Energy Aware Real-Time Scheduling Policy with Guaranteed Security Protection

Wei Jiang[1,3], Ke Jiang[2], Xia Zhang[3], and Yue Ma[4]

[1]*School of Computer Science and Engineering, University of Electronic Science and Technology of China, China*
[2]*Department of Computer and Information Science, Linköping University, Sweden*
[3]*School of Information and Software Engineering, University of Electronic Science and Technology of China, China*
[4]*Department of Computer Science and Engineering, University of Notre Dame, U.S.A.*
weijiang@uestc.edu.cn, ke.jiang@liu.se, zhangxia0414@163.com, yma1@nd.edu

***Abstract*—In this work, we address the emerging scheduling problem existed in the design of secure and energy-efficient real-time embedded systems. The objective is to minimize the energy consumption subject to security and schedulability constraints. Due to the complexity of the problem, we propose a dynamic programming based approximation approach to find the near-optimal solutions with respect to predefined security constraint. The proposed technique has polynomial time complexity which is about half of traditional approximation approaches. The efficiency of our algorithm is validated by extensive experiments.**

I. Introduction

Real-time embedded systems are facing more and more severe security threats, e.g. due to the integration of new communication interfaces. One of the emerging needs is to protect sensitive data in critical embedded systems [1], [2]. Since snooping, spoofing and altering security-critical data can lead to significant losses or serious system failures, resulting in great loss of finance or human life. We refer to such systems as Security-Critical Real-Time Systems (SCRTS). Examples of SCRTS are flight control systems, satellite communication systems and radar tracking systems, which all have strict security requirements. To protect SCRTS against potential threats, a series of security services, i.e. integrity, confidentiality and authentication protection, need to be considered in the design process of SCRTS. With the most suitable security protections regarding the demands, SCRTS would be effectively protected via using the right amount of resources, e.g. CPU utilization.

Energy efficiency is another fundamental requirement in the context of SCRTS [3], [4], [5]. But security protections usually demand a significant amount of energy [2]. Quick energy depletion or early exhaustion of battery may cause failure to mission-critical tasks, resulting in unexpected losses, such as the energy incurred failure of Mars Pathfinder. Hence, how to design energy-efficient SCRTS becomes a great challenge.

Task scheduling policy plays an important role for achieving high performance in embedded systems. Unfortunately, traditional real-time scheduling approaches were mostly designed to guarantee timing requirements only [6]. Recently, security-aware real-time scheduling has become a hot research topic, e.g. [7], [8], [9], [10], [11]. However, all these works did not consider the energy factor in system-level designs, which may deliver solutions with unexpected energy consumptions.

In this paper, we identify the uniprocessor scheduling problem lying in many SCRTS designs considering energy, security and real-time dimensions. Specifically, we want to schedule a set of periodical real-time tasks with the objective of minimizing energy consumption, while satisfying security and timing constraints. The primary contributions of this paper are in two aspects. First, a typical security- and energy-aware SCRTS application is presented. Second, we study the existing policies and propose an efficient approximation approach that guarantees the security requirements. Our approach has polynomial time complexity, and requires bounded memory space. To the best of our knowledge, this is the first work that addresses the security- and energy-aware real-time task scheduling problem.

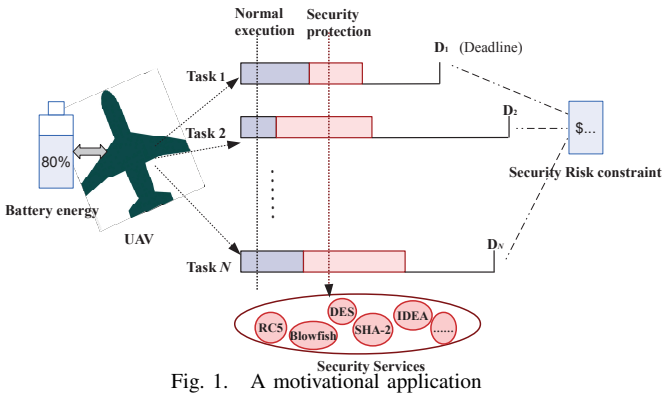

Fig. 1. A motivational application

II. Application and System Model

A. Motivational application

In this paper we focus on the SCRTSs with limited energy budget, for example, an unmanned aerial vehicle (UAV) depicted in Fig. 1. The UAV is battery driven, and has limited resources, i.e. CPU and memory. It runs critical tasks that are periodically released, and exchanges information with other peers or service centers. Each task generates or receives some private data that needs to be transmitted over insecure environments. Different data has different requirements of security and deadline guarantee ratios. In order to make the communication secure (i.e. to protect the confidentiality or integrity of the messages), we need to perform cryptographic algorithms like RC5, DES and SHA-2, on the data before or after the normal executions of corresponding tasks. Thus, the energy consumption of each task consists of two parts that are from the normal execution and extra security protections, which will be discussed in Sec. II(B). Although there are many available cryptosystems, it is hard to obtain the best choices among different solutions having different execution and energy overhead, and the UAV only has limited energy and processing capability. Therefore, how to allocate resources to protect different data becomes an important design trade-off. In other words, we are aiming to schedule a set of periodic real-time tasks with the objective of minimizing energy consumption while satisfying security and schedulability constraints.

B. Task model for security-critical real-time systems

In this paper, we consider a set of periodic security- and energy-aware tasks running on SCRTS. Each task T_i is modeled as a tuple $T_i = \{BE_i, L_i, S_i, S_i^{DM}, V_i, SR_i, P_i\}$. BE_i denotes the worst case execution time (WCET) of its non-security part. L_i is size of the data that is generated or received by T_i and needs to be protected using selected security service. S_i and S_i^{DM} are the chosen and designated security levels of T_i, respectively. If S_i^{DM} is achieved, this task is assumed to be absolutely secure. V_i is the security impact value of T_i representing the relevant importance of the messages processed by T_i. SR_i is the security risk of T_i, and means the potential loss of the security protection, which will be elaborated in Sec.II(D). P_i is the period and also the relative deadline of T_i.

C. Time and energy overhead of security critical tasks

It is known that security protections can be achieved by additional security services, which also compete resources with other executions. For example, doing AES encryption on one message may reduce the available CPU resource for protecting other messages. So it is indispensable to always allocate the right amount of resource to the security protections among tasks in order to reach the best global security protection.

It is still an open problem of quantifying the security strength of different cryptographic algorithms. So we quantify the security level of an algorithm based on its execution time similar to [8]. Then security levels of typical algorithms are enumerated as their relative strengths. Based on the measurement on a S3C2440 ARM board with 500MHz CPU and 64MB SDRAM [12], we obtain the time and energy overheads of six widely used confidentiality services for protecting 1 KB data as shown in Table I. For example, we assign security level 1 to the relatively weakest algorithm RC4 that has the shortest encryption time. In this paper, we only consider periodic tasks, so we assume that the key setup procedures for the security algorithms are prepared before the system starts, and thus, ignored, for the sake of simplicity. Note that this paper just gives a reasonable quantification of security level of cryptography algorithms. It can be changed by more reasonable data. If there exists newly developed algorithm with higher security strength while lower time overhead, we can use it to replace the ones with larger overhead and lower level.

If task T_i generates or receives L_i Bytes security sensitive data, then the execution time of T_i can be formulated as follows,

$$Exe_i = BE_i + \theta(S_i) * L_i \tag{1}$$

$\theta(S_i)$ is the mapping function of security level S_i to unit execution time of the chosen algorithm. Taking $S_i = 3$ for example, $\theta(S_i)$ is thus the unit execution time overhead of BLOWFISH algorithm according to Table I. As can be observed from Table I, there is a close to linear relation between energy consumption and encryption time, i.e. approximately $320mJ/S$, which is recognized as the *power*. We also found that the *power* of task's basic execution is nearly the same situation in our measurement. Thus, the total energy consumption T_i including the data protections is the product of power (POW) and the whole execution time.

$$En_i = POW * Exe_i = POW * (BE_i + \theta(S_i) * L_i) \tag{2}$$

TABLE I
TIME AND ENERGY OVERHEAD OF CONFIDENTIALITY ALGORITHMS

Ciphers	time(ms/KB)	Energy(mJ/KB)	Sec. Level
RC4	0.0063	2.0237	1
RC5	0.0125	4.0340	2
BLOWFISH	0.0170	5.4696	3
IDEA	0.0196	6.2822	4
SKIPJACK	0.0217	6.9658	5
3DES	0.0654	21.0914	6

D. Risk model for security-critical task

To quantify the security quality of tasks, it is necessary to introduce security risk model. In [8] and [10], linear profit model, which is the sum of weighted security levels, is used to evaluate the quality of each security-critical task. Obviously, this model is impractical to capture the real security quality. Since the potential risk is the product of security violation probability and consequence of security breach [13], we model the security risk (SR) as the expected security loss of T_i:

$$SR_i = V_i * Pro_i^{risk}, \tag{3}$$

where V_i is the security impact value of T_i representing the relevant importance of its data. It could be finance loss or other metrics, e.g., 500$ loss if T_i failed. Pro_i^{risk} is the failure probability of T_i with chosen security level S_i formulated as

$$Pro_i^{risk} = \begin{cases} 1 - e^{-\lambda_i(S_i^{DM} - S_i)}, & \text{if } S_i < S_i^{DM} \\ 0, & \text{if } S_i \geq S_i^{DM}. \end{cases} \tag{4}$$

λ_i is the security risk coefficient of T_i, which can be adjusted by the designer based on different scenarios. As implied in Eq. 4, if the assigned security level is greater or equal to the demanded security, we assume no failure will occur when facing attacks. Inversely, the task has the probability to fail, and bigger security demand gap leads to higher probability of security violation. This paper just introduces a more reasonable and practicable metric, i.e. the security risk model, to quantify the quality of security-aware tasks, and uses it to assess the system performance.

III. PROBLEM FORMULATION

A. Original problem

In this paper, we consider a security-critical embedded system that has a set of N tasks. A system monitor gives the security Risk Bound (RB) that defines the current security requirement of the whole system. In another words, if the task executions lead to security violations, i.e. the system cannot satisfy the expected RB, then it is recognized as an unacceptable system. In certain situations, it is demanded that the security risk should not succeed $(1 + \beta)RB$, where β is the risk slack ratio and defined by designers according to the system requirement. Higher security-critical application has less value of β. So the problem is to assign the most suitable security services for tasks using the minimum energy consumption, and satisfy the strict real-time and security constraints.

Before going further, let us introduce the definition of *hyperperiod* (HP) that is the least common multiple of all tasks' periods. Thus, the purpose of this paper becomes to obtain the minimal long-term energy consumption for the task set within a hyperperiod. So our energy minimization scheduling problem can be formulated as

$$\text{Minimize } Energy = \sum_{i=1}^{N} (HP/P_i) * En_i \tag{5}$$

Subject to

$$\begin{cases} \sum_{i=1}^{N}(HP/P_i) * SR_i \leq RB \\ \sum_{i=1}^{N}(BE_i + \theta(S_i) * L_i)/P_i \leq UB_x \\ S^{min} \leq S_i \leq S^{max}, \end{cases}$$

where, UB_x denotes the utilization ratio bound of the x scheduling policy. A set of periodic tasks is schedulable on a processor using real-time scheduling policies, e.g. EDF ($UB_{EDF} = 1$) and RM ($UB_{RM} = 0.693$), if the processor utilization ratio is not more than the given bound. The first two constraints are the system security risk constraint and real-time constraint, respectively. The last constraint makes sure the correctness of the selected security protection method: the permitted security level must lie between S^{min} and S^{max}.

B. Reduced problem

The design problem that we are facing is a multi-constrained optimization problem, which can be transformed into a Multi-Dimensional Multiple Choice Knapsack Problem. It takes large computation overhead to get the optimal solution for large-scale designs. Thereby, we try to find an efficient assignment algorithm that can obtain good solution while satisfying the predefined constraints. In this section, we will reduce the fore-mentioned problem. Combining Eq. 2 and Eq. 5, we can rewrite our optimization objective as

$$\begin{aligned} Energy &= \sum_{i=1}^{N}(HP/P_i) * En_i \\ &= HP * POW * \sum_{i=1}^{N}(BE_i + \theta(S_i) * L_i)/P_i \end{aligned} \quad (6)$$

where,for the fixed hardware platform and application, the hyperperperiod HP, power POW and task's non-security execution utilization $\sum_{i=1}^{N} BE_i/P_i$ are constant values. Consequently, we can rewrite the design problem as follows,

$$\text{Minimize} \sum_{i=1}^{N} \theta(S_i) * L_i/P_i \quad (7)$$

such that,

$$\begin{cases} \sum_{i=1}^{N}(HP/P_i) * SR_i \leq RB \\ \sum_{i=1}^{N} Exe_i/P_i \leq UB_x \\ S^{min} \leq S_i \leq S^{max} \end{cases}$$

Now with the reduced system problem, we only need to consider the CPU utilization ratio caused by security services and security risk for different level. In addition, this design optimization problem will be further transformed into a Markov decision-making procedure in the next section.

IV. Approximation based dynamic programmings

As the reduced system problem is still not easy to be solved, we must find an efficient approximation approach to solve it. In this section, we transform the problem to a multi-stage Markov decision-making procedure, and use approximate Dynamic Programming to address it.

A. Markov decision-making procedure

Considering a set of N periodic tasks, the utilization ratio minimization problem can be formed as an N-stage Markov decision-making procedure. The decision variable for the i-th stage is the chosen security service S_i that needs to be assigned for task T_i. Thus, the purpose can be transformed as to find a combination $S^* = (S_1, S_2, \cdots, S_N)$ with minimum system utilization ratio while satisfying the security risk constraint.

We denote a triple $(\xi_{ik}, \gamma_{ik}, S_{ik})$ to describe the k-th state in i-th stage. ξ_{ik} and γ_{ik} are two values which present the accumulated CPU utilization ratio and the accumulated security risk for the first i tasks, respectively. S_{ik} is the specific value of security decision variable S_i in this state.

The State Set Ω_i for i-th stage is defined as $\Omega_i = \{(\xi_{i1}, \gamma_{i1}, S_{i1}), (\xi_{i2}, \gamma_{i2}, S_{i2}), \cdots, (\xi_{ii^{max}}, \gamma_{ii^{max}}, S_{ii^{max}})\}$, where i^{max} is total number of states in the i-th stage. Given Ω_{i-1} in $(i-1)$-th stage, we can obtain the state subset Ω_{ik} by adding utilization ratio and security risk of task T_i under security level $S_{ik} \in [S^{min}, S^{max}]$ to all states in Ω_{i-1} as follows,

$$\begin{aligned} \Omega_{ik} &= \Omega_{i-1} \biguplus (U_i(S_{ik}), SR_i(S_{ik}), S_{ik}) \\ &= \{(\xi_{i-1,1} + Exe_i(S_{ik})/P_i, \gamma_{i-1,1} + SR_i(S_{ik}), S_{ik}), \\ &\quad (\xi_{i-1,2} + Exe_i(S_{ik})/P_i, \gamma_{i-1,2} + SR_i(S_{ik}), S_{ik}), \cdots\} \end{aligned}$$

Thus, the state set in i-th stage is $\Omega_i = \bigcup_{k=1}^{|S_i|} \Omega_{ik}$. $|S_i|$ is the number of available options for task T_i. In the first stage, there are only $|S_1|$ states, and the number of states in i-th stage is the production of $|S_i|$ and the number of states in stage i-1. The maximal state space is:

$$SS = \prod_{i=1}^{N} |S_i| \quad (8)$$

As can be noticed from Eq. 8, the state space grows exponentially as the numbers of tasks and security choices grow. Thus it is infeasible to apply Dynamic Programming on large system designs. So, we must find methods to reduce the solution space. Inspired by the approximation algorithm of Knapsack problem [14], grouping the security risk into a series of discrete integers is a good approach to reduce the decision states on each stage. Setting Δ as the group factor, then the security risk of each task can be transformed to an integer decided by SR_i/Δ. Thus, in each stage, we just keep the state with minimal security risk when several states have the same risk value. Bigger Δ gives smaller scaled security risk values and smaller number of states in each decision-making stage. Since the number of states in each decision-making step can be maximized to a constant $M = \lceil RB/\Delta \rceil$, we can leverage a two-dimension table to show the states of all stages.

B. Four approximating approaches

1) Round to Ceiling approach: Round to Ceiling (RC) approach means that we round the divided value to the closest bigger integer. Most of the previous related works concerning energy used this policy to approximate the divided values, like [15] and [16]. For each T_i, the security risk value is divided by Δ. According to the RC policy, the result is scaled up to the closet bigger integer, i.e. $RC(SR_i) = \lceil \frac{SR_i}{\Delta} \rceil$.

2) Round to Floor approach: Round to Floor (RF) approach is similar to the fore-mentioned RC policy, but the obtained result is scaled down to its closest integer, i.e. $RF(SR_i) = \lfloor \frac{SR_i}{\Delta} \rfloor$.

3) Round Randomly: As can be observed, RC will bring positive deviation comparing to the real value, while RF leads to negative deviation. In this section, we introduce a Randoml Round (RR) policy, which round the divided value to the

closest two integers with a certain probability i.e.

$$RR(SR_i) = \begin{cases} \lceil \frac{SR_i}{\Delta} \rceil \text{with prob. } \rho_1 = \frac{SR_i}{\Delta} - \lfloor \frac{SR_i}{\Delta} \rfloor \\ \lfloor \frac{SR_i}{\Delta} \rfloor \text{with prob. } \rho_2 = \lceil \frac{SR_i}{\Delta} \rceil - \frac{SR_i}{\Delta} \end{cases} \quad (9)$$

4) Round to Nearest approach: RR can erase the potential deviation for a set of tasks, but in extreme situations, the deviation is still big. Hence, we introduce a simple scaling policy, Round to Nearest (RN) integer, which scales the divided value to the closest integer with minimal deviation as follows.

$$RN(SR_i) = SR_i^{\Delta} = \begin{cases} \lceil \frac{SR_i}{\Delta} \rceil, \text{ if } \frac{SR_i}{\Delta} - \lfloor \frac{SR_i}{\Delta} \rfloor \geq 0.5 \\ \lfloor \frac{SR_i}{\Delta} \rfloor, \text{ if } \frac{SR_i}{\Delta} - \lfloor \frac{SR_i}{\Delta} \rfloor < 0.5 \end{cases} \quad (10)$$

C. $(1+\beta)$ Approximating Analysis

As proposed in the last section, the approximation approach may return results deviating with the real security risks. Thus, in this section, we analyze the deviation of each approach in the whole optimization problem, and identify the most suitable policy to be used in our scheduling mechanism.

Let us assume that there exists N tasks, and RC policy is applied to discretize security risks, then the Overall Deviation (OD) between the real and approximated risk for the whole system is as follows.

$$\begin{aligned} OD^{RC} &= \sum_{i=1}^{N} (SR_i - \Delta * RC(SR_i)) \\ &\geq \sum_{i=1}^{N} (SR_i - \Delta * (\frac{SR_i}{\Delta} + 1)) \\ &= -N\Delta \end{aligned} \quad (11)$$

Similarly, we have

$$OD^{RF} \leq N\Delta \quad (12)$$

and

$$-N\Delta \leq OD^{RR} \leq N\Delta \quad (13)$$

RN policy has two extreme scenarios. The first one is that the fraction part of grouped security risk is always bigger than 0.5. Then, according to Eq. 10, we can obtain the risk deviation as

$$\begin{aligned} OD^{RN} &= \sum_{i=1}^{N} (SR_i - \Delta * RN(SR_i)) \\ &\geq \sum_{i=1}^{N} (SR_i - \Delta * (\frac{SR_i}{\Delta} + 1/2)) \\ &= -N\Delta/2 \end{aligned}$$

For the other case when the fraction part of grouped security risk is always less than 0.5, the overall risk deviation is $OD^{RN} = N\Delta/2$. Hence, we can get that the risk deviation for all cases, since all scenarios are lying between the above two extreme cases, i.e.

$$-N\Delta/2 \leq OD^{RN} \leq N\Delta/2 \quad (14)$$

Bigger Δ reduces the states in the Markov decision procedure, but brings deviation comparing with the real security risk value. Based on the above analyses, bigger Δ will also increase the deviation for all four policies. If the approximation algorithm can find the near-optimal solution within polynomial time while satisfying the security risk deviation ratio β, then it is a good approach.

For RC policy, the real risk value is less than the scaled value, which means it won't exceed the security risk bound (RB). To the satisfy the minus deviation ratio $-\beta$, it should satisfy following equation

$$-\beta * RB \leq -N * \Delta. \quad (15)$$

Then, the maximal Δ is $\Delta = \frac{\beta * RB}{N}$

For RF policy, the real risk value is more than the scaled value. Therefore, in order to satisfy the risk bound as $(1+\beta) * RB$, the maximal Δ is also $\frac{\beta * RB}{N}$. For RR policy, it has the deviation between RC and RF policies. Then, the maximal Δ is also the same as the above two policies to satisfy $(1+\beta) * RB$. Meanwhile, the case for RN policy is different. Given Δ, the deviation of RN policy is less than the other approaches. More specifically, the deviation is only one second of them according to Eq. 14. Hence, given risk slack ratio β, we can get the maximal Δ as

$$\Delta = \frac{2 * \beta * RB}{N} \quad (16)$$

From Eq. 16, we can notice that RN allows twice larger value of Δ than RF, RC and RR, while satisfying the given security risk deviation ratio. In another words, RN policy gives less time complexity, which is about half of other three policies. So RN is the best policy among the four studied methods, and we will present a RN based approximation algorithm for our system optimization problem in next section.

Algorithm 1 RN-based approximation algorithm

```
//Step 1: Schedulability test
if ∑_{i=1}^{N} (HP/P_i) SR_i(S^max) > RB
    or ∑_{i=1}^{N} Exe_i(S^min)/P_i > UB_x then
    Return. /*Given task set is not schedulable*/
//Step 2: Initialization
Compute Δ = 2βRB/N and M = ⌈RB/Δ⌉
Initialize state matrix Ω_{N×M} with each element Ω_{i,j} = (0,0,0)
Initialize Ω_1 by calculate (ξ_1, γ_1, S_1) with each S_1 ∈ [S^min, S^max]
//Step 3: Update the state matrix in N-Stage decision procedure
for i = 2 to N do
    while (ξ_{i-1}, γ_{i-1}, S_{i-1}) ≠ (0,0,0) in Ω_{i-1} do
        for S'_i = S^min to S^max do
            Calculate temporary state (ξ'_i, γ'_i, S'_i)
            if ξ'_i > UB_x or γ'_i > RB then
                Ignore this state and break /*Schedulability or security violated*/
            if state Ω_{i,j}, (j = γ'_i) is not existed then
                Ω_{i,j} = (ξ'_i, γ'_i, S'_i) /*Store new state*/
            else if ξ'_i < ξ_i in Ω_{i,j} then
                Ω_{i,j} = (ξ'_i, γ'_i, S'_i) /*Keep state with smaller utilization*/
//Step 4: Find the minimal energy consumption solution
Find Ω*_{N,j} with minimal utilization ratio ξ*_N
Obtain the final security assignment decision (S_1, S_2, ··· , S_N) by backtracking
Energy* = ξ*_N * HP * POW /*The minimal energy*/
```

D. RN-based Approximating Algorithm

Based on the RN policy and the 2-dimensional states presentation, we proposed a Dynamic Programming based security-aware approximation solution. The main purpose of RN-based Approximation Algorithm (RNAA) is to assign the most suitable security level to each task using the minimum energy while satisfying the security requirements. The detailed optimization procedure is presented as pseudo-code in Algorithm 1.

RNAA is composed of four steps. In the first step, we test the schedulability of the given task set. If the minimal security risk is higher than the risk bound, i.e. all tasks have the maximal security levels, then the task set is not schedulable; if

the minimal CPU utilization ratio is higher than the utilization bound even if only the minimal demanded security level are assigned on each task, then the task set is also not feasible. In Step 2, RNAA initializes the group factor Δ and the 2-dimensional state matrix.

Step 3, the core of RNAA, conducts the upgrading procedure of decision states for every decision-making stage. Based on each non-zero risk state in $(i-1)$-th stage, RNAA calculates every possible state for i-th stage. If a temporarily generated state obtains higher security risk than the given bound RB or more utilization ratio than UB, it is immediately ignored. Furthermore, if there is no such state with the same risk as the temporary state in N rows of the state matrix, or the temporary state has lower utilization ratio comparing with prior state in i-th stage with the same security risk, then RNAA replaces the old state with the new one that has lower utilization.

After all the decision states in each stage have been renewed, RNAA selects the state with the smallest utilization ratio in the N-th stage in Step 4. To obtain the final security protection decision, RNAA goes back from N to the first stage to get the vector of security assignments. In the end of Step 4, RNAA successfully obtains the minimal energy consumption.

Complexity Analysis: In step 1, it takes $O(n)$ to complete the schedulability test. In step 2, it takes $O(1)$. For step 3, there are $N-1$ decision stages (line 11). Based on each state at the $(i-1)$-th stage, it takes $O|S_i|$ to update all states at i-th stage (see lines 13-20). Due to the number of states can be at most M for $(i-1)$-th stage, it will take $O((n-1)*|S_i|*M)$ for the whole step 3. For step 4, it takes $O(n)$ to find the minimal energy solution. Therefore, the time complexity of RNAA can be inferred as $O(RNAA) = O(n) + O(1) + O((n-1)*|S_i|*M) + O(n) = O(n^2|S_i|/2\beta)$. According to the analysis in Sec.IV(D), it will take $O(n^2|S_i|/\beta)$ time overhead by using other policies (RF, RC, and RR). Thus, we conclude that RNAA is polynomial of tasks number n, security choices $|S_i|$ and $1/2\beta$ and has half time complexity of other traditional approximating policies.

V. Experimental results

In this section, we conducted synthetic experiments to verify the performance of the proposed algorithm. We implement a task scheduler that includes security assignment and scheduling in .Net environment. For evaluation purposes, we compare our RNAA algorithm with one group of approximation methods, i.e. RCAA and RSAA, and one group of heuristics, named GRDY and SEAS. RCAA is an approximation approach based on RC policy like in [16], while RRAA is the approximation approach based on RR policy. For GRDY scheme, the security levels are assigned in a greedy fashion. It provides the current highest security level to tasks step by step until all available energy slacks are depleted. SERS is a Security and Energy-aware Real-time Scheduling (similar to SASES [8]). SERS increase gradually the security level of tasks by comparing the risk-energy ratio among tasks, while satisfying the risk and utilization constraints, just like the benefit-cost ratio used in SASES.

The performance metrics in our experiments are energy consumption and risk deviation ratio. Risk deviation ratio is the deviation ratio of real system security risk to the risk bound. We performed two groups of simulations. For both groups, we generate three synthetic sets. Final results are obtained as the average of these three sets. For each task, the basic execution time is normally generated in the range of 5 ms to 10 ms, and the period is between 300 ms and 500 ms. The security demands are randomly assigned between level 6 and 8 for confidentiality protection. The impact value of each task is randomly generated between 5 and 10, and the size of sensitive data ranges from 100KB to 400KB uniformly. The security coefficient λ is set in $[1, 3]$ for all tasks. Other parameters are set correspondingly in each simulation group. For execution time and energy consumption of security services, we use the same values as Table I.

A. Impacts of Risk Bound over different approaches

In this group of simulation, we analyze the performances under different security bounds. Tasks are generated as discussed in the previous section, and are scheduled by EDF. Thus, the schedulability of the system can be tested by checking whether the utilization bound is not bigger than 1. For a set of tasks, the maximal security risk (MAR) and minimal security risk (MIR) can be calculated when each task is assumed to use the maximal security level or minimal level respectively. In this section, we refer to the risk bound as α. So the real risk bound can be obtained by $MIR + \alpha * (MAR - MIR)$. We set the risk slack ratio β to 0.05 and the risk bound varying from 0.4 to 0.9 with step size of 0.1. The overall energy cost is normalized to the energy consumption of all tasks (each with the maximal security level). Fig. 2 and 3 are the obtained results of the five different algorithms that were discussed at the beginning of this section under different security risk bounds.

There are several interesting phenomenons that can be observed in Fig. 2. For example, with the increase of security risk bound α, the energy costs of all approaches are gradually reducing. This is because that loose risk constraint gives larger search space for each algorithm to decrease their security protection levels, which can result in less energy consumption. GRDY has the largest energy cost while RNAA and RRAA have nearly lowest energy costs. The energy costs of RCAA and SERS lie between them. More specifically, RNAA saves 14.5%, 5.9% and 4.3% energy from GRDY, RCAA and SERS, respectively. RRAA has roughly the same average energy costs as RNAA.

From Fig. 3, we can see the real security risk deviation with different risk bound α. All RNAA, RRAA and RCAA can guarantee the risk slack ratio value 0.05. RCAA has the largest negative deviation ratio as -0.025 averagely. From this group of experiments, we can reach to the conclusion that our proposed RNAA uses the minimal energy cost within given security risk constraint among all the five solutions. Comparing with RRAA and RCAA, the time complexity of RNAA is about half of them according to the derivation in section IV. Therefore, RNAA is the most suitable approach for the scheduling optimization problem of this paper (refer to Section III).

B. Impacts of Risk Slack ratio

Different user or system designer may have different security risk deviation demands on approximating approaches. For example, users can tolerate larger risk deviation for less important applications but require smaller risk deviation for security-critical applications. Thus, the goal of this group of simulations is to evaluate the performance under different risk slack ratios β. We set the security risk bound as

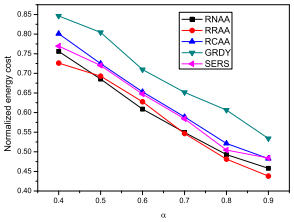

Fig. 2. Impacts of α on energy consumption

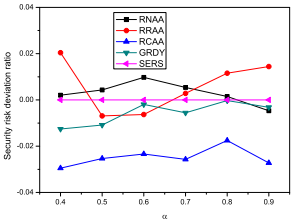

Fig. 3. Impacts of α on risk deviation ratio

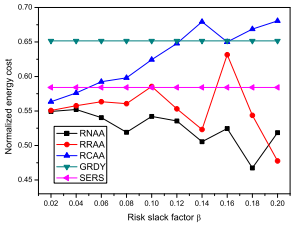

Fig. 4. Impacts of β on energy consumption

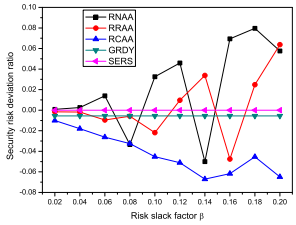

Fig. 5. Impacts of β on risk deviation ratio

$MIR + 0.7 * (MAR - MIR)$. The risk slack ratio varies from 0.02 to 0.2 with step increment of 0.02. The task attributes are generated as discussed in the previous section. The simulation results of overall energy consumption and security risk deviation ratio are shown in Fig. 4 and Fig. 5, respectively.

For security risk in Fig. 4, we can see that RNAA also gives nearly the minimal energy cost on average, while GRDY approach results in the largest energy overhead. The energy costs of GRDY and SERS are constant in this experiment, because GRDY and SERS are not impacted by risk slack ratios. The energy cost of RCAA is increasing with bigger ratios, while the results of RNAA and RRAA demonstrate their random characteristics. RNAA can averagely reduce the energy cost than GRDY, SERS and RCAA by 19.3 percent, 10.0 percent and 16.3 percent, respectively.

Fig. 5 depicts the risk deviation ratios under different risk slack factors. The risk deviation ratios of GRDY and SERS are also constant as they are independent from risk slack factors. All RNAA, RRAA and RCAA can satisfy the given risk slack factor even if the slack factor is very small, e.g. 0.02 and 0.04. RCAA generally obtains negative deviation ratio. RNAA and RRAA have smaller (absolute) risk deviation ratio than RCAA. For example, RNAA can get smaller deviation ratio than RCAA, for example, when it becomes negative with the slack factor value 0.14. The reason is that RNAA and RRAA utilize random scaling policies which cancel out the positive and negative deviations. Based on these two figures, we can find that RNAA is the best algorithm among the five approaches which obtains the lowest energy cost with little security risk deviation.

VI. Conclusion

Energy and security are two important factors for designing mission-critical real-time embedded systems running on constrained resources. This paper addresses one common scheduling problem for security- and energy-critical real-time applications, in which minimizing energy consumption while satisfying the security and schedulability constraints is of central importance. This problem is a multi-dimensional knapsack problem which is proved to be NP-hard. Then, we reduce the problem based on the relationship between energy consumption and CPU utilization. To find the solution efficiently, we introduce and analyze four approximation policies, and then propose our dynamic programming based approximation algorithm to find the near-optimal solutions within predefined security risk constraints efficiently. The proposed algorithm has fully polynomial time complexity that is roughly half of the existing FPTAS approaches. Moreover, our algorithm has also low memory overhead, which is suitable to be used in resource limited embedded systems. Finally, synthetic simulations demonstrate the advantages of our proposed scheduling framework.

VII. Acknowledgement

This work was partly supported by the National Natural Science Foundation of China under Grant No. 61003032 and the Research Fund of National Key Laboratory of Computer Architecture under Grant No. CARCH201104.

References

[1] A. Valenzano, L. Durante, and M. Cheminod, "Review of security issues in industrial networks," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 277–293, 2012.

[2] R. Chandramouli, S. Bapatla, K. Subbalakshmi, and R. Uma, "Battery power-aware encryption," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 2, pp. 162–180, 2006.

[3] V. Chaturvedi, H. Huang, S. Ren, and G. Quan, "On the fundamentals of leakage aware real-time dvs scheduling for peak temperature minimization," *Journal of Systems Architecture*, vol. 58, no. 10, pp. 387–397, 2012.

[4] Y. Wang, H. Liu, D. Liu, Z. Qin, Z. Shao, and E. H.-M. Sha, "Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, no. 2, pp. 14:1–14:32, Apr. 2011.

[5] J. Gan, F. Gruian, P. Pop, and J. Madsen, "Energy/reliability trade-offs in fault-tolerant event-triggered distributed embedded systems," in *Proc. of ASP-DAC*, 2011, pp. 731–736.

[6] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-time systems*, vol. 28, no. 2, pp. 101–155, 2004.

[7] W. Jiang, K. Jiang, and Y. Ma, "Resource allocation of security-critical tasks with statistically guaranteed energy constraint," in *18th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 2012, pp. 330–339.

[8] T. Xie and X. Qin, "Improving security for periodic tasks in embedded systems through scheduling," *ACM Transactions on Embedded Computing Systems*, vol. 6, no. 3, pp. 1–19, 2007.

[9] K. Jiang, P. Eles, and Z. Peng, "Optimization of message encryption for distributed embedded systems with real-time constraints," in *14th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*. IEEE, 2011, pp. 243–248.

[10] M. Lin, L. Xu, L. T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu, "Static security optimization for real-time systems," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 1, pp. 22–37, 2009.

[11] K. Jiang, P. Eles, and Z. Peng, "Optimization of secure embedded systems with dynamic task sets," in *Proc. of DATE*. IEEE, 2013, pp. 1765–1770.

[12] W. Jiang, Z. Guo, Y. Ma, and N. Sang, "Research on cryptographic algorithms for embedded real-time systems: A perspective of measurement-based analysis," in *Proc. of 9th IEEE International Conference on Embedded Software and Systems*. IEEE, 2012, pp. 1495–1501.

[13] B. Karabacak and I. Sogukpinar, "Isram: information security risk analysis method," *Computers & Security*, vol. 24, no. 2, pp. 147–159, 2005.

[14] H. Kellerer and U. Pferschy, "Improved dynamic programming in connection with an fptas for the knapsack problem," *Journal of Combinatorial Optimization*, vol. 8, no. 1, pp. 5–11, 2004.

[15] J. Gong, X. Zhong, and C.-Z. Xu, "Maximizing rewards in wireless networks with energy and timing constraints for periodic data streams," *IEEE Transactions on Mobile Computing*, vol. 9, no. 8, pp. 1187–1200, 2010.

[16] X. Zhong and C.-Z. Xu, "System-wide energy minimization for real-time tasks: Lower bound and approximation," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, p. 28, 2008.