Applying adaptive technology in data security

Éder José Pelegrini[1] João José Neto[1]

[1]Escola Politécnica da Universidade de São Paulo

eder.pelegrini@poli.usp.br, joao.jose@poli.usp.br

Abstract

This paper proposes a new approach for data security and reports the application of adaptivity in the design of an adaptive technology-based experimental cipher system. Our goal with this approach is to build a fast and secure cipher system based on two practices: using the cipher key to encode part of the information on the execution algorithm; and hiding another part of that information as controlled dynamic modifications imposed through changes applied to the execution algorithm. Because of the interesting properties of this proposal, authors expect the main ideas present in this paper be used in actual cryptosystems in a near future.

1. Introduction

A common sense in cryptography is that encrypting and decrypting methods be publicly known, so its secrecy lies on the cryptography key. This idea is stated by Kerckhoff principle. The alternative of keeping hidden the description of the algorithm in order to achieve security is considered an ineffective strategy [Tanenbaum, 2003].

A publicly known algorithm allows cryptanalysts to explore its fragile points, i.e., knowledge on the operation of the algorithm allow inferring more efficient ways for breaking the cipher than simply doing exhaustive trials in order to determine the proper key. For example:

- The obsolete Caesar cipher accepts any integer number as a key. Its ciphering idea consists of replacing each letter in a text with another one, displaced by a fixed amount in the alphabet sequence [Bishop, 2002][Tanenbaum, 2003]. Actually, there are only twenty-six different valid keys in this scheme, whose exhaustive trial may be easily performed.
- The RSA is an asymmetric cipher [Bishop, 2002][Rivest; Shamir and Adleman, 1978][Tanenbaum, 2003] whose security relies on the difficulty of the factorization problem for large integers [Rivest; Shamir and Adleman, 1978]. Solving this problem requires breaking the cipher, despite the difficulty of determining prime factors for large integers.

Modern crypto-algorithms use sophisticated sequences of operations in order to avoid such attacks and increase their security. As an example, the AES algorithm [Bishop, 2002][Daemen and Rijmen, 1999][Tanenbaum, 2003] is based on the Galois Field theory and uses multiple rounds. On each round, substitution, transposition and exclusive-or operations are performed.

This paper presents a novel approach to perform data ciphering (and deciphering) through the use of adaptive technology. It proposes a novel way for the design of a cryptosystem-like ciphering system which hides part of the ciphering/deciphering logic in a way that both ensures Kerckhoff's principle and increases the difficulty to identify and explore its fragilities. Both the information and the logic changes are based on the chosen secret key.

This method allows changing dynamically the hidden logic, driven by the contents of the input text, making it harder to explore its fragilities. Adaptive Technology is used to achieve that goal. Our proposal considers hiding part of the algorithm's logic and changing it in order to get a secure and fast ciphering algorithm for data security. Such approach can eventually be used as a starting point towards the design of a full cryptosystem.

It is important to emphasize that the aim of our proposal is to illustrate the use of an adaptive formalism to investigate new ways for achieving data security. For illustrating purposes only, a simple ciphering system is proposed below by means of a very simple example, in order to communicate ideas and validate concepts.

The remainder of this paper is organized as follows: section 2 presents motivations and previous works; section 3 describes the formalism used in this paper; section 4 explains the concepts and a way for building a cipher system based on adaptive technology; section 5 shows and demonstrate a first development of a cipher system based on this approach; section 6 discusses the security of this approach; at last, section 7 concludes this paper, being followed by the references.

2. Previous Work

Neto [2002] defined adaptive automata as self-modifying rule-driven devices based on structured pushdown automata. In this publication, the classical adaptive automaton's mechanism is slightly modified in the proposal of the adaptive finite-state transducers.

Lewis and Papadimitriou [1981] and Neto [1987] state finite-state transducers as a formalism similar to finite-state automata, except for the optional output associated to each transition. The approach presented here is based on this formal model.

Bishop [2002], Tanenbaum [2003] present issues concerning modern cryptosystems. It is easily possible to identify the gradual historical increase of complexity imposed to the cryptosystem's algorithm, in order to achieve proper security level.

3. Adaptive Technology

Adaptive Technology resulted from searching for formulations that be both simple and sufficiently strong to represent and handle complex phenomena [Pistori, 2003]. It deals with the application of formalisms and devices that react to external stimuli by dynamically activating self-modification actions that change their behavior [Pedrazzi; Tchemra and Rocha, 2005].

Several traditional formalisms, such as finite-state automata, grammars, Markov chains, Statecharts, decision tables and decision trees have their power and expressiveness significantly increased when used as subjacent devices for this approach [Pedrazzi; Tchemra and Rocha, 2005].

Adaptive technology has already been successfully used in a wide range of applications, e.g. robotic navigation, automatic musical composition, computer vision, pattern recognition, natural language processing and compiler construction [Pistori; Neto and Pereira, 2006].

3.1. Finite-State Transducers

A finite-state transducer is a device similar to the classic finite-state automaton [Lewis and Papadimitriou, 1981]. Apart of its language recognition purpose, a finite-state transducer is also expected to convert its input into an output text [Neto, 1987].

A finite-state transducer may be described [Lewis and Papadimitriou, 1981][Neto, 1987] as a 6-tuple (Q, Σ, Λ, δ, q_0, F), where: Q represents a finite, non-empty set of states; Σ represents its input alphabet; Λ represents its output alphabet; q_0 represents its initial state ($q_0 \in Q$); F represents a finite non-empty set of final states ($F \subseteq Q$); δ represents its transition function ($\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow Q \times \Lambda$).

Finite-state transducers may be described by means of state transition diagrams, that are directed graphs with input and output information associated to their transitions.

An operation cycle of a finite-state transducer consists of: reading an input symbol; using the transition function to update the current state; and writing out some corresponding output symbol if necessary.

In order to represent the execution of such machine, the concept of configuration is defined for finite-state transducers [Neto, 1987] as a 3-tuple (current state, input string yet to be read, output string generated so far). Let $w_i w_{i+1} \ldots w_n$ be the part of the input string yet to be read and $u_0 \ldots u_{i-1}$ the already generated output string. The computation step performed by the transducer by applying some transition rule $\delta(q,w_i) \rightarrow (q',u_i)$ is represented by a move from its current configuration to a next one: $(q, w_i w_{i+1} \ldots w_n, u_0 \ldots u_{i-1}) \vdash (q', w_{i+1} \ldots w_n, u_0 \ldots u_{i-1} u_i)$.

3.1. Adaptive Finite-State Transducers

Adaptive finite-state transducers are self-modifying rule-driven devices based on finite-state transducers. At the start of its operation, an adaptive finite-state transducer T may be viewed as an initial finite-state transducer T_0. Whenever a transition is executed, its behavior may change. So, in operation, an adaptive finite-state transducer T describes a path $(M_0,w_0,u_0) \Rightarrow (M_1,w_1,u_1) \Rightarrow \ldots \Rightarrow (M_n,w_n,u_n)$, where $w_0 w_1 \ldots w_n$ ($n \geq 0$) represent the input string, and u_k ($0 \leq k \leq n$) represent the full output substring generated at each step k.

The basic mechanisms that allow adaptive transducers to perform self-modification are called adaptive functions. Adaptive functions may be parametric, and must be declared apart of the transducer's definition.

A similar formulation, presented in [Neto and Pariente, 2002] for adaptive automata, defines adaptive functions as collections of elementary adaptive actions. There are three different elementary adaptive actions: insertion actions, that joins a specified transition to the set of transitions defining the automaton; elimination actions, which remove a specified transition from that set; and inspection actions, that searches that set for transitions matching some given pattern.

Calls to adaptive functions may be attached to the transducer's transitions. In this case, they are called adaptive actions. Adaptive actions are specified to execute before/ after the execution of the associated transition [Neto and Pariente, 2002].

In order to represent transitions with optionally attached adaptive functions the following notation is used: $(q,w_i)B \rightarrow (q',u_i)A$, where: $q, q' \in Q$, $w_i \in \Sigma$, $u_i \in \Lambda$. B/A optionally specify names of adaptive functions to be executed before/after the transition is applied, respectively.

Being extensions of adaptive finite-state automata, it is easy to show they have the same power of Turing Machines [Rocha and Neto, 2000][Neto and Pariente, 2002].

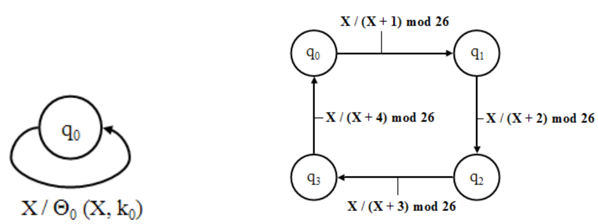
4. Building Data Security Systems with Adaptive Finite-State Transducers

In this section we introduce finite-state transducers as the subjacent abstraction for the implementation of ciphering systems whose desired behavior is being similar to that of a standard cryptosystem: the transducer's underlying finite-state automaton is used as part of the design of ciphering functions. Afterwards, adaptive actions are added to its transitions in order to obtain the proposed ciphering system. The purpose of our ciphering algorithm is to convert some plain text P into a ciphered text C by means of an algorithm using key K. The main idea of such a scheme is to associate different ciphering algorithms/cryptosystems to the states of an adaptive finite-state transducer, so that its output is generated as part of the ciphered text in a specific state by applying the algorithm associated to that state to the input data.

4.1. A Finite-State Transducer Approach

Let us consider a finite-state transducer $T = (Q, \Sigma, \Lambda, \delta, q_0, F)$, where:

- Q: set of n states q_s , $1 \leq s \leq n$ (in the examples below, $q_s \equiv s$ by natural isomorphism);
- Σ: is the input alphabet and it contains all possible values for w_i. Typically, Σ is the set of all elements in $\{0,1\}^p$, $p > 0$. In this case, p is the bit width of the symbols in the input alphabet;
- Λ: is the output alphabet, and it contains all possible output values. Note that Σ and Λ may even be the same, but only $|\Lambda| = |\Sigma|$ is required;
- q_0: is the transducer's initial state ($q_0 \in Q$);
- F: is the set of the transducer's final states (F may even be irrelevant, since typically T is not always intended to operate as an acceptor);
- δ: $Q\times\Sigma \rightarrow Q\times\Lambda$ is the transduction function. Each of its defining transduction rules $(q_s,w_i)\rightarrow(q_j,\lambda_i)$ maps some input symbol $w_i \in \Sigma$ into its corresponding ciphered output $\lambda_I = \Theta_s(w_i, k_s) \in \Lambda$, where Θ_s and k_s are the output parameters associated with the transition origin state q_s, and moves the transducer from its current state q_s to state q_j.



(a) Transducer T_1 (b) Transducer T_2

Figure 1: Examples of Finite-state Transducers. Token X represents any input symbol

The computation performed on an input string $w = w_1 \ldots w_n$ by the finite-state transducer T_1 in Figure 1(a) is $(q_0, w_1...w_n, \varepsilon) \vdash^* (q_0, \varepsilon, \Theta_0(w_1,k_0)... \Theta_0(w_n,k_0))$. Its output is obtained by applying function Θ_0 to each w_i ($1 \leq i \leq n$), with the argument k_0. If Θ_0 is a ciphering function and k_0 is the ciphering key, then T_1 performs a ciphering computation on w_i. A 4-state transducer T_2 is illustrated in Figure 1(b).

In order to decipher the output of T_1 a finite-state transducer T_1^{-1} is needed, where: (i) Q, F, q_0 are the same as in T_1; (ii) The input alphabet of T_1 is the output alphabet of T_1^{-1} and vice-versa; and (iii) for each $(q_s, w_i) \rightarrow (q_j, \lambda_i)$ in T_1 there is some $(q_s, \lambda_i) \rightarrow (q_j, \Theta_s^{-1}(\lambda_i, k_s^{-1}))$ in T_1^{-1}, where $\lambda_i=\Theta_s(w_i, k_s)$ in T_1, k_s^{-1} is the deciphering key associated to the ciphering key k_s and Θ_s^{-1} is the deciphering function for the Θ_s ciphering function.

The following conditions must hold in order to a finite-state transducer be considered a ciphering system: (i) the ciphering finite-state transducer must be deterministic, otherwise it would map an input text into different ciphered texts, turning the deciphering operation also non-deterministic; (ii) finite-state transducers must be fully described, otherwise their operation would stop before input text is fully processed; and (iii) the same function and the same key must apply to all transitions emerging from each state (output parameters are associated to the transition's origin state), otherwise the inverse operation will not be deterministic.

4.2. Adaptive Transducer Approach

Adaptive functions allow automatically changing finite-state transducer's behavior, by modifying transducer's topology and ciphering parameters (key, function, output mapping).

Adaptive functions used in our proposal must be designed in such a way that the transducer remains both fully described and deterministic. That requires restrictions to be imposed to the allowed adaptive actions, e.g., in order to assure that the transducer remains fully described, the removal of each transition must be complemented by the adding another transition emerging from the same state and consuming the same input symbol. Additionally, for efficiency, it is advisable to restrict the use of adaptive actions only to before- or after- ones, not both.

Choosing adequate adaptive actions requires caution and study. An example of a convenient set of simple adaptive functions is: (i) modify the destination state of a transition in order to change transducer's topology. The new destination state may be specified as a parameter for the adaptive function; (ii) modify the adaptive action within a rule by changing the adaptive function invoked by the rule; and (iii) modify the output symbol in some rule. Such a function must consistently replace the output symbol of all rules departing from the affecting state, in order to ensure that different destination states correspond to different outputs. The idea is to modify ciphering parameters. The output symbol may be changed in two ways: (i) for pre-computed values apply some adequate function (e.g. an exclusive-or operation between the original cipher symbol and some parameter); and (ii) for the function/key method, change the function and/or the key used.

4.3. Adaptive Finite-State Automata in Data Security

Preceding sections propose using (adaptive) finite-state transducers in data security. An adaptive version was used to perform ciphering/deciphering. An adaptive finite-state transducers-based ciphering system is an algorithm that uses an adaptive finite-state transducer instead of traditional data keys. The strength of this approach is that it radically disguises the algorithm being executed, and allows it even to change dynamically.

Using a finite-state transducer as a key allows performing symmetric stream ciphering (similar to a stream cipher cryptosystem [Bishop, 2002][Tanenbaum, 2003]) in which part of the operation sequence depends on information encoded within the key. The sequence of keys and/or functions executed depends on the current state of the transducer. Each state determines the function and the key to be used, so such execution path may be denoted by a sequence $((\Theta_{q0}, k_{q0}), (\Theta_{q1}, k_{q1}), \ldots, (\Theta_{qr}, k_{qr}))$, where qr denotes the current state before the execution of the n-th step of computation. Consequently, part of the transduction logic becomes hidden. For illustration purposes, let us consider, for analogy, accepting sentences of a language defined by a regular expression. This problem may be solved by using specific finite-state automata [Lewis and Papadimitriou, 1981]. Algorithms that simulate generic finite-state automata are well-known, but the particular one that accepts a specific language defined by some particular regular expression is encoded in the corresponding transition function. The resulting obscurity in that logic meets the requirements stated by Kerckhoff's principle, provided that the adaptive transducer simulating algorithm be publicly available. It remains hidden the information on what sort of operation is expected to be used at each specific execution step.

However, conventional finite-state transducers may be inferred. Some ciphering key patterns may appear, turning this approach vulnerable to differential attacks. An illustrating example is given in Figure 1(b). Note that the sequence q_0, q_1, q_2, q_3 of states in this automaton corresponds to an execution pattern. Such fragility may be avoided by dynamically modifying the transducer's execution flow. Such feature is the essence of adaptive devices. For example, even topology is not permanent in adaptive finite-state automata.

Adaptive functions allow changing the set of transitions and the associated ciphering parameters. Furthermore, with appropriate adaptive functions it is even possible to change the adaptive actions associated to the transitions.

In brief, this approach has the following features:

- The ciphering key, represented by an adaptive finite-state transducer, holds information on the sequence of computation. The performed ciphering/deciphering operation depends on such sequence.
- The algorithm does not predefine the number of states, so the key size is not constant. Nevertheless, execution time stays constant for each state, so the number of states in the transducer will not affect its ciphering/deciphering speed. However, the larger the number of states, the harder the inference of the correspond transducer (key).
- Transducers are self-modified by the adaptive functions associated to the execution of their transitions.
- The ciphering/deciphering operation also depends on the input text, once different input strings usually have different computation sequences.
- Pre-computed output in the execution algorithm allows attackers to know only the type of adaptive actions used.
- Different instances of a given input block may be ciphered into different outputs. This is a kind of stream cipher algorithm in which the next key depends on both the input text and the topology of the automaton being used as a key.

Nevertheless, one may be aware that: (i) using a transducer as a key often leads to keys that are large compared to the traditional 128- or 256-bit keys used in symmetrical cryptosystems; (ii) execution time depends on the way output symbols are generated: by means of some pre-calculated table or by using function calls. For pre-calculated output, transitions cannot be partitioned, so the number of possible input/output must be limited, otherwise, the transducer becomes too large; (iii) based on the logic of the algorithm and its intrinsic dependence on the input text, both ciphering and deciphering transducers must stay synchronized in order to assure correct operation. With this approach only two-sided communication is feasible; and (iv) With these restrictions only it's possible that two or more different keys execute the same logical operation, i.e. perform the same cipher/decipher operation. In order to this ciphering scheme be properly used as a cryptosystem, some further refinements must be developed to avoid this situation.

Security provided by both hiding information and changing the execution flow allows using straightforward actions, such as the exclusive-or operation, in the output function. We expect that after some refinements, this ciphering scheme will become the basis for a new kind of stream cryptosystem.

5. Experiment and Results: The Adapt_Cipher Algorithm

Adapt_Cipher is an illustrative ciphering algorithm (described in Algorithm 1), derived from the obsolete Ceasar cipher (by using only that cipher as the Θ function), developed around the previously presented ideas. The aim of this ciphering scheme is to exemplify the possibilities of adaptivity for data security applications and to compare the proposed scheme with a well-known classical cipher (the motivation of choosing Ceasar cipher is the ease of understanding). Our main goal is to obtain a fast ciphering system. Although it seems to be susceptible to be broken, it can be significantly hard to predict or infer if the number of states is sufficiently high (e.g. 50 states).

Algorithm 1 Adapt_Cipher

Require: The key (description of an adaptive finite-state transducer)
Execute:

```
    initialize cipher();
    q_current = 0;
    input = readinput();
    while (input != NULL)        {
        q_current = executetransition(q_current, X);    //performs a transducer transition
        output = Θ_qcurrent (input, K_qcurrent);         //generates output symbol (ciphered text)
        executeadaptivefunction(q_current, X);          //executes adaptive action
        input = readinput();        }                   //reads next input symbol
    end;
```

This ciphering algorithm accepts an adaptive finite-state transducer with one transition per state (to reduce the key size, the single token X represents all possible input symbols used in the transition). Each transition is associated to an adaptive function associated (indexed by numbers 0 through 4). Let |Q| be the number of states, and DS the destination state in the transition being executed. The procedure call "executeadaptivefunction" assume one of the following options:

- A_0: Do nothing;
- A_1: Change DS to (DS+1) mod (|Q|);
- A_2: Change the $K_{qcurrent}$ to ($K_{qcurrent}$ + 1) mod 26;
- A_3: Change DS to (DS+1) mod (|Q|) and change the $K_{qcurrent}$ to ($K_{qcurrent}$ + 1) mod 26;
- A_4: Change DS to (DS+2) mod (|Q|).

The ciphering initialization is a function that reads in the key and converts it to an internal representation. Its execution consists of a loop on the input text symbols (the algorithm keeps running until input is exhausted). The loop body includes: reading the next symbol from the input text, performing a transition to the next state of the transducer, generating an output, and executing the required adaptive action.

5.1. Example

This example has been elaborated in order to illustrate some execution details of this method and its differences to the classical Caesar's approach, which is too weak for real uses, since it is vulnerable to two simple attacks: exhaustive testing its 26 valid keys or using frequency analysis over the idiom of the original text.

This example is based on the finite-state transducer in Figure 1(b). The finite-state approach gives the pattern of key (1234). In order to avoid such patterns, adaptive functions have been inserted. The resulting adaptive finite-state automaton is used as a key for the adaptive cipher (described in Table 1). The reduced number of states in this example is not usual, and has been used for easing manual simulation only. Obviously, effective use of this method in actual applications requires a larger number of states to be used for the transducer.

Id	Current State	Input String	Destination State	Output String (K_{state})	Adaptive Function
1	0	X	1	X+1 mod 26 (K_0 = 1)	A_1
2	1	X	2	X+2 mod 26 (K_1 = 2)	A_2
3	2	X	3	X+3 mod 26 (K_2 = 3)	A_0
4	3	X	0	X+4 mod 26 (K_3 = 4)	A_5

Table 1: Adaptive Finite-state Automaton Transitions (visual key for Adapt Cipher)

A step-by-step evolution of the adaptive finite-state transducer on the word "REDISCOVER" is:
(M_0, REDISCOVER, ε) $\vdash_1$ (M_1, EDISCOVER, S) $\vdash_2$ (M_2, DISCOVER, SG)
$\vdash_3$ (M_3, ISCOVER, SGG) $\vdash_4$ (M_4, SCOVER, SGGM) $\vdash_1$ (M_5, COVER, SGGMT)
$\vdash_3$ (M_6, OVER, SGGMTF) $\vdash_4$ (M_7, VER, SGGMTFS) $\vdash_3$ (M_8, VE, SGGMTFSY)
$\vdash_4$ (M_9, R, SGGMTFSYI) $\vdash_1$ (M_{10}, ε, SGGMTFSYIS)
The subscript at each computation step indicates the number of the transition applied.

For comparing results, "REDISCOVER" is now ciphered using the Caesar cipher and the finite-state automaton (without adaptive actions). Table 2 compares the results.

Input	R	E	D	I	S	C	O	V	E	R
Caesar cipher										
Keys	3	3	3	3	3	3	3	3	3	3
Output	U	H	G	L	V	F	R	Y	H	U
Finite-state Automaton in Fig 1 (b)										
Keys	1	2	3	4	1	2	3	4	1	2
Output	S	G	G	M	T	E	R	Z	F	T
Adaptive Finite-state Automaton (table 1) – Adapt_Cipher										
Keys	1	2	3	4	1	3	4	3	4	1
Output	S	G	G	M	T	F	S	Y	I	S

Table 2: Compared results. "Keys" indicate the Caesar cipher key used for each letter.

Although using so few states is not recommended, one can easily observe how even a simple 4-state adaptive finite-state automaton provides an apparently erratic effect in the sequence of "keys". Changing "keys" turns this approach less vulnerable to the previously mentioned attacks to Caesar cipher. Since the transducer changes dynamically, inference becomes a harder task.

6. Discussion about the results: Overview of Strength against Attacks

The security achieved through this method is fundamentally based on hiding information on the ciphering/deciphering process. In addition, adaptivity modifies the transducer every time an adaptive action takes place, so breaking such system seems to be a hard task. The following text comments on possible attacks and their consequences. The security of this algorithm has not been analytically proven yet. Stream cipher cryptosystems are often general design/analysis techniques. Estimating or assuring security for this kind of cipher is more complex than for the block cipher cryptosystem [Schneier, 2000], since the block cipher cryptosystem typically follows some concrete design practice that assures security.

A first possible attack is to build a translation dictionary that maps input text into the corresponding cipher. Using the transducer method, each plain text maps into different ciphered texts, therefore any attack using translation dictionaries results rather ineffective.

Another possibility is the differential attack. It investigates how variations in the input text reflect in the output text and explores high-probabilities in the occurrence of such differences [Heys, 2001]. Since ciphering functions and keys may change from state to state, since the state path depends on the input text and since adaptive functions may change transducer's parameters, the possibility of simultaneously occurring matching cipher differences in correspondence to identical input text differences seems to be very low. Additionally, it is possible to use cipher functions already proven to be secure against this kind of attack.

Perhaps the best way to break this ciphering scheme should be to determine its correct key. For doing that one needs to discover the number of states, the set of state transitions and the adaptive functions and corresponding arguments attached to each transition. Finding such amount of information through brute force is unfeasible, once there exist too many keys (theoretically, infinite keys exist since the number of states is unknown and not limited). A more sophisticated technique can combine key inference and differential attack. Our illustrating example (adaptive ciphering with a 4-state key) is rather easy to break by using a sufficiently large number of input texts together with the respective ciphered texts, and assuming that only Ceasar cipher has been used. However, by increasing the number of states, key inference tends to become unfeasible. Certainly more sophisticated attacks will appear with the diffusion of this approach.

Public knowledge of the algorithm provides information that helps finding ways to determine the key but this information alone seems to be insufficient to endanger the key's confidentiality: (i) the key is encoded as an adaptive finite-state transducer; and (ii) the possible adaptive functions to be executed and output functions (for non-pre-computed output) are given.

7. Conclusions

This work presented some essays toward a novel ciphering method. In our proposal, security is obtained for a ciphering system, either: by masking part of the algorithm's logic; by moving some information into the key; by performing adaptive execution.

The security obtained with both transducers and adaptivity allows using simple arithmetic ciphering operations, or even pre-computed values, reducing the ciphering process to simple replacements of input text blocks with corresponding ciphered text blocks obtained by performing the related transduction.

By reducing the complexity of such operations, the ciphering process may turn faster than with traditional cryptosystem algorithms. In this case, execution time depends on three components: (i) the time needed to perform a state transition; (ii) the time needed to obtain an output value; and (iii) the time needed to perform the changes specified by adaptive actions. With fast output algorithms it is possible to achieve good performance in applications that require the encryption of large amounts of data (e.g. ciphering digital video).

As a practical result of this research we expect to apply these ideas in near future to the design of a secure and efficient cryptosystem, with a unique key (the approach developed so far does not guarantee that two different keys do not generate the same output).

References

[Bishop, 2002] Bishop, M. (2002). Computer security art and science. 1st edition. Addison-Wesley Professional.

[Daemen and Rijmen, 1999] Daemen, J. and Rijmen, V. (1999). AES proposal: Rijndael. http://csrc.nist.gov/Crypto Toolkit/aes/rijndeal/

[Heys, 2001] Heys, H. H. (2001). A tutorial on linear and differential cryptanalysis. Centre for Applied Cryptographic Research, Departament of Combinatorics and Optimization, University of Waterloo.

[Lewis and Papadimitriou, 1981] Lewis, H. R. and Papadimitriou, C. H. (1981). Elements of the theory of computation. Prentice-Hall, Inc.

[Neto, 1987] Neto, J. J. (1987). Introdução à compilação. LTC - Livros Técnicos e Científicos Editoras S.A.

[Neto and Pariente, 2002] Neto, J. J. and Pariente, C. A. B. (2002). Adaptive automata - a revisited proposal. Implementation and Application of Automata 7th International Conference - CIAA 2002. Tours, France.

[Pedrazzi; Tchemra and Rocha, 2005] Pedrazzi, T.; Tchemra, A. H. and Rocha, R. L. A. (2005). Adaptive decision tables - a case study of their application to decision-taking problems. Proceedings of International Conference on Adaptive and Natural Computing Algorithms - ICANNGA 2005. Coimbra, Portugal.

[Pistori, 2003] Pistori, H. (2003). Tecnologia adaptativa em engenharia de computação: Estado da arte e aplicações. Universidade de São Paulo (USP), São Paulo.

[Pistori; Neto and Pereira, 2006] Pistori, H.; Neto, J. J. and Pereira, M. C. (2006). Adaptive non-deterministic decision trees: General formulation and case study. INFOCOMP Journal of Computer Science. Lavras, Brasil.

[Rivest; Shamir and Adleman, 1978] Rivest, R. L.; Shamir, A. and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21(1):120-126.

[Rocha and Neto, 2000] Rocha, R. L. A. and Neto, J. J. (2000). Autômato adaptativo, limites e complexidade em comparação com máquina de turing. Second Congress of Logic Applied to Technology - LAPTEC'2000. São Paulo, Brasil.

[Schneier, 2000] Schneier, B. (2000). A self-study course in block-cipher cryptanalysis. Cryptologia, 24(1): 18-33.

[Tanenbaum, 2003] Tanenbaum, A. S. (2003). Computer networks. 4th edition. Pearson Education, Inc. Prentice Hall PTR.