10th Brazilian Congress on Computational Intelligence (CBIC'2011), November 8 to 11, 2011, Fortaleza, Ceará Brazil

DECISION TREE-BASED ENSEMBLES TO SPECIFY SECRET KEYS FOR CELLULAR AUTOMATA CRYPTOGRAPHIC MODEL

Luiz G. A. Martins and Gina M. B. Oliveira

Faculdade de Computação, Universidade Federal de Uberlândia
gustavo@facom.ufu.br; gina@facom.ufu.br

Abstract – A cryptographic method based on cellular automata (CA) was previously proposed which employs transition rules as secret keys. However, some rules belonging to the possible key space present undesirable behaviors that must be avoided. In a previous work, it was investigated the secret key specification for this cryptography model associating rules performance in ciphering with CA static parameters. A genetic algorithm-based data mining was performed to discover adequate key specification and it was employed to filter the set of all possible radius 2 CA rules. It was able to discover good secret key specifications. However, such filter provokes a significant decay in the number of good keys, while still keeping some underperforming rules. Adequate secret key specifications are investigated here using decision tree ensembles: bootstrap aggregating (bagging), boosting and random forest. The new filters are compared to the previous ones. By applying the new methodology, it was possible to find filters able to eliminate almost all underperforming rules and keeping a higher number of adequate secrete keys.

Keywords – Ensembles, decision trees, genetic algorithm, data mining, cellular automata, cryptography.

1 Introduction

Cellular automata are particularly well suited for cryptographic application and there are several previous studies in this topic [1]-[10]. Since CA rule is simple, local and discrete, it can be executed in easily-constructed parallel hardware at fast speed. Considering CA reverse interaction, given a lattice in time *t*, a possible antecessor lattice is determined for time *t-1*. This process is also known as pre-image computation. In previous papers [9-10], the application of the reverse interaction [12] as a cipher algorithm was investigated. Besides, static parameters were employed in [9] to specify CA rules as appropriate secret key. A rule was considered adequate in [9] if it presents a guarantee of pre-image existence for any possible lattice. Using a spatial entropy measure to evaluate the ciphering quality, the main conclusion in [9] is that the simple adoption of the reverse algorithm is not possible because only rules with 100% guarantee of pre-image existence are not appropriate for ciphering, since they do not exhibit a chaotic dynamics. A new approach has been emerged from this previous study [11]: it employs toggle transition rules as secret keys and it alternates the original reverse algorithm and a variation that uses extra bits in encryption when the pre-image computation fails. This variation is similar to pre-image computation adopted in Gutowitz's model [7] which also employs toggle rules. Since it is expected that in practice few failures happen, the ciphertext length will be close to the plaintext. This method was named Variable-Length Encryption Method (VLE) [11].

CA rules used as secret keys in VLE must be properly specified to exhibit two characteristics: *(i)* they must have low probability of failure during pre-image computation to returning a ciphertext length close to the original text; *(ii)* they must add a high level of entropy during the encryption to ensure a good ciphering quality. In [14], the secret key specification was better investigated. First, a representative rule set formed by all radius 2 right-toggle rules was analyzed. These rules represent about 50% of the possible secret keys in radius 2 space, being that the other 50% are dynamically equivalent (radius 2 left-toggle rules). The main conclusion is that there are some undesirable behavior rules in the complete set that must be avoided as secret keys. An analysis based on CA static parameters [13] was employed in [14] to capture the pattern associated to underperforming rules. Using nine static parameters and a genetic algorithm (GA) [15] to mine this pattern, it was able to find a good specification of rules to be used as valid secret keys. However, the number of available good rules decreases significantly and some undesirable ones are maintained in the filtered set.

In the present work, new secret key specifications to VLE encryption method are investigated using decision trees-based ensembles. Decision trees employ fast learning algorithms and they are able to generate comprehensible classifiers. In our experiments, machine learning algorithms available in Weka® were used. First, a single decision tree based on C4.5 algorithm was employed. Although this method results a good filter to adequate rules, it kept a significant number of undesirable rules due to the natural unbalancing of the data set. Subsequently, we adopted three ensemble methods - bootstrap aggregating (bagging), boosting and random forest - attempting to deal with this unbalanced nature of the data set. Ensemble methods use multiple models to obtain better predictive performance than could be obtained from any of the base models. The principle behind ensembles is that prediction can base be improved by aggregating weaker independent classifiers. The main goal of this work is to filter the undesirable rules of the entire secret key space, without drastically reducing the number of adequate secret keys. The new classifier presents a great improvement, mainly in the underperforming rules reduction.

2 CA Applied in Cryptography

A cellular automaton consists of a lattice of cells and a transition rule. Each cell presents in each time *t* one of *k* distinct states. A cell is updated in discrete time steps and its new state depends on the states of the *2R+1* neighborhood cells, where *R* is the

CA radius. In the case of a deterministic one-dimensional CA, whose example is showed in Figure 1, the state of the cell i at time $t+1$ is determined by the transition rule τ:

$$a_i^{(t+1)} = \tau[a_{i-R}^{(t)}, ..., a_i^{(t)}, ..., a_{i+R}^{(t)}] \quad (1)$$

The dynamics of a cellular automaton is associated with its transition rule. In order to help forecast the dynamic behavior of CA, several static parameters have been proposed [13], some of them are: Z derived from the pre-image computation algorithm and it is composed by Z_{left} and Z_{right}. [12]; symmetry level (S) of a rule transition [9]; neighborhood dominance (ND), activity propagation, sensitivity and absolute activity (AA) [13]. The idea behind such static parameters is to perform a simple calculus over the rule transition output bits. Based on the result, one can predict the most probable behavior when the rule is applied to an arbitrary lattice. For example, if a transition rule has low Z (next to 0) it is expected a fixed point behavior, while if it has high Z (next to 1), the chaotic behavior is more probable.

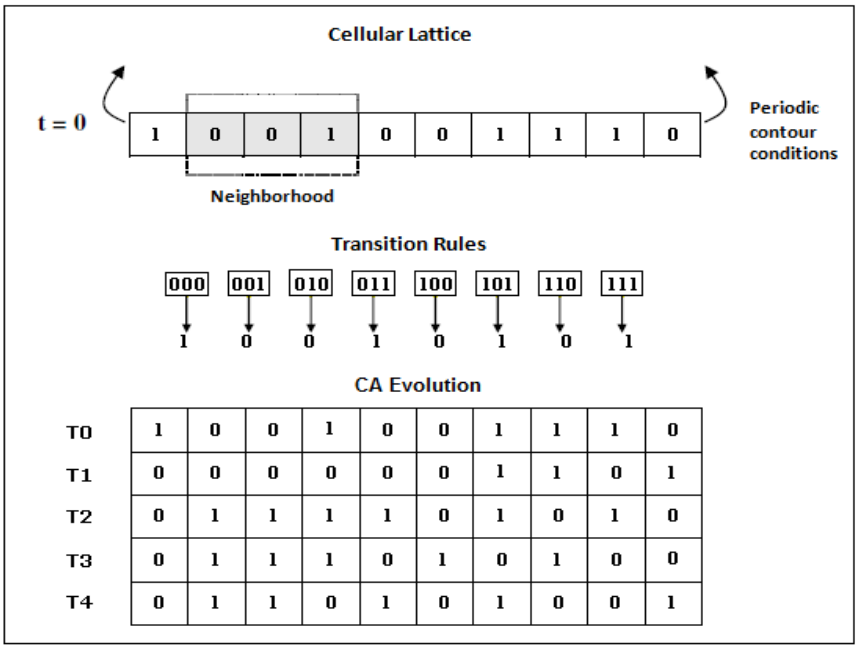

Figure 1 – Example of Deterministic 1D Cellular Automata.

Basically, CA-based cryptographic models can be divided into three classes: (*i*) models that use CA to generate binary sequences with good pseudo-random properties, which are used as cryptographic keys, but the effective ciphering process is made by another function [1-4]; (*ii*) models based on additive, non-homogeneous and reversible CA, that use algebraic properties of this kind of rules to generate automata of maximum and/or known cycle [5-6]; and (*iii*) models based on irreversible CA, which uses the backward interaction of cellular automata in the ciphering process and the forward interaction to decipher [7-11], as the cryptographic model discussed here.

Gutowitz has previously proposed a cryptographic model based on backward evolution of irreversible CA [7]. Toggle CA transition rule is used as the secret key in his model. A pre-image of an arbitrary lattice is calculated adding extra bits in each side of the lattice. This increment is pointed as the major flaw in the model. An efficient reverse algorithm was proposed by Wuensche and Lesser [12] for a periodic boundary condition, keeping the pre-image with the same size of the original lattice. Such algorithm was evaluated as encryption method in [9]. However, its usage has the disadvantage that there is no guarantee of pre-image existence for any given lattice and any given rule. Thus, the pre-image computation can fail if a Garden-of-Eden state [12] is found during ciphering. The only rules with assurance of pre-image existence are not appropriate for ciphering because they do not exhibit a chaotic dynamics.

3 VLE Method

3.1 General Description

Due to drawback of the previous methods, a new approach was proposed in [9] and developed in [11], which alternates the original reverse algorithm and the variation that uses extra bits, using the second only when the pre-image computation fails. This variation is similar to pre-image computation adopted in Gutowitz model [7]. Although this approach needs to add bits to the ciphertext when a failure occurs, it is expected that in practice few failures happen and the ciphertext length will be equal or close to the plaintext size. In the resultant method, encryption always succeeds and the final length of the ciphertext is not fixed. This method was named Variable-Length Encryption Method (VLE) [11]. VLE method works as it alternates rounds of pre-image computation performed by reverse algorithm (a variation of Gutowitz's model for periodic conditions) with few or

none steps of pre-image computation performed by Gutowitz's model. Ciphering is made by computing P consecutive pre-images starting from a lattice of size N corresponding to the plaintext. The secret key is a radius-R CA rule τ generated with an appropriate specification based CA static parameters.

Suppose that VLE started to calculate pre-images using reverse algorithm and the secret key τ and it fails in the K-th pre-image such that $K \leq P$. In such situation the ciphering process uses the modified reverse algorithm with extra bits to calculate the K-th pre-image. Thus, the K-th pre-image will have N+2R cells. Ciphering returns again to the original reverse algorithm for remaining pre-image computations. If all the subsequent pre-images computation succeeds the final ciphertext will have a size of $N + 2R$. If the pre-image computation fails again, the ciphering process changes and adds 2R more bits to the lattice. If the process fails in F pre-images ($F \leq P$) the final lattice will have N+2FR. Starting from a lattice of N cells, the size of the ciphertext after P pre-images computation is given by $N \leq$ ciphertext size $\leq N +2PR$.

For practical reasons related to the speed of the encryption process, it can be better to limit the method to operate with only toggle rules. A CA transition table is said to be a toggle rule if it is sensible in respect to a specific neighborhood cell, that is, if any modification of the state on this cell necessarily provokes a modification on the new state of the central cell, considering all possible neighborhoods [7]. Therefore, it speeds up the pre-image computation, since that will be removed the ambiguities and there is no possibility of the algorithm to come back in the computation before arriving at the end of the lattice.

Deciphering is executed applying the forward interaction of cellular automata rules. By starting from the ciphertext the recipient needs to apply the transition rule τ forward by P steps and the final lattice will be the plaintext. He also needs to know in which pre-images failures happened to recover the original text.

3.2 Evaluation of VLE Secret Key Space

Applying VLE method, any toggle rule is able to complete the ciphering process starting from any plaintext (initial lattice). However a short length ciphertext depends on the secret key specification. Some experiments were performed in [14] to analyze VLE's performance and to evaluate rules specification using the complete set of radius 2 right-toggle rules: all of them have $Z_{left} = 1$ and $0 < Z_{right} < 1$. As a radius 2 toggle rule is defined by only 16 bits (since the other 16 bits are deterministically defined due to toggle property), this set is composed by 65536 (2^{16}) rules. These rules represent 50% of the possible secret keys in radius 2 space. The other 50% of secret keys are the all radius 2 left-toggle rules ($Z_{right} = 1$), which are dynamically equivalent to the set of right-toggle rules.

VLE-based environment was employed in [14] to cipher a hundred 256-bits plaintexts using each right-toggle rule (of 65536 rules), by calculating 48 consecutive pre-image steps (P). The number of consecutive pre-images was experimentally determined by preliminary investigations. Based on the results, it is possible to determine some performance metrics as following:

- *Ciphertext Length* (L_{mean}): by applying VLE method, the final length of the ciphertext can be between N and N+2PR. Aiming to evaluate if the expected final length is in fact close to N, it was calculated the mean length (L_{mean}) of the ciphertexts (final lattices), related to the mean number of failures (F_{mean}) occurred during the ciphering. In the experiments performed in [14], a number of fails above 10 was considered inadequate because it returns a ciphertext with 300 bits or more starting from a plaintext of 256 bits. The ciphertext length obtained for all 65536 right-toggle rules ciphering 100 plaintexts highlights the existence of several secret keys returning at least one ciphertext with size equal or above 300 bits. About 800 rules returned long ciphertext length for at least one random plaintext evaluated and 478 rules returned L_{mean} above 300 bits considering the 100 evaluated plaintexts.

- *Ciphering quality* (E_{mean}): by comparing ciphertexts generated from two very similar plaintexts it is possible to evaluate the encryption quality and specially its protection against a differential cryptanalysis-like attack. Cryptanalysis tries to find the plaintext after getting the ciphertext without knowing the secret key [16]. Sen *et al*. (2002) have used the same idea to analyze their CA cryptosystem named CAC comparing it with DES and AES cryptosystems [17]. In such analysis, several pairs of plaintext (X, X'), that differ one of the other by a fixed and small difference D, is used to generate a pair of ciphertexts (Y, Y'), which differ one of the other by a difference D'. This difference is obtained by Hamming distance. That is, applying XOR operations between Y and Y' and counting the number of 1s in D'. In the experiments performed in [14], the difference D between two plaintexts X and X' was fixed in only one bit in any arbitrary position over the lattice and the value of D' was calculated for the complete set of right-toggle rules. The computation of D' to each pair (Y, Y') was performed to obtain the entropy. This measure aims to verify if D' does not keep any pattern that eventually could help a cryptanalyst. Spatial entropy [14] was calculated on D' to evaluate the existence of some undesirable regularity on this difference. Entropy below 0.5 indicates a strong pattern in difference D', in other words, a low ciphering quality. When ciphering quality metric E_{mean} was calculated for all 65536 right-toggle rules ciphering the 100 plaintexts [14], it was possible to verify that there are rules with mean values of entropy below 0.5, indicating that these rules does not perform an adequate encryption of the plaintexts in average. The most probable behavior in such cases is that the rule only shifts the initial lattices, not performing an actual encryption of these plaintexts. This behavior cannot be allowed in a secure cryptosystem. Considering the entire rule set (right and left-toggle rules), about 3200 rules returned entropy below 0.5 for at least one pair Y and Y' and 879 rules returned E_{mean} below 0.5 considering the 100 evaluated plaintexts.

Based on performance metrics L_{mean} and E_{mean}, the main conclusion in [14] was that there are undesirable behavior rules in the complete set analyzed - considering a cryptographic purpose - that must be avoided as secret keys. Therefore the entire rule space formed by all radius 2 right-toggle rules is not appropriate to be applied as secret keys in VLE method: 1357 rules (~2% of the key space) must be avoided: 879 due to low entropy and 478 due to long ciphertext length.

3.3 Previous Specifications Using Static Parameters and GA-Based Data Mining

Secret key specification was first proposed in [9] trying to filter undesirable behavior rules when using the pure reverse algorithm [12] as ciphering process: static parameters *S* and the components Z_{left} and Z_{right} were used. However, some preliminary experiments using the complete set of right-toggle rules had evidenced that their application is not effective as supposed, when VLE method is used. Aiming to better understand the relation between CA static parameters and underperforming rules, an analysis was developed in [14]. A series of CA parameters were calculated trying to identify a pattern to filter inadequate rules of the complete key space formed by all radius 2 right-toggle rules. Nine parameters were used in this analysis: Z_{right}, *S*, BWLR_Symmetry (*BWLR*), LR_Symmetry (*LR*), Absolute Activity (*AA*), Neighborhood Dominance (*ND*), Sensitivity (μ) and Activity Propagation (*AP*) [13-14]. A new parameter was also proposed and used in [14]: it is the spatial entropy associated to the 16 bits that define the 32-bits toggle rule (the rule core), named here as *core entropy* (*CE*). All these parameters were calculated to each one of the 65536 right toggle radius 2 rules. The parameter values are normalized between 0 and 1. A database was elaborated in which each register corresponds to one right-toggle rule and the fields are composed by the values of the nine parameters calculated for each rule, and the values of the performance metrics (L_{mean} and E_{mean}) where calculated when the rule is applied to cipher 100 random plaintexts.

As a pattern associating parameters with the underperformance rules was not possible to recognize by a simple visual inspection, a GA-based data mining process was applied with this goal. Standard genetic algorithm was elaborated based on the model described in [18]. As illustrated in Figure 2, the individual is composed by *I* genes, where *I* is the number of CA static parameters analyzed (*I*=9). The *i*-th gene is subdivided into three fields: weight (W_i), operator (O_i) and value (V_i). Each gene corresponds to one condition in the antecedent part (IF) and the individual as a whole is the rule antecedent. The weight field is an integer variable and its value is between 0 and 10. This field determines the insertion or no of the correspondent gene in the rule antecedent. If this value is lesser than a boundary-value this gene will not appear in the rule, otherwise the gene appears. The value 7 was used as the boundary-value [14]. The operator field can be < (minor), ≥ (larger or equal) or ≠ (different). The value field is a floating-point number that can vary between the 0 and 1, because each parameter was normalized in such range.

Gene 1			**Gene 2**			…	**Gene 9**		
Z_{right}			***S***			…	***CE***		
W_1	O_1	V_1	W_2	O_2	V_2		W_9	O_9	V_9

Figure 2 –Individual representation.

To establish the consequent part of the rule, the fields F_{mean}, and E_{mean} of each register of the database was analyzed aiming characterize the underperformed rules in specific classes. Field *Class* was added to the database with the classification of each register in one of these classes:

- *Class 1*: rules with low mean entropy ($E_{mean} < 0.5$). There are 879 right-toggle rules.
- *Class 2*: rules with large mean ciphertext length ($L_{mean} \geq 300$ bits). There are 478 right-toggle rules.
- *Class 3*: rules with adequate features to secret key. There are 64,179 reminiscent rules.

The individual illustrated in Figure 1 represents only the antecedent part of the rule (IF). The consequent part is always in the format *THEN Class = C*, being that *C* can be 1, 2 or 3. However, it is omitted in individual's representation. Conversely, it is a fixed execution parameter of GA. Thus, if the GA is executed with $C = 1$, all the rules of population represent classification rules in the format **IF** *ANTECEDENT* **THEN** *Class is "Low entropy"*. All registers are considered either *Class 1* or *not Class 1*. Fitness quantifies the quality of the rule associated to each individual. Two indicators commonly used in classification are *Sensitivity* and *Specificity* [18]. In [14], individual fitness is given by a weighted sum between *Sensitivity* and *Specificity*. Stochastic tournament with Tour = 3 is used as the matting selection method. Two-point crossover is applied and a specific mutation operator is used to each type of gene field with a rate of 30%. Each GA experiment was formed by 100 runs, using a population of 100 individuals, which were evolved by 100 generations. The classification rule that was indeed intended to be mined was Class 3, because it represents appropriate rules to be used in cryptography. However, the other two classification rules (classes 1 and 2) were also important to achieve this goal, because they better characterize low entropy rules and long ciphertext ones, given information to prune the rules returned by GA for Class 3. After several executions and manual post-processing pruning procedures the rule following was found in [14]:

IF ($S \neq 1$ **AND** $ND \leq 0.57$ **AND** $CE > 0.65$ **AND** $Z_{right} \neq 1$) **THEN** Class = 3

This rule employs 4 of the 9 CA parameters to characterize adequate secrete keys. It was applied as a filter in the complete radius 2 right-toggle CA rule set. The filtered set has 51,495 right-toggle rules: it reduces 21.4% of the entire key

space. When the underperforming rules are analyzed, the advantage of such filter is evidenced: only 28 rules with such inappropriate low entropy remains (out of 879 rules) and 217 rules with a ciphertext length above 300 bits remains (out of 750 rules). Therefore, it was concluded in [14], the GA-mined filter is a good specification for CA secret keys.

4 Ensemble-Based Models

Ensemble methods combine multiple hypotheses in order to form a hopefully better hypothesis. That is, an ensemble is a technique for combining many weak classifiers (base classifiers) in an attempt to produce a strong one [19]. Generally, the ensemble decision is generated from the base classifiers hypothesis by voting. Therefore, ensemble hypothesis is not necessarily contained within the hypothesis space of the models from which it is built. Thus, ensembles can be shown to have more flexibility in the functions they can represent. Diversity and accuracy are features desirable for good performance of the ensembles [19]. Diversity can be obtained when the base classifiers are independents (its errors are not correlated). Accuracy can be obtained if the base classifiers results individually better performances than random classification (> 0.5). Evaluating the prediction of an ensemble typically requires more computation than evaluating the prediction of a single model. Thus, fast algorithms such as decision trees are commonly used with ensembles, although slower algorithms can benefit from ensemble techniques as well. Other advantage of decision tree is its capability of comprehensibility, that is, it is able to understand the criteria used by model to classify a rule.

In present work, three kinds of ensembles were tested: two meta-algorithms based on instances manipulation (bootstrap aggregating-bagging and boosting) and one algorithm based on instances and attributes manipulation (random forest). All ensembles adopted decision trees as base classifiers. The models was generated, trained and tested through the Weka® Explorer framework. In bagging method, each model in the ensemble vote with equal weight. In order to promote model variance, bagging trains each model in the ensemble using a different subset of the training set. Bagging randomly generates each new training subset by uniform sampling examples from original one. In this process is used the bootstrap approach, which employed sampling with replacement. Boosting involves incrementally building an ensemble by training each new model instance (base classifier) to emphasize the training examples that previous models misclassified. In other words, data is dynamically reweighted: misclassified examples increase their weight and corrected classified examples decrease their weight. In some cases, boosting has been shown to yield better accuracy than bagging, but it also tends to be more likely to overfitting training data. The most common implementation of boosting is Adaboost, which was used in this work. Random forest is an ensemble classifier that consists of many decision trees. The ensemble hypothesis is obtained by the mode of the outputs of the individual trees. The random forest algorithm combines random decision trees (attributes selection) with bagging (instances sampling) to achieve very high classification accuracy. For each node of the tree, the algorithm randomly chooses of k attributes (being k << total of attributes) and calculates the best split based on these attributes in the training subset. The selection of a random subset of attributes is a way to implement stochastic discrimination [20]. Each tree is fully grown and it is not pruned. Random forest presents: *(i)* a fast learning algorithm; *(ii)* it runs efficiently on large databases; and *(iii)* it estimates what variables are important in the classification. In other hand, it is more likely to overfitting some datasets, being more pronounced in noisy classification or regression tasks.

5 Experiments

Although GA-mined filter obtained in [14] defines a good specification for VLE cryptographic model, it provokes a significantly reduction of available keys. Besides, some undesirable rule still remains in the resultant filtered key set. In this section new experiments are described which were carry out aiming to identify new secret key specifications based in decision trees ensemble methods.

5.1 Preprocessing

This work also uses the complete radius 2 right-toggle rules set employed in GA mining experiments described in [14]. The full set formed by 65,536 radius 2 rules presents an unbalancing in the registers quantity between the classes: 879 rules with low entropy (class 1); 478 rules with large ciphertexts length (class 2); and 64,179 rules classified as adequate secret keys (class 3). Due to such high unbalance, the algorithms guide the learning process to identify the majority class, prejudicing the detection of the minority classes. Therefore, the complete rule set was worked in order to aid the machine learning algorithms to find adequate classifiers. Initially, a random sampling of the registers of majority class was made in order to decrease the unbalancing; being the number of instances of the class 3 is limited up to four times the total of instances of two others (classes 1 and 2). The new subset contains approximately 5400 class 3 instances. In this process, three different random seeds were employed for generation of subsets. Each subset was again divided in training and test subsets through a stratified 10-fold cross-validation sampling, which are applied to evaluate of the classifiers performance [19]. Therefore, 30 subsets were used for each machine learning method. Furthermore, two types of experiment were performed: the first type uses 3 classes as defined previously (low entropy, large ciphertext and adequate rules); the second type joined the undesirable rules into only one class defining only two classes (inadequate and adequate rules). All preprocessing was carried on Matlab®, as well the generation of data files in the Weka® format (*arff*).

5.2 Ensembles-Based Experiments

Each machine learning method (single decision tree and ensembles approaches) was applied through the Weka® Explorer framework to all training and test subsets obtained in pre-processing step. Table 1 presents mean values of performance metrics obtained using learning methods in 2-classes test subsets. These values are grouped by random seeds. This table contain the quantity of adequate rules selected (TP - true positive); underperforming rules effectively detected (TN - true negative); adequate rules incorrectly classified as underperforming ones (FN - false negative); underperforming rules incorrectly classified as adequate ones (FP - false positive); as well the indicators *Sensitivity*; and *Specificity*.

Table 1 – Mean values of the methods specifications applied to test subsets with 2 classes.

Run	Metrics	Single DT	Bagging	Boosting	Random Forest
0	TP	524	529.5	529.7	527.1
	FN	18.8	13.3	13.1	15.7
	FP	25.2	26.4	6.1	7.9
	TN	110.5	109.3	129.6	127.8
	Sensit.	**0.965**	**0.975**	**0.976**	**0.971**
	Specif.	**0.815**	**0.805**	**0.955**	**0.941**
1	TP	526.5	529.7	529.1	527.5
	FN	16.3	13.1	13.7	15.3
	FP	26.8	27	6.7	8.5
	TN	108.9	108.7	129	127.2
	Sensit.	**0.970**	**0.976**	**0.975**	**0.972**
	Specif.	**0.802**	**0.802**	**0.950**	**0.938**
2	TP	526	528.8	528.7	528.7
	FN	16.8	14	14.1	14.1
	FP	27.8	26	6.8	7.7
	TN	98.9	109.7	128.9	128
	Sensit.	**0.969**	**0.974**	**0.974**	**0.974**
	Specif.	**0.745**	**0.808**	**0.950**	**0.943**

The executions of single decision tree-based methods aim to generate baseline performance metrics for evaluate of ensemble classifiers and determine the values of configuration parameters of the decision tree (DT) employed in the others methods. The adopted decision tree after some preliminary experiments uses a prune confidence factor of 0.25; allows at least six instances for each leave node; uses C4.5 algorithm and 10-fold cross-validation to training validation. The training process of this kind of classifier spends about 5.5 seconds and generates decision trees with 125 nodes and 63 leave nodes in average. The best decision tree obtained was applied as a filter over the complete radius 2 right-toggle CA rule set. The filtered set has 62,443 rules (reduction of 5% considering the entire key space), remaining 202 underperforming rules (97 rules with low entropy and 105 with a larger ciphertext length).

Three kinds of ensembles also were evaluated in this work: bagging, boosting and random forest. All ensembles approaches adopted the same parameter configuration used in the single decision tree executions to their base classifiers. In addition, each method has its specific parameters.

Considering the ensemble method based on bagging, it was performed 5 iterations during the training, where the size of each bag is the same of the training subset. The choice of the number of iterations was also defined after some exploratory experiments aiming to reduce training processing time without compromising the accuracy of the model, returning a mean processing time of 30 seconds. However, its processing time is greater than the single decision tree model, since that this method generates 5 decision trees from different bootstrap sampling. The decision trees generated are bigger than those obtained using single decision tree models. They have 153 nodes and 72 leaves in average. The best bagging ensemble obtained was applied over full set, generating a filtered set of 62,972 instances (about 3.9% of reduction), being that 95 rules belonging to class 1 (low entropy) and 106 rules belonging to class 2 (large ciphertext length).

In boosting approach, Adaboost M1 (Weka® algorithm) was used with 10 iterations. The adoption of such number of iterations (greater than the number used in bagging) aims better performance of boosting. The best boosting ensemble obtained was applied as filter over the complete radius 2 right-toggle rule set. The filtered set has 62,768 rules (reduction of 4.2% of the entire key space), remaining 6 underperforming rules (4 rules with low entropy and 2 with a larger ciphertext length). As it can be observed, there is a great reduction of underperforming rules in this method. We believe it happens due to the better regulation of the attribute weights realized during training phase, since there are more specialized/adjusted trees. In the other hand, it provokes a significantly increase of the mean processing time during training to approximately 72 seconds. This is the worst processing time between learning methods investigated here. Besides, executions with boosting returning the biggest decision trees (298 nodes and 150 leaves in average).

Random forest method experiments were set to generate five decision trees per execution and employees 4 input attributes. This configuration returned a performance similar to the boosting taking a processing time close to these obtained to

single decision tree method (7.5 seconds in average). When applied over the full rule set, the best random forest ensemble obtained a filtered set of 62,556 reminiscent rules (about 4.6% of reduction), being 5 rules with low entropy and 5 rules with large ciphertext length.

As it can be noted in Table 1, the usage of a single decision tree and the bagging approaches generated the worst performances. It was biased by the great number of instances in majority class, resulting in greater values of false positive (underperforming rules classified as adequate keys). In other hand, boosting and random forest approaches presented the best performances, resulting in similar metrics.

For better comparing the two approaches, three classifiers (optimist, pessimist and mean ensembles of each seed) were applied in full set. The mean values of the metrics obtained by the best filter of both methods as illustrated in Table 2. As one can see, the values returned by both approaches are very similar, although boosting has a bit of advantage since it returns the same sensitivity and a higher specificity, which is the major capability we desire (because it is related to the misclassification of inadequate rules).

Table 2 – Results of the best ensembles applied over the full rule set with 2 classes.

Method	TP	FN	FP	TN	*Sens*	*Spec*
Boosting	62865	1314	5	1352	**0.980**	**0.996**
Random Forest	62693	1486	9	1348	**0.980**	**0.993**

Attempting to better identify which method generates the best filter, it was verified the behavior of ensembles generated using 3 classes instead of 2. For simplicity, we used boosting and random forest approaches applied only to the training and test subsets in which the best ensembles were obtained in 2-classes problem. The resultant 3-classes specifications were applied to full rule set and the confusion matrixes are presented in Table 3.

Analyzing this table, boosting obtains the best trade-off, that is, it provided a more reduction of underperforming rules, without considerably compromising the quantity of available rules in filtered set. However, the result obtained by random forest also is very good, being able to be used.

Table 3 – Confusion matrixes obtained in 3 classes full set.

	Boosting			**Random Forest**		
	C1	**C2**	**C3**	**C1**	**C2**	**C3**
C1	875	0	4	870	4	5
C2	4	472	2	0	473	5
C3	990	427	62762	1038	595	62546

Aiming to compare the best filters obtained using each approach, Table 4 presents the results found. Each row corresponds to the used filter criteria applied to database, except for the first row which represents the complete radius 2 toggle rule set (*Fullset*). The other rows presents: filter generated through GA-based data mining process (*GA-Subset*) described in [14]; single decision tree-based filter (*DT-Subset*); ensemble filters based on bagging, boosting and random forest approaches (*BA-Subset*, *BO-Subset* and *RF-Subset*, respectively). The second column corresponds to total number of reminiscent rules in each filtered set. Other columns correspond to number of rules in each class (1, 2 or 3). It is desirable a small quantity of rules in classes 1 (low entropy) and 2 (large ciphertext length); and a high number of rules in class 3 (adequate keys).

Table 4 – Reminiscent rules after filtering (3 classes).

Filter Criteria	Number of rules	Low Entropy (C=1)	Large Length (C=2)	Good Rules (C=3)
Fullset	65536	879	478	64179
GA-Subset [14]	51495	28	217	51250
DT-Subset	62443	97	105	62241
BA-Subset	62972	95	106	62771
BO-Subset	62768	4	2	62762
RF-Subset	62556	5	5	62546

5 Conclusions

The appropriate specification of adequate cellular automata rules as secret keys for VLE cryptographic method was deeper investigated in the present work. It was used a set formed by all the 65536 radius 2 right-toggle rules as potential secret keys and different machine learning approaches were used to relate the adequacy of rules and their static parameters. Initially, we analyzed specification obtained on previous work [14], which used a standard genetic algorithm and several CA static parameters to mine adequate rules in this same data set. Although a good specification was found in [14], its usage as a filter

over the key space provokes a severe decay of the number of available good rules and some undesirable ones are maintained. In order to improve filter specification, the usage of single or ensemble methods based on decision trees is investigated here.

Considering the mean convergence time (training), the usage of single decision tree or random forest ensemble were must faster than bagging and boosting approaches. Boosting has the double of iterations in relation to bagging turning it the slowest approach. Besides, boosting generates the biggest decision trees, with twice number of nodes in relation to bagging. The smallest decision trees are generated by single C4.5 algorithm. Analyzing the performance metrics obtained in experiments, does not exist a significant variation in rate of false negative between the methods. This indicates a similarity in relation to majority class prediction. However, considering the reduction of underperforming rules in filtered set, it is possible to rank the methods in relation to false positives: single decision tree and bagging classifiers as the worst filters. Although they return large number of rules in filtered sets (bagging generated the biggest one) and they are able to decrease the number of rules with large ciphertext length in relation to previous specification published in [14], they also increase the number of reminiscent rules with low entropy. Such kind of rules is prohibitive for cryptography specifications. Boosting and random forest approaches present a significant improvement on the prediction of minority classes (they returned the best results), being that the values obtained by boosting method are a bit better, while random forest method is much faster (random forest is almost ten time faster than boosting). Both methods also returned a small reduction of the secret key space, specially when compared with the previous specification obtained in [14]. Therefore, both methods produced excellent specifications to secret keys and they overcame the specification obtained by genetic algorithm in [14].

6 References

[1] S. Wolfram, Cryptography with cellular automata. **Int. Cryptology Conf. (Crypto'85)**. LNCS. 218 (1986), 429-432.

[2] M. Tomassini, M. Perrenoud, Stream Ciphers with One and Two-Dimensional Cellular Automata. **Parallel Problem Solving from Nature VI**. LNCS. 1917(2000), 722-731.

[3] F. Seredynski, P. Bouvry, A.Y. Zomaya, Secret key cryptography with cellular automata. **Workshop on Nature Inspired Distributed Computing**, (2003), 149-155.

[4] M. Benkiniouar, M. Benmohamed, Cellular Automata for Cryptosystem. **IEEE Conference Information and Communication Technologies**, (2004), 423-424.

[5] J. Kari, Cryptosystem based on reversible cellular automata, **Personal communication**, (1992), Apud in (Seredynski, Bouvry and Zomaya, 2003).

[6] S. Nandi, B. Kar, P. Chaudhuri, Theory and Applications of CA Automata in Cryptography, **IEEE Trans. on Computers**, 43 (1994), 1346-1357.

[7] H.Gutowitz, Cryptography with Dynamical Systems,**Cellular Automata and Cooperative Phenomena**,1(1995),237-274.

[8] G.M.B. Oliveira, A. Coelho, L. Monteiro, Cellular Automata Cryptographic Model Based on Bi-Directional Toggle Rules, **Int. Journal of Modern Physics C**, 15 (2004), 1061-1068.

[9] G.M.B. Oliveira, H. Macêdo, A. Branquinho, M. Lima, A cryptographic model based on the pre-image computation of cellular automata, **Automata-2008: Theory and Applications of Cellular Automata**, (2008), 139-155.

[10] A. Wuensche, Encryption using cellular automata chain-rules, **Automata-2008: Theory and Applications of Cellular Automata**, (2008), 126-138.

[11] G.M.B. Oliveira, L.G.A. Martins, L.S. Alt, G.B. Ferreira, Investigating a Cellular Automata-Based Cryptographic Model with a Variable-Length Ciphertext. **International Conference on Scientific Computing**, (2010).

[12] A. Wuensche, M. Lesser, Global Dynamics of Cellular Automata. **Addison-Wesley**, (1992).

[13] G.M.B. Oliveira, P. de Oliveira, N. Omar, Definition and applications of a five-parameter characterization of 1D cellular automata rule space, **Artificial Life**, 7:3 (2001), 277-301.

[14] G.M.B. Oliveira, L.G.A. Martins, G.B. Ferreira, L.S. Alt, Secret Key Specification for a Variable-Length Cryptographic Cellular Automata-Based Model, **Int. Conf. on Parallel Problem Solving from Nature**, LNCS, 6239(2010), 381-390.

[15] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, **Addison-Wesley**, (1989).

[16] W. Stallings, Cryptography and Network Security: Principles and Practice, **Prentice Hall**, (2003).

[17] S. Sen, C. Shaw, D. Chowdhuri, N. Ganguly, P. Chaudhuri, **Cellular Automata based Cryptosystem (CAC)**. LNCS, 2513 (2002), 303-314.

[18] M. Fidelis, H. Lopes, A. Freitas, Discovery comprehensible classification rules with a genetic algorithm. **Cong. on Evolutionary Computation**, (2000), 805-810.

[19] K. Faceli, A.C. Lorena, J. Gama, A.C.P.L.F. de Carvalho, Inteligência Artificial: Uma abordagem de Aprendizado de Máquina, **unpublish**, (2011).

[20] E. Kleinberg, An Overtraining-Resistant Stochastic Modeling Method for Pattern Recognition, **Annals of Statistics**, 24:6 (1996), 2319–2349.