallel applications with deadline and security constraints on cluster, in: Proc. of the 7th IEEE International Conference on Cluster Computing, 2005, pp. 1–10.
[15] C. Liu, J. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, 1973.
[16] J. Stankovic, M. Spuri, K. Ramamritham, G. Buttazzo, Deadline Scheduling for Real-Time Systems–EDF and Related Algorithms, Kluwer Academic Publishers, Boston, MA, 1998.
[17] K. Ramamritham, J. Stankovic, Dynamic task scheduling in distributed hard real-time system, IEEE Trans. Softw. 1 (3) (1984) 65–75.
[18] B. Schneier, D. Whiting, Fast software encryption: Designing encryption algorithms for optimal software speed on the Intel Pentium processor, in: International Workshop Proceedings on Fast Software Encryption, 1997, pp. 242–259.
[19] S. Fluhrer, I. Mantin, A. Shamir, Weaknesses in the key scheduling algorithm of RC4, in: The 8th Annual Workshop on Selected Areas in Cryptography, 2001, pp. 1–24.
[20] B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, Wiley, 1996.
[21] A. Bosselaers, Fast implementations on the Pentium, http://homes.esat.kuleuven.be/~bosselae/fast.html.
[22] Lingo, http://www.lindo.com.