Optimization of Secure Embedded Systems with Dynamic Task Sets

Ke Jiang, Petru Eles, Zebo Peng
Department of Computer and Information Science, Linköping University
{ke.jiang, petru.eles, zebo.peng}@liu.se

***Abstract*—In this paper, we approach embedded systems design from a new angle that considers not only quality of service but also security as part of the design process. Moreover, we also take into consideration the dynamic aspect of modern embedded systems in which the number and nature of active tasks are variable during run-time. In this context, providing both high quality of service and guaranteeing the required level of security becomes a difficult problem. Therefore, we propose a novel secure embedded systems design framework that efficiently solves the problem of run-time quality optimization with security constraints. Experiments demonstrate the efficiency of our proposed techniques.**

I. Introduction

It is common today to find embedded systems controlling safety and reliability critical applications. Recently, security concerns in such systems are emerging rapidly as more and more attacks occur. Moreover, the increasing interactions with the outside world urge the need for security protections in critical embedded systems. At the same time, applications are often functioning under a dynamic load with the number and functionality of active tasks changing during run-time. This leads to the concept of dynamic task set, under which security demands are even more difficult to be achieved. In order to satisfy *timing* and *security* constraints, and provide a high quality of service within limited amount of resources, embedded systems need to be carefully designed. Therefore, it is important to consider the security aspects, together with other constraints, already in the early design stages.

Among the key components of information security, confidentiality is of central importance. This is also valid in the embedded systems domain, because the generated data, especially those controlling safety or reliability critical applications, often contain sensitive information which should not be disclosed to unauthorized parties. In this paper, we focus on providing confidentiality protection for the communication. Moreover, we also consider intrusion detection during the design of secure embedded systems to ensure that malicious remote access can be detected.

Security aspects have been overlooked in most of the previous works on embedded systems design. More recently, however, researchers started to address the issue of security in the context of embedded systems. For example, the global design challenges of secure embedded systems were described in [1]. In [2], the authors presented an automatic hardware-software design flow for detecting code injection attacks in MP-SoCs. A hardware/software co-design technique was presented to protect embedded systems against buffer overflow attacks in [3]. Design of differential power analysis resistant secure embedded systems has been approached in [4], [5]. Delivering sound security protections under real-time constraints has been studied and validated in [6], [7]. We have proposed techniques for designing secure embedded systems for different scenarios, e.g. cost minimization codesign of secure distributed embedded systems and security optimization under various constraints (real-time, energy, etc.), in [8], [9], [10].

None of the previous works, however, has addressed the global, security constrained QoS optimization problem, let alone in the context of dynamic task sets. In the present paper, we propose an approach to this problem.

The rest of this paper is organized as follows. Section II introduces related background. Sections III and IV present our system model and an illustrative example, respectively. We formally formulate the design optimization problems in Section V. The proposed techniques and experimental results are presented in Section VI and VII, respectively. Section VIII concludes the paper.

II. Preliminaries

We will consider three dimensions of optimization in this paper: 1) Quality of Service (QoS) delivered by the application tasks, 2) confidentiality protection of the communication, 3) intrusion detection.

A. Quality of Service (QoS)

At any moment in time, a set of tasks is active, representing the current mode M of the system. Each task t_i is composed of two parts. The first part is mandatory to be executed, if t_i is released, with a fixed execution time $\mathcal{E}_i^m$. The second part is optional, and its execution time e_i^o can be in range $[0, \mathcal{E}_i^o]$. The mandatory part provides constant QoS $\mathcal{Q}_i^m$, and the optional part delivers additional QoS depending on the allocated execution time e_i^o. The QoS produced by t_i is then

$$QoS_{t_i} = \mathcal{Q}_i^m + f_i(e_i^o), \tag{1}$$

where $f_i(e_i^o)$ is a user defined non-decreasing reward function. We define the total QoS provided by the system in mode M as

$$QoS_M = \sum_{t_i \in M} QoS_{t_i}. \tag{2}$$

B. Confidentiality

Confidentiality is achieved by message encryption/decryption. In this paper, we assume the use of iterated block ciphers (IBCs), one type of symmetric-key cryptography, that arguably are the most widely adopted message encryption algorithms. In order to be able to adapt the system to a given context, we need to find an efficient trade-off between protection strength and encryption/decryption time.

Quantifying the strength of different IBCs is still an open problem. We propose a solution in which the protection strength of an IBC is quantified as the logarithm of the number of plaintext-cipher pairs that are required to break the algorithm by the best known cryptanalysis attack. We have conducted extensive studies on four representative IBCs, i.e. RC5, RC6, Rijndael and Blowfish [11], [12], [13], [14], [15]. In this paper, we selected the RC6 algorithm because of the sound protection strength, high flexibility, and fast encryption speed on embedded processors [16] that we observed in our studies. Seven variants of RC6 that are assumed to be used in the system are listed in Table I. The first row presents the referred RC6 variants.

978-3-9815370-0-0/DATE13/©2013 EDAA

RC6 variant	4	6	8	10	12	14	16
Strength	29	45	61	78	94	110	118
Time	17	26	35	44	52	61	70

TABLE I
THE STRENGTH AND ENCRYPTION TIME OF SELECTED CIPHERS

They correspond to increasing number of encryption/decryption rounds employed (4, 6, 8, etc). The second and third rows list the protection strength and execution time (cycles/16Bytes) of the corresponding variant, respectively. The encryption and decryption time are very similar for RC6 according to [17], so we assume them equal for simplicity of the presentation. Note that our proposed design framework is general enough to be applied to other quantification methods and cryptographic algorithms too, if similar strength/time relations can be derived.

Assuming that the selected RC6 variant to encrypt/decrypt message m_{ij} generated by task t_i is $\mathcal{C}_{m_{ij}}$, then the quality of confidentiality (QoC) protection for m_{ij} is quantified as

$$QoC_{m_{ij}} = \frac{e^{\text{Strength}(\mathcal{C}_{m_{ij}})/MAX} - 1}{e - 1} * l_{ij} * w_{ij}, \tag{3}$$

where

$$l_{ij} = \lceil \frac{m_{ij} \text{ length in bits}}{\text{block length in bits}} \rceil$$

is the length (in number of blocks) of m_{ij}, and w_{ij} is the importance weight of m_{ij}. MAX is the highest protection strength value provided in the system, i.e. 118 in Table I. The QoC delivered by the system in a mode M is:

$$QoC_M = \frac{\sum_{t_i \in M} \sum_{m_{ij} \in \mathcal{L}_i} QoC_{m_{ij}}}{\sum_{t_i \in M} \sum_{m_{ij} \in \mathcal{L}_i} l_{ij} * w_{ij}}, \tag{4}$$

where $\mathcal{L}_i$ is the set of all messages over which task t_i interacts with the environment.

C. Intrusion detection

Intrusion detection (ID) is realized as a host-based independent task [18]. It monitors the system calls and the changes to system status, e.g. variables and log records, for potential risks. It has a constant execution time $\mathcal{E}^{ID}$ for examining all necessary information, but with variable periods p^{ID} depending on the available resources and on the estimated level of threats. The intrusion detection accuracy ($0 < IDA \leq 1$) of mode M is calculated based on the period p^{ID} as follows,

$$IDA = \frac{\mathcal{P}^{ID}_{min}}{p^{ID}}, \tag{5}$$

where $\mathcal{P}^{ID}_{min}$ is the shortest period that the ID task needs to adopt in case of the highest threat level.

III. System Model

The hardware platform considered in this paper has a central processor that handles all computations and a communication module via which it communicates (by wire or wireless) with other peers or service centers. The set of active tasks that is running in the system is dynamically changing at run-time. The set of active tasks, at a certain time, defines the current mode $M \in \mathcal{M}$ (similar to the definition in [19]). At each task arrival or task termination, the system switches to a new mode. The whole set of tasks $\mathcal{T} = \{t_1, t_2, ..., t_n\}$ that might occur in the system is known, and referred to as the *root* mode M_r. The complete set of modes is the power set $\mathcal{M} = 2^{\mathcal{T}}$ having cardinality of $|\mathcal{M}| = 2^{|\mathcal{T}|}$. Certain modes are excluded due to functionality constraints. We shall refer to the modes that can occur at run-time as *functional* modes $\mathcal{M}^{func}$. Mode M is called a *supermode* of M', if $M, M' \in \mathcal{M}$ and $M' \subset M$. Similarly, M' is a *submode* of M. The set of supermodes and submodes of M are denoted as $\overline{\mathcal{M}}(M)$ and $\underline{\mathcal{M}}(M)$ respectively.

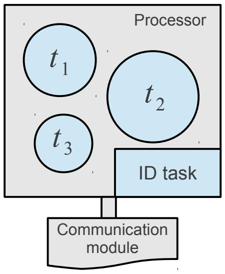

Fig. 1. An illustrative system

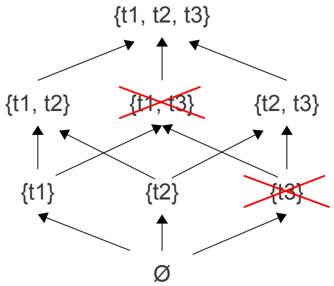

Fig. 2. The hasse diagram of modes for tasks of Fig. 1

Task	$\mathcal{P}$	$\mathcal{E}^m$	$\mathcal{E}^o$	$\mathcal{L}$	$\mathcal{Q}^m_i$	Reward function
t_1	1500	600	800	$\{m_{11}\}$	5	$\sqrt{e^o_1}/50$
t_2	1300	300	700	$\{m_{21}\}$	7	$0.015 * e^o_2$
t_3	2100	500	900	$\emptyset$	3	$(e^o_3)^2/20000$

TABLE II
TASK ATTRIBUTES

The tasks are preemptable and independent from each other. A task t_i is composed of a mandatory and an optional part, and is associated with several attributes ($\mathcal{P}_i$, $\mathcal{E}^m_i$, $\mathcal{E}^o_i$, $\mathcal{L}_i$, $\mathcal{Q}^m_i$, f_i). $\mathcal{P}_i$ is the period and deadline of t_i. $\mathcal{E}^m_i$ is the execution time of the mandatory part. $\mathcal{E}^o_i$ is the maximal amount of time for the optional execution. $\mathcal{L}_i$ is the set containing all messages by which task t_i interacts with the environment. Each message m_{ij} is characterized by its length l_{ij} (in number of blocks) and a weight w_{ij} representing its confidentiality importance. In addition, for each message m_{ij}, a minimum level of required confidentiality $QoC^{min}_{m_{ij}}$ is indicated. $\mathcal{Q}^m_i$ is the constant QoS value from the mandatory execution of t_i. f_i is the reward function as introduced in Section II-A, expressing the amount of additional QoS produced with increasing the execution time e^o_i of the optional part. The execution time and minimal period of the ID task is given as $\mathcal{E}^{ID}$ and $\mathcal{P}^{ID}_{min}$. We consider the encryption/decryption operations for the messages m_{ij} as an additional load for task t_i. Assuming that the system is scheduled by EDF, the schedulability constraint in a certain mode M is the following

$$U_M = \frac{\mathcal{E}^{ID}}{p^{ID}} + \sum_{t_i \in M} \left(\frac{\mathcal{E}^m_i + e^o_i}{\mathcal{P}_i} + \sum_{m_{ij} \in \mathcal{L}_i} \frac{\mathcal{E}^E_{ij} * l_{ij}}{\mathcal{P}_i} \right) \leq 1, \tag{6}$$

where $\mathcal{E}^E_{ij}$ is the encryption/decryption time (per block) of the chosen RC6 variant on the output/input message m_{ij}.

An illustrative system is depicted in Fig. 1. Three application tasks may occur in the system, and the corresponding partial order capturing the relations of all potential modes is shown as a Hasse diagram in Fig. 2. Modes $M^{13} = \{t_1, t_3\}$ and $M^3 = \{t_3\}$ are functionally excluded, and the rest are functional modes. Table II presents the attributes of the three application tasks. We assume that only two messages, m_{11} and m_{21}, are generated in the system by t_1 and t_2, respectively, and to be sent over the communication module. They have $l_{11} = 2$, $l_{21} = 3$, $w_{11} = 0.7$, and $w_{21} = 0.3$.

IV. Motivational Example

Now let us consider the situation when the system in Fig. 1 runs in mode $M^{12} = \{t_1, t_2\}$, in which task t_3 is not active. The ID task has execution time 390, and its period delivering maximal security protection ($IDA = 1$) is 1200 time units.

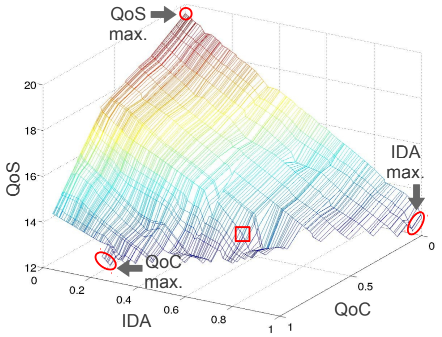

Fig. 3. The Pareto surface of M^{12}

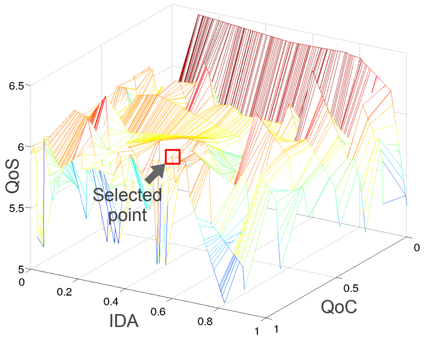

Fig. 4. Derived surface for M^1 from M^{12}

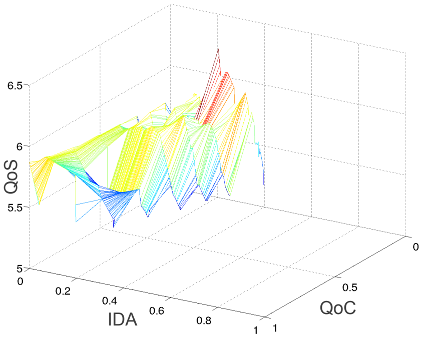

Fig. 5. Derived surface for M^1 from M^{123}

With the given attributes in Table II, the processor utilization of the mandatory parts of task t_1, t_2 is $\mathcal{E}_1^m/\mathcal{P}_1+\mathcal{E}_2^m/\mathcal{P}_2 = 0.63$. So we have 37% free processor load without doing any optional task executions or security protections. If only considering the maximization of the generated QoS, the optimal resource allocation is to give $e_1^o = 30$ and $e_2^o = 454$ time units respectively as optional execution times of tasks t_1 and t_2. Then we obtain a $QoS = 19.58$. However, there is absolutely no security protection in this case, since the total processor load is dedicated to the application tasks and no message encryption or ID operation is performed.

Alternatively, we can achieve the highest possible confidentiality protection by encrypting both m_1 and m_2 with the RC6 variant that provides the highest protection level in Table I leading to $QoC = 1$. In this case, 140 and 210 time units are spent on encrypting m_1 and m_2, respectively. By this, we have already utilized $U = 88.6\%$ of the processor. Consequently, we can only perform very limited optional application executions or carry out the ID task with a long period leading to poor QoS and IDA. Similarly, if the highest ID level ($IDA = 1$) is demanded, we need to carry out the ID task with period $\mathcal{P}_{min}^{ID}$, i.e. 1200 time units. By this, the processor is almost fully utilized ($U = 95.6\%$) leaving no resources for either QoS or QoC improvements.

The three scenarios outlined above represent cases in which either security is completely ignored, or one security service is provided at the very highest possible level, leaving an small fraction of resources to the actual application. In reality, a reasonable balance between the amounts of resources assigned to application tasks, confidentiality protection and intrusion detection should be achieved. What we are facing is a multi-objective optimization problem along these three dimensions. The solutions to this problem is captured as a Pareto surface[1]. The Pareto surface corresponding to mode $M^{12} = \{t_1, t_2\}$ is depicted in Fig. 3. The points representing the three extreme situations outlined above are marked with circles. Note that the second scenario is related to a set of points that correspond to different allocations of the small free processor load (11.4%).

Based on the current status of the system and threat level from the environment, a security monitor determines the required security level as QoC^R and IDA^R. When entering a new mode, the operation point is determined which corresponds to the values e_1^o and e_2^o that produce the highest QoS under the given circumstances with respect to the QoC^R and IDA^R constraints. For example, if the current $QoC^R = 0.6$ and $IDA^R = 0.6$, then we can decide on a resource allocation defined by the point $(13.38, 0.61, 0.61)$ (marked in the red square in Fig. 3). In fact, 13.38 is the maximal achievable QoS satisfying both security constraints. Assuming that Pareto surfaces are available at run-time for all functional modes, the above procedure will allow determining the optimal operation points based on the actual security requirements on-line.

Producing the Pareto surfaces at run-time is infeasible due to the large overhead. Instead, they can be generated off-line, and utilized on-line. However, there are two problems that have to be considered:

1) Due to the huge complexity of the optimization problem, and the potentially very large number of functional modes (growing exponentially with the number of tasks), it might be impossible, at design time, to generate Pareto surfaces for all functional modes.
2) The available memory space at run-time limits the number of Pareto surfaces that can be stored.

Thus, Pareto surfaces can be stored only for a limited amount of functional modes. If the system enters a mode for which there is no available Pareto surface, the operation point has to be extrapolated at run-time based on the available surfaces. For example, let us assume that the system enters mode $M^1 = \{t_1\}$, but no Pareto surface is available for this mode. The system then tries to use the Pareto surfaces of its implemented supermodes $M' \in (\overline{\mathcal{M}}(M^1) \cap \mathcal{M}^{impl})$ in order to find an good operation point for M^1 efficiently.

Let us assume that there exist available Pareto surfaces for modes M^{12} and M^{123}. As a first step, the run-time system will construct the surface that can be derived from M^{12} and M^{123} by ignoring the component produced due to tasks in $M^{12} \backslash M^1 = \{t_2\}$ and $M^{123} \backslash M^1 = \{t_2, t_3\}$ respectively. Fig. 4 and 5 depict the obtained surfaces for M^1 in these two cases. The surface obtained from M^{12} (Fig. 4) provides higher quality points for M^1 which in this case is easily observable from the two figures. Thus, it will be used to find a first approximation of the operation point for the new mode. For example, if the current $QoC^R = 0.85$ and $IDA^R = 0.6$, then we can select the solution producing the point $(6.16, 1, 0.6)$ (marked in Fig. 4) on the derived surface, since it delivers the highest QoS, and satisfies both security constraints. This point, however, since it is derived from a surface produced for M^{12} by ignoring task t_2 and corresponding resource allocation, is sub-optimal. The current processor utilization is $U = 73.3\%$. By allocating, in the next step, the full amount of available processor load to t_1, we extend e_1^o to 400.6 time units, and improve the QoS of the system to $QoS = 7.83$ accordingly. The on-line and off-line phases of the above procedure will be discussed in Section VI.

[1] We refer to a finite set of the solutions as the Pareto surface in the rest of this paper.

V. Problem Formulation

At design time, we formulate the design process for each particular mode as a multi-objective optimization problem. There exist $O(2^{|\mathcal{T}|})$ instances of the problem to be solved, which becomes infeasible as $|\mathcal{T}|$ grows. Thus, we want to explore only a subset of $\mathcal{M}$, depending on the available optimization time and run-time memory needed to store the generated solutions. *At run-time*, the system needs to adapt to a new mode, meeting the current security requirements such that the provided QoS is as high as possible.

A. Design Time

For a given mode, we have considered three objectives, i.e. QoS (Eq. 2), QoC (Eq. 4) and IDA (Eq. 5). The set of optimal solutions can be produced as a Pareto surface. An implementation $\mathcal{I}_M$ of a mode M consists of a set of solutions on the Pareto surface, and is stored in memory. An individual solution $s \in \mathcal{I}_M$ is characterized by a certain delivered QoS (as result of the allocated optional execution cycles to application tasks), QoC (as result of the encryption/decryption cycles allocated), and IDA (due to the assigned period of the ID task).

Before going further, let us introduce a relation between two surfaces $\mathcal{I}_i$ and $\mathcal{I}_j$. It is based on their hypervolumes [20] $\mathcal{H}(\mathcal{I}_i)$ and $\mathcal{H}(\mathcal{I}_j)$ calculated using the method developed in [21]. If $\mathcal{H}(\mathcal{I}_i) > \mathcal{H}(\mathcal{I}_j)$, then $\mathcal{I}_i$ outperforms $\mathcal{I}_j$. If $M \in \mathcal{M}^{impl}$, where $\mathcal{M}^{impl}$ is the set of implemented modes (for which a Pareto surface has been generated and stored at design time), then the hypervolume $\mathcal{H}_M$ of M is calculated as $\mathcal{H}(\mathcal{I}_M)$ from $\mathcal{I}_M$. If $M \notin \mathcal{M}^{impl}$, then $\mathcal{H}_M$ is calculated as $\mathcal{H}(\mathcal{I}_M^{M'})$, where $\mathcal{I}_M^{M'}$ is the derived solution surface for M from $M' \in (\overline{\mathcal{M}}(M) \cap \mathcal{M}^{impl})$ providing the highest hypervolume after removing the resources occupied by redundant tasks $M' \backslash M$.

We divide our design phase optimization into two subproblems. The first subproblem is to find the Pareto surface $\mathcal{I}_M$ for one given mode M. The second subproblem is to explore the Hasse diagram efficiently to obtain the subset $\mathcal{M}^{impl} \subset \mathcal{M}$, and use the solution to the first subproblem to obtain the Pareto surfaces for all modes $M \in \mathcal{M}^{impl}$.

Subproblem I: We take the following parameters as input,

- the active tasks in mode M,
- attributes ($\mathcal{P}_i$, $\mathcal{E}_i^m$, $\mathcal{E}_i^o$, $\mathcal{L}_i$, $\mathcal{Q}_i^m$, f_i) for all $t_i \in M$, as well as l_{ij}, w_{ij}, and $Q_{m_{ij}}^{min}$ corresponding to all messages $m_{ij} \in \mathcal{L}_i$,
- execution time $\mathcal{E}^{ID}$ and period $\mathcal{P}_{min}^{ID}$ for the ID task,
- a designer provided strength/time tradeoff table of selected cryptographic algorithms as Table I.

The desired output is the Pareto surface corresponding to the optimization objectives QoS, QoC, and IDA. Schedulability, as described in Eq. 6, must be satisfied in all $M \in \mathcal{M}^{impl}$.

Subproblem II: The inputs to this subproblem are

- the set of all functional modes $\mathcal{M}^{func}$,
- the top functional modes $\mathcal{M}_{\uparrow}^{func}$ that do not have any functional supermodes, and thus must be implemented.

Our desired output is implementations of selected modes $\{M \in \mathcal{M}^{impl} : \mathcal{M}^{impl} \subset \mathcal{M}^{func}\}$. Each functional mode $M \in \mathcal{M}^{func}$ must have its own implementation or an implementation of at least one of its supermodes, i.e.

$$\forall M \in \mathcal{M}^{func}, (\{M\} \cup \overline{\mathcal{M}}(M)) \cap \mathcal{M}^{impl} \neq \emptyset. \quad (7)$$

The design objective for this problem is to implement a set of modes $\mathcal{M}^{impl}$ such that the total hypervolume value H of all functional modes is maximized, i.e.

$$\max H = \sum_{M \in \mathcal{M}^{func}} \mathcal{H}_M, \quad (8)$$

where

$$\mathcal{H}_M = \begin{cases} \mathcal{H}(\mathcal{I}_M), & \text{if } M \in \mathcal{M}^{impl} \\ \mathcal{H}(\mathcal{I}_M^{M'}), & \text{otherwise.} \end{cases}$$

The number of modes $|\mathcal{M}^{impl}|$ that can be explored and implemented is limited by the available optimization time and run-time memory size.

B. Run time

At run-time, the system has to find the appropriate operation solution according to the current mode and security requirements. The system takes the following as input parameters for the on-line procedure,

- available implementations $\{\mathcal{I}_M : M \in \mathcal{M}^{impl}\}$ that are stored in memory,
- QoC and IDA requirements, QoC^R and IDA^R, received from the run-time security monitor.

The objective is to find a solution s, based on the available implementations, that delivers the highest QoS value while satisfying all constraints, i.e. $QoC_s \geq QoC^R$, $IDA_s \geq IDA^R$, and $U_s \leq 1$.

VI. Proposed Techniques

A. Design Time

Subproblem I: We cannot afford finding the optimal Pareto surface for a given mode, due to the huge computational complexity. Therefore, we use the genetic algorithm based multi-objective optimization framework NSGA-II [22] for generating the Pareto surface $\mathcal{I}_M$ for mode M. The parameters, e.g. the population size, number of generations, and mutation rates, for NSGA-II are fine-tuned for different problem sizes. The population size in NSGA-II gives the maximal number $|\mathcal{I}_M|$ of valid solutions to be saved in memory for one obtained surface.

Subproblem II: For solving the second subproblem defined in the previous section, we traverse the modes in the Hasse diagram using an improvement factor λ to limit the depth of exploration. Before going further, let us introduce the definition of immediate submode of M as

$$\mathcal{M}^-(M) = \{M' \in \underline{\mathcal{M}}(M) : |M| - 1 = |M'|\}. \quad (9)$$

For example, $M^1 = \{t_1\}$ is a immediate submode of $M^{12} = \{t_1, t_2\}$, but not of $M^{123} = \{t_1, t_2, t_3\}$; thus, $M^1 \in \mathcal{M}^-(M^{12})$.

The optimization procedure is outlined in Algorithm 1. $\mathcal{M}^{wait}$, $\mathcal{M}^{impl}$, and $\mathcal{M}^{skip}$ keep the modes that are waiting to be processed, have been implemented, and ignored respectively. The top functional modes $\mathcal{M}_{\uparrow}^{func}$ must be implemented to ensure correctness (see Eq. 7). So we start from these modes when generating implementations (Line 2-3). After implementing a mode M_t, we add its immediate submodes $\mathcal{M}^-(M_t)$ into $\mathcal{M}^{wait}$ (Line 4).

While the list $\mathcal{M}^{wait}$ is not empty, we take out and remove the first mode M' from $\mathcal{M}^{wait}$ (Line 5-6). If M' is a functional mode, and has not been implemented or skipped, then we generate the derived surfaces $\mathcal{I}_{M'}^{M''}$ (see Section V-A) for all $M'' \in (\overline{\mathcal{M}}(M') \cap \mathcal{M}^{impl})$, and keep the one among them with the maximal hypervolume $\mathcal{H}_{MAX}^D$ (Line 7-10). Then we generate the actual surface $\mathcal{I}_{M'}$ for M', and compare the hypervolumes $\mathcal{H}(\mathcal{I}_{M'})$ and $\mathcal{H}_{MAX}^D$. This comparison gives us

Algorithm 1 Heuristic for offline design phase

```
 1: Initialize list $\mathcal{M}^{wait} := empty$, sets $\mathcal{M}^{impl}, \mathcal{M}^{skip} \leftarrow \emptyset$
 2: for each $M_t \in \mathcal{M}_{\uparrow}^{func}$ do
 3:   Generate Pareto surf. $\mathcal{I}_{M_t}$, and $\mathcal{M}^{impl} \leftarrow \mathcal{M}^{impl} \cup \{M_t\}$
 4:   Add each $M \in \mathcal{M}^{-}(M_t)$ into the list $\mathcal{M}^{wait}$
 5:   while $\mathcal{M}^{wait} \neq empty$ do
 6:     Take out the first element $M'$ from $\mathcal{M}^{wait}$
 7:     if $M' \in \mathcal{M}^{func} \backslash (\mathcal{M}^{impl} \cup \underline{\mathcal{M}}^{skip})$ then
 8:       for each $\mathcal{I}_{M''}$ of $M'' \in (\overline{\mathcal{M}}(M') \cap \mathcal{M}^{impl})$ do
 9:         Calculate $\mathcal{H}(\mathcal{I}_{M'}^{M''})$ of derived surface for $M'$
10:         $\mathcal{H}_{MAX}^{D} = MAX(\mathcal{H}(\mathcal{I}_{M'}^{M''}), \mathcal{H}_{MAX}^{D})$
11:       Generate Pareto surface $\mathcal{I}_{M'}$ for $M'$
12:       if $\mathcal{H}(\mathcal{I}_{M'}) \geq \mathcal{H}_{MAX}^{D} * (1 + \lambda)$ then
13:         $\mathcal{M}^{impl} \leftarrow \mathcal{M}^{impl} \cup \{M'\}$
14:         Add the modes in $\mathcal{M}^{-}(M') \backslash (\mathcal{M}^{impl} \cup \mathcal{M}^{skip})$ to the
            end of $\mathcal{M}^{wait}$
15:       else
16:         $\mathcal{M}^{skip} \leftarrow \mathcal{M}^{skip} \cup \{M'\} \cup \underline{\mathcal{M}}(M')$
17:         Remove the modes in $\underline{\mathcal{M}}(M')$ from $\mathcal{M}^{wait}$
```

an indication about how much we gain by storing an actual implementation for M' as opposed to using a derived surface at run-time. If $\mathcal{H}(\mathcal{I}_{M'}) \geq \mathcal{H}_{MAX}^{D} * (1+\lambda)$, we consider the gain sufficiently large for storing $\mathcal{I}_{M'}$, and the immediate submodes of M', $\mathcal{M}^{-}(M')$, are added into $\mathcal{M}^{wait}$ to be further processed (Line 11-14). Otherwise, no implementation of M' will be stored, and the search along its descendants is stopped (Line 16-17). After the algorithm terminates, the modes that have been implemented are kept in $\mathcal{M}^{impl}$.

The designer can traded-off solution quality with optimization time and needed memory space by tuning the improvement factor $\lambda (\lambda >= 0)$ in Line 12. With smaller value of λ, the numbers of explored and stored implementations increase.

B. Run time

After the design phase, the system has a set of implementations $\{\mathcal{I}_M : M \in \mathcal{M}^{impl}\}$ stored in memory. At run-time, the system changes modes dynamically, and must adapt to the security requirements corresponding to the current state of the environment and threat level. Two situations may occur when the system is required to switch to mode M: an implementation of M is available ($M \in \mathcal{M}^{impl}$), and is not available ($M \notin \mathcal{M}^{impl}$).

1) $M \in \mathcal{M}^{impl}$: In this case, M was implemented, and a set of solutions $\mathcal{I}_M$ were stored in memory. Knowing the security requirements QoC^R and IDA^R, the system selects the operation point $s \in \mathcal{I}_M$, which delivers the highest QoS_s, and satisfies $QoC_s \geq QoC^R$ and $IDA_s \geq IDA^R$. It might be possible that, under the given amount of resources, there does not exist any schedulable solution satisfying QoC^R and IDA^R, even if the optional cycles for all application tasks are zero (there is no $s \in \mathcal{I}_M$ that satisfies the security constraints). In this case, the run-time monitor will be notified and emergency measures must be taken.

2) $M \notin \mathcal{M}^{impl}$: In this case, we first select the supermode $M' \in (\overline{\mathcal{M}}(M) \cap \mathcal{M}^{impl})$ that results in the best hypervolume of the derived surface $\mathcal{H}(\mathcal{I}_M^{M'})$ for M, i.e.

$$\mathcal{H}(\mathcal{I}_M^{M'}) \geq \mathcal{H}(\mathcal{I}_M^{M''}), \forall M'' \in (\overline{\mathcal{M}}(M) \cap \mathcal{M}^{impl}). \qquad (10)$$

As a second step, an appropriate solution point $s \in \mathcal{I}_M^{M'}$ has to be selected. It should be reminded that all solutions on $\mathcal{I}_M^{M'}$ are fixed such that the resources assigned to tasks $t_i \in M' \backslash M$ are not used. Thus, in order to determine the best resource allocation for the new mode M, the unused resources must be redistributed. We have to consider two cases depending on, if according to $\mathcal{I}_M^{M'}$, the current security requirements QoC^R and IDA^R can be satisfied or not.

If QoC^R and IDA^R can be satisfied in $\mathcal{I}_M^{M'}$: we first select the solution $s \in \mathcal{I}_M^{M'}$ as in case 1) discussed above. Then we allocate the free processor bandwidth $U_s^{free} = 1 - U_s$ among the tasks $t_i \in M$ in the priority order of the derivative $f_i'(e_i^o)$ of the reward function (refer to Eq. 1).

If QoC^R and IDA^R cannot be satisfied in $\mathcal{I}_M^{M'}$: we start from the solution $s \in \mathcal{I}_M^{M'}$ that has the closest Euclidean distance between (QoC_s, IDA_s) and (QoC^R, IDA^R). Then we check whether IDA^R can be satisfied by allocating bandwidth from the unused processor load $U_s^{free} = 1 - U_s$ in order to achieve the required period of the ID task $p_s^{ID} = \mathcal{P}_{min}^{ID} / IDA^R$. If this is possible, we fix the period of the ID task accordingly and continue with a similar procedure regarding the requirement QoC^R. If QoC^R is currently not satisfied, we distribute the now available processor bandwidth to provide higher encryption level to the messages corresponding to tasks $t_i \in M$. The available bandwidth is assigned for increasing the encryption strength with priority to the messages with higher w_{ij} on confidentiality. If bandwidth is still available, it will be distributed for increasing QoS, as indicated further above. In case the security requirements QoC^R and/or IDA^R cannot be satisfied, the run-time monitor will be signaled.

VII. Experimental Results

We have conducted experiments on a Linux machine having a four-core Intel Xeon CPU with 2.66GHz frequency and 8GB RAM. We have studied five different problem sizes having task numbers $|\mathcal{T}| = 3, 4, 6, 8,$ and 10 respectively. On each problem size, 20 test applications were generated randomly. The ID task execution time $\mathcal{E}^{ID}$ and minimal period $\mathcal{P}_{min}^{ID}$ were set as 390 and 1200 time units, respectively.

The mandatory execution time $\mathcal{E}_i^m$ and maximal optional execution time $\mathcal{E}_i^o$ of task t_i were randomly generated from the intervals $[500, 1400]$ and $[100, 800]$, respectively. Each task was associated with a set of messages, each of which had $l_{ij} \in \{1, 2, ..., 7\}$ blocks and a random weight $w_{ij} \in [0, 1]$. The reward functions were assumed to be of the form $f_i(e_i^o) = \alpha_i * e_i^o + \beta_i * \sqrt{e_i^o}$.

A. Design Time

We first evaluated our *design time* optimization technique. Five different improvement factors λ were studied in the experiments, i.e. 0, 0.2, 0.4, 0.8, and 1.6, to evaluate the trade-off between optimization time and performance. If $\lambda = 0$, the algorithm exhaustively explored all functional modes, and generated a Pareto surface for each mode.

For comparison, we chose the baseline as the situation when only the top functional modes were implemented, i.e. $\mathcal{M}^{impl} = \mathcal{M}_{\uparrow}^{func}$. Under this scenario, the total hypervolume $H_{\mathcal{M}_{\uparrow}^{func}}$ of the system was actually the minimum that could be obtained without violating the coverage constraint in Eq. 7. For each λ on a given test application, we computed the achieved performance improvement (*PI*) as follows,

$$PI = \frac{H_{\mathcal{M}^{impl}} - H_{\mathcal{M}_{\uparrow}^{func}}}{H_{\mathcal{M}_{\uparrow}^{func}}}, \qquad (11)$$

where $H_{\mathcal{M}^{impl}}$ was the total hypervolume of all functional modes under the situation when only $\mathcal{M}^{impl}$ were implemented (this is the value of the cost function in Eq. 8).

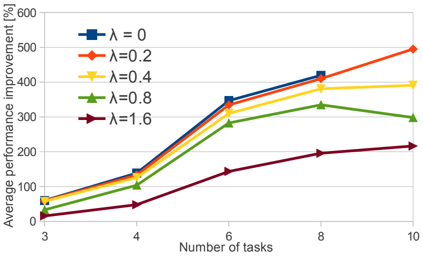

Fig. 6. Performance improv. of off-line phase

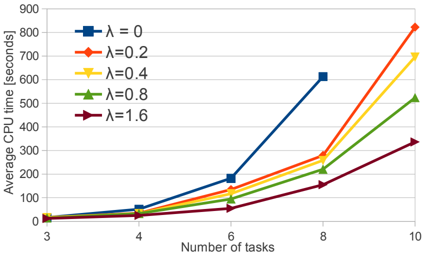

Fig. 7. Optimization time of off-line phase

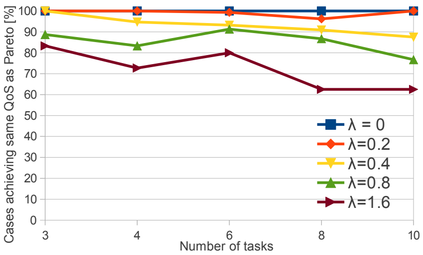

Fig. 8. Performance of on-line solution selection

Fig. 6 shows the obtained results relative to the baseline solution $\mathcal{M}^{impl} = \mathcal{M}^{func}_{\uparrow}$. The horizontal and vertical axes indicate the number of tasks $|\mathcal{T}|$ and average *PI* in each experimental setup, respectively. It can be observed that bigger improvements can be achieved with smaller λ, since more modes are explored and implemented. Therefore, it is important to implement more modes than the top functional modes in the system design stage, especially in large system designs. The overall trend also shows that, with a fixed λ, better improvements are obtained on bigger problem sizes. Fig. 7 shows the average optimization time (in seconds) for each λ and problem size. As can be noticed, no result was shown for $|\mathcal{T}| = 10$ and $\lambda = 0$ in Fig. 6 and 7 because of the long runtimes. These two figures together demonstrate the trade-off between optimization time and result quality for different problem sizes. In large problem sizes, it is infeasible to explore a very large portion of the Hasse diagram as the optimization time grows aggressively for small λ. However, good performance improvement can still be achieved using a larger λ.

B. Run time

We simulated the run-time operation phase optimization on the same test applications having 3-10 tasks as above considering the same five values for λ. For each application and λ, we randomly generated 20 modes to simulate the dynamic changes of the active task set. Each mode M_i was assigned with random QoC^R_i and IDA^R_i constraints to emulate the uncertain security requirements received on-line. We performed our proposed run-time solution selection technique on each of the 20 modes.

For evaluation, we counted the number of cases where the QoS value delivered by the obtained solution reached the ideal case according to QoC^R and IDA^R. In other words, the obtained solution delivered the same QoS as if the Pareto surface was available for M_i. Fig. 8 shows the percentage of the cases in which the ideal QoSs were obtained. Meanwhile, for those solutions that differ from the ideal ones, the average achieved QoSs of all applications and value of λ were above 90.1% of the achievable value if all implementations are available.

VIII. Conclusion

In this paper, we have presented a novel security aware design optimization framework for modern embedded systems that have dynamic task sets. Confidentiality protection and intrusion detection are considered in the design process together with QoS optimization. In order to master complexity issues, we propose efficient heuristics to be used in the off-line and on-line stages, respectively. Our off-line solution can be tuned by the designer for pursuing better solution quality vs. shorter optimization time and less run-time memory. Experiments have demonstrated the efficiency of the proposed technique.

References

[1] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, “Security in embedded systems: Design challenges,” *ACM trans. on Embedded Computing Systems*, 3:461-491, aug 2004.

[2] K. Patel and S. Parameswaran, “Shield: a software hardware design methodology for security and reliability of mpsocs,” in *Design Automation Conference (DAC)*, 2008.

[3] Z. Shao, C. Xue, Q. Zhuge, M. Qiu, B. Xiao, and E.-M. Sha, “Security protection and checking for embedded system integration against buffer overflow attacks via hardware/software,” *IEEE trans. on Computers*, pp. 443–453, apr. 2006.

[4] C. H. Gebotys, “A table masking countermeasure for low-energy secure embedded systems,” *IEEE trans. on VLSI Systems*, 14:740-753, jul. 2006.

[5] K. Tiri and I. Verbauwhede, “A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation,” in *Des., Autom. and Test in Europe Conf. (DATE)*, 2004, pp. 246–251.

[6] M. Lin, L. Xu, L. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu, “Static security optimization for real-time systems,” *IEEE trans. on Industrial Informatics*, pp. 22–37, feb. 2009.

[7] T. Xie and X. Qin, “Improving security for periodic tasks in embedded systems through scheduling,” *ACM Trans. Embedded Computing Systems*, vol. 6, 2007.

[8] K. Jiang, P. Eles, and Z. Peng, “Co-design techniques for distributed real-time embedded systems with communication security constraints,” in *Des., Autom. and Test in Europe Conf. (DATE)*, 2012, pp. 947–952.

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

[10] W. Jiang, K. Jiang, and Y. Ma, “Resource allocation of security-critical tasks with statistically guaranteed energy constraint,” in *Embedded and Real-Time Computing Systems and Appl. (RTCSA)*, 2012, pp. 330–339.

[11] A. Biryukov and E. Kushilevitz, “Improved cryptanalysis of rc5,” in *Advances in Cryptology (EUROCRYPT)*, 1998, pp. 85–99.

[12] L. Knudsen and W. Meier, “Correlations in rc6 with a reduced number of rounds,” in *Fast Software Encryption*, 2001, pp. 94–108.

[13] W. Zhang, W. Wu, and D. Feng, “New results on impossible differential cryptanalysis of reduced aes,” in *Information Security and Cryptology (ICISC)*, 2007, pp. 239–250.

[14] J. Lu, O. Dunkelman, N. Keller, and J. Kim, “New impossible differential attacks on aes,” in *Prog. in Crypto.*, 2008, pp. 279–293.

[15] S. Vaudenay, “On the weak keys of blowfish,” in *Fast Software Encryption*, 1996, pp. 27–32.

[16] O. Hyncica, P. Kucera, P. Honzik, and P. Fiedler, “Performance evaluation of symmetric cryptography in embedded systems,” in *International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*, 2011, pp. 277–282.

[17] R. L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin, “The rc6 block cipher,” in *First Advanced Encryption Standard (AES) Conference*, 1998.

[18] D.-Y. Yeung and Y. Ding, “Host-based intrusion detection using dynamic and static behavioral models,” *Pattern Recognition*, 36:229-243, 2003.

[19] S. Samii, P. Eles, Z. Peng, and A. Cervin, “Quality-driven synthesis of embedded multi-mode control systems,” in *Design Automation Conference (DAC)*, july 2009, pp. 864–869.

[20] E. Zitzler and L. Thiele, “Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study,” in *Conference on Parallel Problem Solving from Nature (PPSN V)*, 1998, pp. 292–301.

[21] E. Zitzler, (2001) Hypervolume metric calculation: ftp://ftp.tik.ee.ethz.ch/pub/people/zitzler/hypervol.c.

[22] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *Evolutionary Computation*, 6:182-197, apr 2002.