Hindawi
Security and Communication Networks
Volume 2022, Article ID 4118994, 16 pages
https://doi.org/10.1155/2022/4118994

Research Article

Timed Automaton-Based Quantitative Feasibility Analysis of Symmetric Cipher in Embedded RTOS: A Case Study of AES

Yawen Ke [1] **and Xiaofeng Xia** [1,2]

[1]*School of Bigdata and Software Engineering, Chongqing University, Chongqing 401331, China*
[2]*Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing 400044, China*

Correspondence should be addressed to Xiaofeng Xia; xiaxiaofeng@cqu.edu.cn

Received 4 October 2021; Accepted 9 December 2021; Published 12 January 2022

Academic Editor: Shahram Babaie

The real-time operating system (RTOS) has a wide range of application domains and provides devices with the ability to schedule resources. Because of the restricted resources of embedded devices and the real-time constraints of RTOS, the application of cryptographic algorithms in these devices will affect the running systems. The existing approaches for RTOS ciphers' evaluation are mainly provided by experimental data performance analysis, which, however, lack a clear judgment on the affected RTOS performance indicators, such as task schedulability, bandwidth, as well as a quantitative prediction of the remaining resources of RTOS. By focusing on task schedulability in RTOS, this paper provides a timed automaton-based quantitative approach to judge the feasibility of ciphers in embedded RTOS. First, a cryptographic algorithm execution overhead estimation model is established. Then, by combining the overhead model with a sensitivity analysis method, we can analyze the feasibility of the cryptographic algorithm. Finally, a task-oriented and timed automaton-based model is built to verify the analysis results. We take AES as a case study and carry out experiments on embedded devices. The experimental results show the effectiveness of our approach, which will provide specific feasibility indicators for the application of cryptographic algorithms in RTOS.

1. Introduction

1.1. Motivation. For applications in critical fields such as industrial control systems (ICSs), aerospace, national defense and military, and communications, most of the embedded devices play an important role and undertake a large number of operation and control tasks. In embedded devices with complex functions, embedded systems are generally used to improve resource utilization. To take the needs of real-time control into account, embedded devices are usually equipped with RTOS, including μ C/OS-II, eCos, FreeRTOS, Mbed OS, RTX, and Vxworks. In recent years, with the development of the Internet of Things (IoT), increasing embedded devices are connected to the Internet for data transmission and realize more functions. From the perspective of security, this also brings new challenges. As a key component, the security of embedded devices is related to the overall security of the system, so it is necessary to add security technology protection to embedded devices.

However, due to its resource constraints and real-time restrictions of RTOS, most embedded RTOSs will encounter an adaptability problem when implementing security technologies. Taking cryptographic algorithms as an example, in embedded RTOS, the application of ciphers will cause additional overhead based on the original tasks, which not only leads to an increase in the utilization of hardware resources such as processor, RAM, and FLASH, but the longer execution time after encryption also affects the schedulability of tasks in the system and further affects the RTOS real-time performance even at the normal realization of system functions.

It can be seen that when the current common cryptographic algorithms are implemented on resource-constrained devices, the performance of the embedded RTOS may be clearly affected by them. At present, the most

common cryptographic algorithms are aimed at PCs, servers, and smart mobile devices that do not require high real-time performance and have sufficient resources. Therefore, how to ensure that they can be effectively deployed in embedded RTOS has become a new challenge. Furthermore, how to quantitatively evaluate the cryptographic algorithm configuration allowed by embedded RTOS has also become the focus of security researchers.

On the evaluation of cryptographic algorithms on embedded devices, the performance [1, 2] of cryptographic algorithms on different devices and simple models of cryptographic overhead [3–6] are presented in prior work. Meanwhile, in the task scheduling problem of RTOS, the existing work indicates that there is a sensitivity analysis method, which can calculate the feasible region of the task set. This brings us to present: we wonder whether we can combine the cryptographic algorithm overhead model and the sensitivity analysis method to achieve a quantitative measure of the cryptographic algorithm feasibility in the RTOS task set. This is the primary motivation for this study.

1.2. Our Contribution. In this paper, we present a method to provide a quantitative evaluation of the feasibility of ciphers in embedded RTOS. First, we propose a model for symmetric cryptographic algorithm overhead prediction; then, we combine the sensitivity analysis method to analyze the feasibility of the cryptographic algorithm. Finally, the model based on task automaton is used to verify the analysis results formally. Our major contributions can be summarized as follows:

(i) We analyze the encryption and decryption processes of symmetric ciphers (*AES* – Rijndael) and construct a model according to the instruction set of the application platform to predict the time overhead of the cipher execution

(ii) By applying the sensitivity analysis method to handle the scheduling problem, we analyze the parameter feasible space of the original task set and combine the cost model to quantify the performance upper bound of the cryptographic algorithm in this task set, that is, the upper throughput bound

(iii) We use the labeled transition system (LTS) to give the definition of schedulability of the task automaton model and verify our analysis results by modeling the case in the formal verification tool Uppaal to evaluate the impact of ciphers on the real-time performance of the system

(iv) We port FreeRTOS to a real embedded platform and implement our case. The tracked running data prove the practicability of our proposed evaluation method.

1.3. Organization. The rest of this paper is organized as follows: Section 2 introduces RTOS and the execution on STM32F103 core Cortex-M3. Section 3 introduces the symmetric cryptographic algorithms in detail and models the algorithm's overhead estimation on the platform. Section 4 defines the task automaton and the quantitative definition of schedulability. Section 5 mainly describes the experimental method of this paper, including embedded experiments and validation of the automaton model with formal tools. Before giving the conclusions and prospects in Section 8, we present the experimental data and analyze our evaluation model in Section 6 and discuss prior work related to this paper in Section 7.

2. Background

2.1. RTOS and FreeRTOS. A real-time operating system (RTOS) is a kind of operating system that guarantees the completion of functions within a certain time limit. RTOS is divided into hard real time and soft real time. Hard RTOS requires the operations to be completed within the specified time, which is guaranteed by the design of the operating system, while for soft RTOS, the operations are completed as quickly as possible according to the priority of the tasks. A real-time task scheduler must be included in an RTOS to allocate CPU time strictly according to priority.

In RTOS, a task is an executable program unit. Tasks can be divided into two types: periodic and nonperiodic, depending on whether there are periodic changes during task execution. On the one hand, for periodic tasks, the external device periodically sends out an excitation signal to the computer, requiring it to execute cyclically according to a specified period so as to periodically control a certain external device. On the other hand, nonperiodic tasks are different. Excitation signals sent by external devices have no obvious periodicity, but they must be linked to a deadline. It can be divided into two parts: start deadline (task must be executed before a certain time) and completion deadline (task must be completed before a certain time).

The task cycles of periodic tasks generally include task initial offset, deadline, WCET (worst-case execution time), and WCRT (worst-case response time). A periodic task model can be expressed, as shown in Figure 1.

In the embedded field, RTOS is widely used. The application of embedded RTOS can make more reasonable and effective use of CPU resources, simplify the design of application software, shorten the system development time, and ensure the real-time performance and reliability of the system better. FreeRTOS is a mini RTOS kernel with the following characteristics: open-source code, portability, scalable, and flexible scheduling strategy. It can be easily ported to various embedded devices. FreeRTOS has no limit on the number of system tasks, and meanwhile, the scheduling algorithm supports priority and round-robin scheduling algorithms.

2.2. Execution on STM32 Cortex-M3. The internal structure of the CPU divides the execution of an instruction into three stages: fetch, decode, and execute. In the first phase, one or more instructions are loaded from the program counter points. Then, in the next state, the instruction is decoded, the opcodes in the instruction register are taken out, and the

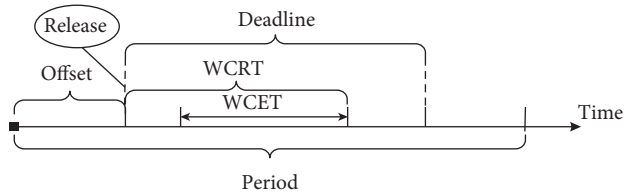

Figure 1: Periodic task execution parameters.

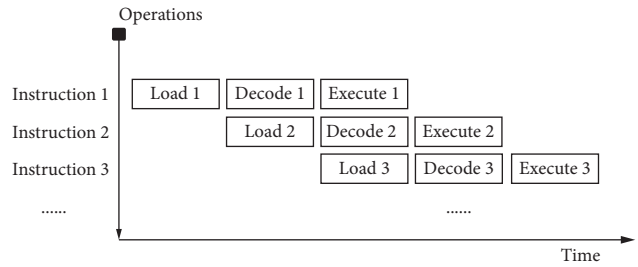

Figure 2: Pipelined instruction execution.

instruction is analyzed. After recognition, the instruction will be executed. During execution, it may also need to access the memory bus, depending on what the opcode does.

The popular 32-bit Cortex-M architecture uses the ARM Thumb-2 instruction set [7]. This instruction set opcodes are encoded in a narrow 16-bit or a wide 32-bit form. Most of the instructions in Cortex-M require one or two clock cycles to execute, and for load/store operation, there is an additional overhead that is proportional to the amount of data to load or store.

Like the other architectures in the Cortex-M series, M3 is based on Harvard memory architecture. This means that data and instructions are accessed through different buses, so the loading, decoding, and execution of instructions can be parallelized. This feature is called pipelining, as shown in Figure 2. When the CPU executes the first instruction, it can fetch the next instruction in the same clock cycle. However, if the CPU has prefetched the instruction and fills the pipeline, but it notices that the prefetched instruction is not the next line to execute during the decoding phase, the pipeline needs to be refreshed and repopulated with the correct instruction.

Besides pipeline operation, bus contention and cache also have a great impact on the actual runtime of the system. When devices on the bus are working beyond their designed speed, the bus mediation may fail, which leads to bus competition. Cache misses are associated with the overhead of performing actual reads from the bus because the access to cached data is usually the orders of magnitude faster.

STM32 series products have an official cryptographic algorithm library. It provides a series of compact implementations of cryptographic algorithms [8] on the platform. We decompile the binary files generated by compiling the cryptographic algorithm library to get the assembly code under the Thumb-2 instruction set; then, we analyze the code structure based on the implementation of the cryptographic algorithm. We also classified the instructions for the Thumb-2 instruction set in Table 1.

In Table 1, P represents the pipe reload. Branches take 1 cycle for instruction and then pipeline reload for target instruction. Nontaken branches are 1 cycle totally. Taken branches with an immediate are normally 1 cycle of pipeline reload (2 cycles total). Taken branches with register operand are normally 2 cycles of pipeline reload (3 cycles total). $(+P)$ means that the pipe reload is counted conditionally when the PC is destined or loaded. N represents the count of elements. Generally, load-store instructions take 2 cycles for the first access and 1 cycle for each additional access. Stores with immediate offsets take 1 cycle.

3. Cryptography Algorithm Analysis

3.1. Cipher Analysis Model. To evaluate the impact of the use of cryptographic algorithms on embedded devices, we need to model the resource usage of these ciphers in terms of clock cycles. For a certain microcontroller, the software implementation of cryptographic algorithms is finally determined by the corresponding instructions, and the overhead of the assembly instruction is also fixed. On this basis, we propose an instruction-level overhead model for cryptographic algorithm implementation. The dominant idea of this method is to extract and simplify the assembly instructions in implementing the ciphers; then, by counting the number of instructions and clock cycles required for execution, we can estimate the computational overhead. Essentially, a mathematical model for the overheads of ciphers can be expressed as follows:

$$O_{\text{cipher}} = \sum O_{\text{ins}_i} N_{\text{ins}_i}, \tag{1}$$

where O_{cipher} represents the overhead of a kind of the cryptographic algorithm, O_{ins_i} and N_{ins_i} are the overhead and occurrence number of each kind of assembly instructions.

From the perspective of assembly instruction-level analysis, we can use the instruction cycle to estimate the running cost of the program. For a cryptographic algorithm, we can abstract the model that expresses its structural features according to the algorithm flow. Taking the block cipher as an example, it can be represented by the following model:

$$\begin{aligned} O_{\text{exec}}(\text{text\_len}) &= O_{\text{initial}} + O_{\text{append}} + O_{\text{finish}} \\ &= O_{\text{misc\_proc}} + \left(N_r \cdot O_r + O_{b\_\text{proc}}\right) \cdot N_b \\ &= O_{\text{misc\_proc}} + \left(N_r \cdot O_r + O_{b\_\text{proc}}\right) \\ &\quad \cdot \left\lceil \frac{\text{text\_len}}{\text{block\_size}} \right\rceil, \end{aligned} \tag{2}$$

where $O_{\text{exec}}(\text{text\_len})$ represents the overhead required to process a piece of content with a length of text_len, O_{initial}, O_{append}, and O_{finish} are the overhead of these 3 phases of the cryptographic algorithm, $O_{\text{misc\_proc}}$ includes all other overheads except block processing, including initial and finish overhead, O_r and N_r are the overhead of each round and the number of rounds, respectively, O_{b_proc} is the overhead of

Table 1: Thumb-2 instruction grouping and cycle count.

Groupname	Description	Examples	Cycle count
Addsub	Addition and subtraction	ADC, ADD, SBC, SUB, ADDW, SUBWADC {S}, ADD {S}, RSB {S}, SBC {S}, SUB {S}	1 (+P)
Branches	Branch on condition code	B<cond>, B, BL, BX, BLX	1 + P
Bitwise	Bitwise operations	AND, BIC, EOR, ORR, NEG, LSL, LSR, TST, REV, UBFX, AND {S}, BIC {S}, EOR {S}, ORR {S}, LSL{S}, LSR {S}	1 (+P)
ls single	Load store single operations	LDR, LDRB, LDRH, LDRSB, LDRSH, STR, STRB, STRH	2 (+P)
ls multiple	Load store multiple operations	LDMIA, POP, PUSH	1 + N (+P)
Mov	Family of move operations	MOV, MOV {S}, MOVW, MOVT	1 (+P)
Extended	Extended	IT and NOP (includes YIELD).	0-1
Misc	Miscellaneous	SXTB, SXTH, UXTB, UXTH, CMP, CBZ	1

processing and appending blocks, which depends on the block size and cryptographic mode it applied, and N_b is the number of blocks.

3.2. AES-Rijndael. Rijndael is an iterative block cipher with a variable block and a key length. AES (the Advanced Encryption Standard) [9] is a special implementation of the Rijndael algorithm. The block size of AES is fixed to be 128 bits (16 bytes), the key can be 128, 192, and 256 bits (16, 24, and 32 bytes), and the number of rounds N_r can be 10, 12, and 14, respectively.

The input of the AES algorithm is a 128-bit block mapped to a byte array named "State", and the operations of AES encryption and decryption are performed on the State. The number of rounds N_r depends on the key length; a 16-byte key corresponds to 10 rounds, a 24-byte key corresponds to 12 rounds, and a 32-byte key corresponds to 14 rounds. The first $N_r - 1$ rounds comprise 4 different transformations: SubBytes, shiftRows, Mix Columns, and AddRoundKeys, while the last round only contains 3 transformations except the Mix Columns.

The Key Expansion for AES-Rijndael is a simple expansion using XOR and cyclic shifts, which takes a word as a basic unit (a word is 4 bytes), and the purpose is to expand the input key into $N_r + 1$ subkeys.

3.2.1. AES-Rijndael Model. For the AES implementation on an instruction set, according to the above formula, it can be expressed in the following form:

$$T_{\text{exec}}(\text{text\_len}) = \frac{\left[\sum O_{i_m} N_{i_m} + \left(N_r \cdot \sum O_{i_r} N_{i_r} + \sum O_{i_b} N_{i_b}\right) \cdot \lceil \text{text\_len/block\_size} \rceil\right]}{\text{processor\_freq}}. \quad (3)$$

In this model, we use the instruction-level overhead expression in formula (1) to replace the content in formula (2), and then, we get formula (3). The number of instruction cycles in the Thumb-2 instruction set can be queried in Table 1. The processor_freq is the basic frequency of the microcontroller in the running platform. We can calculate the clock cycle, which is the reciprocal of the basic frequency. Based on formula (3), we can estimate the AES execution time [10] of a specific configuration on Cortex-M3.

3.3. Model Analysis. To evaluate the accuracy of the model, we performed it on an embedded platform STM32F103VE as a concrete example. We use different modes of the AES [11] algorithm to encrypt 64-byte plaintext on Thumb-2. Under our model, the statistical results of the amount of instruction groups are shown in Table 2. Figures 3 and 4 show the comparison between the actual encryption process overhead and our predicted results. Figure 3 mainly shows the difference between the predicted and actual value when the length of the plaintext is fixed to 64 bytes. The predicted execution time is greater than the actual execution time with several configurations of the AES. This is because we chose the longest program flow to calculate in the instruction-level static analysis of the AES assembly file, so our prediction can be viewed as the WCET of this AES implementation for a specific plaintext length. While Figure 4 shows the distinction of different plaintext lengths, we can see that the predicted value of our model is still higher than the actual execution time in terms of changing plaintext length. It can be seen that our model can more accurately predict the implementation cost of the AES algorithm on the instruction set.

4. TA Model and Schedulability Analysis

4.1. Timed Automaton (TA). Timed automaton is a set of theories for modeling and verifying [12] real-time systems. Many verification tools (such as Uppaal [13]) are based on the theory of timed automaton. A timed automaton is a finite

Table 2: Thumb-2 assembly instructions used in AES encryption.

Groupname	AES-128 ECB	AES-192 ECB	AES-256 ECB	AES-128 CBC	AES-192 CBC	AES-256 CBC	AES-128 CTR	AES-192 CTR	AES-256 CTR
Addsub	174	190	206	210	226	242	161	177	193
Branches	84	89	149	85	89	149	84	92	152
Bitwise	3219	3563	3935	3251	3595	3967	3219	3563	3935
ls single	1782	2102	2422	1798	2118	2438	1816	2136	2456
ls multiple	10	10	10	10	10	10	10	10	10
Mov	466	554	646	464	552	556	461	549	641
Extended	6	6	6	1	1	1	2	2	2
Misc	818	966	1122	820	968	1124	820	972	1124

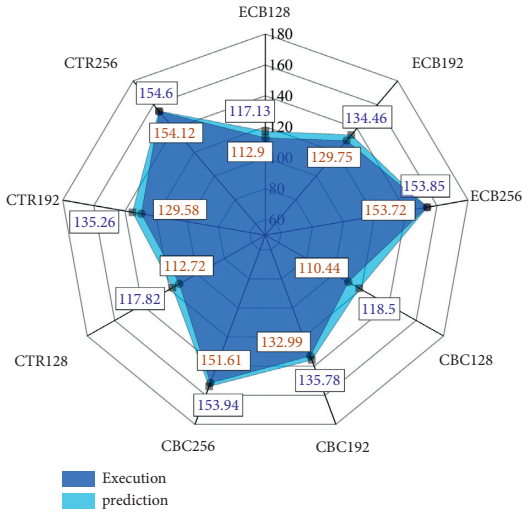

Figure 3: AES encryption prediction accuracy.

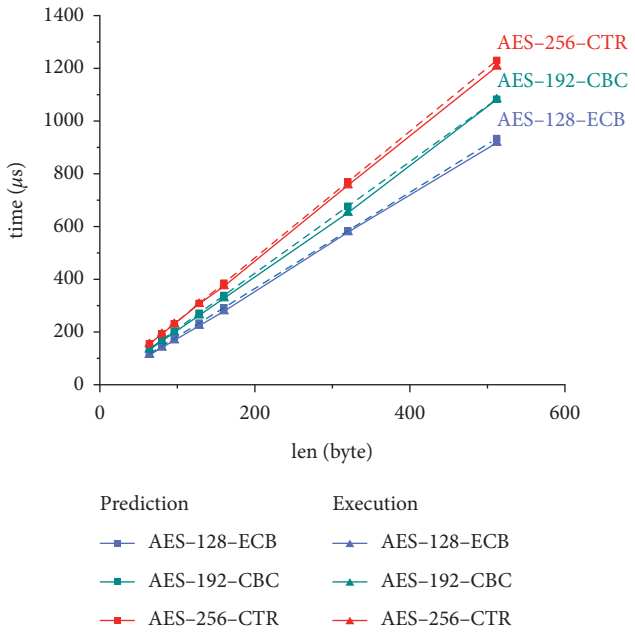

Figure 4: AES encryption with variable plaintext length.

automaton (a graph composed of a finite number of nodes or locations and edges) attached to several real-valued clocks. The transition between the states can only occur if the clock constraints are met. Timed automaton is an abstract model of the time-sequential [14] system. In this paper, the task set scheduling problem in RTOS is our focus, so we choose timed automaton theory for modeling.

Definition 1 (timed automaton). Assuming a finite set of real-valued clocks $\mathscr{C}$ and $\mathscr{B}(\mathscr{C})$ is the set of constraints on the clocks in $\mathscr{C}$. The clock constraints (guards) are conjunctions of expressions of the form $x \infty N$ and $x - y \infty N$, where $x, y \in \mathscr{C}, N \in \mathbb{N}$ and $\infty \in \{<, \leq, =, \geq, >\}$. A timed automaton over the set of clocks C is a tuple $\mathscr{A} = \langle L, l_0, \Sigma, \mathscr{C}, I, E \rangle \in \mathscr{A}$, where

(i) L: a set of locations that represent the system status

(ii) $l_0 \in L$: the initial location

(iii) Σ: a set of actions

(iv) $\mathscr{C}$: a set of clocks. All clocks are initialized to 0 at l_0 and may be reset after executing a transition.

(v) I: $L \longmapsto \mathscr{B}(\mathscr{C})$: a function assigning each location with a clock constraint (a location invariant)

(vi) $E \subseteq L \times \mathscr{B}(\mathscr{C}) \times \Sigma \times 2^{\mathscr{C}} \times L$ is the set of transitions, where transition $\langle l, g, a, r, l' \rangle$ from location l to location l' labeled with action a is executed only if guard g is true and clocks in $r \subseteq \mathscr{C}$ are reset

A transition in TA is denoted by $l \longrightarrow^{g}, a, r\ l'$, where l and l' represent the source and destination locations, respectively; a is the action taken during the transition; g is the guard clock that should hold to execute the transition; r is the subset of clocks to be reset when the transition is over. When a timed automaton $\mathscr{A} \in \mathscr{A}$, we use $L(\mathscr{A}), l_0(\mathscr{A}), \Sigma(\mathscr{A}), \mathscr{C}(\mathscr{A})$ to represent its system status, initial locations, set of actions, and set of clocks. It is similar for other automatons.

Definition 2 (timed automaton semantics). Let $\mathscr{A} = \langle L, l_0, \Sigma, \mathscr{C}, I, E \rangle \in \mathscr{A}$ be a timed automaton with the configuration (l, u), where location $l \in L$ and clock valuation $u \in \mathscr{C}$. The semantics are defined by a $LTS < S, s_0, \longrightarrow >$ as follows:

(i) $S \subseteq L \times \mathbb{R}^{\mathscr{C}}$: a set of state

(ii) $s_0 = (l_0, u_0)$: the initial state of TA, $u_0 = 0$

(iii) $\longrightarrow \subseteq S \times (\mathbb{R}^{\geq 0} \cup \Sigma) \times S$: the transition relation such as–

- (i) $(l,u) \longrightarrow^{d} \mathscr{A} \quad (l,u+d)$ and $\forall d'$, if $0 \le d' \le d$, $u+d' \vDash I(l)$ where $d \in \mathbb{R}^{\ge 0}$,
- (ii) $(l,u) \longrightarrow^{a} \mathscr{A}(l',u')$, if $\exists e = (l,a,g,r,l') \in \Sigma$ and $u' = [r \longmapsto 0]u$, $u' \vDash I(l')$.

When an action $a \in \Sigma$ received or sent a clock value that satisfies the clock guard g $(u \vDash g)$, the transition $l \longrightarrow^{g}, a, r\ l'$ will happen, and the TA will change its location as well.

4.2. Timed Automaton with Tasks

4.2.1. Tasks. We consider a multiprocessor system with m identical processors and a set of n independent, preemptive real-time tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. A real-time task τ_i, where $\tau_i \in \Gamma$, is represented by a tuple as follows:

$$\tau_i = (T_i, D_i, O_i, P_i, R_i, B_i, W_i), \tag{4}$$

where τ_i is the task identification, T_i is the period (or equivalently, the minimum interarrival time), $D_i \le T_i$ is the relative deadline, O_i is the initial offset, and P_i is the priority, smaller values of P_i indicate lower priorities. Without loss of generality, we assume global fixed-priority scheduling and that $P_1 > P_2 > \ldots > P_n$ so that τ_1 is the highest priority task. R_i is the resource, B_i and W_i are the best and worst execution time of the task, respectively, such that $B_i \le W_i \le D_i$ and $0 < W_i$.

We shall use τ_i to denote a task instance, and τ_i can also be shown as $(T_i, D_i, O_i, P_i, R_i, B_i, W_i)$. Then, a task queue can be denoted by a poset ordering by the priority of task set Γ. A set of all task queues including instances of tasks from Γ is denoted Q_Γ.

4.2.2. Scheduling Strategy. The research of this paper is scheduling problems on single-processor systems, and the task queue mentioned in the previous part can be assumed as a list sorted with a specific scheduling strategy. The first element in this list is the task executed by the processor, and the others are tasks waiting for resources. A scheduling strategy can be assumed as a function that can insert released tasks into the task queue, such as FIFO (first in first out), FPS (fixed-priority scheduling), and EDF (earliest deadline first).

A scheduling strategy can be indicated by $\mathscr{SCH}: \Gamma \times Q_\Gamma \longmapsto Q_\Gamma$. this function can insert a task instance into the task queue and keep the other tasks in the queue preserved when receiving a task instance and a task queue. Finished tasks are removed from the task queue, and the tasks must finish before its WCET.

4.2.3. Task Automaton. The task automaton model is based on the timed automaton and adds a clock named x_{finished}. It resets at the time when a task finishes its computation removed from the queue and prepares to start a new period.

Definition 3 (task automaton). A task automaton is a timed automaton extended with tasks. A task automaton over actions Σ, clocks $\mathscr{C}$, and tasks Γ is a tuple $\langle L, l_0, E, I, M, x_{\text{finished}} \rangle$, where

(i) L: a set of locations that represents the system status

(ii) $l_0 \in L$: the initial location

(iii) $E \subseteq L \times \mathscr{B}(\mathscr{C}) \times \Sigma \times 2^{\mathscr{C}} \times L$ is the set of transitions, indicated by edges

(iv) I: $L \longmapsto \mathscr{B}(\mathscr{C})$: a function assigning each location with a clock constraint (a location invariant)

(v) M: $L \longrightarrow \Gamma$: a partial function assigning locations with tasks

(vi) $x_{\text{finished}} \in \mathscr{C}$: the clock which is reset when the task finished

A transition in task automaton is denoted by $l \longrightarrow^{g}, a, r\ l'$.

We use $\mathscr{V}(\mathbb{R} \ge 0)$ to denote the set of clock assignments for $\mathscr{C}$; a semantic state of an automaton is a triple (l,u,q), where l is the current location, $u \in \mathscr{V}$ denotes the current values of clocks, and q is the current task queue. By u_0, we denote a clock assignment such that $u_0(x) = 0$ for all clocks x. We can then present the operational semantics in LTS.

Definition 4 (scheduling semantics). Given a scheduling strategy SCH, the semantics of an automaton $\mathscr{A} = < L, l_0, E, I, M, x_{\text{finished}} >$ is a labeled transition system A_{SCH} with an initial state $(l_0, u_0,)$, and the transitions are defined by the following rules:

(i) $(l,u,q) \xrightarrow{a}_{\text{SCH}} (l', u[r], \text{SCH}(M(l'), q))$ if $l \longrightarrow^{g}, a, r\ l', u \vDash g$, and $u[r] \vDash I(l')$,

(ii) $(l,u,) \longrightarrow^{t}_{\text{SCH}} (l, u+t,)$ if $t \in \mathbb{R}_{\ge 0}$ and $(u+t) \vDash I(l)$,

(iii) $(l,u,\tau_i::q) \longrightarrow^{t}_{\text{SCH}} (l, u+t, \quad \text{Exec} \quad (\tau_i)::q, t)$ if $t \in \mathbb{R}_{\ge 0}$, $t \le W_i$ and $(u+t) \vDash I(l)$,

(iv) $(l,u,\tau_i)::q) \longrightarrow^{\text{finish}}_{\text{SCH}} (l, u[x_{\text{finished}}], \quad q)$ if $B_i \le 0 \le W_i$ and $u[x_{\text{finshed}}] \vDash I(l)$,

where $\tau_i::q$ denotes the queue with the task instance τ_i inserted into q (at the first position), denotes the empty queue, and finish $\notin \Sigma$ is a distinct action name.

Definition 5 (reachability). We can use $(l,u,q) \longrightarrow_{\text{SCH}} (l',u',q')$ to indicate $(l,u,q) \longrightarrow^{a}_{\text{SCH}} (l',u'\prime,q')$ for an action a or $(l,u,q) \longrightarrow^{t}_{\text{SCH}} (l',u',q')$ for a delay t or $(l,u,q) \longrightarrow^{\text{finish}}_{\text{SCH}} (l',u',q')$. We define that (l,u,q) is reachable for $(l_0,u_0,)$ with the scheduling strategy SCH by $(l_0,u_0,) \longrightarrow^{*}_{\text{SCH}} (l,u,q)$.

As all deadlines in task automaton are hard, we define schedulability for a given scheduling strategy as an impossibility of reaching a state where some deadline is missed. We use $q_{\ell err}$ to denote queues containing a task instance $(T_i, D_i, O_i, P_i, R_i, B_i, W_i)$ with $D_i - W_i < 0$.

Definition 6 (schedulability). A task automaton $\mathscr{A}$ with the initial state $(l_0,u_0,)$ is nonschedulable with SCH if $(l_0,u_0,) \longrightarrow^{*}_{\text{SCH}} (l,u,q_{\ell err})$ for some l and u. Otherwise, we say that $\mathscr{A}$ is schedulable with SCH. More generally, we say that $\mathscr{A}$ is schedulable if and only if there exists a scheduling strategy SCH with which $\mathscr{A}$ is schedulable.

If each task automaton in the task set is schedulable in terms of scheduling strategy and system resources, the task set is schedulable. Fersman et al. and Krcal. [15, 16] proved that the problem of checking schedulability relative to a preemptive scheduling strategy is decidable for task automaton with fixed computation times [17]. Therefore, we fixed the execution time of the task to WCET [18] so that the task automaton is decidable.

4.3. Schedulability Analysis. One of the most interesting properties of task automaton is schedulability. In Definition 6, we define schedulability in the task automaton; then, in this section, we study the schedulability problems with cryptographic parameters [19] related to the overhead model presented in Section 3. Merely, yes or no answers about schedulability do not provide useful results to help designers understand how changes in task parameters will affect the schedulability of the system. [20] Therefore, the schedulability analysis [21, 22] should show the amount of slack in each task that can be used to plan other actions and accurately measure the allowable changes. Thus, we not only consider whether the system is schedulable but also apply sensitivity analysis [23] to provide information about how changes in task WCET parameters affect the system. Starting from a feasible task set, sensitivity analysis will provide an accurate amount of change that can be tolerated in the task calculation time to keep the task set feasible.

4.3.1. Feasibility Region in the W Space. The space where each coordinate represents a task's WCET (W_i) is called W space. When working in the W space, we assume that the variable is only W_i, and the period (T_i), deadline (D_i), and other parameters of tasks are fixed. The origin of W space is O.

In our feasible region analysis, the task set is regarded as a point in a specific space of task parameters, just as W space for WCET feasible region analysis. Therefore, the judgment of schedulability can be equivalent to verifying whether a point belongs to the feasible region of the task set. In this way, the relationship between task W_i parameters can be established, and the feasible value of W_i for a specific task can be determined. In particular, using the FPS scheduling strategy, the feasible region $\mathscr{M}_n$ of the task set Γ is defined in the following way:

$$\mathscr{M}_n(T_1,\ldots,T_n,D_1,\ldots,D_n)=\{(W_1,\ldots,W_n)\in R_+^n:\ \Gamma_n \text{ is schedulable by FPS.}\} \tag{5}$$

Lehoczky et al. [24] completed the first attempt to analyze and characterize the $\mathscr{M}_n$ region in W space. In their work, the deadline is assumed to be equal to the period, and they proved the following theorem:

Theorem 1. *Given a periodic task set Γ under fixed priorities,*

(i) τ_i is feasibly schedulable if and only if

$$L_i=\min_{t\in S_i}\frac{\sum_{j=1}^{i}\lceil t/T_j\rceil W_j}{t}\le 1, \tag{6}$$

where $S_i=\{rT_j:\ j=1,\ldots,i,\ r=1,\ldots,\lfloor T_i/T_j\rfloor\}$.

(ii) The entire task set is feasibly schedulable if and only if

$$\max_{i=1,\ldots,n} L_i\le 1. \tag{7}$$

Theorem 2. *This can be restated in a more expressive form:*

$$\begin{aligned}&\max_{i=1,\ldots,n}\min_{t\in S_i}\frac{\sum_{j=1}^{i}\lceil t/T_j\rceil W_j}{t}\le 1\\ \Leftrightarrow&\bigwedge_{i=1,\ldots,n}\min_{t\in S_i}\frac{\sum_{j=1}^{i}\lceil t/T_j\rceil W_j}{t}\le 1\\ \Leftrightarrow&\bigwedge_{i=1,\ldots,n}\bigvee_{t\in S_i}\frac{\sum_{j=1}^{i}\lceil t/T_j\rceil W_j}{t}\le 1.\end{aligned} \tag{8}$$

Theorem 3. *This provides a way to express the feasible region of $\mathscr{M}_n$ under rate monotonic (RM) priorities, which is optimal in static scheduling [25]. It can be expressed by the following theorem in the form of logical operators:*

Theorem 4. *When D_i is equal to T_i, the region of the schedulable task sets $\mathscr{M}_n$ is defined as follows:*

$$\mathscr{M}_n(T_1,\ldots,T_n,D_1,\ldots,D_n)\mid_{D_i=T_i}=\left\{(W_1,\ldots,W_n)\in R_+^n:\ \bigwedge_{i=1,\ldots,n}\bigvee_{t\in S_i}\sum_{j=1}^{i}\lceil\frac{t}{T_j}\rceil W_j\le t\right\}, \tag{9}$$

where $S_i = \left\{ [rT_j \colon j = 1, \ldots, i, r = 1, \ldots, \left\lfloor T_i/T_j \right\rfloor] \right\}$.

Bini and Buttazzo [26] simplified the above theorem and significantly reduced the number of equations required to define the $\mathscr{M}_n$ region. Theorem 3 is proposed to facilitate the establishment of the relationship between the task set parameters, by which we can infer the feasible value of W_i.

$$\mathscr{M}_n(T_1, \ldots, T_n, D_1, \ldots, D_n) | D_i = T_i = \left\{ (W_1, \ldots, W_n) \in R_+^n \colon \bigwedge_{i=1,\ldots,n} \bigvee_{t \in \text{sched } P_i} \sum_{j=1}^{i} \lceil \frac{t}{T_j} \rceil W_j \le t \right\}, \tag{10}$$

where Sched P_i *is a set of scheduling points defined as sched* $P_i = \mathscr{P}_{i-1}(D(i))$, *and* $\mathscr{P}_i(t)$ *is given by the expression as follows:*

$$\begin{cases} \mathscr{P}_0(t) = \{t\}, \\ \mathscr{P}_i(t) = \mathscr{P}_{i-1}\left(\lfloor \frac{t}{T_i} \rfloor T_i \right) \cup \mathscr{P}_{i-1}(t). \end{cases} \tag{11}$$

4.3.2. Sensitivity Analysis in W Space. Changes to the task content in RTOS often lead to changes in the schedulability of the task set. In many cases, people hope to accurately describe the slack degree of task scheduling for system scheduling problems. At this time, sensitivity analysis [27–29] is very useful. For example, when a developer wants to add a new function to a task, sensitivity analysis can give the allowable W_i range of the task, which can help change the task without affecting the schedulability of the task.

Continuing to focus on W space, a task set Γ can be expressed as a point W_Γ in W space, where $W_\Gamma = (W_1, \ldots, W_n)$. Changes to tasks in the task set Γ will lead to linear changes of W_i, and the modified point $W_{\Gamma'}$ can be expressed in the following formal form:

$$W_{\Gamma'} = W_\Gamma + \varepsilon \vec{d}. \tag{12}$$

$\vec{d}$ is a non-negative vector used to set the direction of change, and ε is a scalar used to measure the change from the original position. In this paper, only the schedulability after the modification of the original schedulable task set is considered, so $\varepsilon > = 0$.

When the change in the task set is determined, and $\vec{d}$ is also determined accordingly, the satisfaction of the task set Γ' schedulability condition is equivalent to restricting $W_{\Gamma'}$ in the feasible region $\mathscr{M}_n$ so that the range of slack ε can be determined. If we denote the first i elements of $\vec{d}$ by d_i and W_i' is consisted of the first i WCET in Γ', then Γ' is schedulable if and only if:

Theorem 5. *Given a periodic task set* Γ *under fixed priorities, when* D_i *is equal to* T_i, *its feasible region* $\mathscr{M}_n$ *can be defined as follows:*

$$\begin{aligned} & \bigwedge_{i=1,\ldots,n} \bigvee_{t \in \text{sched } P_i} \vec{n_i} \cdot \vec{W_i'} \le t \\ \Leftrightarrow & \bigwedge_{i=1,\ldots,n} \bigvee_{t \in \text{sched } P_i} \vec{n_i} \cdot \left(\vec{W_i} + \varepsilon \vec{d_i} \right) \le t \\ \Leftrightarrow & \bigwedge_{i=1,\ldots,n} \bigvee_{t \in \text{sched } P_i} \varepsilon \le \frac{t - \vec{n_i} \cdot \vec{W_i}}{\vec{n_i} \cdot \vec{d_i}} \\ \Leftrightarrow & \varepsilon \le \min_{i=1,\ldots,n} \max_{t \in \text{sched } P_i} \frac{t - \vec{n_i} \cdot \vec{W_i}}{\vec{n_i} \cdot \vec{d_i}}, \end{aligned} \tag{13}$$

where $\vec{n_i} = (\lceil t/T_1 \rceil, \lceil t/T_2 \rceil, \ldots, \lceil t/T_{i-1} \rceil, 1)$.

This formula provides the maximum linear slack along the $\vec{d}$ direction.

Changing a task τ_k is considered in a given schedulable task set Γ, and its WCET will change accordingly, $W_k' = W_k + \Delta_k$, the direction vector $\vec{d} = (0, \ldots, 1, \ldots 0)$ in the change. Except the k-th bit, which is set to 1, all other bits are all 0. We can use the above formula to calculate the maximum value of Δ_k. The actual meaning of Δ_k is the maximum WCET increase amount allowed by task τ_k under schedulability.

$$\Delta W_k^{\max} = \min_{i=k,\ldots,n} \max_{t \in \text{sched } P_i} \frac{t - \vec{n_i} \cdot \vec{W_i}}{\lceil t/T_k \rceil}, \tag{14}$$

where $\vec{n_i} = (\lceil t/T_1 \rceil, \lceil t/T_2 \rceil, \ldots, \lceil t/T_{i-1} \rceil, 1)$.

4.3.3. CPU Utilization Constraints. The analysis in Section 2 is based on a situation where no additional constraints are imposed. Now, adding utilization constraints to a single CPU system is considered. The ratio $U_i = W_i/T_i$ is called the utilization of task τ_i and represents the processor clock cycles used by that task, and the value is as follows:

$$U_p = \sum_{i=1}^{n} U_i. \tag{15}$$

U_p is called total processor utilization; it represents the proportion of processor time used by the periodic task set. Obviously, if $U_p > 1$, there is no feasible plan for this task set. So, we can set the CPU utilization of the original task set as follows:

$$U_{ori} = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{W_i}{T_i}. \tag{16}$$

Then, after changing the task k, the new CPU utilization is as follows:

$$U_{\text{new}} = \frac{W_1}{T_1} + \cdots + \frac{W_k + \Delta W_k}{T_k} + \cdots + \frac{W_n}{T_n}. \tag{17}$$

With constrained $U_{\text{new}} <= U_{\max}$, $U_{\max}$ is the constraints of CPU utilization, when $U_{\text{new}} > 1$, the CPU is overloaded. Obviously, this configuration is not feasible. The constraint of ΔW_k under this condition is

$$\Delta W_k \le T_k \left(U_{\max} - U_{ori}\right). \tag{18}$$

Combined with the sensitivity analysis in Section 4.3.2, under the condition of CPU utilization constraints, the maximum value of the task k whose execution time changes is

$$\Delta W_k^{\max} = \min\left\{ \min_{i=k,\ldots,n} \max_{t \in \text{sched } P_i} \frac{t - \vec{n_i} \cdot \vec{W_i}}{\lceil t/T_k \rceil}, T_k\left(U_{\max} - U_{ori}\right) \right\}, \tag{19}$$

where $\vec{n_i} = (\lceil t/T_1 \rceil, \lceil t/T_2 \rceil, \ldots, \lceil t/T_{i-1} \rceil, 1)$.

4.3.4. Throughput Evaluation of Cryptographic Algorithms. In Section 3, we propose a model for estimating the CPU clock cycle cost of a specific configuration of the cryptographic algorithm on the chip and the corresponding instruction set. In Sections 4.3.1–4.3.3, we introduce a sensitivity analysis method for determining the parameter feasible region of a task set. Then, in this section, we combine the model and the sensitivity analysis method to give the evaluation index of the application of the cryptographic algorithm in RTOS: throughput. It can reflect the performance of cryptographic algorithms.

First, Formula (3) is simplified; when the block cipher is the subject of investigation, we have

$$T_{\text{exec}}(\text{text\_len}) = T_{\text{misc}} + T_{\text{block}} \cdot \lceil \frac{\text{text\_len}}{\text{block\_size}} \rceil. \tag{20}$$

Encrypting and decrypting a piece of content with a length of text_len is considered as the added part of the task k. From the previous formula, we know

$$\begin{aligned} & T_{\text{exec}}(\text{text\_len}) \le \Delta W_k^{\max} \\ \Leftrightarrow & T_{\text{misc}} + T_{\text{block}} \cdot \lceil \frac{\text{text\_len}}{\text{block\_size}} \rceil \le \Delta W_k^{\max} \\ \Leftrightarrow & \text{text\_len} \le \frac{\left(\Delta W_k^{\max} - T_{\text{misc}}\right) \cdot \text{block\_size}}{T_{\text{block}}}. \end{aligned} \tag{21}$$

The throughput of a cryptographic algorithm is defined as the length of plaintext/ciphertext. It processes in a unit time. Then, according to the above formula, it can be concluded that in RTOS, when a fixed priority scheduling strategy is used on the premise that the task set is schedulable and the CPU occupancy rate is within $U_{\max}$, a specific configuration of cryptographic algorithm protection for task k is added; then, the upper bound of throughput can be expressed as follows:

$$TPS_{\text{Crypto}} = \frac{\text{text\_len}}{T_k} \le \frac{\left(\Delta W_k^{\max} - T_{\text{misc}}\right) \cdot \text{block\_size}}{T_k \cdot T_{\text{block}}}. \tag{22}$$

Among them, T_{misc} is the initial cost of encrypting and decrypting each piece of text, block_size is the size of each block; taking AES as an example, there are three choices of 128, 192, and 256 (in bits), and T_{block} is the time to process each block.

5. Experimental Method

This section explains the experimental steps of this paper, in addition to the description of the automata model in detail.

Our research goal is to determine the schedulability changes on a single-core processor before and after including a cryptographic algorithm to a specific task set; then, we can determine the applicability of the cryptographic algorithm on this hardware platform under this configuration. The process is outlined as follows:

(i) Step 1. Use the cryptographic overhead estimation model in Section 3 to evaluate a cryptographic algorithm on the target platform at the instruction level and get the estimated overhead of the cryptographic algorithm under a certain configuration.

(ii) Step 2. Perform sensitivity analysis on the original task set in RTOS and analyze the allowable variation of the task WCET under the guarantee of schedulability. The other parameters of the task are fixed.

(iii) Step 3. Add the estimated overhead of the cryptographic algorithm in Step 1 to the task and judge whether the WCET of this task meets the schedulability constraint analyzed in Step 2.

(iv) Step 4. Model the task scheduling of RTOS according to the content of Section 4, construct a set of task automata models with time constraints, set parameters for the automata according to the

research case, and verify the schedulability of the task set through the automata operation.

(v) Step 5. Port RTOS to the embedded platform and set task attributes. After running the system, use the Tracealyzer [30] to track the execution of the tasks and compare with the results in the previous steps.

(vi) Step 6. Compare and analyze the experimental data to judge the accuracy of this process and the feasibility of this method.

The fundamental idea of the verifying step is to convert the schedulability analysis problem into the reachability problem of timed automata and use the real-time model validator Uppaal to find the WCRT and processor utilization, and check whether all deadlines are satisfied. In the response time analysis, to prove schedulability, it is sufficient to calculate the WCRT of each task and compare it with the deadline [31]. If the WCRT of each task does not exceed the corresponding deadline, then the system is schedulable and vice versa. Some tasks are periodic, and some are sporadic, but we simplify the model by treating all tasks as periodic.

We use STM32F103VE (Cortex-M3) [32] as a development platform and port FreeRTOS [33] as the operating system on this platform, which employs a preemptive fixed priority scheduler and a time slice round scheduler under the same priority. In a hard RTOS, a task that misses the deadline means that the system is not schedulable. The workflow of our experiment is shown in Figure 5.

As the cryptographic overhead prediction, automaton theory, and sensitivity analysis are mentioned in the former sections, we will introduce the model construction of task automata in detail.

We use Uppaal [34] as a formal verification tool to verify our schedulability analysis model by using verification queries [35]. We use a stopwatch to record the progress of the task, and the stopwatch will stop running when the task is preempted. In Uppaal, the stopwatch is implemented as clock derivatives, where the running rate can be set to 1 or 0.1 means the clock runs at a normal speed, and 0 means the clock stops. Syntactically, the stopwatch expression appears in the invariant constraint of the node as $x' == c$, where x is declared as a clock type and c is an integer. By verifying whether the stopwatch satisfies the clock constraints, you can check whether the task meets the real-time restrictions of the system.

5.1. The Uppaal Framework. The Uppaal framework [36] consists of the following process models: a fixed-priority preemptive scheduling strategy, a system resource model, a task model, and a model to ensure global invariance. The template we provide for the task model is parameterized using an explicit sequence of task attributes, and we can customize the protocols that the tasks share on specific resources [37]. The main modeling elements are summarized below:

(i) A system resource sharing template. The system has two states: Idle and Inuse. The states switch through the interaction with tasks and scheduling strategy; therefore, the resource utilization rate can be calculated;

(ii) A scheduling strategy template to process tasks in the queue according to different scheduling strategies;

(iii) A task template that can set attributes according to the definition in Section 4.2 task automaton. Each task uses the following clocks and data variables:

(iv) the task and its clocks are parameterized by the identifier id; the local clock **x** is used to control the execution status of its own tasks, and the global clock time [id] is used to control the execution status of the task set in the entire system.

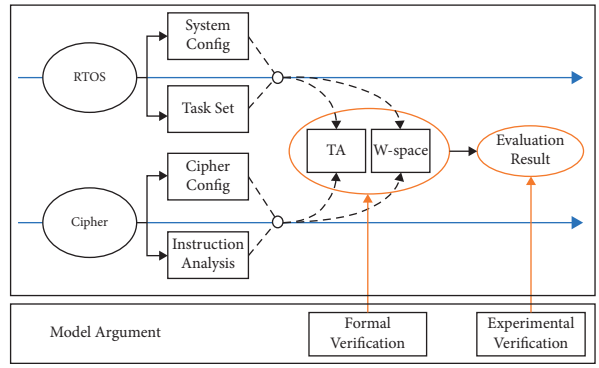

Figure 5: Workflow diagram.

5.2. Resource and Scheduler Model. Figures 6 and 7 show a model of system resources (CPU) and scheduling strategy. First, the system resource model is initialized. Since no user task is running, the CPU runs idle tasks and is in the Idle position. When a task is ready, it sends the signal ready [id]! on the resource preparation channel so that the system resource moves from Idle to an intermediate state position and then sends the signal insert_task [policy]! on the channel to insert the new task into the scheduling queue. Then, the scheduling strategy model FPS receives the signal. If the scheduling queue is not empty, the priority of the current task is compared with the priority in the queue in turn and then inserted into the task queue by insert_task_in_buffer (); otherwise, the current task is directly inserted into the head position of the task queue by insert_at (0, ready_task, id). Then, the resource selects the highest priority task from the task queue and enters the running state position Inuse.

The task releases the CPU through the signal finished [id]!. In this case, the scheduling policy model removes the task from the task queue by removeTask (). If the task queue is empty at this time, the CPU runs the idle task and moves to the Idle position. If it is not empty, the highest priority task is extracted from the task queue, and the CPU is still in the running state, Inuse position. It also uses the stopwatch usedTime to calculate the CPU usage time and then to calculate the total CPU load as used Time/global Time.

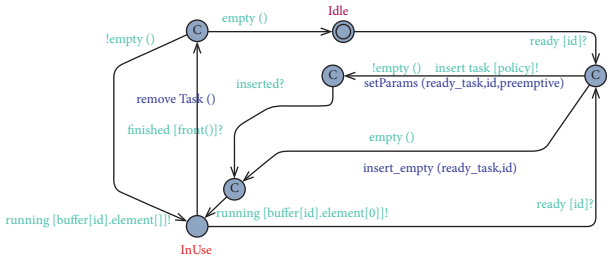

Figure 6: Resource.

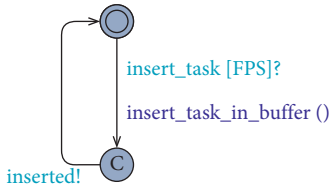

Figure 7: Scheduler.

5.3. Task Template. Figure 8 shows the task model. The task model is built according to the task automaton template proposed in Section 4.2. Each task is instantiated by in it_off set, min_period, max_period, off set, dead line, BCET, WCET, resource, and priority. First, the task is initialized; after initialization, a new cycle is started, and it is on the WaitingOffset position and waits for the initial offset. Then, the task is on the WaitingDepedency position and waits for the end of the predecessor task it depends on. When the dependency constraint is reached, the task will send ready[id]! to the CPU resource. The system resource and scheduling strategy will insert the task into the queue, and the task will enter the Ready position. After that, the task will switch between Running and Suspend according to its position in the task scheduling queue, at a different rate of local clock x. When completed, the task will send finished[id]! to the CPU resource, and the system resource and scheduling policy will remove the task from the scheduling queue. At last, the task is completed; the periodic task will continue to start a new cycle, and the nonperiodic task will enter the Done position.

5.4. Verification Queries. We use the following query statement to verify schedulability:

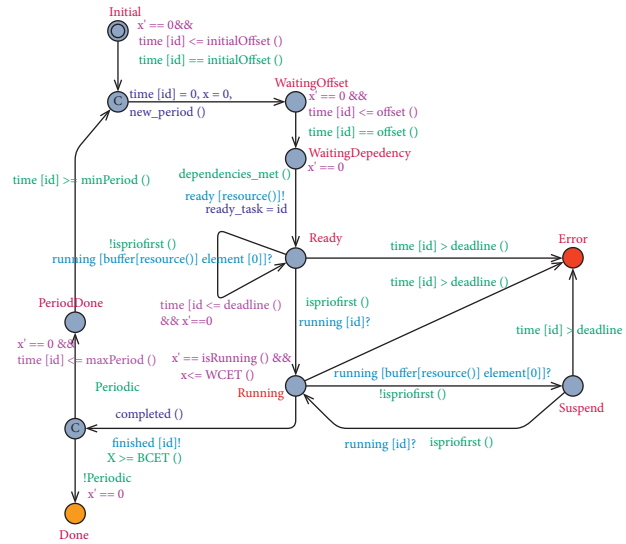

Figure 8: Task template.

(i) Check the schedulability of the system (the error location is not reachable):

A[] forall (i: t_id) not Task (i).Error

(ii) Find the total worst CPU usage:

sup: usedTime, idleTime, globalTime

(iii) Find the worst-case response times in n tasks:

sup:WCRT[0], ..., WCRT[n-1]

6. Case Study and Data Analysis

We present a case study to evaluate our method. The original task set configuration is shown in Table 3. The column attributes include WCET, period, deadline, priority, scheduled point, and CPU utilization of each task. Now, adding the AES algorithm to task 3 is considered to protect it without affecting its schedulability. Task 3 needs to encrypt 5.0kb of data and send it in each period. And the CPU occupancy rate is required to be less than 95%. Under this condition, we evaluate the feasibility of the AES algorithm with different configurations. Figure 9 shows the algorithm's flow diagram for the analysis of tasks related to AES.

Sensitivity analysis is performed on the original task set, and the maximum allowable change in WCET of task 3 is calculated. According to the formula, we have

$$\Delta W_3^{\max} = \min\left\{ \max_{t \in sche\ dP_3} \frac{t - \vec{n_3} \cdot \vec{W_3}}{t/T_3}, T_3 \cdot \left(U_{\max} - U_{ori}\right) \right\},$$

$$\Delta W_3^{\max} = \min\{\max\{150000 - 148539, 160000 - 158589, 180000 - 188739, 200000 - 198789\}, 3636\}, \tag{23}$$

$$\Delta W_3^{\max} = 1461.$$

Table 3: Task set configuration.

i	Wi (μ s)	Ti (ms)	Di (ms)	Pi	SchedPi (ms)	Ui (%)
1	10 050	30	30	3	{30}	33.5
2	30 150	80	80	2	{30, 60}	37.69
3	37 989	200	200	1	{150, 160, 180, 200}	18.99

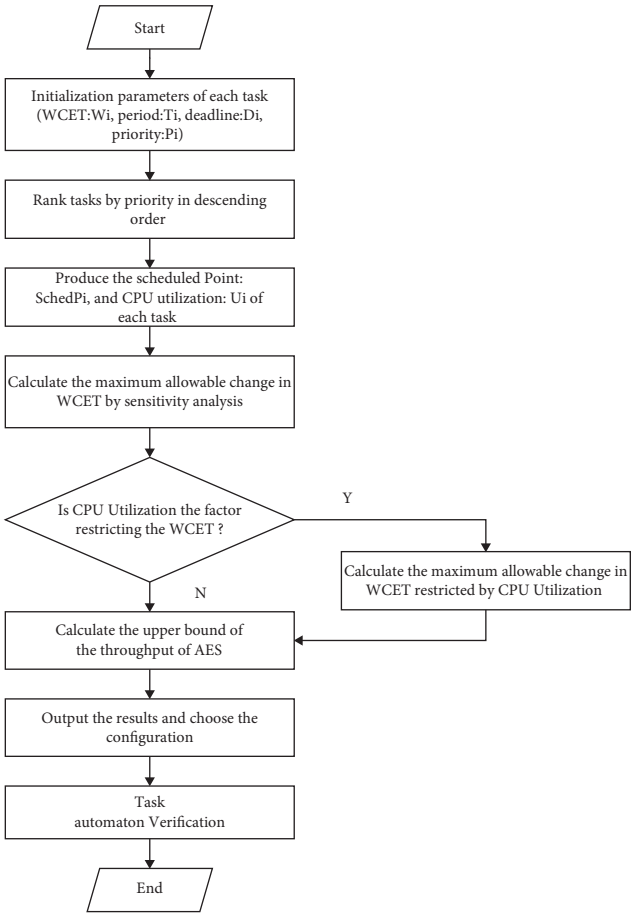

Figure 9: AES-related analysis process.

We can see that in the sensitivity analysis of this task set, the factor restricting the WCET of task 3 is schedulability rather than CPU utilization. After calculating the maximum change in WCET of task 3, we calculate the upper bound of the throughput of different configurations of AES under this constraint, which can be expressed as

$$TPS_{AES} \leq \frac{\left(\Delta W_3^{\max} - T_{\text{misc}}\right) \cdot \text{block\_size}}{T_3 \cdot T_{\text{block}}}. \tag{24}$$

Through formula (24) and the instruction-level overhead prediction model in Section 3, we can calculate the upper throughput bound of AES with different key lengths and encryption modes in the task set of the case. As a reference, it can be a performance measurement for the selection of cryptographic algorithms and encryption mode. For example, for the configuration of AES-192-CBC, the upper bound of its throughput rate is 27.52kb/s, and for AES-256-CBC, the value is 23.68kb/s.

The encryption requirement of task 3 is 5.0kb per period, so the required throughput rate is 25kb/s. According to the sensitivity analysis, we can conclude that AES-192-CBC is feasible but AES-256-CBC is not. Next, we use the task automaton model to verify this conclusion.

The results of our model-based framework include three parts: task set schedulability judgment, worst-case response time estimation, and cycle-limited verification benchmark CPU utilization estimation.

Schedulability. In the task automata model, the task state that violates the time constraints will be transferred to Error. By retrieving the state space list, the formal verification tool Uppaal can transform the schedulability problem into the automata reachability problem. By using query statement 1 in Section 5.4, the correctness of our analysis results has been verified.

Worst-case response time. In the response time analysis, the WCRT of each task is calculated and compared with the deadline. This is a visual proof of schedulability that if the WCRT of each task does not exceed the corresponding deadline, the system is schedulable. Table 4 compares the WCRT of the original task set and the task set after adding AES-192-CBC and AES-256-CBC, respectively. Task 3, which adds AES-256-CBC, will enter the Error state in Uppaal, and the task will be terminated, so its WCRT cannot be estimated.

CPU utilization. In the periodic task model of RTOS, tasks do not have a fixed progress measurement. To control the running progress of tasks and even the entire task set, we use the cycle count to indicate the number of cycles up to the current time. When the cycle count reaches a preset upper limit (CYCLELIMIT), it is reset to 0 along with all global clocks.

Before resetting the cycle counter Cycle, exploring the hyperperiod (the least common multiple of periods) of all periodic tasks is an effective way to simplify the set of infinite periodic tasks. There are three task periods in the case: 30000μs, 80000μs, and 200000μs, so the potential hyperperiod is 1200000μs. If the global cycle (CYCLE) is set to 120000μs, the number of hyperperiods is 10 global cycles. The results are shown in Table 5.

In this case, the CPU utilization is not the major factor limiting the task WCRT, so only the CPU utilization of the task set using AES-192-CBC under different global cycles is shown. The CPU utilization is verified by query 2 in Section 5.4. When the cycle counter Cycle is 1 to 10; that is, during the first hyperperiod, the CPU utilization fluctuates. When the cycle counter Cycle is 10, the CPU utilization stabilizes to 90.861% for the first time, and the fluctuation of the CPU utilization gradually decreases. When the cycle counter Cycle is a multiple of 10, that is, when the time window taken is an integer number of hyperperiod, the CPU utilization is stable at 90.861%.

6.1. STM32 Experimental Analysis. We port FreeRTOS10.0.0.1 on the STM32F103VE platform and implement the case. We record the experimental results to judge the accuracy of our method and the correctness of task automata verification. The results are shown in Figures 10–13.

Table 4: Task set WCRT verification results.

Task set	Task	Period (μs)	Deadline (μs)	WCET (μs)	WCRT (μs)
Original task set	1	30 000	30 000	10 050	10 050
	2	80 000	800 000	30 150	50 250
	3	200 000	200 000	37 989	148 539
Task set with AES-192-CBC	1	30 000	30 000	10 050	10 050
	2	80 000	800 000	30 150	50 250
	3	200 000	200 000	39 347	149 897
Task set with AES -256-CBC	1	30 000	30 000	10 050	10 050
	2	80 000	800 000	30 150	50 250
	3	200 000	200 000	39 529	—

Table 5: Task set (with AES-192-CBC) CPU utilization.

Cycle limit	Idle (μs)	Used (μs)	Global (μs)	Utilization (%)
1	0	120 000	120 000	100
2	103	239 897	240 000	99.957
3	10 156	349 844	360 000	97.179
4	29 906	450 094	480 000	93.770
5	39 959	560 041	600 000	93.340
6	50 062	669 938	720 000	93.047
7	69 762	770 238	840 000	91.695
8	79 865	880 135	960 000	91.681
9	79 865	1000 135	1080 000	92.718
10	109 668	1090 332	1200 000	90.861
12	109 771	1330 229	1440 000	92.377
15	149 627	1650 373	1800 000	91.673
18	189 533	1970 467	2160 000	91.687
20	219 336	2180 664	2400 000	90.861
30	329 004	3270 996	3600 000	90.861
40	438 672	4361 328	4800 000	90.861

Through Tracealyzer's tracking results, we can see that in the original task set and the task set of adding AES-192-CBC and AES-256-CBC, the overall CPU usage is stable at 89.6%, 90%, and 90.6%, respectively. These are similar to automata estimation and are all lower than the required values because in this case, the CPU usage is not the key factor affecting feasibility. In Figure 13, we can see that, compared to task 3, to which the AES algorithm is directly added, the WCRT of Tasks 1 and 2 has no major changes before and after the application of the AES algorithm. There are only small fluctuations, and they are all within their deadline. It can prove the schedulability of these two tasks. For Task 3, in the original task set and after the application of AES-192-CBC, its WCRT is within the deadline and can be scheduled; but after the AES-256-CBC is added, its WCRT exceeds the deadline range, and Task 3 cannot be scheduled. This is also consistent with our analysis and formal verification results.

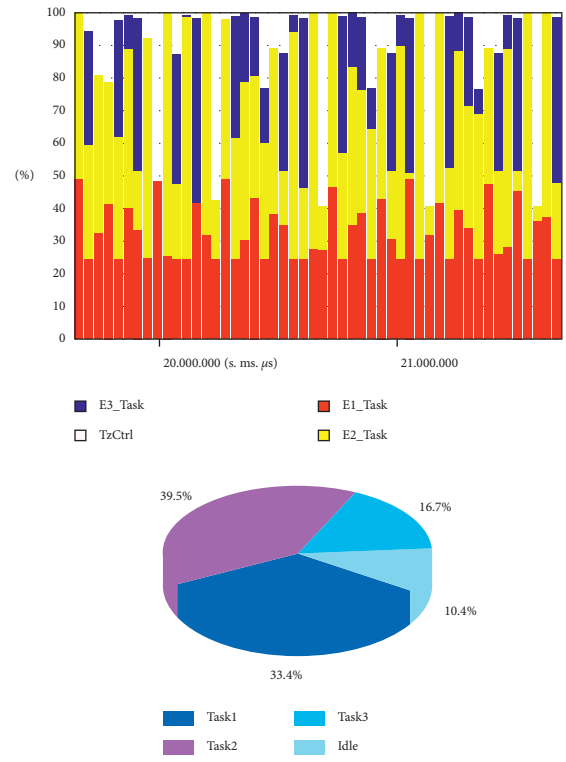

Figure 10: Original CPU utilization.

7. Related Work

7.1. Cipher Overhead Evaluation. Liu et al. [4] proposed a mathematical model based on the basic operation cost frequently used in cryptographic algorithms to predict the cost of these algorithms. Since these basic operations cannot be ignored in implementation, the prediction result can be seen as the lower bound of the overhead. Bauer and Freiling [38] gave an overview of cycle-accurate simulation problems and showed a method for cycle-accurate ARM Thumb-2 simulators. The classification of the Thumb-2 instruction set was used for reference. Granelli and Boato [5] proposed a method to evaluate the complexity and computational cost of different block ciphers that are independent of the platform on which they are implemented. As a representative of resource-constrained devices, the cost evaluation and measurement of cryptographic algorithms on wireless sensor networks [2, 6, 39–41] have been widely studied.

7.2. Schedulability Analysis. Yalcinkaya et al. [42] determined the sensitivity of task execution time. The paper measured the influence of parameters on schedulability and proved its correctness and optimality. Punnekkat et al. [43] provided a general method for sensitivity analysis of the task set. This method can help developers integrate changes to the system while ensuring that schedulability remains

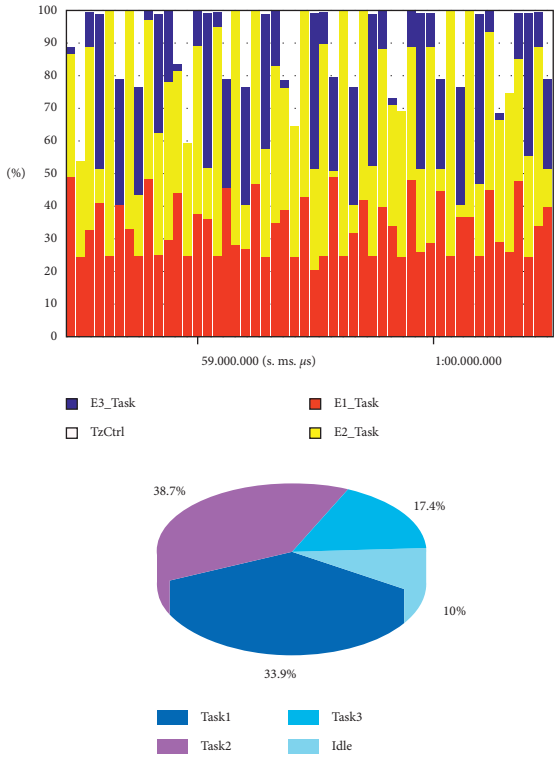

Figure 11: AES-192-CBC CPU utilization.

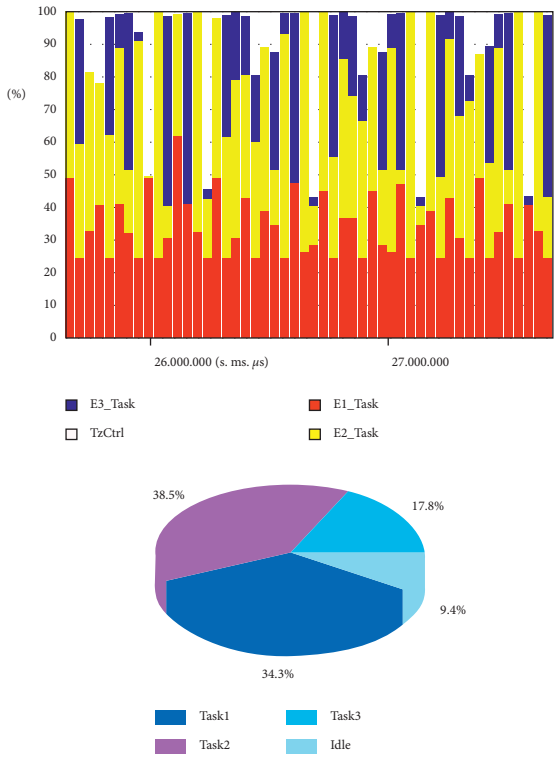

Figure 12: AES-256-CBC CPU utilization.

unchanged. Bini and Buttazzo [26] proposed a novel method to analyze the schedulability of periodic tasks under RM priority allocation [25]. This method can accurately describe

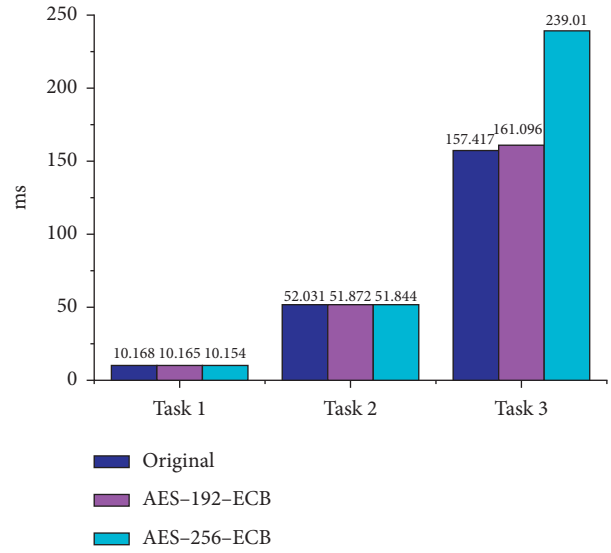

Figure 13: Task set WCRT comparative analysis.

the feasible region in the task calculation time space (C space). Afterwards, they [44] summarized a theoretical method for the sensitivity analysis of a real-time system containing a set of periodic tasks. The proposed method allows designers not only to verify the feasibility of the application but also to determine the specific operations to be performed on the design variables to ensure the schedulability of the task set or improve resource utilization.

7.3. Formal Verification. Fersman et al. [15] provided a model, task automaton, for real-time systems with non-uniform repetitive computing tasks. It is an extended version of the timed automaton, which can be used to specify resource constraints and strict timing restrictions in computation. The paper proves that the schedulability check problem related to preemptive scheduling is decidable for task automaton. Mikučionis et al. [45] proposed a modeling framework that uses Uppaal real-time model checker to perform schedulability analysis. Contributions included the modeling framework, its application in industrial case studies, and the comparison of the results with classical response time analysis.

The related work is elaborated in three parts: the overhead estimation analysis and modeling of cryptographic algorithms on embedded platforms, task schedulability and sensitivity analysis in RTOS, and the establishment of RTOS models based on timed task automaton. The related research results in the three fields are introduced, respectively. However, as far as we know, there is no method to analyze the application feasibility of cryptographic algorithm in the RTOS task set, which stimulates the thinking and research in this paper.

8. Conclusions and Future Work

In this paper, we analyze the instruction-level overhead of commonly used symmetric cryptographic algorithms on a chip, establish an automaton model based on the task set in

RTOS, and introduce the parameter-based sensitivity analysis. In the experimental part, for a concrete case, we conducted a feasibility analysis of AES in different modes, and then, we construct a model for RTOS on a single-core CPU with a fixed priority preemptive scheduling strategy using Uppaal and verify its schedulability before and after adding a cryptographic algorithm for tasks. Comparing the results with the running data on a real embedded RTOS tracked by the tracealyzer tool, it shows that the estimation results of our method are substantially consistent with the actual data. It is proved that this method can effectively estimate the impact of a cryptographic algorithm on RTOS system task scheduling and provide a quantitative index for the application of cryptographic algorithms on embedded platforms.

In future work, we intend to establish a more accurate estimation model for the overhead of cryptographic algorithms and incorporate more cryptographic algorithms into our evaluation model. We also plan to conduct a sensitivity analysis of other factors affecting RTOS scheduling and study the impact of different scheduling strategies, processor frequency, jitter, and deadlines on models and analysis methods, to build a model that can describe the running process of tasks more accurately.

Data Availability

The figures, tables, codes, and models used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

We thank the support of the National Foundation for Development of Cryptography, under grant no. MMJJ20180211.

References

[1] G. C. Pereira, R. C. Alves, F. L. D. Silva, R. M. Azevedo, B. C. Albertini, and C. B. Margi, "Performance Evaluation of Cryptographic Algorithms over Iot Platforms and Operating Systems," *Security and Communication Networks*, vol. 2017, Article ID 2046735, 16 pages, 2017.

[2] W. Liu, R. Luo, and H. Yang, "Cryptography overhead evaluation and analysis for wireless sensor networks," in *Proceedings of the 2009 WRI International Conference on Communications and Mobile Computing*, vol. 3, pp. 496–501, Kunming, China, January 2009.

[3] B. J. Mohd and T. Hayajneh, "Lightweight block ciphers for iot: energy optimization and survivability techniques," *IEEE Access*, vol. 6, Article ID 35966, 2018.

[4] W. Liu, B. Ying, H. Yang, and H. Wang, "Accurate modeling for predicting cryptography overheads on wireless sensor nodes," in *Proceedings of the 2009 11th International Conference on Advanced Communication Technology*, vol. 2, pp. 997–1001, Phoenix Park, February 2009.

[5] F. Granelli and G. Boato, "A Novel Methodology for Analysis of the Computational Complexity of Block Ciphers: Rijndael, Camellia and Shacal-2 Compared," Technical Report DIT-04-004, University of Trento, Trento, Italy, 2003.

[6] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, and M. Sichitiu, "Analyzing and modeling encryption overhead for sensor network nodes," in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pp. 151–159, San Diego, CA, USA, September 2003.

[7] ARMDeveloper, "Cortex-m3 technical reference manual," 2021, https://developer.arm.com/documentation/ddi0337/h.

[8] STMicroelectronics, "Stm32 cryptographic user manual," 2021, https://www.st.com/resource/en/user_manual/cd00208802-stm32-cryptographic-library-stmicroelectronics.pdf.

[9] J. Daemen and V. Rijmen, "Aes Proposal: Rijndael," 1999, https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf.

[10] P. Schwabe and K. Stoffelen, "All the aes you need on cortex-m3 and m4," in *Proceedings of the International Conference On Selected Areas In Cryptography*, pp. 180–194, Newfoundland & Labrador (Memorial University of Newfoundland), St. John's, Canada, October 2017, https://sacworkshop.org/SAC-history.html.

[11] P. Prasithsangaree and P. Krishnamurthy, "Analysis of energy consumption of rc4 and aes algorithms in wireless lans,"vol. 3, pp. 1445–1449, in *Proceedings of the GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, vol. 3, IEEE, San Francisco, CA, USA, December 2003.

[12] W. Damm and H. Hermanns, "Computer Aided Verification," in *Proceedings of the Computer Aided Verification: 19th International Conference, CAV 2007*, vol. 4590, Springer, Berlin, Germany, July 2007.

[13] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal—a tool suite for automatic verification of real-time systems," in *Proceedings of the International Hybrid Systems Workshop*, pp. 232–243, Springer, New Brunswick, NJ, USA, October 1995.

[14] J. Bengtsson and W. Yi, "Timed Automata: Semantics, Algorithms and Tools," in *Proceedings of the Advanced Course On Petri Nets*, pp. 87–124, Springer, Bad Honnef, Germany, September 2003.

[15] E. Fersman, P. Krcal, P. Pettersson, and W. Yi, "Task automata: schedulability, decidability and undecidability," *Information and Computation*, vol. 205, no. 8, pp. 1149–1172, 2007.

[16] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Schedulability analysis of fixed-priority systems using timed automata," *Theoretical Computer Science*, vol. 354, no. 2, pp. 301–317, 2006.

[17] P. Krčál and W. Yi, "Decidable and undecidable problems in schedulability analysis using timed automata," in *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 236–250, Springer, Barcelona, Spain, April 2004.

[18] B. Fang, G. Li, D. Sun, and H. Cai, "Schedulability analysis of timed regular tasks by under-approximation on wcet, Dependable Software Engineering: Theories, Tools, and Applications," in *Proceedings of the International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pp. 147–162, Springer, Beijing, China, November 2016.

[19] Y. Sun, R. Soulat, G. Lipari, É. André, and L. Fribourg, "Parametric schedulability analysis of fixed priority real-time distributed systems," in *Proceedings of the International Workshop on Formal Techniques for Safety-Critical Systems*, pp. 212–228, Springer, Shenzhen, China, November 2013.

[20] A. Burns, "Preemptive priority based scheduling: an appropriate engineering approach," *Principles of Real-Time Systems*, Prentice Hall, Hoboken, NJ, USA, 1993.

[21] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[22] R. Yerraballi, R. Mukkamala, K. Maly, and H. Wahab, "Issues in schedulability analysis of real-time systems," in *Proceedings of the 7th Euromicro Workshop on Real-Time Systems*, pp. 87–92, IEEE, Odense, Denmark, June 1995.

[23] F. Dorin, P. Richard, M. Richard, and J. Goossens, "Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities," *Real-Time Systems*, vol. 46, no. 3, pp. 305–331, 2010.

[24] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Proceedings of the Real-Time Systems Symposium*, vol. 89, pp. 166–171, Santa Monica, CA, USA, December 1989.

[25] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[26] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1462–1473, 2004.

[27] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns, "A review of priority assignment in real-time systems," *Journal of Systems Architecture*, vol. 65, pp. 64–82, 2016.

[28] R. Racu, A. Hamann, and R. Ernst, "Sensitivity analysis of complex embedded real-time systems," *Real-Time Systems*, vol. 39, no. 1-3, 2008.

[29] F. Zhang, A. Burns, and S. Baruah, "Sensitivity analysis for edf scheduled arbitrary deadline real-time systems," in *Proceedings of the 2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 61–70, IEEE, Macau, China, August 2010.

[30] Percepio, "Tracealyzer user manual," 2021, https://percepio.com/docs/FreeRTOS/manual/index.html#Introduction___Welcome_to_Tracealyzer.

[31] M. S. AbouTrab, M. Brockway, S. Counsell, and R. M. Hierons, "Testing real-time embedded systems using timed automata based approaches," *Journal of Systems and Software*, vol. 86, no. 5, pp. 1209–1223, 2013.

[32] STMicroelectronics, "Stm32 stm32f10x user manual," 2021, https://www.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD00171190.pdf/files/CD00171190.pdf/jcr:content/translations/en.CD00171190.pdf.

[33] ARMDeveloper, "The Freertos Reference Manual," 2021, https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf.

[34] G. Behrmann, A. David, and K. G. Larsen, *A Tutorial on Uppaal 4.0*, Department of computer science, Aalborg university, Aalborg, Denmark, 2006.

[35] P. Pettersson, *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*, Department of Computer Systems Uppsala University, Sweden, Europe, 1999.

[36] A. David, J. Illum, K. Larsen, and A. Skou, "Model-based framework for schedulability analysis using uppaal 4.1," *Model-based design for embedded systems*, vol. 1, no. 1, pp. 93–119, 2009.

[37] J. Madsen, M. R. Hansen, K. S. Knudsen, J. E. Nielsen, and A. W. Brekling, "System-level verification of multi-core embedded systems using timed-automata," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 9302–9307, 2008.

[38] J. Bauer and F. Freiling, "Towards cycle-accurate emulation of cortex-m code to detect timing side channels," in *Proceedings of the 2016 11th International Conference on Availability, Reliability and Security (ARES)*, pp. 49–58, IEEE, Salzburg, Austria, September 2016.

[39] M. R. Doomun and K. Soyjaudah, "Analytical comparison of cryptographic techniques for resource-constrained wireless security," *IJ Network Security*, vol. 9, no. 1, pp. 82–94, 2009.

[40] Y. W. Law, J. Doumen, and P. Hartel, "Survey and benchmark of block ciphers for wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, no. 1, pp. 65–93, 2006.

[41] Y. Xiao, H.-H. Chen, B. Sun, R. Wang, and S. Sethi, "Mac security and security overhead analysis in the ieee 802.15. 4 wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2006, no. 1, Article ID 093830, 2006.

[42] B. Yalcinkaya, M. Nasri, and B. B. Brandenburg, "An exact schedulability test for non-preemptive self-suspending real-time tasks," in *Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1228–1233, IEEE, Florence, Italy, March 2019.

[43] S. Punnekkat, R. Davis, and A. Burns, "Sensitivity analysis of real-time task sets, Advances in Computing Science - ASIAN'97," in *Proceedings of the 3rd Asian Computing Science Conference on Advances in Computing Science*, pp. 72–82, Springer, Heidelberg, Germany, December 1997.

[44] E. Bini, M. Di Natale, and G. Buttazzo, "Sensitivity analysis for fixed-priority real-time systems," *Real-Time Systems*, vol. 39, no. 1-3, pp. 5–30, 2008.

[45] M. Mikučionis, K. G. Larsen, J. I. Rasmussen et al., "Schedulability analysis using uppaal: herschel-planck case study," in *Proceedings of the International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pp. 175–190, Springer, Heidelberg, Germany, October 2010.