An Improved Security-Cognizant Scheduling Model

Fatima Raadia
Wayne State University
fatima.fr@wayne.edu

Nathan Fisher
Wayne State University
fishern@wayne.edu

Thidapat Chantem
Virginia Tech
tchantem@vt.edu

Sanjoy Baruah
Washington University in St. Louis
baruah@wustl.edu

***Abstract*—Security is increasingly a primary concern in the design of safety-critical embedded systems, yet balancing it with timing constraints is challenging due to limited computing resources. The Multi-Phase Secure (MPS) Sporadic Task Model, proposed in an ISORC-2023 paper, addressed this by balancing overhead from security mechanisms (e.g., trusted-execution environments) with real-time scheduling constraints. However, this model assumed a somewhat pessimistic view of the overhead involved in switching between security mechanisms, often overestimating the necessity of these switches. This paper refines the MPS Sporadic Task Model to more accurately assess when switching security mechanisms is unnecessary, thereby avoiding undue overhead. Our refined model demonstrates a substantial improvement in the schedulability ratio when the utilization of the system approaches one (approximately 15% improvement) for randomly-generated security-aware task systems.**

***Index Terms*—Embedded System Security; Graph Transformation; Limited-Preemption Scheduling; Earliest-Deadline First.**

I. Introduction

A paper that was presented at ISORC last year [1] had proposed a new task model called the *Multi-Phase Secure* or MPS task model, for modeling recurrent real-time workloads in a manner that is cognizant of security considerations. The need for such security-cognizant models is evident: security is now a primary concern in many real-time systems, e.g., [2], [3], [4], [5], [6], [7], [8], as electronic devices now permeate our daily lives and become more and more interconnected. However, implementing a given security measure usually comes with additional resource requirements (e.g., computation) or may restrict application behaviors (e.g., requiring non-preemptable computation or isolation). For example, control flow integrity (CFI) checks may be needed to ensure correct program execution. However, such checks, which require CPU time in addition to normal code execution, must be carried out at specific time points (e.g., after branching) and allowing for preemption may result in an arbitrary computation being performed but not detected. As another example, a task that is responsible for taking sensor readings may need to execute in isolation in order to ensure that another task cannot deduce when an event of interest occurs [8].

Since implementing security measures requires some of the same resources that the real-time tasks need to advance their execution, a co-design approach that explicitly considers security cost/requirements along with real-time requirements is potentially more effective at managing limited computational resources. For instance, *trusted execution environments* (TEEs) provide isolation of code and data in hardware at the expense of setup and teardown costs (in the order of microseconds for Arm Cortex-M [9] and hundreds of microseconds or even tens of millseconds for Arm Cortex-A [10]). A scheduling approach that does not consider this specific security-driven overhead may elect to switch between the secure world (i.e., executing in TEE) and the normal world (no TEE) indiscriminately. This may result in an excessive amount of overhead and cause deadline misses. A security-cognizant scheduler, on the other hand, would make judicious decisions based on both security and real-time requirements, e.g., by bundling up multiple TEE executions and executing them one immediately after the other so as to have to pay for setup and teardown cost only once [10]. A prior ISORC paper [11] had proposed and developed algorithms that are able to provide provable correctness of both the timing and some security properties. The design philosophy underpinning these algorithms was articulated in [11] as follows: security for safety-critical real-time embedded systems can be achieved by (i) explicitly representing specific security considerations within the same formal frameworks that are currently used for specifying real-time workloads (thereby extending notions of correctness to incorporate both the timing and the security aspects); and (ii) extending previously-developed techniques for achieving provable timing correctness to these models (thus assuring that both timing and security properties are proved correct).

This work. As mentioned earlier, an ISORC-2023 paper [1] had applied the methodology of [11] to the following problem in system design for real-time plus security. Consider computer platforms upon which *multiple different security mechanisms* (such as TEEs, encryption/ decryption co-processors, FPGA-implemented secure computations, etc.) co-exist. Different code sections may require varied security mechanisms based on their specific security needs. It is therefore assumed that the code is broken up into *phases*, with different successive phases needing to use different security mechanisms — the security mechanism used by each phase is specified for the phase. There is a setup/ tear-down overhead cost (for data communication, initialization, etc.), expressed as an execution duration, associated with switching between different security mechanisms. I.e., there is a time overhead associated with switching between the execution of different phases.

The workload model discussed in the above paragraph was formalized in [1] as the *Multi-Phase Secure* (MPS) task model, in which multiple independent recurrent processes of this kind

979-8-3503-7128-4/24/$31.00 ©2024 IEEE

are assumed to execute upon a single shared preemptive processor. Specifically, a model in which each recurrent process is represented using the widely-used 3-parameter sporadic task model [12] was studied in great depth in [1], and an additional generalization was briefly considered that models *conditional* execution within each recurrent process. However, this generalization in [1] to conditional code made some fairly significant conservative approximations (which we discuss in detail in Section II below). The major **contribution** of the current work is to remove these approximations and provide an optimal algorithm (and corresponding analysis and evaluation, both theoretical and empirical) for schedulability-analysis and run-time scheduling of systems of conditional MPS tasks upon preemptive uniprocessor platforms.

Organization. The remainder of this manuscript is organized as follows. Section II provides a concise description and summary of the MPS sporadic task model and associated analysis that was presented in [1]. Section III describes the refinements to the MPS Sporadic task model that permit a reduction in the over-approximation of the security overhead. Section IV provides the steps required to use the model refinements of the previous section to improve the schedulability analysis for the system. Section V discusses an empirical evaluation of the improvements of updated schedulability analysis over randomly-generated task systems (including randomly-generated conditional graphs). Section VII provides some concluding remarks and thoughts on future directions.

II. The Conditional MPS Sporadic Task Model

In this section we briefly summarize the current state of the art concerning the conditional MPS sporadic task model. We will start out in Section II-A discussing a simpler model (referred to in [1] as the "linear" MPS sporadic task model, to distinguish it from the conditional model), and then proceed to the conditional MPS sporadic task model in Section II-B.

A. Linear MPS Sporadic Tasks

As defined in [1], a (linear) MPS sporadic task system Γ comprises multiple independent tasks that are to be scheduled together upon a single shared preemptive processor. Each task is characterized by a relative deadline, a period, and n *phases* denoted $v_1, v_2, \ldots, v_n$. Successive phases are assumed to execute using different security mechanisms (i.e., phase v_i and phase v_{i+1} execute using different security mechanisms for each $i, 1 \leq i < n$). The worst-case execution time (WCET) of phase v_i is denoted by $c(v_i)$, and the sum of the setup and the teardown cost associated with the security mechanism used by phase v_i is denoted by $q(v_i)$. The actual WCET experienced by an individual job of the task during any particular execution depends upon the number of times it gets preempted within each of its phases. That is, let us suppose that during some execution the i'th phase of a particular job of the task gets preempted a total of k_i times for each $i, 1 \leq i \leq n$. Then the total WCET of this job during this execution is

$$\sum_{i=1}^{n} \Big(c(v_i) + (k_i + 1) \times q(v_i) \Big)$$

The approach presented in [1] for scheduling a given a linear MPS sporadic task system Γ comprising multiple independent tasks upon a single preemptive processor is as follows.

1) Each task in Γ is first mapped onto a *limited-preemption task* [13]. As discussed in [13], a limited-preemption task is characterized, alongside the WCET, deadline, and period parameters, by a *chunk size* – the maximum duration for which each job of the task may execute non-preemptively without causing deadline misses. The mapping defined in [1] computes these chunk sizes for each task in Γ.
2) Next, a limited-preemption EDF schedulability analysis algorithm [13], [14] is executed to determine whether the resulting limited-preemption system is schedulable.
3) If so, then during run-time the original linear MPS sporadic task system is scheduled using the limited-preemption EDF scheduling algorithm [13], with chunk-sizes as determined by the mapping in Step 1 above.

This approach was proved [1, Theorems 1 and 2] to be *optimal*[1]: if a given linear MPS sporadic task system Γ can be scheduled to consistently meet all deadlines, then the approach in [1] will ensure that it always meets those deadlines.

B. Conditional MPS Sporadic Tasks

In many event-driven real-time application systems, the code modeled by a task may include conditional constructs ('if-then-else' statements) in which the outcome of evaluating a condition depends upon factors (such as the current state of the system, the values of certain external variables, etc.), which only become known at run-time, and indeed may differ upon different invocations of the task. Tasks exhibiting such conditional execution are frequently modeled [15] as directed acyclic graphs (DAGs) in which the vertices represent execution of straight-line code, with a piece of straight-line code ending in a conditional expression represented by a vertex that has out-degree > 1, as in the example below:

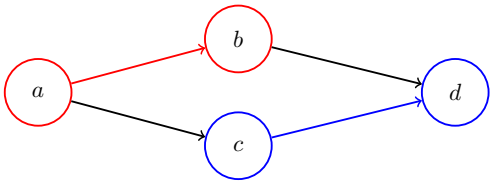

Fig. 1. A task modelled as a directed acyclic graph (DAG)

[1] Under some not-very-restrictive assumptions that require potential preemption points to be statically assigned – please see [1] for a detailed discussion.

In Figure 1, the vertex a denotes a piece of straight-line code that ends with the execution of a conditional expression. Depending upon the outcome of this execution, the straight-line code represented by either the vertex b or the vertex c executes, after which the straight-line code represented by the vertex d executes.

The *conditional MPS sporadic task model* that was introduced in [1] is capable of modeling such conditional execution. A system Γ of conditional MPS sporadic tasks is modeled as comprising multiple independent tasks, in which each task is characterized by a 3-tuple (G, D, T). As in the linear MPS task model, D and T are non-negative integers denoting the relative deadline and period of the task. G is a directed acyclic graph (DAG): $G = (V, E)$ where $E \subset V \times V$. Each node $v_i \in V$ signifies a computational phase that needs a specific security mechanism for execution. A wcet function is defined, denoted as $c : V \to \mathbb{N}$, where $c(v_i)$ represents the Worst Case Execution Time (WCET) of node v_i. Additionally, an overhead function, represented as $q : V \to \mathbb{N}$, specifies the setup or tear-down costs related to the security mechanism used by vertex v_i during execution.

Let us return to our example DAG. Suppose that vertices a and b use the same (red) security mechanism, and vertices c and d also use the same security mechanism (that is blue, and so different from the red one used by a and b). If during some execution of this task vertex b were to execute after vertex a, then there is no teardown or setup cost associated with the transition from a to b but such a cost is incurred in transiting from b to d; if instead vertex c were to execute after vertex a, then a teardown and setup cost is incurred on this transition but none is incurred in transiting from c to d. The analysis in [1], however, makes the conservative assumption that overhead costs are incurred in *every* transition, even when both vertices use the same security mechanism: "...we assume that [...] the tear-down cost associated with the security mechanism of v_j, and the setup cost associated with the security mechanism of v_k, is always paid upon traversing an edge (v_j, v_k)" [1].

Example 1. *Suppose (i) Figure 1 is the entire task; (ii) the chunk size*[2] *for this task is* 5*, and (iii) WCETs are as follows:*

vertex	*a*	*b*	*c*	*d*
WCET	*3*	*2*	*2*	*3*

Suppose, too, that the tear-down + startup overhead associated with going from the red security mechanism to the blue one (or vice versa) are equal. In reality, the maximum execution duration of any individual job of this task is 8 *plus one (tear-down + startup) overhead, but the analysis of [1] would model it as being equal to* 8 + *two (tear-down + startup) overheads.* □

The conservative assumption allows the model of [1] to associate a tear-down plus setup overhead cost of $q(v_i)$ with each vertex v_i, just as in the linear model of Section II-A. In this paper, we eliminate this conservative over-approximation: transitioning from one vertex in the DAG to another that uses the same security mechanism no longer incurs an overhead cost. While this refinement more accurately reflects reality (and as we will see, enables superior analysis), it requires modification to the model specification: the tear-down/ setup overhead associated with a vertex is now context-dependent and cannot be directly specified as a parameter that is only associated with the vertex. In Section III below we describe how we have modified the conditional MPS task model of [1] to enable such refined analysis.

[2]Recall from Section II-A that a task's *chunk size* is the maximum duration for which each job of the task may execute non-preemptively under limited-preemption EDF without causing deadline misses.

III. A More Accurate Model

As we had discussed in Section II above, the algorithms presented in [1] for the schedulability analysis of systems of conditional MPS tasks make the assumption that tear-down and setup costs are always paid upon transitioning between any pair of jobs, regardless of whether both jobs use the same or different security mechanisms. But of course this is needlessly conservative: no tear-down and subsequent setup are performed if both jobs use the same security mechanism. In this section we propose some enhancements to the conditional MPS sporadic task model of [1], in order to be able to not have to be conservative in this manner.

As described in Section II above, a system Γ of conditional MPS sporadic tasks comprises multiple independent tasks, with each task characterized by a 3-tuple (G, D, T) where D and T are the relative deadline and period respectively, while $G = (V, E)$ represents a Directed Acyclic Graph (DAG). Each node $v_i \in V$ signifies a computational phase that uses a specific security mechanism. The WCET function $c : V \to \mathbb{N}$ specifies the worst-case execution times of the individual vertices.

Previously (i.e., in [1]), tear-down and setup overheads were specified by specifying a parameter $q(v_i)$ for each $v_i \in V$. But as we had pointed out in Section II-B, this implicitly implies that the overhead is always paid in transitioning from one vertex to another, even when the vertices use of the same security mechanism. In this paper, we wish to provide more accurate (i.e., less conservative) analysis that only counts the overhead when it is actually incurred; in order to be able to do so, it is necessary that we explicitly specify which security mechanism is used by each vertex. Accordingly, let

$$\mathcal{D} = \{\delta_1, \delta_2, \ldots, \delta_{|\mathcal{D}|}\}$$

denote the distinct security mechanisms that are available on the platform, and let

$$\text{cost} : \mathcal{D} \to \mathbb{N}$$

denote the sum of the overhead costs that are associated with the tearing down and setting up operations for each security mechanism: $\text{cost}(\delta_i)$ is the sum of the tear-down and setup

costs associated with security mechanism δ_i. We additionally specify a function

$$\sigma : V \to \mathcal{D}$$

such that $\sigma(v_i)$ denotes the security mechanism used by vertex v_i. (Hence the function $q(v_i)$ from [1] satisfies the equivalence $q(v_i) \equiv \text{cost}(\sigma(v_i))$: the tear-down and setup overhead cost associated with a vertex v_i is the overhead cost for the security domain $\sigma(v_i)$.)

Summarizing the new model. A conditional MPS sporadic task is now specified by

- a 3-tuple (G, D, T) as in [1], with $G = (V, E)$ a DAG and D and T non-negative integers;
- the wcet function $c : V \to \mathbb{N}$ (again as in [1]);
- the set of security mechanisms $\mathcal{D}$ that are used by the task;
- the function $\text{cost} : \mathcal{D} \to \mathbb{N}$ denoting the tear-down plus setup overhead cost associated with each security mechanism; and
- the function $\sigma : V \to \mathcal{D}$ denoting which security mechanism is used by each vertex.

IV. Improved Analysis for Conditional MPS Sporadic Task Systems

In Secton III above, we had proposed some enhancements to the conditional MPS sporadic task model of [1] that enables more accurate accounting of the overhead costs arising from switching between security mechanisms. In this section, we obtain an algorithm that converts any conditional MPS sporadic task $\tau = (G, D, T)$ specified in this enhanced model to a different conditional MPS sporadic task $\tau' = (G', D, T)$, satisfying the two properties that

P1. Each edge in G' connects jobs that use different security mechanisms; and

P2. For any conditional MPS sporadic task system Γ that is limited-preemption EDF-schedulable upon a preemptive processor, the MPS sporadic task system

$$\Gamma' = \bigcup_{\tau \in \Gamma} \{\tau'\} \tag{1}$$

is limited-preemption EDF-schedulable upon the same preemptive processor.

Suppose we were to perform schedulability-analysis of Γ' using the schedulability-analysis algorithm of [1]. Since each task in Γ' satisfies Property **P1**, it is not a conservative over-approximation to charge a tear-down plus setup overhead when transitioning across each edge of the DAG, as the analysis of [1] does – such an overhead cost is indeed actually incurred. And it follows by Property **P2** that Γ' is schedulable if and only if Γ is. Hence, the two-step process

1) convert each task $\tau \in \Gamma$ to a task τ' that satisfies the properties P1 and P2 above to obtain the task system Γ'; and
2) use the schedulability-analysis algorithm of [1] to determine whether Γ' is schedulable or not

constitutes an ***optimal*** schedulability-analysis algorithm for systems with conditional MPS sporadic tasks.

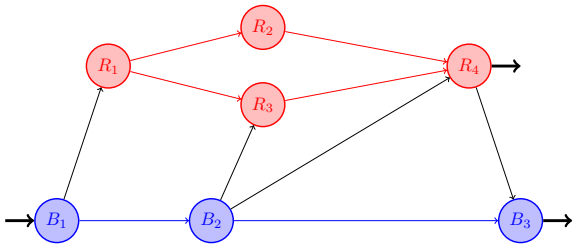

Fig. 2. An example task (discussed in Section IV-A)

A. An Illustrative Example

In this section we illustrate, via a simple example (Figure 2), the intended outcome of the process of converting a conditional MPS sporadic task to another such that the properties P1 and P2 listed above are satisfied, and try to provide some intuition behind our algorithm for doing so. The actual algorithm is presented in Section IV-B.

Our example task in Figure 2 is defined for an execution environment that uses (at least) two different security mechanisms (i.e., $|\mathcal{D}| \geq 2$) that are denoted in Figure 2 by different colors: blue and red. The vertices labeled R_1–R_4 are assumed to use the red security mechanism, and the vertices labeled B_1–B_3 the blue security mechanism.

For any given security mechanism $\delta \in \mathcal{D}$, We refer to the set of all vertices using the same security mechanism V_δ as the ***security domain*** for security mechanism δ. For instance, consider in Figure 2 that its security mechanisms are given by $\mathcal{D} = \{\delta_R, \delta_B\}$ where R and B are the mechanisms for the red and blue nodes respectively. Thus $V_{\delta_R} = \{R_1, R_2, R_3, R_4\}$ comprises the red security domain for our example task, whilst $V_{\delta_B} = \{B_1, B_2, B_3\}$ comprises the blue security domain.

We identify the set of all entry and exit vertices for each security domain. The ***entry*** vertices for a security domain are the source nodes and/or all those vertices that have an incoming edge from a vertex that belongs to a different security domain; analogously, ***exit*** vertices are the sink nodes and/or have an outgoing edge to a vertex belonging to a different security domain. For our example task, vertices R_1, R_3 and R_4 are entry vertices, and R_4 is the sole exit vertex, for the red security domain; for the blue security domain, vertices B_1 and B_3 are entry vertices, and all three of B_1, B_3 and B_3 are exit vertices.

We will first describe how the set of vertices of the converted graph is obtained. The converted graph will have, for each security domain, a single vertex to represent each combination of entry vertex and exit vertex of that security domain. The WCET parameter assigned to a vertex representing a particular

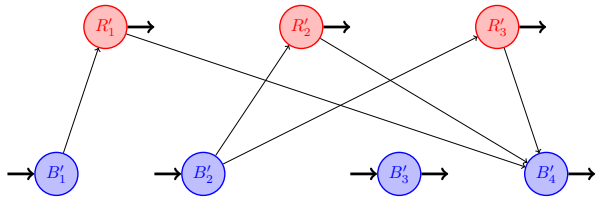

Fig. 3. The example task of Figure 2 after conversion

(entry vertex, exit vertex) ordered pair is set equal to the largest cumulative WCET of any path from the entry vertex to the exit vertex, that lies entirely within the security domain. Figure 3 depicts the converted task for the initial task of Figure 2, with the vertices R'_i and B'_j representing (entry vertex, exit vertex) ordered pairs as follows:

Node	Entry	Exit	WCET
R'_1	R_1	R_4	$c(R_1) + \max(c(R_2), c(R_3)) + c(R_4)$
R'_2	R_3	R_4	$c(R_3) + c(R_4)$
R'_3	R_4	R_4	$c(R_4)$
B'_1	B_1	B_1	$c(B_1)$
B'_2	B_1	B_2	$c(B_1) + c(B_2)$
B'_3	B_1	B_3	$c(B_1) + c(B_2) + c(B_3)$
B'_4	B_3	B_3	$c(B_3)$
B'_5	B_3	B_1	infeasible
B'_6	B_3	B_2	infeasible

Notice that vertices B'_5 and B'_6 are infeasible in the sense that in the original task (Figure 2) it is not possible to traverse the blue security domain from B_3, the entry vertex associated with both B'_5 and B'_6, to either B_1 or B_2, the exit vertices associated with B'_5 and B'_6 respectively. Hence, B'_5 and B'_6 are not depicted in Figure 3.

Now, we will describe how the set of edges of the converted graph is obtained: for each edge (u, v) in the original task that has its two end-points lie in different security domains, we add an edge from each vertex representing an ordered pair in which vertex u is the second (i.e., exit) vertex, to each vertex representing an ordered pair in which vertex v is the first (i.e., entry) vertex. Consider, for instance, the edge (B_2, R_3) in Figure 2. Since B_2 is the exit vertex for B'_2 (it is also the exit vertex for the infeasible vertex B'_6 that is not depicted in Figure 3) whereas R_3 is the entry vertex for R'_2, the edge (B'_2, R'_2) is added to the task in Figure 3.

As another example, consider the edge (R_4, B_3) of Figure 2. Vertex R_4 is the exit vertex for all three vertices R'_1, R'_2, and R'_3, while B_3 is the entry vertex for the vertex B'_4. Consequently, in Figure 3 there are edges from each of R'_1, R'_2, and R'_3 to B'_4.

It is evident from the description of how the edges of the transformed task are obtained that the transformed task satisfies Property P1: each edge connects vertices belonging to different security domains. To see how it also satisfies Property P2, observe that

- Any execution path through the original DAG can be broken up into segments, each comprising a maximal sequence of consecutively-executed vertices that belong to the same security domain.
- But each such maximal sequence of consecutively-executed vertices is represented by a single vertex in the transformed graph.
- Hence, the execution path through the original DAG is modeled by a path in the transformed DAG in which each maximal sequence of consecutively-executed vertices in the path is represented by the corresponding vertex in the transformed DAG that is associated with the first and last vertices of the maximal sequence as the entry and exit vertices, and each edge in the original path that transitions from one security domain to another is represented by the corresponding edge in the transformed DAG.

Consider, for instance, the path

$$\langle B_1 \rightarrow R_1 \rightarrow (R_2 \text{ or } R_3) \rightarrow R_4 \rangle$$

in Γ; this maps to the path

$$\langle B'_1 \rightarrow R'_1 \rangle$$

in Γ'. Similarly, the path

$$\langle B_1 \rightarrow B_2 \rightarrow R_3 \rightarrow R_4 \rightarrow B_3 \rangle$$

in Γ maps to the path

$$\langle B'_2 \rightarrow R'_2 \rightarrow B'_4 \rangle$$

in Γ'.

B. A FORMAL DESCRIPTION OF THE ALGORITHM

In this subsection, we now formalize the process outlined in the example of the previous section for creating transformed DAGs that satisfy Properties P1 and P2. The process comprises follow two steps:

1) Define a set of vertices V' for the converted graph.
2) Create edges E' between the vertices in V'.

Identifying the security mechanism δ associated with each vertex in the graph G is crucial since we construct subgraphs induced by a particular domain V_δ. Let G_δ represent the subgraph induced by the security domain V_δ; that is, $G_\delta = (V_\delta, E_\delta)$ where V_δ is the security domain for δ and E_δ are the edges that exist between nodes of V_δ (i.e., $E_\delta = \{(v_i, v_j) \in E \mid v_i \in V_\delta \wedge v_j \in V_\delta\}$). These subgraphs are defined as described in the pseudocode provided in Figures 4 and 5.

Description of Algorithm on Example: We first identify the set of entry and exit nodes for each domain. Notice that the graph G in Figure 2 contains two domains – *red* and *blue*. For the *red* domain, the entry nodes are: $T1 = \{R_1, R_3, R_4\}$ and the exit nodes are: $T2 = \{R_4\}$. Similarly, for the *blue* domain, the entry nodes are: $T1 = \{B_1, B_3\}$ and the exit nodes are:

Input: Graph $G = (V, E)$
1 $V' \leftarrow \emptyset$ // Initialize V'
2 **for** *each security mechanism* $\delta \in \mathcal{D}$ **do**
3 Identify the corresponding security domain V_δ and induced subgraph G_δ.
4 $T1 \leftarrow \{v_a, v_b, \ldots\}$; // Set of entry nodes of domain V_δ
5 $T2 \leftarrow \{v_x, v_y, \ldots\}$; // Set of exit nodes of domain V_δ
6 **for** *each pair* (v_i, v_j) *such that* $v_i \in T1$ *and* $v_j \in T2$ **do**
7 create a new node X_k;
8 $V' \leftarrow V' \cup \{X_k\}$;
9 $X_k.WCET \leftarrow$ ComputeLongestPath(v_i, v_j, G_δ); // Longest Path in the subgraph induced by domain V_δ
10 $X_k.entry \leftarrow v_i$;
11 $X_k.exit \leftarrow v_j$;
12 **end**
13 **end**
14 **return** SET OF VERTICES V'

Fig. 4. Pseudocode for processing domains to create new nodes V' for the transformed graph.

Input: Graph $G = (V, E)$, set of vertices V'
1 $E' \leftarrow \emptyset$; // Initialize set of edges
2 **for** *each pair of vertices* (v'_i, v'_j) *in* V' **do**
3 **if** $((v'_i.exit, v'_j.entry)$ *edge exists in* $G \wedge (v'_i.domain \neq v'_j.domain))$ **then**
4 Add edge (v'_i, v'_j) to E';
5 **end**
6 **end**
7 **return** SET OF EDGES E'

Fig. 5. Pseudocode for creating edges E' for the transformed graph.

$T2 = \{B_1, B_2, B_3\}$. For every pair within the sets of entry and exit nodes:

- Generate a new node and assign a label.
- Determine its Worst Case Execution Time (WCET) by finding the longest path in the subgraph formed by domain V_δ. Note that the subgraph is also DAG, so the longest path problem can be solved in linear time in $|V_\delta|$ and $|E_\delta|$.
- Record the entry and exit nodes of the newly created node for reference.

These processes result in the creation of nodes (V') within the graph G', as outlined in the tables provided in Section IV-A. Subsequently, to establish connectivity, an examination is conducted for each pair of nodes in V' to determine whether any edges were present in the original graph G, as specified in Line 3 of Figure 5. For instance, consider the pair of nodes within V':

- (R'_1, B'_1) :
  $(R'_1.exit, B'_1.entry) = (R_4, B_1)$
  $(R4, B1) \notin E$
  Thus, there exists no edge in E' for (R'_1, B'_1).
- (R'_2, B'_4) :
  $(R'_2.exit, B'_4.entry) = (R_4, B_3)$
  $(R4, B3) \in E$
  Thus, $(R'_2, B'_4) \in E'$.

The remaining pairs of vertices in V' are checked similarly to form the edges in E'.

C. RUN-TIME COMPLEXITY

The time complexity of the algorithm in Figure 4 is dominated by the inner for-loop, which iterates up to n^2 times where n is the (maximum) number of phases. Inside the inner for-loop, the most time consuming operation involves computing the longest path, which takes $O(|V| + |E|)$. Hence, the time complexity of this algorithm (Figure 4) is $O(|\mathcal{D}| \cdot n^2 \cdot (|E| + |V|))$, i.e., $O(|\mathcal{D}| \cdot |V|^2 \cdot (|V| + |E|))$.

Similarly, the time complexity of the algorithm presented in Figure 5, is $O(|V|^2 + |V| \cdot |E|)$. However, this time complexity can be reduced to $O(|V|^2)$ if an adjacency matrix is used to represent the graph instead of an adjacency list.

V. EMPIRICAL EVALUATION

Having introduced an updated conditional MPS sporadic task model (Section III) and updated analysis (Section IV), we now focus our attention upon empirically quantifying the improvement compared with the original conditional MPS sporadic model/analysis of [1]. Since our proposed approach removes unnecessary startup/ teardown costs between nodes in the same security domain, the total security overhead for each DAG will never increase (and will likely decrease) when compared to the approach that requires a startup/teardown for each node. So, the updated model/analysis will dominate the original MPS model; however, it remains to be seen if these updates represent a significant change with respect to the schedulability of a security-cognizant system comprising conditional MPS sporadic tasks.

A. EXPERIMENTAL SETUP

We implemented both the uniprocessor limited-preemption EDF schedulability test for the original conditional MPS sporadic model [1] and the graph transformation algorithm described in Section IV. All algorithms and the EDF-schedulability test were implemented in `python`.

We evaluated our proposed approach against the original approach [1] over randomly-generated task systems. The conditional MPS sporadic model requires the generation of both the DAG structure as well as the generation of each task's

timing parameters. We now describe the methodology for our task generation of these two aspects separately.

Graph Generation Methodology. For each task, we generate a DAG using the Erdös-Rènyi method [16], [17]. The method, denoted $G(n, p)$ and provided by the `networkx` library [18], takes as input the number of nodes in the graph n and the probability that an edge appears between two nodes p. The method returns an undirected graph. We create a directed acyclic graph from the returned undirected graph by randomly permuting the vertices of the graph in an ordered list; we then add direction to ensure that an edge only points from an earlier node in the list to a later node. This approach provides direction and ensures that no cycles are introduced into the graph. Finally, to ensure that the graph is connected and has a single starting point (source) and a single termination point (sink), we create a dummy source/sink that we link to the source/sink nodes of the original graph.

In the presented experiments, the number of nodes n for a task is uniformly selected from the range $[2, 10]$. The probability of an edge p is determined by a predefined value, in this case, set to 0.5. Furthermore, each node in the graph (except the dummy sink/source) is uniformly assigned a security domain from the set of security domains. The number of security domains for a task is uniformly drawn from the range $[1, 3]$.

Timing-Parameter Generation Methodology. Once the graph structure $G = (V, E)$ for a task has been generated, we can generate its timing parameters. For each node $v \in V$, we randomly generate a worst-case execution time $c(v)$ uniformly as an integer from the range $[1, 10]$ (except the dummy source/sink are assigned zero execution time). Additionally, for each security mechanism δ (and domain V_δ), we randomly assign an integer cost $\text{cost}(\delta)$. Using the above parameters, the initial "weight" of a node $v \in V$ is determined by $c(v) + \text{cost}(\sigma(v))$. Using this initial weight, we can find the longest path from dummy source to dummy sink; we represent the length of this path as $L(G)$. Intuitively, this represents the maximum execution time of the conditional MPS sporadic task if each node were executed non-preemptively and all nodes performed the security mechanism start-up/ teardown at the beginning/ end of their execution. (The approach of [1] and our paper modify this cost, but this longest path gives a common reference to determine the utilization of the task).

The *utilization* u of a task is defined by the weight of its initial longest path $L(G)$ over the period T. (We assume implicit-deadline tasks for the experiments; that is, $D = T$ for all tasks). The utilization of the system U is the sum of the individual task utilization. To obtain the utilizations of the system and its tasks, we use the UUniFast algorithm [19] which takes as input the target system utilization U and number of tasks $|\Gamma|$ and returns a $|\Gamma|$-length vector of non-negative real numbers $(u_1, u_2, \ldots, u_{|\Gamma|})$ that sum to exactly U. These u_ℓ values represent the target assigned utilization for each task. For each task, we take $L(G_\ell)/u_\ell$ (rounded to the nearest integer) to obtain the corresponding task period T_ℓ. Since the rounding of period may cause some small deviation from the task's target utilization, we check that the generated system utilization is within a threshold of 0.05; if not, then we discard this task system and regenerate another system.

B. Schedulability Results

As an assessment criterion, we analyze the schedulability ratio across various utilization values, varying from 0.1 to 1.0. Figure 6 shows the schedulability ratios obtained for the different utilizations obtained for the Preprocessing Algorithm [1] of the original Conditional MPS model (previous approach) and our improved Conditional MPS model (new approach).

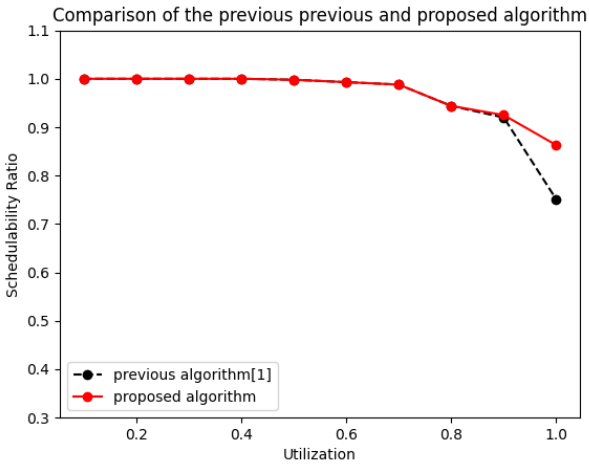

Fig. 6. Schedulability ratio for different utilizations. The ratio is calculated for 1000 tasksets, each containing 5 tasks with at most 3 distinct security domains

The graph illustrates the comparison of the *Schedulability Ratio* between the two algorithms, considering various levels of system utilization. At lower utilization values, both algorithms demonstrate near-unity schedulability ratios, indicating high reliability in meeting task deadlines. At $U = 0.5$, the schedulability ratio drops marginally to 0.998 for both the previous and the proposed algorithms. At $U = 0.6$, $U = 0.7$, and $U = 0.8$, the schedulability ratios of the two algorithms remain constant at 0.993, 0.988, and 0.944, respectively. As system utilization increases, a noticeable divergence emerges between the two algorithms. Specifically, at $U = 0.9$, the previous algorithm shows a decline, reaching a schedulability ratio of 0.920, while the proposed algorithm maintains a higher schedulability ratio of 0.925. This trend persists as the utilization further increases, with the previous algorithm experiencing a sharper decline compared to the proposed algorithm. Notably, at $U = 1.0$, the schedulability ratio of the previous algorithm drops to 0.751, indicating a significant impact on its schedulability. In contrast, the proposed algorithm demonstrates consistently high performance, with a schedulability ratio of 0.863 at the same utilization. This represents approximately a 15% improvement for high-utilization task systems.

We observe that the schedulability of the previous algorithm declines more rapidly as the additional overhead between each node, which, as the system utilization surpasses 100%, is likely to exert a greater impact on the non-transformed graph. In contrast, the transformed graph in the proposed algorithm employs the algorithm in Section IV-B to minimize overheads, ensuring a more gradual decrease in schedulability even under higher workloads. As the schedulability ratio of the proposed algorithm remains comparatively higher than that of the previous algorithm, this suggests that the proposed algorithm offers improved schedulability under higher workload conditions.

We also tested edge probabilities p of 0.25 and 0.75, alongside the original case. The results revealed notable similarities across all scenarios, suggesting that variations in edge probabilities do not significantly affect schedulability. It can be inferred that the connectivity of the graph does not seem to exert a substantial influence on the ability to mitigate inter-node pessimism, as the schedulability remained consistent across different edge probabilities

VI. Conclusions

In this work, we have presented a refinement to the Multi-Phase Secure (MPS) Sporadic Task Model, addressing a potential source of pessimism in the security-cognizant scheduling of real-time systems. Our approach significantly reduces the over-approximation in the overhead assumptions inherent in previous models by accurately identifying instances where switching between security mechanisms is unnecessary.

The proposed approach can only *de*crease the overhead when compared to the previous mechanism, and therefore dominates it in terms of schedulability. The results from our empirical evaluation demonstrate the improvement in schedulability ratios for security-aware task systems. Thus our proposed refinement furthers the balancing of security needs and timing constraints in systems with limited computing resources.

Furthermore, our refined model contributes to the broader discourse on the integration of security considerations into real-time system design. By offering a more precise and effective scheduling methodology, we facilitate the development of systems that are both secure and time-predictable, addressing the increasing complexity and security demands of modern embedded applications.

Future work will focus on extending this model to multi-processor and distributed systems and exploring its application in various real-world scenarios. The potential for further optimization and adaptation to different types of embedded systems presents an exciting avenue for research, promising even greater contributions to the field of secure and real-time system design.

VII. Acknowledgments

This research was supported in part by NSF Grants CNS-2038609, CNS-2141256, CNS-2211641, CPS-2229290, and CPS-2038726. We thank the anonymous reviewers for their constructive remarks.

References

[1] Sanjoy K. Baruah, Thidapat Chantem, Nathan Fisher, and Fatima Raadia. A scheduling model inspired by security considerations. In *26th IEEE International Symposium on Real-Time Distributed Computing, ISORC 2023, Nashville, TN, USA, May 23-25, 2023*, pages 32–41. IEEE, 2023.

[2] T. Xie and X. Qin. Scheduling security-critical real-time applications on clusters. *IEEE Trans. Computers*, 55(7):864–879, July 2006.

[3] M. Lin, L. Xu, L.T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu. Static security optimization for real-time systems. *IEEE Trans. Industrial Informatics*, 5(1):22–37, February 2009.

[4] Y. Ma, W. Jiang, N. Sang, and X. Zhang. ARCSM: A distributed feedback control mechanism for security-critical real-time system. In *Proc. Int. Symp. Parallel and Distributed Processing with Applications*, pages 379–386, July 2012.

[5] K. Jiang, A. Lifa, P. Eles, Z. Peng, and W. Jiang. Energy-aware design of secure multi-mode real-time embedded systems with FPGA co-processors. In *Proc. Int. Conf. Real-Time Networks and Systems*, pages 109–118, October 2013.

[6] M. Pajic, N. Bezzo, J. Weimer, R. Alur, R. Mangharam, N. Michael, G.J. Pappas, O. Sokolsky, P. Tabuada, S. Weirich, and I. Lee. Towards synthesis of platform-aware attack-resilient control systems. In *Proc. Int. Conf. High Confidence Networked Systems*, pages 75–76, April 2013.

[7] M. Yoon, S. Mohan, C. Chen, and L. Sha. Taskshuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, April 2016.

[8] Sibin Mohan, Man Ki Yoon, Rodolfo Pellizzoni, and Rakesh Bobba. Real-time systems security through scheduler constraints. In *Proceedings of the 2014 Agile Conference*, AGILE '14, pages 129–140, 2014.

[9] Tanmaya Mishra, Thidapat Chantem, and Ryan Gerdes. Teecheck: Securing intra-vehicular communication using trusted execution. In *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, RTNS 2020, page 128–138, New York, NY, USA, 2020. Association for Computing Machinery.

[10] Anway Mukherjee, Tanmaya Mishra, Thidapat Chantem, Nathan Fisher, and Ryan Gerdes. Optimized trusted execution for hard real-time applications on cots processors. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, RTNS '19, page 50?60, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Sanjoy K. Baruah. Security-cognizant real-time scheduling. In *25th IEEE International Symposium On Real-Time Distributed Computing, ISORC 2022, Västerås, Sweden, May 17-18, 2022*, pages 1–9. IEEE, 2022.

[12] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

[13] Sanjoy Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, 2005.

[14] Marto Bertogna and Sanjoy Baruah. Limited preemption EDF scheduling of sporadic task systems. *IEEE Transactions on Industrial Informatics*, 2010.

[15] Sanjoy Baruah. A general model for recurring real-time tasks. In *Proceedings of the Real-Time Systems Symposium*, pages 114–122, Madrid, Spain, December 1998. IEEE Computer Society Press.

[16] P. Erdös and A. Rènyi. On random graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[17] Daniel Cordeiro, Gregory Mouni, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frederic Wagner. Random graph generation for scheduling simulations. In *Proceedings of the 3rd international ICST conference on simulation tools and techniques*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[18] Networkx. https://networkx.org/.

[19] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2), 2005.