Chapter 2

An Adaptive Lightweight Security Framework Suited for IoT

Menachem Domb

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.73712

Abstract

Standard security systems are widely implemented in the industry. These systems consume considerable computational resources. Devices in the Internet of Things [IoT] are very limited with processing capacity, memory and storage. Therefore, existing security systems are not applicable for IoT. To cope with it, we propose downsizing of existing security processes. In this chapter, we describe three areas, where we reduce the required storage space and processing power. The first is the classification process required for ongoing anomaly detection, whereby values accepted or generated by a sensor are classified as valid or abnormal. We collect historic data and analyze it using machine learning techniques to draw a contour, where all streaming values are expected to fall within the contour space. Hence, the detailed collected data from the sensors are no longer required for real-time anomaly detection. The second area involves the implementation of the Random Forest algorithm to apply distributed and parallel processing for anomaly discovery. The third area is downsizing cryptography calculations, to fit IoT limitations without compromising security. For each area, we present experimental results supporting our approach and implementation.

Keywords: IoT, anomaly detection, entropy, machine learning, random forest, cryptography, RSA

1. Introduction

The area of the Internet of Things [IoT] is rapidly growing, raising severe security concerns to the entire network. Due to its high traffic volume and real-time operation, a security framework is essential. The system should timely predict possible attacks and react accordingly. Standard security systems are widely implemented in the industry. These systems consume considerable computational resources and cannot operate in IoT devices (i.e., sensors) due to their very limited memory and computation power. To cope with these limitations, two alternatives come to mind, i.e., the development of novel security measures tailored to IoT [1] or downsizing existing security processes to enable properly operation in IoT devices. We apply the latter option as it is highly recommended to use proven algorithms, which have been extensively analyzed and tested, while new algorithms exposes the user to vulnerability.

We introduce lightweight versions of several known security processes. We analyze each relevant process and its corresponding limitations, and then we divide each complex and large process into a collection of smaller processes. These small processes are distributed and executed by sensors connected to the same network, based on its available capacity. Once all small processes are completed, we collect the partial results and input them into a complementary process that integrates the partial results to compose the desired result. The final result is the same as if the original process was generated. In this chapter, we describe three areas, where we minimize the required storage space and processing power. The first is the classification process required for ongoing anomaly detection, whereby values accepted or generated by a sensor are classified as valid or abnormal. We collect historic data and analyze it using machine learning techniques to draw a contour, and all streaming values are expected to fall within the contour space. The detailed collected data are no longer required, thereby considerably reducing the storage space. The second area involves the implementation of the Random Forest algorithm to apply distributed and parallel processing for anomaly discovery, resulting in the use of limited processing power. The third area is downsizing cryptography calculations, such as RSA, a public-key cryptosystem, to fit IoT limitations. The rest of this chapter is divided into three sections, one dedicated to each downsized area. In the last section, we conclude this chapter.

The rest of this chapter is organized as follows: In Section 2, we describe the preparation stage of the classification process, which minimizes the need for the entire historic data and then the anomaly detection processes using the outcome of the previous stage. In Section 3, we describe the use of the Random Forest algorithm for distributed and parallel processing of automatic classification and anomaly detection. In Section 4, we present an improved implementation of RSA to allow high class cryptography that runs in an IoT configuration. In Section 5, we conclude this chapter and discuss our ongoing and future work.

2. Classification framework for data streaming anomaly detection

To predict the behavior of a system, we usually examine its past data to discover common patterns and other classification issues. This process consumes considerable computational power and data storage. In this section, we describe an approach and a system, which requires much less resources without compromising prediction capabilities and accuracy. It employs three basic methods: a common behavior graph, the contour surrounding the graph, and entropy calculation methods. When the system is about to be implemented for a specific domain, the optimized combination of these three methods is considered, such that it fits the unique nature of the domain and its corresponding type of data. In addition, we present a framework and a process that will assist system designers in finding the optimal methods for the case at hand. We use a case study to demonstrate this approach with meteorological data collected over 15 years to classify and detect anomalies in new data.

This section is organized as follows: We begin by defining the problem, proceed with various solutions proposed in the literature, and then present our adjustable contour approach. We then show how it is applicable for IoT. We proceed with a case study demonstrating the build-up of the contour and how it is used for instant anomaly detection. We conclude with a summary of the section.

2.1. Problem definition

The problem we attempt to solve is the optimization of the amount of sampling data collected to maintain a proper balance between the quantity of sampling data and the information extracted from it. The problem statement focuses on extracting concepts, methods, rules, and measurements, so that at the end of the process, the original sampling data become redundant and no longer need to be stored. However, to keep improving and adjusting the extracted items to natural changes in the behavior of the sampled mechanism, we incorporate in the approach an ongoing learning process. In addition, in the study, we concentrate on time-dependent streaming sampling data, divided by fixed periods, so that we can repeat the analysis process for each period/cycle. Thus, while there are many classification algorithms using time series sampling, the aim is not to compare the performance of yet another classifier, but rather present a flexible method to compactly represent the data with several parameters that can be chosen and adjusted. We suggest an independent framework that allows a flexible adaptation of the contour to the nature of the given domain. Indeed, some of the reviewed works, such as Reeves et al. [6], can be revised and adjusted to the problem statement and serve as a valid alternative to the approach we present. We are striving for the best sampling strategy given sequential data, generated from IoT devices.

The input given is a set of time series: $D = \{d^{(1)}, d^{(2)}, \ldots, d^{(n)}\}$, where each time series $d^{(\mathrm{i})}$ contains pairs (timestamp and numeric value). The required output is an optimal set $Dw = \{a_1, a_2, \ldots, a_m\}$, where *ai* can be any sampling item, such as a minimal data set, trends, graphs, measurements, or rules, which strongly represents and supports the purpose of the original data set *D*.

We consider the set *Dw* and the full data set *D* as containing the same information, if they produce the same classifier. That is, if $f(d) = fw(d) \in \{-1, 1\}$ for every new data series *d*, where *f* is a classifier learned from *D* and *fw* is a classifier based on *Dw*. For instance, we can judge whether a series of yearly temperatures represent an El Nino (EN) year or not, or whether a series of sensor data is characteristic of a suspected intrusion or not. Here, we consider two sets *D* and *Dw* as containing the same (or similar) information if both can predict the future pattern of an initial series *d*. That is, we can use either *D* or *Dw* to predict a future item *dn* with similar accuracy.

2.2. Literature review

Real-world data typically contain repeated and periodic patterns. This suggests that the data can be effectively represented and compressed using only a few coefficients of an appropriate basis. Mairal et al. [2] study modeling data vectors as sparse linear combinations of basic elements generating a generic dictionary and then adapt it to specific data. Jankov et al. [3] present an implementation of a real-time anomaly detection system over data streams and report experimental results and performance tuning strategies. Vlachos et al. [4] formulate the problem of estimating lower/upper distance bounds as an optimization problem and establish the properties of optimal solutions to develop an algorithm which obtains an exact solution to the problem. Sakurada and Yairi [5] use auto-encoders with nonlinear dimensionality reduction for the anomaly detection task. They demonstrate the ability to detect subtle anomalies where linear PCA fails. Reeves et al. [6] present a multi-scale analysis to decompose time series and to obtain sparse representations in various domains. Chilimbi and Hirzel [7] implement a dynamic pre-fetching scheme that operates in several phases. The first is profiling, which gathers a temporal data reference profile from a running program. Next, an algorithm extracts hot data streams, which are data reference sequences that frequently repeat in the same order. Then, a code is dynamically injected into appropriate program points to detect and pre-fetch the hot data streams. Finally, the process enters the hibernation phase where the program continues to execute with the added pre-fetch instructions. At the end, the program is deoptimized to remove the inserted checks and pre-fetch instructions and control returns to the profiling phase. Lane and Brodley [8] claim that features can be extracted from object behavior and a domain heuristic. Experiments show that it detects anomalous conditions, and it is able to identify a profiled user from other users. They present several techniques for reducing 70% of the storage required for user profile. Kasiviswanathan et al. [9] proposed a two-stage approach based on detection and clustering of novel user-generated content to derive a scalable approach by using the alternating directions method to solve the resulting optimization problems. Aldroubi et al. [10] show that for each dataset there is an optimized collection of cells spanning the entire space and so generate the optimized sampling set.

The common underlying idea of the reviewed approaches is the definition of the problem they are aiming to solve. The problem attempted to be solved is optimizing the size of the collected sampling data so that it keeps the proper balance between the quantity of sampling data and the information extracted from it.

2.3. Contour-based approach

Briefly, we analyze sampling data collected over several periods. We divide the period into time-units. For example, for a period of a year, we divide it into daily time-units. For each time-unit, we extract one value that represents it. This is done by averaging the samples collected during the time-unit. In the example, we may calculate the average value of all samples of that day. We may also decide to select one of the samples to represent the day, e.g., the first or last sample. We then calculate the average value for each time-unit from the collected values for the same time-unit in all periods, resulting in an average value for a given time-unit. We repeat this process for all time-units in the period and obtain a graph that represents the average values for an average and common period.

Assuming we have the average graph line for an average period, we now calculate the contour around this average. The generated contour represents the standard range of values, such that an unanalyzed period can be compared to this contour. If its graph value is completely within the contour, the period is a standard period. If it is completely out of the contour, then it is purely not standard. If the sections of the graph are within the contour, while others are out of it, we use an entropy measure to calculate the overall "distance" of the given period from the standard contour. Assuming an existing entropy threshold, we can decide whether the period is a standard one or not. We apply the same concept at the unit level and decide whether a specific time-unit in a period is within the standard or not. This specific check is relevant, for example, to anomaly detection of IoT behavior.

In conclusion, the entire process is based on three key elements: the average graph per period, the contour around the average graph, and an entropy value representing the overall distance of a period from the contour. Each of these elements—average, contour, and entropy—can be one of the several possibilities. For the contour, a simplistic choice would be minimum and maximum (min-max) values. Alternatively, the *SD* or confidence interval (CI) could be employed. These three elements affect each other, and every choice of such a triplet—average, contour, and entropy—will produce a different behavior of the compressed classifier. The object is to find the best triplet that will be able to disregard the original data after extracting the representative contour, without compromising the ability to successfully analyze future series. In our work, we consistently use the arithmetic average and classical entropy and focus on finding the best contour.

2.3.1. Finding the optimal contour

We begin with a supervised learning approach, for classification, in which each time series is labeled as one of two classes. To demonstrate, using the data set from the experiments, the time series are year-long recordings of temperature samplings, labeled as positive, if the corresponding year was an EN year, or otherwise negative. We now describe in detail the process of building the classifier, with emphasis on finding the optimal contour.

Constructing the best contour is described in **Figure 1**. We begin with raw data collected during *N* periods, where each record corresponds to a specific time-unit. These cycles have already been classified positive or negative according to some classification criteria. These classified cycles will later be used to determine the best contour.

The process is divided into four stages. In stage one, we use a selected average method and calculate the average graph line representing the *N* given cycles. This is done horizontally by calculating the average of the values related to the same time-unit across all *N* cycles. For example, we calculate the average of the values for January 1st across the various years. Doing so for all time-units will generate the average graph line. In stage two, we select several distance calculation methods, and for each method, we construct its associated contour. This is done by calculating the distance value for each distance method, e.g., the min-max difference, *SD*, and CI. Taking the distance value, we add and subtract it from the average line to get the contour around the average. We repeat this process for all distance methods. At this stage, we have constructed several contours around the average line. The goal now is to select the contour, which is most effective in classifying unclassified cycles. This is done in stages three and four. In stage three, we calculate the prediction power for each contour and select the one with the highest prediction power. This is done by summing, for each contour, the number of cases in which its prediction was right and calculating the average entropy of these correctly classified cycles. We do the same for wrong

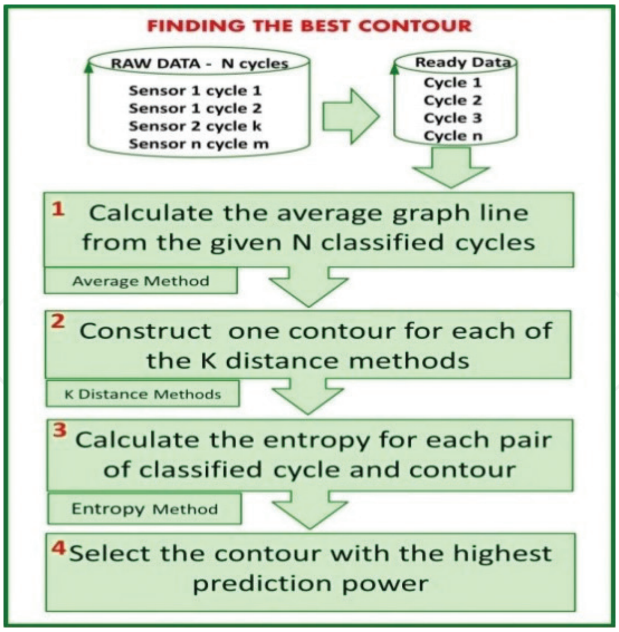
FINDING THE BEST CONTOUR
RAW DATA - N cycles
Sensor 1 cycle 1
Sensor 1 cycle 2
Sensor 2 cycle k
Sensor n cycle m
Ready Data
Cycle 1
Cycle 2
Cycle 3
Cycle n
1 Calculate the average graph line from the given N classified cycles
Average Method
2 Construct one contour for each of the K distance methods
K Distance Methods
3 Calculate the entropy for each pair of classified cycle and contour
Entropy Method
4 Select the contour with the highest prediction power

Figure 1. Process of finding the optimal contour.

predictions. In stage four, we use one entropy method with an associated threshold value. An unclassified cycle with an entropy value lower than the threshold will be classified positive and otherwise negative. For each contour, we calculate the entropy of the given classified cycles. The result is a set of entropy values, where some are below the threshold and others are above it.

a. We repeat this for all classified cycles. We then sum up the number of correct predictions and their total entropies. We do the same for wrong predictions. We then subtract the total wrong numbers from the correct numbers. We repeat this process for all the constructed contours and select the contour with the highest prediction power.

b. Calculating the entropy.

The entropy of a period, given a contour, is calculated as follows:

- Marking for every timestamp whether the cycle's value at that timestamp is below, within, or above the contour.
- Calculating the frequency of each of these three possibilities: below (p_1), within (p_2), and above (p_3)
- Using these as a ternary probability distribution, its entropy is calculated according to the formula: $p_1 \log^{(p1)} + p_2 \log^{(p2)} + \mathrm{p}_3 \log^{(p3)}$
- The entropy measure is expected to return its minimum value at the two extreme cases: When the cycle graph is entirely contained within the contour and when the cycle graph lies entirely outside of the contour. All other cycles are expected to fall mostly within the contour, and those which diverge enough from the contour, will have a high entropy value which will lead to the right conclusion

c. Classifying a cycle/period

Figure 2 describes the process of classifying unlabeled data cycles, as listed below:

1. Apply the given data cycle to the contour and match it according to timestamps.
2. Noting for each timestamp whether the data point is below the contour, within it, or above it.
3. Marking these cases respectively as −1, 0, and +1.
4. Calculating the frequencies of each of the three values: −1 ($p1$), 0 ($p2$), and +1($p3$).
5. Calculating the entropy of the distribution defined by $p1$, $p2$, and $p3$.
6. Classifying as belonging to the contour, if the entropy is below the threshold determined in the learning phase.

2.3.2. Advantages of the proposed technique

The proposed technique has several advantages over other methods. The technique is a family of sampling methods and is defined by the three parameters described above. It is reasonable to expect that different datasets will require different parameters for the best sampling. Different combinations can be tested and evaluated to ensure optimal treatment of the data. The technique we propose is therefore flexible and adjustable and thus suits every given data set. Secondly, this technique can be applied not only for classification but also for prediction of time series.

Thirdly, the technique can be used to evaluate reliability of data online. In cases of high fluctuations or sharp changes in the cycle graph, which do not conform to either of the two class contours, suspicion may arise that the reliability of the data has been compromised. This can indicate that the sensor is damaged or that there has been a security breach.

Fourthly, the approach allows self-learning and automatic adjustments in cases of common behavior changes and a new standard has been established. Lastly, occasionally, a post-mortem may be run to check the system's reaction to actual behavior and thereafter adjust the system parameters accordingly.

2.4. Anomaly detection for IoT security

IoT devices generate time-related data, i.e., structured records containing a timestamp and one or more numeric values. In many cases, we can identify recurrent time frames where the system behavior has a repetitive format. Hence, IoT data have a structure to which the contour approach is highly applicable.

IoT security utilizes common data patterns and quantitative measurements. Based on the identified patterns and measurements, we can extract logical rules that will be executed once an exception is discovered. An exception may be any violation of predefined patterns, measurements, and other parameters, which represent normal, standard, and permitted behavior.

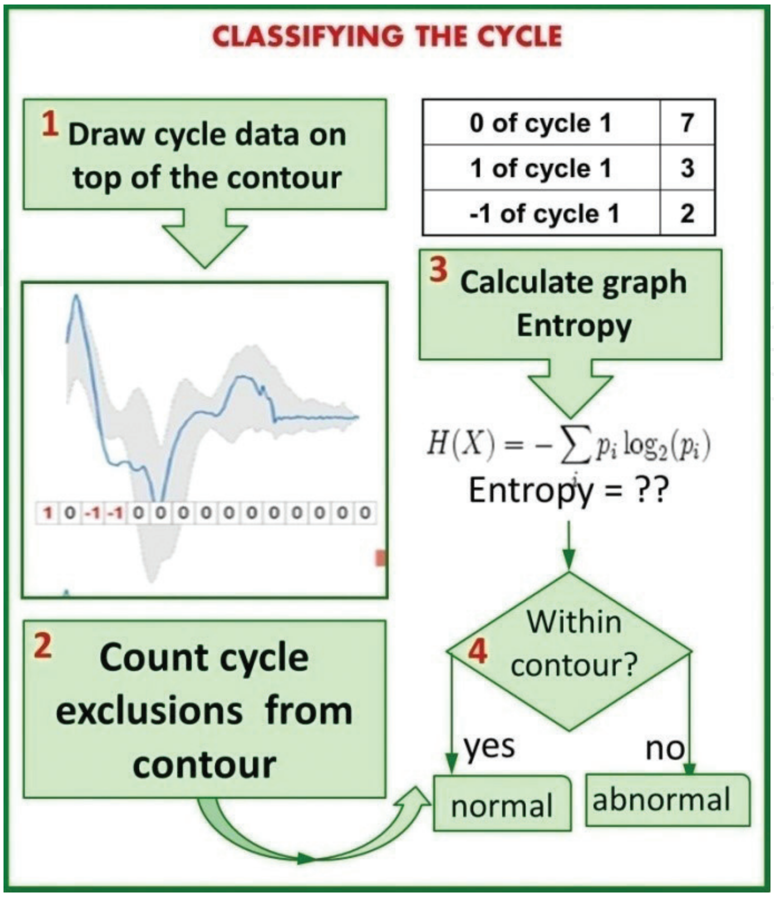

Figure 2. Classifying a cycle.

In IoT, there is an abundance of possible patterns, starting with column level patterns up to a super internet controlling several IoT networks. The goal is to find the methods and tools to define standard patterns and how they can be identified. Once this is done, we can apply the contour method. In our work, we show a two-dimensional contour. Using the same concept, we can expand it to be a multi-dimensional contour. This case is common where there is a dependency among several columns within one record and the same applies for the case where there are dependencies among networks of IoT systems.

2.5. Case study

In the following case study, we used meteorological data collected on EN years (positive class) and NEN years (negative class) from 1980 to 1998. For the positive contours, we took data from the EN years 1982, 1983, 1987, 1988, 1991, and 1992. All other years in the range were NEN years. We tested three methods for generating contours: (a) max-min over all cycles; (b) average cycle ± *SD*; and (c) CI.

Figures 3 and **4** depict the contours for NEN years. **Figure 3** shows the NEN contour in black according to the average ± *SD* and depicts how EN years diverge from this contour, as compared to the NEN year—1995. The 1992 and 1988 (EN years) show clear divergence from the contour while 1995 (a NEN) is more contained within the contour. This is nicely captured by the entropy values, which for 1992 was 0.4266 and for 1988 was 0.3857—above the threshold, leading to the conclusion that they are not NEN years—while for 1995, the entropy was 0.3631—significantly lower than those of the EN years, leading to the correct conclusion that 1995 was indeed a NEN year.

Figure 4 shows two contours: the min-max contour and the average ± *SD* contour. The *Y*-axis in these graphs is the temperature value, and the *X*-axis is the time. Within each contour, the year 1995 (a NEN year) is graphed. Its entropy is 0.3631 for the average *SD* contour and 0.2932 for the min-max contour. Both are the threshold, which leads to the correct conclusion that it should indeed be classified as NEN.

In the case study, we compared the constructed contours, by using the average graph ± *SD* and the average graph ± min-max. For the *SD* contour, we obtained a significant entropy value difference between a classified EN case and a NEN case. In comparison, the min-max contour resulted in close values of entropy for the EN cycle and the NEN cycle. Thus, the ability to differentiate between two extreme situations using entropy depends on the parameter used to build the contour.

2.6. Section summary

In this section, we dealt with the classification problem of an unclassified cycle of IoT streaming data. We introduced the contour approach to draw the borders around the standard area representing a specific class. If there was an unclassified cycle, we measured its distance from

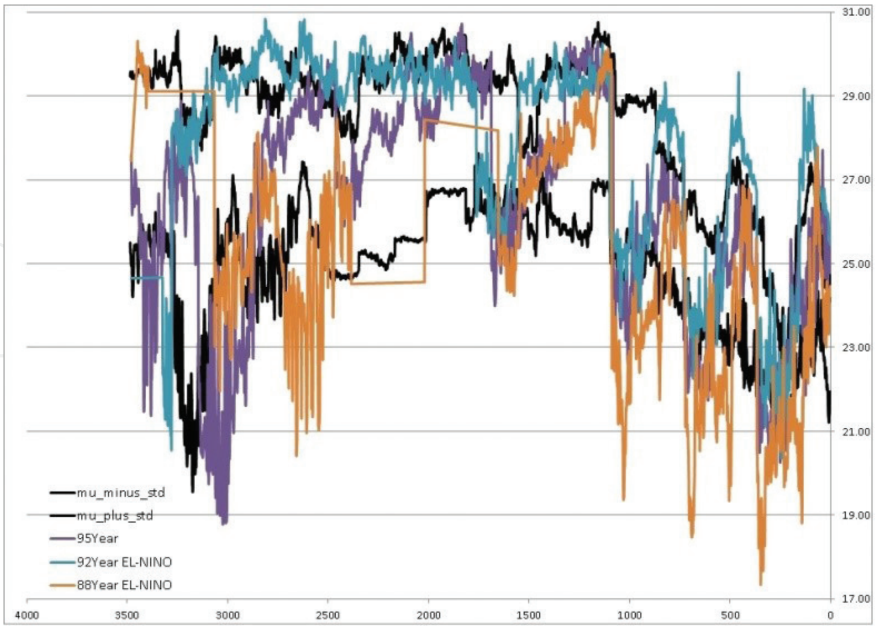

Figure 3. EN cycles on NEN average ± *SD* contour.

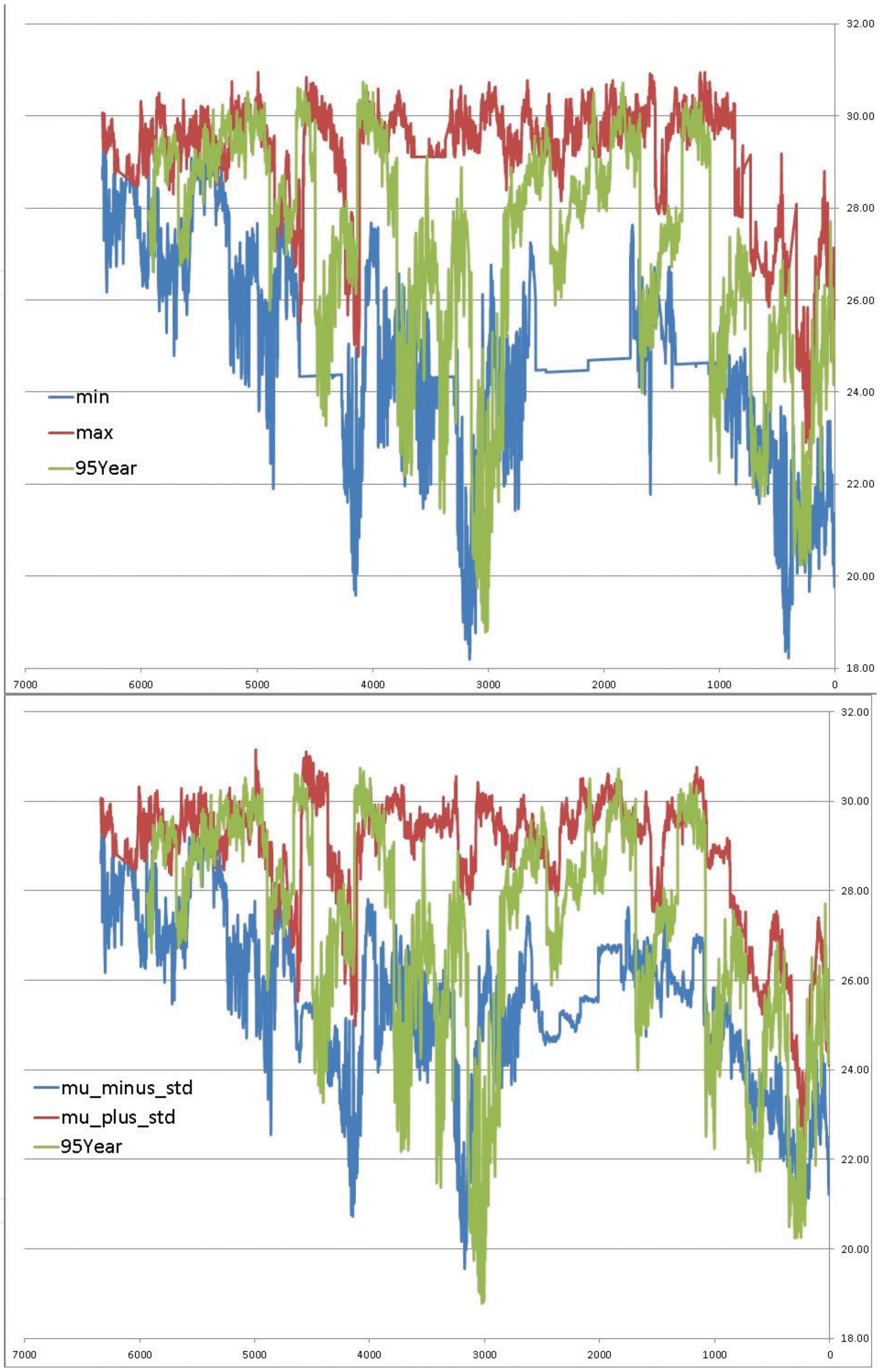

Figure 4. NEN contours—min-max and *SD*.

the contour using an entropy formula. Then, we compared the result to a predefined threshold. If the entropy value is below the threshold, the cycle is of the same class.

We propose a process for constructing the best contour that will presumably classify the correct underlying class. The process is based on three measurement methods: average, distance, and entropy. For each method, there are several alternate formulas that we may use. Each combination of these three methods may result in different contour and hence different entropy value for the same unclassified cycle. We select the combination with the maximum difference between positive and negative values.

In addition to the initial construction of the class contours from the given data, we suggest ongoing improvements of the initial contours. Namely, we recalculate the class averages and their contours to refine and revise the contours for improved classification performance.

In this manner, we are able to improve the contour approach, in reference to several aspects, such as determining the minimal number of classified cycles required to define the best contour, expanding the use of the contour to discover early trends or discover significant changes in behavior and adjusting the contour accordingly, exploring the possibility of dividing one cycle into several segments, and associating a different contour method to each segment.

3. Lightweight adaptive random forest for rule generation and execution

The volume of transmitted data over the various sensors continuously grows. Sensors typically are low in resources of storage, memory, and processing power. Data security and privacy are part of the major concerns and drawbacks of this growing domain. An IoT network intrusion detection system is required to monitor and analyze the traffic and predict possible attacks. Machine leaning techniques can automatically extract normal and abnormal patterns from a large set of training sensors data. Due to the high volume of traffic and the need for real-time reaction, accurate threat discovery is mandatory. This section focuses on designing a lightweight comprehensive IoT rules generation and execution framework. It is composed of three components, a machine learning rule discovery, a threat prediction model builder and tools to ensure timely reaction to rules violation and unstandardized and ongoing changes in traffic behavior. The generated detection model is expected to identify exceptions in real time and notify the system accordingly.

We use random forest (RF) as the machine learning platform for the discovery of rules and real-time anomaly detection. To allow RF adaptation for IoT, we propose several improvements to make it lightweight and propose a process that combines IoT network capabilities, messaging and resource sharing, to build a comprehensive and efficient IoT security framework.

The rest of this section is organized as follows: We begin with an introduction followed by the relevant literature review. We then discuss rules extraction using machine learning techniques. We present random forest as the most suitable ML for IoT. We proceed with various improvements, utilizing RF and IoT attributes. We then outline an experiment that executes RF building and its corresponding classifications using 15 different configurations, each based on a unique combination of the number of processors and the forest size.

3.1. Introduction

IoT is a network of objects, consisting of sensors, Internet, software, and exchange of data. This generates critical issues of security, which must be addressed. Since to date there is no standard for sensors, any system under development at this stage must consider the possibility that soon a standard will be defined, and the systems must be able to easily adjust to it. Along with the limited processing power and the fact that the security issues must be dealt with in real time, we realize the immediate need for a flexible and lightweight solution. The solution should be dynamic, open, scalable, distributed and decentralized. The analysis discovers patterns and measurements from the data, which are then translated into anomaly detection rules associated with actions to be executed when a rule is violated. The rules are then deployed in the IoT devices. When data are received from, or transmitted to an IoT device, the rules are executed. If the result is positive, the corresponding action is triggered to cope with the situation.

3.2. Literature review

Mansoori et al. [11] proposed a systematic process for retrieving fuzzy rules from a given data set. To improve performance, the retrieved rules are then crystallized based on its effectiveness and applicability. Dubois et al. [12] use Sugeno integrals, which are qualitative criteria aggregations where it is possible to assign weights to groups of criteria. They show how to extract if-then rules that express the selection of situations based on local evaluations and rules to detect bad situations. Sumit-Gulwani, Hart, and Zorn [13] deal with converting data into an appropriate layout, which requires major investment in manual reformatting. The paper introduces a synthesis engine to extract structured relational data. It uses examples to synthesize a program using an extraction language. Bharathidason et al. [23] presented a fast and compact decision rules algorithm. The algorithm works online to learn rule sets. It presents a technique to detect local drifts by taking advantage of the modularity of the rule sets. Each rule monitors the evolution of performance metrics to detect a concept drift. It provides useful information about the dynamics of the process generating data, faster adaptation to changes, and generates more compact rule sets. Jafarzadeh et al. [15] used averaging techniques to propose a method in which a previous algorithm for association rules mining is improved upon to specify minimum support. It uses fuzzy logic to distribute data in different clusters and then tries to provide the user with the most appropriate threshold automatically. Limb et al. [16] used Fuzzy ARTMAP and Q learning to build a data classification and rule mining model. To justify the classification, the model provides a fuzzy conditional rule. Q-values are used to minimize QFAM prototyping. Mashinchi et al. [17] proposed a granular-rules extraction method to simplify a data set into a granular-rule set with unique granular rules. It performs in two stages to construct and prune the granular rules. Yang H. et al [18] proposed an anomaly detection algorithm of Quick Access Recorder (QAR) data, based on attribute support of a rough set. The method retains the time characteristics of QAR data and strengthens the relation between the condition and decision attributes. Tang [19] described an approach of data mining with Excel, using the XLMiner add-in. This is an example of mining association rules to illustrate all the steps of this approach. Tong S and Koller D. [20] introduced an algorithm for choosing which instances to request next, in a setting in which the learner has access to a pool of unlabeled instances and can request the labels for some number of them. The algorithm is based on a theoretical motivation for using support vector machines (SVMs). Osungi et al. [21] proposed an active learning algorithm that balances exploration by dynamically adjusting the probability to explore each step. Lang T et al. [22] proposed an active learning method for multi-class classification. The method selects informative training compounds to optimally support the learning progress. Bharathidason et al. [23] improved the performance and the accuracy by including only uncorrelated high performing trees in a random forest.

The reviewed literature focuses on improvements to known rule discovery mechanisms, such as machine learning, to transform them into lightweight systems that can be executed in limited resources settings. In most cases, the proposed solutions remain for general purposes but can run with less required resources. We are seeking a solution that takes advantage of the unique IoT attributes and utilizes them to build a combined comprehensive framework for IoT security.

3.3. Rules generation and deployment process

The process consists of seven stages (see **Figure 1**). Stage 1 collects training data from the IoT network, removes irrelevant records, and complements data in records with missing data. In stage 2, we apply discovery techniques to extract important measurements and patterns. Stage 3 consists of generating a rule for each measurement and pattern. In stage 4, we evaluate the effectiveness of each rule with a set of training data. If the number of times a rule has been executed is below a given threshold, the rule is removed from the rules set. Next, in stage 5, we check the completeness and the integrity of the generated set of rules. Rules that contradict another rule are removed and missing rules are added. Stage 6 runs a simulation with the same training data with the presumption that all the designated rules will be executed. Finally, in stage 7, we deploy the generated rules set. At this point, the system is ready to accept the IoT traffic data in real time and automatically check it against the set of rules.

3.4. Extracting rules from training data

A typical sensor record contains the sensor ID, timestamp, and one or more values per feature. The main source for extracting rules is data collected from the concrete processes involved in the explored domain. The significance to IoT is taking the accurate decision in real time and react in real time to security alerts, notifications, automation, and predictive maintenance. To ensure the completeness and the integrity of the generated set of rules, we use a consistent multi-layer process of accumulating rules, starting with the simplest rules up to the most complicated and multi-stage rules. Simple rules are extracted at the single feature level, and then we proceed with rules extracted from a combination of any number of features having a common relation, such as features of sensors sharing the same workflow. The generated rules at this level relate to basic data such as maximum, minimum, average, standard deviation, median, and most frequent value. More complex relations, such as proportions among subsequent values, sequence trends, and significant patterns, require reasoning capabilities and can be reached by machine learning and data mining techniques. The outcomes are measurements, thresholds, and patterns used to draw the corresponding decision trees. These decision trees tend to grow fast, consuming large storage, and memory space along with high runtime when pruning and analyzing it to find the specific rule. The depth of the tree grows linearly with the number of variables, but the number of branches grows exponentially with the number of states. Decision trees are useful when the number of states per variable is limited. It becomes complicated when the state of the variables depends on a threshold or complex computations. Communicating this rationale requires labeling every edge and then tracing the tree path to understand the logic incorporated in it. Complex event processing (CEP) engines are popular in IoT. They support matching time series data patterns that originate from different sources. However, they suffer from the same modeling issues as trees and pipeline processing.

Rule engines have two major drawbacks in the context of IoT, the logic representation is not compact and the use of it requires much processing power and time. We will cope with these drawbacks in two ways. 1. Reduce the number of decision trees and improve the search navigation scope, resulting in a reasonable and acceptable search time. 2. Utilize IoT attributes and functionality to optimize the tree navigation flow and process sharing.

In the following sections, we present the random forest machine learning and propose several improvements where the known drawbacks are removed.

3.5. Decision automation using random forest

Random forest employs bootstrap aggregation for training. While the predictions of a single tree are sensitive to noise in its training set, the average of many uncorrelated trees is not. Bootstrap sampling is a way of decorrelating the trees by showing them different training sets. Many trees reduce the depth and width of each tree and so save pruning and analysis time, which suit IoT constraints.

The algorithm has two key parameters: the number of K trees to form a random forest and the number of features F, randomly sampled features for building a decision tree. For large and high dimensional data, a large K should be used. Estimating the performance of random forest for one core is based on the following parameters: # trees [K], # features [F], # rows [R], and maximum depth [D]. The estimated runtime is influenced by the number of features. Hence, keeping only the most important features lowers the number of records and maintains the maximum depth low, which will improve the overall random forest performance.

Random forest performance is better than the classical tree decision algorithm. However, it may still be insufficient for IoT due to the memory space and processing power it requires. Hence, building a lightweight RF process and utilizing IoT networking are required.

In the following section, we describe four proposals that make random-forest lightweight.

3.6. Improving RF performance and consumption of resources

a. Randomization may cause occurrence of redundant, irrelevant or even contradicting trees, which may lead to redundant searches or even to the wrong decision. Therefore, selection of trees with high classification accuracies leads to improved performance and better decision accuracy. A decision process is effective when the difference among the relevant alternatives is significant. RF contains many decision trees, where each of them may contribute to the final decision. Many such trees generally require wider searches and thus expand the decision process. On the one hand, reducing the number of the searched trees will shorten the process but on the other hand may increase the probability of making the wrong decision. Therefore, a selection criterion for removing the "redundant" trees is required. An initial approach is to remove similar trees as correlated trees hardly contribute to reaching the correct decision. Thus, for effective RF decisions, we strive to remove uncorrelated trees [14]. The correlation between two trees may be defined in various ways, such as:

1. Distance—we transform the tree into a sequence of values, and then we apply a hashing function on this sequence and get a score. Two trees are correlated if the difference between the scores is below a predefined threshold.
2. Common components—count the number of similar components and compare.
3. Empirically by removing the tree and trying a vast number of cases, we will reach the same decisions as we would if the tree was included, which means that the tree has no effect on practical decisions.

b. Prioritize trees by simulation using labeled and already classified cases.

Instead of removing trees, we propose prioritizing them. The prioritization can be an empirical study of the historical use and effectiveness in true/false decisions. Another way is to run a Monte-Carlo intensive simulation and prioritize trees accordingly.

3.7. Prioritize trees by its threat level

We define several security levels: low, normal, high, and emergency. For each level, we associate the most effective trees and the order of the trees to be visited. For each network, we designate a security manager device, which collects messages from its network devices, assesses it, and determines the network security level. When the network is initiated, the designated level is low. As time passes, messages arrive at the security manager device, which analyzes the input and decides to change the security level. Then, a message is distributed requesting a security level change. Once the level is changed, the local system activates the new tree search schedule.

3.8. Messaging assisted, best trees selection

MQTT is a lightweight messaging protocol, over TCP, adjusted to the IoT domain. Given MQTT, we can utilize the IoT network itself to improve performance. We use it to transfer messages and data from one device to another. For example, in case of a suspicious occasion detected by one of the sensors, using the protocol, the device sends alert messages to other members. The messages include data strings and unique data patterns that receivers should expect to receive and thus detect a malicious situation. The message may also include the most effective trees that may cope with the suspected threat.

When suspicious data reach a sensor, it is analyzed locally, and the best tree sequence is identified. This device sends a message to the security manager, containing the data with the detected anomaly and the sequence of trees to visit and act accordingly. The messaging protocol is an adjustment of HTTP.

3.9. Experiment using the random forest in an IoT

In this section, we describe a comprehensive test, simulating the building of various random forests and then runs several classification cycles for a given set of anonymous records. We used a computer with eight processors running the random forest PMI platform with 10–1000 trees per forest. It contained a random forest builder, an anonymous records classification process, and a configuration tool. We sought the best configuration, suitable for the optimized performance and accuracy of a random forest simulation. A configuration in this context is measured by the combination of the number of processors and the number of trees in a forest. For the simulation, we used 500 anonymous records and 3350 already classified samples, where each sample has 95 attributes. We ran 30 test cycles where each cycle represented a unique configuration—number of processors: 2, 4, 6, 8, and 16 and the number of trees per forest: 10, 100, 250, 500, 750, and 1000. For comparison, all test cycles used the same data set. In cases of similar trees, we ran a process that removes similar trees. The performance of the entire 30 test cycles is evaluated by its accuracy and processing time.

Figures 5 and **6** show that accuracy, performance of each of the processes and combined are best achieved when using 10 trees per forest and 8 processors. Based on the above simulations, it seems that for the example at hand, using a relatively small number of trees per forest and multi-core processors is recommended for optimal performance and high accuracy. However, this may not be the common case. Therefore, prior to implementing RF-based anomaly detection, it is recommended that a simulation test be run with the main data. In addition, we propose a prototype of an IoT environment. The prototype is composed of one server and six Arduino OS devices. We built two configurations, A and B. In configuration A, all the devices are connected via WIFI 14 to the server, where the data transmission between two devices is done through the server. The entire RF is loaded in the server while the devices have one tree installed in them. The data flow of an incoming event in configuration A can be one of the following: 1. An event arrives at a device, the device forwards it to the server, which then runs the RF and classifies the event. 2. An event arrives at a device and the device forwards it to the server. The server forwards it to all devices. Each device checks the event against the appropriate local tree and sends the result to the server. The server then counts the results and sends the reply to the sender, which acts accordingly. The flow in configuration B is as follows: An event arrives at a device, the device propagates it to other devices, checks it against its own tree, and propagates the results back to the sender. The sender classifies the event and acts accordingly. To test the feasibility of the prototype, we used the trees built by the simulation tool and loaded it to the server and devices accordingly. We transmitted 500 events to the devices in

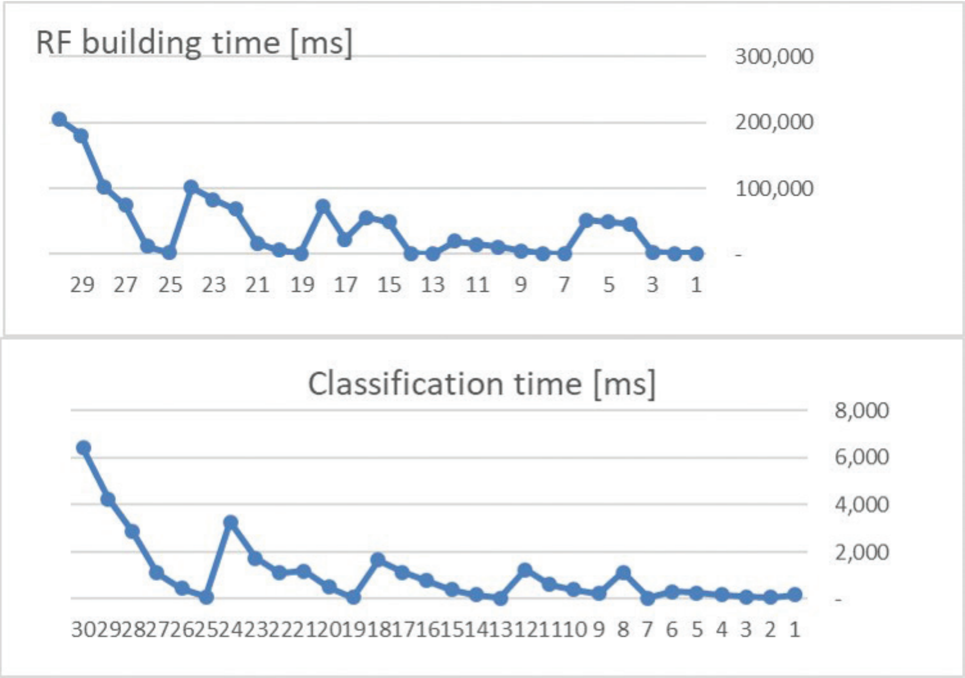

Figure 5. Results of running the 30 classification processes.

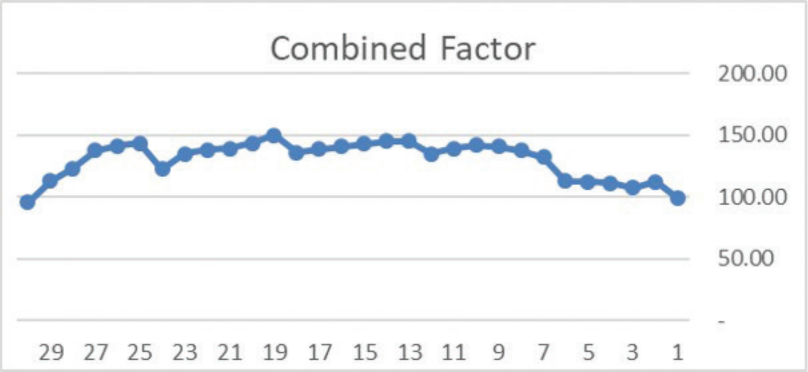

Figure 6. Accuracy and combined results of running the 30 classification processes.

a round robin schedule. The resulting accuracy level was similar to the level we found in the previous simulation. Performance was out of the scope of the prototype stage. Nonetheless, we did not notice streaming interruptions or delays. In future work, we intend to design and perform consistent and comprehensive tests of the device and other similar devices. Based on the results, we will be better able to determine which rules are to be executed in real time and which are to be executed online or in batch mode.

4. Lightweight public key cryptographic processor suited for IoT

Due to the vast number of IoT devices and high transmission volumes, a robust and adaptive cryptography system is required. However, since IoT devices have limited memory and computation power, they are unable to execute public key cryptographic systems. To cope with this limitation, we propose a lightweight RSA process. A combination of symmetric and asymmetric encryption systems is commonly used by the industry. Symmetric encryption systems require moderate computation resources and consequently are already used in IOT. However, asymmetric public key encryption requires vast computation resources, and as a result cannot be executed by most IOT devices. In this section, we describe a lightweight RSA encryption, where three improvements are incorporated: acceleration of modular exponentiation calculation, parallel and distributed multi-core processing, and splitting the original message if the message length is very long. After each part is completed, the system collects the intermediate results and loads them into a consolidation and integration process, which generates the result. We ran comprehensive encryption and decryption processes on messages of various lengths. The results prove that lightweight RSA is ready to be incorporated in IoT devices.

The rest of this section outlines the relevant literature review. Then, we describe an example of smart modular exponential calculation, which runs efficiently in an IoT architecture.

4.1. Literature review

Lin et al. [24] proposed the execution in parallel on CPU/GPU hybrids, of the Montgomery algorithm, to improve RSA performance and security. Fadhil and Younis [25] proposed a hybrid system, running RSA on multi-core CPU and multi GPU cores. For comparison purposes, they implemented variants of RSA, Crypto++, and the sequential counterpart. Multi-thread CPU improved performance by 6, over the sequential CPU implementation, and with GPU, it improved 23 times over the sequential implementation. The throughput gained for 1024 bits was ~1800 msg/sec, and for 2048 bits, it was ~250 msg/sec. Yanga et al. [26] suggested a parallel block Wiedemann algorithm in cloud to enhance the performance of GNFS and reduce communication costs, involved in solving large and sparse linear systems over GF.

4.2. Example of the acceleration of a modular exponentiation calculator

The calculation of "a factor b modulo n" is the heart of RSA cryptography and is also the most resource consuming component. Dividing this calculation into smaller parts will allow distributed and parallel processing of this calculation, where each smaller part is calculated by one sensor and later is integrated to obtain the result of "a factor b modulo n." The underlying concept is the following conceptual equation: ((a mod n) * (b mod n)) mod n = (a*b) mod n. This concept is used by the following algorithm to calculate modular exponentiation. Step 1: Translate the input into a binary number. Step 2: Start at the rightmost digit, let k = 0, for each positive digit calculate the value of 2^k, Step 3: Calculate mod n of the powers of two ≤ b, Step 4: Use modular multiplication properties to combine the calculated mod n values. Steps 2 and 3 can be executed in parallel by several connected sensors. The results from the sensors are then sent to the sensor requested the encryption/decryption, to execute step 4 and obtain the final result. Using a network of 7688 devices, we ran a comprehensive test, which proves the feasibility of executing RSA using parallel and distributed processing.

5. Conclusion

Connecting sensors to the Internet exposes the entire network to malicious penetrations. This is due to poor computation resources in standard sensors, which do not allow the execution of robust security systems. Hence, lightweight primitive systems should be implemented in IoT. To maintain current Internet security level, we adjusted implementations of known security concepts and mechanisms, which contribute to the security of the Internet of things. In this chapter, we focused on three key security elements where downsizing is feasible without compromising security: (a) Eliminating the frequent use of detailed data in the classification process. (b) Adjusted random forest machine learning to work in a distributed and parallel mode, when building the forest and during the detection process. (c) Adjust RSA cryptography calculations which are executed in parallel and distributed. The proposed solutions have smaller footprints, are efficient, and in most cases demonstrate better performance. We prove that downsizing and parallel processing are the most appropriate approaches for implementing comprehensive concepts for proper operation in constrained environments of IoT.

We are currently working on expanding current research areas. For example, additional improvements in RF implementation and exploring other machine learning technologies to check its applicability to IoT anomaly detection. We are exploring other asymmetric cryptography systems to check their applicability to IoT. In parallel, we are investigating authentication methods and technologies to discover a suitable one for IoT, or we are considering building an IoT-specific authentication.

Author details

Menachem Domb

Address all correspondence to: dombmnc@edu.aac.ac.il

Ashkelon Academic College, Computer Science Department, Ashkelon, Israel

References

[1] Aldosari HM, Snasel V, Abraham A. A new security layer for improving the security of internet of things (IoT). Ineternational Journal of Computer Information Systems and Industrial Management Applications. 2016;**8**:275-283. ISSN: 2150-7988

[2] Mairal FB, Ponce J, Sapiro G. Online dictionary learning for sparse coding. In: Proceedings of the 26th Annual International Conference on Machine Learning. Montreal, Quebec, Canada: ACM; 2009. pp. 689-696

[3] Jankov D, Sikdar S, Mukherjee R, Teymourian K, Jermaine C. Real-time high performance anomaly detection over data streams: Grand challenge. In: Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems. Tokyo, Japan: ACM; 2017. pp. 292-297

[4] Vlachos M, Freris NM, Kyrillidis A. Compressive mining: Fast and optimal data mining in the compressed domain. The VLDB Journal. 2015;**24**(1):1-24

[5] Sakurada M, Yairi T. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In: Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis. ACM; 2014. p. 4

[6] Reeves G, Liu J, Nath S, Zhao F. Managing massive time series streams with multi-scale compressed trickles. Proceedings of the VLDB Endowment. 2009;**2**(1):97-108

[7] Chilimbi TM, Hirzel M. Dynamic hot data stream prefetching for general-purpose programs. In: ACM SIG-PLAN Notices. Berlin, Germany: ACM; 2002;**37**(5):199-209

[8] Lane T, Brodley CE. Temporal sequence learning and data reduction for anomaly detection. ACM Transactions on Information and System Security (TISSEC). 1999;**2**(3):295-331

[9] Kasiviswanathan SP, Melville P, Banerjee A, Sindhwani V. Emerging topic detection using dictionary learning. In: Proceedings of the 20th ACM international conference on Information and knowledge management. ACM; 2011. pp. 745-754

[10] Aldroubi A, Cabrelli C, Molter U. Optimal non-linear models for sparsity andsampling. Journal of Fourier Analysis and Applications. 2008;**14**(5-6):793-812

[11] Mansoori EG, Zolghadri MJ, Katebi SD. SGERD: A steady-state genetic algorithm for extracting fuzzy classification rules from data. IEEE Transactions on Fuzzy Systems. Aug 2008;**16**(4):1061-1071. ISSN: 1063-6706

[12] Dubois D, Durrireu C, Prade H, Rico A, Ferro Y. Extracting decision rules from qualitative data using sugeno integral: A case-study. In: Proceedings of the 13th European Conference, ECSQARU 2015. Compiègne, France: Springer; Jul 2015;**9161**:14-24. ISBN: 978-3-319-20806-0; ISSN: 0302-9743

[13] Daniel W. Gulwani S, Hart T, Zorn B. FlashRelate: extracting relational data from semi-structured spreadsheets using examples. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation. Vol. 50, Issue. 6. New York: ACM; Jun 2015. pp. 218-228

[14] Kosina P, Gama J. Very fast decision rules for classification in data streams. Data Mining and Knowledge Discovery. Jan 2015;**29**(1):168-202. ISSN: 1384-5810

[15] Jafarzadeh H, Torkashvand R, Asgari C, Amiry A. Provide a new approach for mining fuzzy association rules using apriori algorithm. Indian Journal of Science and Technology. Apr 2015;**8**(S7):127-134. ISSN: 0974-6846

[16] Pourpanaha F, Peng Limb C, Mohamad Saleh J. A hybrid model of fuzzy ARTMAP and genetic algorithm for data classification and rule extraction. Elsevier, Expert Systems with Applications. 2016;**49**(7):4-85

[17] Mashinchi R, Selamat A, Ibrahim S, Krejcar O. Granular-rule extra action to simplify data. In: Intelligent Information and Database Systems. Vol. 9012 of the Series LNCS. Mar 2015. pp. 421-429. ISBN: 978-3-319-15704-7

[18] Yang H, Xiao C, Qiao Y. Study on anomaly detection algorithm of qar data based on attribute support of rough set. International Journal of Hybrid Information Technology. 2015;**8**(1):371-382. DOI: http://dx.doi.org/10.14257/ijhit.2015.8.1.33. ISSN: 1738-9968. IJHIT Copyright © 2015 SERSC

[19] Tang H. A simple approach of data mining in excel. In: 4th International Conference Browse Conference Publications, IEEE Xplore; 2008

[20] Tong S, Koller D. Support vector machine active learning with applications to text classification. Journal of Machine Learning Research, Leslie Pack Kaelbling. 2001:45-66

[21] Osugi T, Kun D, Scott S. Balancing exploration and exploitation: A new algorithm for active machine learning. In: Fifth IEEE International Conference on Data Mining. IEEE Xplore, 2005. DOI: 10.1109/ICDM.2005.33

[22] Lang T, Flachsenberg F, Luxburg U, Rarey M. Feasibility of Active Machine Learning for Multiclass Compound Classification. 2016. PMID: 26740007. DOI: 10.1021/acs.jcim.5b00332

[23] Bharathidason S, Jothi Venkataeswaran C. Improving classification accuracy based on random forest model with uncorrelated high performing trees. International Journal of Computer Applications (0975-8887). Sep 2014;**101**(13)

[24] Lin C, Liu J, Li C-C, Chu P-W. Parallel modulus operations in RSA encryption by CPU/GPU hybrid computation. In: Taiwan, Conference Paper, IEEE Xplore: 29. Jan 2015. DOI: 10.1109/AsiaJCIS.2014.25

[25] Fadhil HM, Younis MI. Parallelizing RSA algorithm on multicore CPU and GPU. International Journal of Computer Applications (0975-8887). Feb 2014;**87**(6)

[26] Yanga LT, Huanga G, Jun Feng B, Xua L. Parallel GNFS algorithm integrated with parallel block Wiedemann algorithm for RSA security in cloud computing. Information Sciences, Elsevier. 2016