Electronic Supplementary Material 3 (ESM3)

Data Analysis, Weighted Scoring Model Development Pseudocode and Implementation Details for "Machine Learning–Based Cryptographic Selection System for IoT Devices Based on Computational Resources"

Authors: Qi Ming Ng, Julia Juremi and Nor Azlina Abd Rahman
Version: v0.1 (30 October 2025)

Purpose Note:

This supplementary material includes pseudocode, implementation details, configuration of parameters, and the corresponding Python source files for analyzing raw data produced during the cryptographic simulation and data generation phase outlined in ESM 2, as discussed in the main manuscript. This document is intended for review and will be structured in accordance with the Springer journal's ESM requirements upon approval.

Contents Summary

File	Description
ESM3.docx	Main supplementary document containing pseudocode, implementation details, and supporting explanations
project_root.zip	Contains Python source files for the main data-analysis module, along with input and output folder structures. The core module (under Data_Analysis/) includes: (1) Jupyter notebook Data_Analysis_Performance.ipynb, and (2) Python script Data_Analysis_Performance.py. The input folder (Data_Simulation/Simulation Results/) and output folder (Data_Analysis/Summary/) are empty because the dataset is too large to include in this archive. Environment files in Data_ Analysis/ include generate_env_info_auto.py (environment descriptor module) and its output appendix_environment_concise.txt, plus requirements.txt (pinned packages)
README.txt	Guide for scripts execution and interpretation
Simulation Results.zip	Input data generated from the cryptographic simulation and data generation phase
Standardized.zip	Byproduct data generated from the Data_Analysis_Performance.py scripting
Summary.zip	Data generated from the Data_Analysis_Performance.py scripting

Reader Guide

This document contains organized pseudocode listings (Algorithms C.1–C.11), parameter configurations, and accompanying explanatory notes related to the implementation of data analysis and scripting for the development of a weighted scoring model, as discussed in Section 4.3 of the main text. The pseudocode prioritizes clarity and reproducibility over precise runtime equivalency. The accompanying Python scripts in the zip file offer the reference implementation utilized for data analysis and the building of the weighted scoring model in the manuscript. It is recommended to utilize the accompanying requirements.txt file or adhere to the execution environment outlined in the appendix for establishing the necessary environment before executing the attached Python scripts. The enclosed README.txt provides additional information and comprehensive instructions for utilizing the Python scripts.

Table of Contents

Section 0 – Algorithm Relationships and Workflow Overview

This section clarifies the relationship between Algorithms C.1 and C.11 utilized in the data analysis and the formulation of the weighted scoring model for cryptographic performance assessment. The section is segmented into three subsections: the first presents a narrative dependency mapping, the second features a workflow diagram illustrating the complete pipeline, and the final section contains a dependency mapping table for rapid reference.

Section 0.1 – Algorithm Dependency Mapping

This section shows the processing pipeline described in this ESM proceeds through the following algorithmic stages:

- **Algorithm C.1** initializes all required libraries and dependencies.
- **Algorithm C.2** loads the raw simulation data and requires initialization from C.1.
- **Algorithm C.3** refines the dataset by removing host-specific and duplicate metrics from C.2 output.
- **Algorithm C.4** filters the data by chosen threat and sensitivity levels, using C.3 output.
- **Algorithm C.5** validates decryption success of the filtered dataset from C.4.
- **Algorithm C.6** groups the validated dataset from C.5 into IoT device specification profiles.
- **Algorithm C.7** applies normalization and inversion of performance metrics for each group from C.6.
- **Algorithm C.8** computes encryption and decryption weighted scores using the metrics generated by C.7.
- **Algorithm C.9** aggregates the directional scores into total performance scores and feasibility results.
- **Algorithm C.10** concatenates and exports standardized results from all device-profile groups processed via C.9.
- **Algorithm C.11** selects the best-performing cryptographic configuration per device profile based on the outputs produced in C.10.

Section 0.2 – Workflow Diagram for Algorithm C.1-C.11

This section presents the sequential data-processing and scoring workflow executed by Algorithms C.1 through C.11. The complete pipeline is illustrated in Fig. 1.

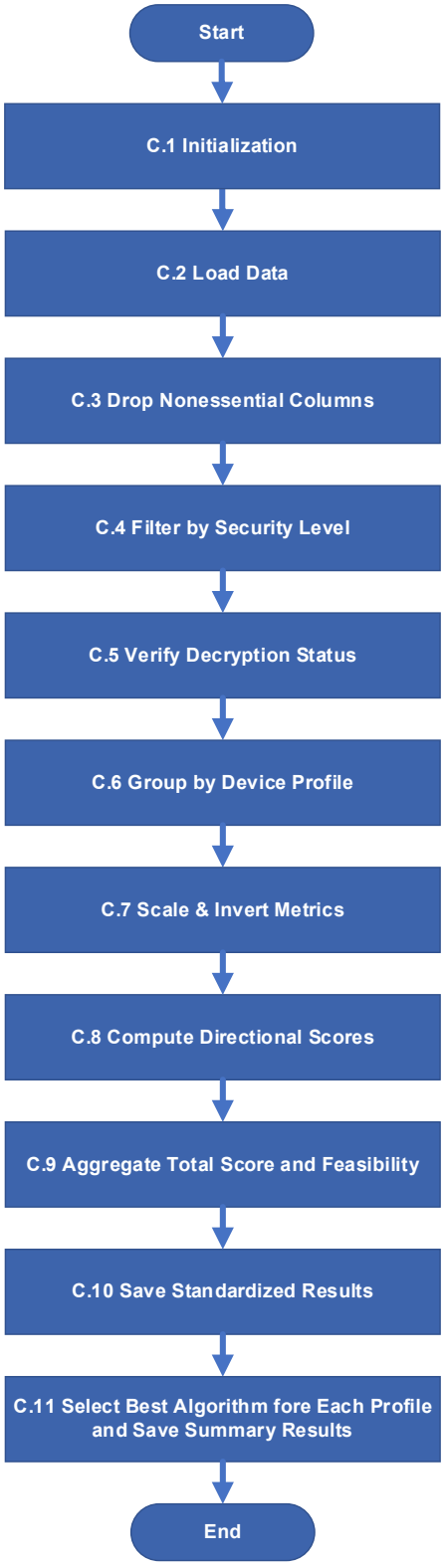

Fig. 1 Workflow diagram for the data-processing and weighted scoring pipeline (Algorithms C.1–C.11)

Section 0.3 – Algorithm Dependency Table

This section summarizes the dependencies among the algorithms and outlines their primary roles within the data-processing and weighted-scoring pipeline as shown in Table 1.

Table 1 Algorithm dependencies and functional roles

Algorithm	Depends On	Provides Output For	Functional Role
C.1	—	C.2–C.11	Initialize libraries, dependencies, and environment
C.2	C.1	C.3	Load raw simulation results into DataFrame
C.3	C.2	C.4	Remove nonessential and host-specific columns
C.4	C.3	C.5	Filter records by security parameters
C.5	C.4	C.6	Validate decryption success and remove failures
C.6	C.5	C.7	Group cleaned dataset by IoT hardware profiles
C.7	C.6	C.8	Normalize and invert performance-related metrics
C.8	C.7	C.9	Compute directional weighted scores
C.9	C.8	C.10	Aggregate total performance score and feasibility
C.10	C.9	C.11	Export standardized results for downstream processing
C.11	C.10	—	Identify best-performing cryptographic algorithms per profile

Section 1 – System Initialization

This section specifies the necessary libraries and dependencies for establishing the data analysis and weighted scoring model development environment, aimed at transforming raw data into a dataset suitable for supervised machine learning, which is essential for the subsequent phase. Algorithm C.1 specifies the libraries and dependencies necessary for the development of the data analysis and weighted scoring model, which are positioned at the beginning of the modules titled ‘Data_Analysis_Performance.ipynb’ and ‘Data_Analysis_Performance.py’. All library dependencies and version specifications utilized in this work are documented in the supplementary ESM files—requirements.txt and appendix_environment_concise.txt—with an accompanying human-readable summary presented in Appendix A.

1: **Inputs:**

—

2: **Output:**
Environment prepared with essential data-handling and scaling utilities

3: **Procedure** *InitializeDependencies()*
4: Activate the primary module responsible for table-based data manipulation
5: Enable the component used for pattern-matching and text filtering
6: Make available the display helper used for rendering structured outputs
7: Load the normalization tool required for metric scaling
8: **endProcedure**

Algorithm C.1 InitializeDependencies

Section 2 – Data Analysis and Processing

This section presents the pseudocodes employed for the raw data analysis and the formulation of the weighted scoring model utilized to ascertain the optimal cryptographic method based on performance among the 16 candidate algorithms. All the pseudocode listings in this section are located within the module titled 'Data_Analysis_Performance.ipynb' and 'Data_Analysis_Performance.py', immediately following the system initialization scripts. The Jupyter Notebook version was initially utilized for data analysis because of its superior visualization capabilities, and subsequently, the script was switched to the Python version for enhanced looping efficiency. Algorithm C.2 was employed to import raw data from six CSV files, each containing 6,220,800 rows and 54 columns, resulting in a merged dataset comprising 37,324,800 rows and 54 columns for analysis in the Jupyter Notebook. Algorithms C.3–C.6 was employed to refine the dataset by eliminating unnecessary columns, eradicating data duplication, verifying the decryption status of each entry, and categorizing pertinent data entries based on their IoT device specification profiles. This process reduced the dataset size from 37,324,800 rows to 6,220,800 rows and subsequently to 248,832 rows, while the number of columns decreased from 54 to 32 and further to 30. Algorithm C.7 was employed to execute MinMax scaling on essential metrics for selecting the optimal cryptographic algorithm, including duration, RAM usage, ROM usage, and battery consumption for both encryption and decryption operations, subsequently inverting the scores to reflect higher values, which signify superior performance or enhanced resource efficiency. Algorithm C.8 served as the pseudocode for computing the weighted score of encryption and decryption by multiplying the four metrics by their corresponding priorities. This approach identifies the superior cryptographic algorithm among the 16 candidates regarding IoT device performance, assigning a score of 0 if any feasibility check for RAM, ROM, or battery fails. Algorithm C.9 was employed to compute the total weighted score by aggregating the weighted scores for encryption and decryption, thereby facilitating an overall performance comparison. It also assigned a score of 0 to those deemed infeasible based on one of the feasibility assessments for RAM, ROM, and battery. Algorithm C.10 was employed to export the processed data and save it as a CSV file prefixed with 'standardized_results,' including the corresponding data size and priority ranking

for documentation purposes. Algorithm C.11 was employed to determine the optimal cryptographic algorithms from 16 candidates for each IoT device specification profile, based on the highest total weighted score. Profile groups lacking feasible cryptographic algorithms will be designated as 'Infeasible' to serve as instructional material for supervised machine learning training. The final datasets will be exported as CSV files prefixed with 'summary_best_config,' along with their corresponding data size and priority ranking.

1: **Inputs:**
    datasize — label indicating which simulation dataset to load

2: **Output:**
    A consolidated table containing simulation entries

3: **Procedure** *LoadSimulationRecords(datasize)*
4: Form the complete file path using the supplied datasize descriptor
5: Import the selected simulation file into a structured data table
6: Report the total number of rows and columns
7: Inspect the structural properties and field types of the loaded dataset
8: Display representative samples from the beginning and end of the table
9: Return the populated dataset
10: **endProcedure**

Algorithm C.2 LoadSimulationRecords

1: **Inputs:**
    data — dataset containing raw simulation attributes

2: **Output:**
    Dataset containing only IoT-relevant performance fields

3: **Procedure** *PruneIrrelevantAttributes(data)*
4: Identify attributes related solely to host device hardware and timing
5: Exclude these attributes from the dataset to produce a refined table
6: Return the reduced dataset
7: **endProcedure**

Algorithm C.3 PruneIrrelevantAttributes

1: **Inputs:**
    data — refined dataset
    threat_level — selected threat category
    sensitivity_level — required sensitivity category

2: **Output:**
Filtered dataset containing only entries satisfying the chosen levels

3: **Procedure** *ApplySecurityFilters(data, threat_level, sensitivity_level)*
4: Select records whose threat level matches the specified requirement
5: Retain only those entries whose sensitivity aligns with the target level
6: Remove the filter fields after the selection is complete
7: Inspect the resulting structure and preview the final rows
8: Return the security-filtered dataset
9: **endProcedure**

Algorithm C.4 ApplySecurityFilters

1: **Inputs:**
data_m — dataset filtered by security parameters

2: **Output:**
Dataset with validated decryption records and a count of any failures

3: **Procedure** *ValidateDecryption(data_m)*
4: Locate all entries indicating unsuccessful decryption
5: Compute the total number of such entries
6: If no failures are detected:
7: Remove the decryption-status indicator
8: Else:
9: Display the affected records for assessment
10: Return the validated dataset and failure count
11: **endProcedure**

Algorithm C.5 ValidateDecryption

1: **Inputs:**
data_m — decryption-validated dataset

2: **Output:**
Profile descriptors and grouped dataset structures

3: **Procedure** *FormDeviceProfileGroups(data_m)*
4: Define the set of IoT hardware characteristics used for grouping
5: Partition the dataset according to these profile descriptors
6: Return the attribute list and grouped data object
7: **endProcedure**

Algorithm C.6 FormDeviceProfileGroups

1: **Inputs:**
group — records associated with a single device profile
weights — mapping of metric names to priority coefficients

2: **Output:**
Group enriched with normalized and direction-adjusted metrics

3: **Procedure** *NormalizeAndInvert(group, weights)*
4: Create an operational copy of the received group
5: Identify which performance metrics require normalization
6: Apply a scaling mechanism to express each metric on a comparable scale
7: Produce inverted forms of the scaled metrics to ensure higher values reflect higher efficiency
8: Return the transformed group
9: **endProcedure**

Algorithm C.7 NormalizeAndInvert

1: **Inputs:**
group_m — normalized dataset for a specific profile
weights — priority mapping used for scoring

2: **Output:**
Dataset annotated with feasibility flags and directional weighted scores

3: **Procedure** *ComputeDirectionalScores(group_m, weights)*
4: Define the resource feasibility indicators for encryption
5: Define the resource feasibility indicators for decryption
6: For each entry in the dataset:
7: Determine if encryption operations are feasible
8: Determine if decryption operations are feasible
9: Assign directional scores by combining inverted metrics with their priorities
10: Return the enriched dataset
11: **endProcedure**

Algorithm C.8 ComputeDirectionalScores

1: **Inputs:**
group_m — dataset with directional scores
version_name — scoring configuration identifier
datasize — dataset variant descriptor
group_info — profile-specific details

2: **Output:**
Dataset containing total scores and combined feasibility status

3: **Procedure** *AggregateOverallScore(group_m, version_name, datasize, group_info)*
4: For each entry, compute a total score by summing directional components
5: Mark entries as performance-feasible when both directional scores exceed zero
6: Record a summarizing annotation for analysis tracking
7: Display the updated dataset
8: Return the aggregated dataset
9: **endProcedure**

Algorithm C.9 AggregateOverallScore

1: **Inputs:**
processed_results — collection of all processed groups
datasize — dataset variant
version_name — scoring configuration label

2: **Output:**
A merged dataset saved as a standardized output file

3: **Procedure** *ExportStandardizedTables(processed_results, datasize, version_name)*
4: If no processed results are available:
5: Issue a warning and terminate the procedure
6: Combine all processed groups into one comprehensive table
7: Prepare a filename using the dataset and scoring identifiers
8: Save the combined table to the designated output directory
9: Return the standardized dataset
10: **endProcedure**

Algorithm C.10 ExportStandardizedTables

1: **Inputs:**
scaled_table — standardized dataset
datasize — dataset identifier
version_name — scoring configuration

2: **Output:**
A table identifying the optimal cryptographic method for each device profile

3: **Procedure** *DetermineBestConfiguration(scaled_table, datasize, version_name)*
4: For each device-profile group:
5: Initialize the column indicating which algorithm is selected

```
6:          If all entries are operationally infeasible:
7:              Label the profile as infeasible and flag the first entry as the
                representative row
8:          Else:
9:              Mark infeasible rows accordingly
10:             Identify the feasible entry with the highest overall weighted score
11:             Mark that entry as the best-performing configuration
12:     Remove intermediate analysis fields
13:     Reorder fields so that decision indicators appear at the end
14:     Write the resulting table to the summary output directory
15:     Return the final summary table
16: endProcedure
```

Algorithm C.11 DetermineBestConfiguration

Appendix A – Execution Environment

Scope

This appendix summarizes the execution environment used to generate the data analysis and machine learning experiments associated with the manuscript. The complete machine-readable metadata is provided in appendix_environment_concise.txt, automatically generated by generate_env_info_auto.py.
All Python package versions are pinned in requirements.txt for reproducibility.

Table A.1 Runtime Environment Summary

Area	Description
Operating system	Windows 11 (10.0.26200, 64-bit, SP0)
CPU	Intel64 Family 6 Model 183 Stepping 1 (GenuineIntel)
Python	3.13.2 (build tags/v3.13.2:4f8bb39, Feb 4 2025 15:23:48)
Virtual environment	venv
Environment snapshot	2025-10-26 17:14:08

Key Third-party Packages

- IPython v9.4.0
- pandas v2.3.0
- scikit-learn v1.7.1
  (Complete list available in requirements.txt.)

Local Project Modules

None detected — all functionalities were implemented using third-party and built-in Python modules.

Standard Library / Built-in Modules Used

re

Reproducibility Notes

This environment summary provides the human-readable overview only. For exact replication, recreate the Python environment using requirements.txt and rerun generate_env_info_auto.py to regenerate appendix_environment_concise.txt. If discrepancies arise, the machine-readable files (appendix_environment_concise.txt and requirements.txt) are considered authoritative.