Efficient FPGA Implementations and Cryptanalysis of Automata-based Dynamic Convolutional Cryptosystems

Dragoş Trincă
Department of Computer Science and Engineering
University of Connecticut
Storrs, CT 06269, USA
dtrinca@engr.uconn.edu

Abstract

With the exception of the recently proposed class of cascaded dynamic convolutional cryptosystems, all the symmetric cryptosystems studied so far in the literature are static, in the sense that their structure do not change at all during encryption/decryption. In this paper, we propose and analyze a new class of dynamic symmetric cryptosystems, called automata-based dynamic convolutional cryptosystems (ADCCs). The paper is organized as follows. First, we provide the reader with a brief introduction to convolutional codes. Second, we give the definition of an ADCC, and then show how to use such a cryptosystem for encryption/decryption. Third, we provide a thorough security analysis of ADCCs, and then discuss their practical advantages. The conclusion of our cryptanalysis is that an ADCC is very hard to break completely, but quite easy to break partially. Fourth, an extension of ADCCs, called nonlinear cascaded ADCCs, is proposed and shown to be much more secure in practice than ADCCs. Finally, an efficient FPGA implementation of nonlinear cascaded ADCCs is presented.

Keywords: Automata, Convolutional Codes, Cryptanalysis, Cryptography, FPGAs

1 Introduction

Symmetric cryptosystems such as DES [2], triple DES [12], AES [1], and many others, have been all designed as *static* ciphers, in the sense that their structure do not change at all during encryption/decryption. Even if the current encryption standard, which is AES (Advanced Encryption Standard), is considered very secure, the power of such a static cipher is quite limited, since finding its only structure breaks the cipher completely. This means that as long as we find the structure of the static cipher that was used to encrypt a certain plaintext, we can decrypt any cryptotext that was obtained using that cipher. Such a weakness has been advantageously explored by the cryptanalytic attacks proposed so far (linear cryptanalysis [8], differential cryptanalysis [3], linear-differential cryptanalysis [5], and many others). To the best of our knowledge, the only dynamic symmetric cryptosystems proposed so far in the literature are the cascaded dynamic convolutional cryptosystems [11], whose structure is based on globally invertible convolutional transducers.

In [11], we have studied the time performance of cascaded dynamic convolutional cryptosystems, and then designed a parallel variant using the well-known PRAM model of computation [7]. However, due to its linear structure, a cascaded dynamic convolutional cryptosystem can be broken completely quite easily. From this reason, we have concluded that their structure should be significantly changed by adding several nonlinear components and memory in order to obtain a much more secure cipher. In this paper, we propose and thoroughly analyze a new class of dynamic symmetric cryptosystems, called automata-based dynamic convolutional cryptosystems (abbreviated ADCCs). It will be shown that, at least from some point of view, an ADCC is much

more secure in practice than a cascaded dynamic convolutional cryptosystem, mainly due to the addition of several new parameters. However, we will also show that even if an ADCC is very hard to break completely, it is quite easy to break partially. Such a drawback can be easily eliminated by introducing an extension of ADCCs, called nonlinear cascaded ADCCs, which will be shown to be much more secure in practice than ADCCs. More precisely, a nonlinear cascaded ADCC will be shown to be not only hard to break completely, but also hard to break partially.

The paper is organized as follows. In Section 2, we provide the reader with a concise introduction to convolutional codes, convolutional transducers, and convolutional cryptosystems. The ADCCs are introduced in Section 3, where we show concretely how to use an ADCC for encryption/decryption. We also show that an ADCC is more secure in practice than a cascaded dynamic convolutional cryptosystem. In Section 4, we provide a thorough security analysis of ADCCs. Precisely, we develop and analyze several cryptanalytic attacks based on Gaussian elimination. We take concrete examples and show exactly how such attacks work in practice. The conclusion of our cryptanalysis is that an ADCC is very hard to break completely, but quite easy to break partially. This drawback is eliminated in Section 5, where we propose and analyze an extension of ADCCs, called nonlinear cascaded ADCCs. Roughly speaking, a nonlinear cascaded ADCC is a cascade of ADCCs with substitutions and permutations between every two consecutive ADCCs. The conclusion of our security analysis is that a nonlinear cascaded ADCC is not only hard to break completely, but also hard to break partially. Finally, an efficient implementation of nonlinear cascaded ADCCs in FPGAs with embedded memory [14] is presented.

2 Convolutional codes: a short survey

Convolutional codes [4, 9, 10] are a well-known class of error-correcting codes, currently used in practice worldwide to encode digital data before transmission over noisy channels. During encoding, k input bits are mapped to n output bits to give a rate k/n coded bitstream. At the receiver, the bitstream can be decoded to recover the original data, correcting errors in the process. The optimum decoding method is *maximum-likelihood decoding*, where the decoder attempts to find the closest "valid" sequence to the received bitstream. The most popular algorithm for maximum-likelihood decoding is the *Viterbi algorithm* [13].

Even though convolutional codes have been primarily designed for error detection and correction, they can be successfully used in related areas such as cryptography, as we will see throughout this presentation. (To the best of our knowledge, the only previous work in this direction is [11].) The aim of this section is to provide the reader with a brief introduction to convolutional codes.

As stated in [4], a key step in understanding convolutional codes is to distinguish between the convolutional encoder, the convolutional encoding operation, and the convolutional code. Rigorous definitions of all these concepts are provided below.

We denote by $\mathcal{B}_{i\times j}$ the set of $i \times j$ arrays with binary components. If $u \in \mathcal{B}_{i\times j}$, then the number of components of u is denoted by $|u|$, i.e., $|u| = ij$. Also, we denote by $u[q, -]$ the q-th row of u, and by $u[-, q]$ the q-th column of u. If u is a row vector (or a column vector), then we will denote by $u[i]$ the i-th element of u, and by $u_{i:j}$ the subvector $[\, u[i] \ldots u[j] \,]$. If $u_1, \ldots, u_h$ are vectors, then we denote by $vect(u_1, \ldots, u_h)$ the vector consisting of the components of $u_1, \ldots, u_h$, in the same order. For example,

$$vect([\,0\ 0\,], [\,1\ 1\,]) = [\,0\ 0\ 1\ 1\,].$$

Definition 1 *Let n, k, and m be nonzero natural numbers. An (n, k, m)* convolutional transducer *is a function $t : \cup_{i=1}^{\infty} \mathcal{B}_{1\times ki} \mapsto \cup_{i=1}^{\infty} \mathcal{B}_{1\times ni}$ given by*

$$t(u) = u\boldsymbol{G}_{t,|u|}, \tag{1}$$

where

$$\boldsymbol{G}_{t,kp} = \begin{bmatrix} G_{t,0} & G_{t,1} & \dots & G_{t,m} & & & \\ & G_{t,0} & G_{t,1} & \dots & G_{t,m} & & \\ & & \ddots & & & \ddots & \\ & & & G_{t,0} & G_{t,1} & \dots & G_{t,m} \end{bmatrix}$$

is an element of $\mathcal{B}_{kp\times(pn+mn)}$, $G_{t,i} \in \mathcal{B}_{k\times n}$ *for all* $i \in \{0, 1, \dots, m\}$, *and the arithmetic in (1) is carried out over the binary field* $GF(2)$. *The entries left blank are assumed to be filled in with zeros. An* (n, k, m) *convolutional transducer is usually called a rate* k/n *convolutional transducer.*

Definition 2 *Let* $t : \cup_{i=1}^{\infty}\mathcal{B}_{1\times ki} \mapsto \cup_{i=1}^{\infty}\mathcal{B}_{1\times ni}$ *be an* (n, k, m) *convolutional transducer. The* (n, k, m) convolutional code *induced by* t *is the image* $t(\cup_{i=1}^{\infty}\mathcal{B}_{1\times ki})$ *of* t.

Definition 3 *Let* $t : \cup_{i=1}^{\infty}\mathcal{B}_{1\times ki} \mapsto \cup_{i=1}^{\infty}\mathcal{B}_{1\times ni}$ *be an* (n, k, m) *convolutional transducer. An* (n, k, m) convolutional encoder *is a realization by linear sequential circuits of the semi-infinite generator matrix*

$$\boldsymbol{G}_t = \begin{bmatrix} G_{t,0} & G_{t,1} & \dots & G_{t,m} & & \\ & G_{t,0} & G_{t,1} & \dots & G_{t,m} & \\ & & \ddots & \ddots & & \ddots \end{bmatrix}$$

associated with t.

For a more detailed introduction to convolutional codes, we refer the reader to [4]. Let us take an example.

Example 1 *Let* $t : \cup_{i=1}^{\infty}\mathcal{B}_{1\times i} \mapsto \cup_{i=1}^{\infty}\mathcal{B}_{1\times 2i}$ *be a* $(2, 1, 2)$ *convolutional transducer with*

$$G_{t,0} = [\,0\ 1\,], G_{t,1} = [\,1\ 0\,], G_{t,2} = [\,1\ 1\,].$$

Thus, at each step, the number of input bits is $k = 1$, *the number of output bits is* $n = 2$, *and the number of memory registers is* $km = 2$. *The associated convolutional encoder can be represented graphically as in Figure 1. Note that the two output bits at each step are serialized using a multiplexer.*

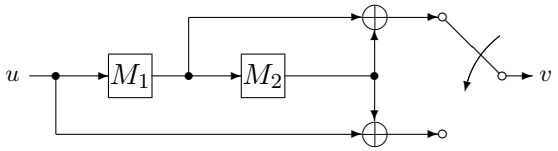

Figure 1: a $(2, 1, 2)$ convolutional encoder

Let us now describe the encoding mechanism. Let b *be the current input bit being encoded, and let* b_1 *and* b_2 *be the current bits stored in the memory registers* M_1 *and* M_2, *respectively. The first output bit is*

$$bG_{t,0}[1] \oplus b_1G_{t,1}[1] \oplus b_2G_{t,2}[1] = b_1 \oplus b_2,$$

whereas the second output bit is

$$bG_{t,0}[2] \oplus b_1G_{t,1}[2] \oplus b_2G_{t,2}[2] = b \oplus b_2.$$

After both output bits are obtained, b_1 *is shifted into the memory register* M_2, *and* b *is shifted into the memory register* M_1. *Let us assume that the input vector* u *has length* kp. *The actual input vector is* u *followed by* km *zeros. Thus, the total length of the output vector is* $pn + mn$. *For*

example, let us take $u = [\ 0\ 1\ 0\ 1\ ]$ *as the input vector. Then, one can verify that the output vector is* $t(u) = u\boldsymbol{G}_{t,4} = [\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ ]$*, where*

$$\boldsymbol{G}_{t,4} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & & & & & & \\ & & 0 & 1 & 1 & 0 & 1 & 1 & & & & \\ & & & & 0 & 1 & 1 & 0 & 1 & 1 & & \\ & & & & & & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

As usual, the missing entries in $\boldsymbol{G}_{t,4}$ *are assumed to be zeros. Note that the size of the output vector* $t(u)$ *is* $pn + mn = 12$*.*

Throughout the paper, we should be interested only in *globally invertible* (k, k, m) convolutional transducers, i.e., convolutional transducers with the property that each output block of k bits can be uniquely decrypted into the corresponding block of k input bits. Let us explain how we encrypt an input vector using a globally invertible (k, k, m) convolutional transducer.

Example 2 *Let* t_1 *and* t_2 *be* $(2, 2, 2)$ *convolutional transducers with*

$$G_{t_1,0} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, G_{t_1,1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, G_{t_1,2} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix},$$

and

$$G_{t_2,0} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, G_{t_2,1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, G_{t_2,2} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

The corresponding convolutional encoders are represented graphically in Figure 2 (a) and (b), respectively. It is easy to see that only t_1 *is globally invertible. For example, assume that* $M_1^1 = 0, M_2^1 = 1, M_1^2 = 0, M_2^2 = 1$*, and the two output bits being decrypted are* 0 *and* 1*, respectively. Given that the first output bit depends on* M_2^1 *and the first input bit, we conclude that the first input bit was a* 1*. Then, given that we already know the first input bit, we find that the second input bit was a* 0*. Thus, we conclude that* t_1 *is globally invertible, since at each step we can decode uniquely the current block of* k *output bits. The convolutional transducer* t_2 *is not globally invertible, since each of the two output bits depends on both input bits. Therefore, the current block of* k *output bits cannot be uniquely decoded into the corresponding input block.*

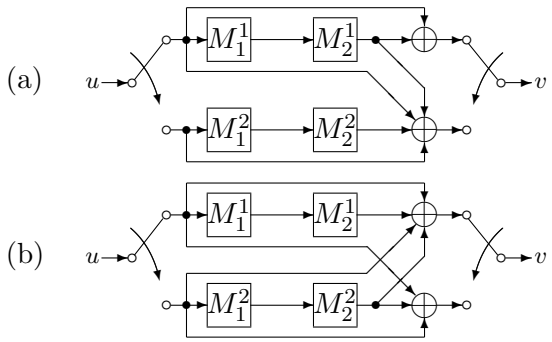

Figure 2: (a) a globally invertible $(2, 2, 2)$ convolutional encoder, and (b) a $(2, 2, 2)$ convolutional encoder that is not globally invertible

Let $u = [\ 0\ 1\ 1\ 0\ ]$ *be an input vector. More precisely, we have* $p = 2$ *blocks of size* $k = 2$ *each. One can verify that*

$$t_1(u) = u\boldsymbol{G}_{t_1,kp} = [\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ ].$$

If the matrices $G_{t_1,0}, G_{t_1,1}, G_{t_1,2}$ are kept private, then we can encrypt u by $v = [\,0\ 1\ 1\ 1\,]$, i.e., the first $kp = 4$ bits of $t_1(u)$. Given that t_1 is globally invertible, we can uniquely decrypt v into u as long as we know the matrices $G_{t_1,0}, G_{t_1,1}, G_{t_1,2}$

Definition 4 *A* (k, k, m) convolutional cryptosystem *is a globally invertible (k, k, m) convolutional transducer in which the matrices $G_{t,0}, G_{t,1}, G_{t,2}$ are considered private keys.*

Remark 1 *Since we always need for encryption only the first $|u|$ bits of $t(u)$, we can change the standard definition of an (n, k, m) convolutional transducer by replacing equation (1) with*

$$t(u) = u\boldsymbol{H}_{t,|u|}, \tag{2}$$

where $\boldsymbol{H}_{t,|u|}$ is the restriction of $\boldsymbol{G}_{t,|u|}$ to the first $|u|$ columns. Thus, if t is a (k, k, m) convolutional cryptosystem, then we always encrypt u by $t(u)$.

3 Automata-based dynamic convolutional cryptosystems

Most of the symmetric cryptosystems that have been studied so far in the literature are *static*, i.e., their structure does not change during the encryption and decryption procedures, but in this paper we shall be interested only in *dynamic* symmetric cryptosystems, whose structures *do* change during the encryption and decryption procedures. To the best of our knowledge, the only *dynamic* symmetric cryptosystem proposed so far is the one introduced in [11]. The idea in [11] is based on linear N-cascaded dynamic convolutional transducers, and is comprised in the following definition.

Definition 5 *Let k and m be nonzero natural numbers. A* (k, k, m) linear N-cascaded dynamic convolutional transducer with propagation *is an $(N+1)$-tuple $(t, \mathcal{S}_1, \ldots, \mathcal{S}_N)$, where*

$$t(u) = u\boldsymbol{H}^1_{t,kp}(x_0)\boldsymbol{H}^2_{t,kp}(x_1)\ldots\boldsymbol{H}^N_{t,kp}(x_{N-1}) \tag{3}$$

for all $u \in \mathcal{B}_{1\times kp}$, $x_0 = u$, $x_i = x_{i-1}H^i_{t,kp}(x_{i-1})$ for all $i \in \{1, \ldots, N-1\}$, $\boldsymbol{H}^i_{t,kp}(w)$ is the restriction of

$$\boldsymbol{G}^i_{t,kp}(w) = \begin{bmatrix} G^{i,0}_{t,0,z} & \ldots & G^{i,m}_{t,m,z} & & \\ & \ddots & & \ddots & \\ & & G^{i,p-1}_{t,0,z} & \ldots & G^{i,m+p-1}_{t,m,z} \end{bmatrix}$$

to the first kp columns, $z = vect(w, [\underbrace{0\ldots0}_{mk}])$, $G^{i,0}_{t,j,z} = G^i_{t,j}(0)$, $G^{i,r}_{t,j,z} = G^i_{t,j}(f(r-1, z))$ for all $r \in \{1, 2, \ldots, m+p-1\}$,

$$f(s, z) = (z_{sk+1:(s+1)k}[1] + \ldots + z_{sk+1:(s+1)k}[k]) \bmod 2,$$

and

$$\mathcal{S}_i = (G^i_{t,0}(0), G^i_{t,1}(0), \ldots, G^i_{t,m}(0), G^i_{t,0}(1), G^i_{t,1}(1), \ldots, G^i_{t,m}(1))$$

is the tuple of state matrices *corresponding to the i-th transducer of the cascade. As usual, the vector-matrix products in (3) are performed over the binary field $GF(2)$.*

A *nonlinear* cascaded dynamic convolutional transducer can be obtained from its linear variant by inserting substitutions and permutations between every two consecutive transducers of the cascade, in the same manner as it will be shown in Section 5.

Remark 2 *It can be seen that equation (3) can be written equivalently as $t(u) = x_{N-1}\boldsymbol{H}^N_{t,kp}(x_{N-1})$.*

Definition 6 *A (k,k,m) linear N-cascaded dynamic convolutional cryptosystem with propagation is a globally invertible linear (k,k,m) N-cascaded dynamic convolutional transducer in which the tuples $\mathcal{S}_1,\ldots,\mathcal{S}_N$ are kept private.*

Remark 3 *A complete example of a linear $(2,2,2)$ 2-cascaded dynamic convolutional cryptosystem with propagation is given in [11]. Shortly, such a cryptosystem consists of a cascade of N dynamic convolutional cryptosystems, and each of those dynamic convolutional cryptosystems has associated two states (i.e., structures). The current state of each of the N cryptosystems depends on whether the previous encrypted block has an odd or an even number of ones.*

The aim of this section is to propose a new class of dynamic convolutional cryptosystems, called *automata-based dynamic convolutional cryptosystems*, and then to explain how to use them for encryption and decryption. Comparisons between the new class of cryptosystems and the one given in Definition 5 will be provided as well.

Definition 7 *Let k,m be nonzero natural numbers. A* (k,k,m) automata-based dynamic convolutional transducer *(abbreviated ADCT)* with q states *is a $(q+2)$-tuple $(t,f,\mathcal{S}_1,\ldots,\mathcal{S}_q)$, where*

$$t(u)=u\boldsymbol{H}_{t,kp}(u) \tag{4}$$

for all $u\in\mathcal{B}_{1\times kp}$, $f:\{1,2,\ldots,q\}\times\mathcal{B}_{1\times k}\to\{1,2,\ldots,q\}$ is the transition function, *$\boldsymbol{H}_{t,kp}(u)$ is the restriction of*

$$\boldsymbol{G}_{t,kp}(u)=\begin{bmatrix} G^0_{t,0,z} & \cdots & G^m_{t,m,z} & & \\ & G^1_{t,0,z} & \cdots & G^{m+1}_{t,m,z} & \\ & & \ddots & & \ddots \\ & & & G^{p-1}_{t,0,z} & \cdots & G^{m+p-1}_{t,m,z} \end{bmatrix}$$

to the first kp columns, $z=vect(u,[\underbrace{0\ldots0}_{mk}])$, $G^0_{t,0,z}=G^1_{t,0}$, $G^i_{t,j,z}=G^{g(i,z)}_{t,j}$,

$$g(i,z)=\begin{cases} f(1,z_{1:k}) & \text{if } i=1,\\ f(g(i-1,z),z_{(i-1)k+1:ik}) & \text{if } i\in\{2,\ldots,m+p-1\},\end{cases}$$

and

$$\mathcal{S}_i=(G^i_{t,0},G^i_{t,1},\ldots,G^i_{t,m})$$

is the tuple of matrices corresponding to the i-th state.

Definition 8 *A* (k,k,m) automata-based dynamic convolutional cryptosystem *(abbreviated ADCC)* with q states *is a globally invertible (k,k,m) ADCT with q states in which the transition function f and the tuples $\mathcal{S}_1,\ldots,\mathcal{S}_q$ are kept private.*

Example 3 *Let $(t,f,\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3)$ be a $(2,2,2)$ automata-based dynamic convolutional cryptosystem with three states, where*

$$G^1_{t,0}=\begin{bmatrix}1&1\\0&1\end{bmatrix},G^1_{t,1}=\begin{bmatrix}0&0\\0&0\end{bmatrix},G^1_{t,2}=\begin{bmatrix}1&1\\0&1\end{bmatrix},$$

$$G^2_{t,0}=\begin{bmatrix}1&1\\0&1\end{bmatrix},G^2_{t,1}=\begin{bmatrix}0&0\\0&0\end{bmatrix},G^2_{t,2}=\begin{bmatrix}1&1\\1&1\end{bmatrix},$$

$$G^3_{t,0}=\begin{bmatrix}1&1\\0&1\end{bmatrix},G^3_{t,1}=\begin{bmatrix}0&0\\0&0\end{bmatrix},G^3_{t,2}=\begin{bmatrix}1&0\\1&1\end{bmatrix},$$

and the transition function f is given in Table 1. Also, let $u=[\,0\ 1\ 1\ 1\ 0\ 1\,]$ be an input vector. Thus, $p=3$ in this case.

Table 1: $f(\{1,2,3\}, \mathcal{B}_{1\times 2})$

f	[0 0]	[0 1]	[1 0]	[1 1]
1	2	3	3	1
2	3	1	2	3
3	3	2	1	2

Having the parameter t in the set of public keys means that only the structure of the matrix $\boldsymbol{H}_{t,kp}(u)$ is known, and not its actual bits. Also, by examining the matrices $G^1_{t,0}, G^2_{t,0}, G^3_{t,0}$, one can remark that t is globally invertible no matter its current state is, so t can indeed be used as a symmetric cryptosystem. Having that said, it can be easily checked that

$$t(u) = u\boldsymbol{H}_{t,6}(u) = u \begin{bmatrix} G^0_{t,0,z} & G^1_{t,1,z} & G^2_{t,2,z} \\ & G^1_{t,0,z} & G^2_{t,1,z} \\ & & G^2_{t,0,z} \end{bmatrix} = [\,0\ 1\ 1\ 0\ 1\ 0\,],$$

since the encoder reaches the third state after encoding the first block [0 1], then reaches the second state after encoding the second block [1 1], and finally reaches again the first state after encoding the third block [0 1]. The entire process is illustrated in Figure 3.

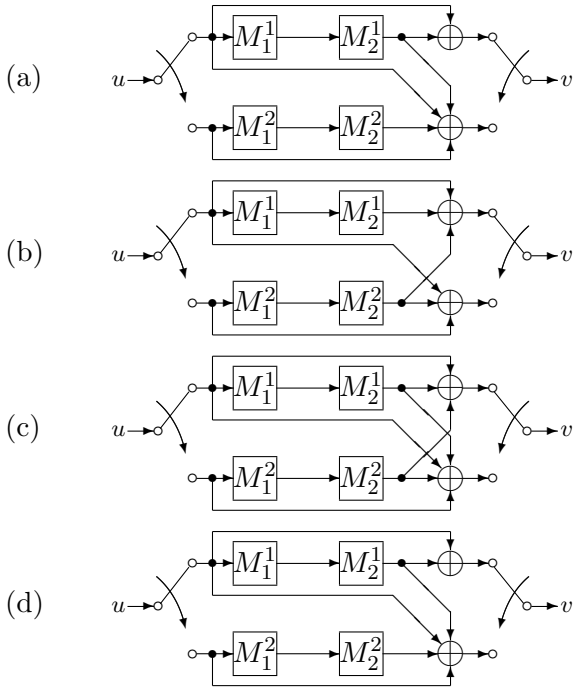

Figure 3: (a) the encoder is in the first state before encoding the first block, then (b) reaches the third state after encoding the first block, then (c) reaches the second state after encoding the second block, and finally (d) goes back to the first state after encoding the third block

Remark 4 (Practical advantages) *The reader now probably wants to know what practical advantages this new class of cryptosystems brings. Suppose that we would like to apply a cryptanalytic attack that finds all the states of a linear N-cascaded dynamic convolutional cryptosystem. Such an attack would break the system completely. This is simply not the case for the new class of cryptosystems, due to the addition of the private parameter f. An attack aimed at finding all the states of an automata-based dynamic convolutional cryptosystem (such as Attack-I from the next*

section) might reveal some values of the function f, but in practice most of them would remain unknown. The answer would become obvious in the next section, when we discuss security issues.

Remark 5 *Since the transition function of an ADCC requires $q2^k$ memory locations, we should define it in such a way that the amount of memory required is significantly reduced. For a $(256, 256, 32)$ ADCC with ten states, the transition function requires $10 \cdot 2^{256}$ memory locations, which is unacceptable. One possibility would be to transform f into a periodic function. For a (k, k, m) ADCC, this can be done by defining f by*

$$f(i, u_1) = f(i, u_2) \quad \text{if and only if} \quad int(u_1) \bmod P = int(u_2) \bmod P,$$

where

$$int(u) = 2^{k-1}u[1] + \ldots + 2^0 u[k]$$

and $P \in \{2^0, 2^1, \ldots, 2^k\}$. It can be easily seen that the period is P. Thus, the number of memory locations required by the transition function is reduced to qP.

4 Gaussian elimination attacks on automata-based dynamic convolutional cryptosystems

An automata-based dynamic convolutional cryptosystem is considered broken as soon as the attacker finds

- the bits of the matrices $G^1_{t,0}, \ldots, G^1_{t,m}, \ldots, G^q_{t,0}, \ldots, G^q_{t,m}$,
- and the values of the function f, i.e. $f(i, u)$ for all (i, u).

Breaking the cipher completely is a *very* difficult task, as we will see in Section 4.2. However, finding partial information about the cipher is quite easy, and the main idea of such an attack, described in detail in Section 4.1, is based on Gaussian elimination.

4.1 The main idea

Let $(t, f, \mathcal{S}_1, \ldots, \mathcal{S}_{10})$ be a $(256, 256, 32)$ ADCC with ten states. For an input u of length 256000, the matrix $\boldsymbol{H}_{t,256000}(u)$ has 256000 rows and 256000 columns. Even from the beginning, it is worth mentioning that the complexity of the Gaussian elimination increases with m, since m determines how many variables we have on each column of the matrix $\boldsymbol{H}_{t,256000}(u)$. Except for the first $nm = 8192$ columns, all the columns have $256(m + 1) = 8448$ variables. Regarding the first 8192 columns, we note that

- the columns $1, 2, \ldots, 256$ have 256 variables each,
- the columns $257, 258, \ldots, 512$ have 512 variables each,
- $\ldots$
- the columns $7937, 7938, \ldots, 8192$ have 8192 variables each.

If we denote the output by v, then the equation $u\boldsymbol{H}_{t,256000}(u) = v$ can be re-written as

$$(S) \begin{cases} u_{1:256}G^0_{t,0,z} = v_{1:256}, \\ u_{1:256}G^1_{t,1,z} \oplus u_{257:512}G^1_{t,0,z} = v_{257:512}, \\ \ldots \\ u_{247553:247808}G^{999}_{t,32,z} \oplus \ldots \oplus u_{255744:256000}G^{999}_{t,0,z} = v_{255744:256000}. \end{cases}$$

Definitely, we can simplify the notation by denoting the i-th 256-bit block of u by u_i, and the i-th 256-bit block of v by v_i. In this case, the system of equations is

$$(S') \begin{cases} u_1 G^0_{t,0,z} = v_1, \\ u_1 G^1_{t,1,z} \oplus u_2 G^1_{t,0,z} = v_2, \\ \dots \\ u_{968} G^{999}_{t,32,z} \oplus \dots \oplus u_{1000} G^{999}_{t,0,z} = v_{1000}. \end{cases}$$

Let $(w_1, y_1), \dots, (w_{256}, y_{256})$ be *known* plaintext-cryptotext pairs satisfying the first equation of (S'). In order to learn the first column of $G^0_{t,0,z}$, i.e. $G^0_{t,0,z}[-, 1]$, we consider the equation

$$AG^0_{t,0,z}[-, 1] = b, \tag{5}$$

where A is a 256×256 matrix and b is a 256-bit column vector. More precisely, $A[i, -] = w_i$ and $b[i] = y_i[1]$. The complexity of the Gaussian elimination in this case is $256^3 = 2^{24}$.

Let us now consider the second equation of (S'), which can be written equivalently as $u_{1:512}X = v_2$, where

$$X = \begin{bmatrix} G^1_{t,1,z} \\ G^1_{t,0,z} \end{bmatrix}$$

is a 512×256 matrix. In order to learn the first column of X, assume that we have 512 *known* plaintext-cryptotext pairs $(w'_1, y'_1), \dots, (w'_{512}, y'_{512})$ satisfying the equation $u_{1:512}X = v_2$, and consider the equation

$$BX[-, 1] = c, \tag{6}$$

where B is a 512×512 matrix and c is a 512-bit column vector, with $B[i, -] = w'_i$ and $c[i] = y'_i[1]$. Clearly, the complexity of the Gaussian elimination in this case is $512^3 = 2^{27}$. So, we have just found that

- the first 256 columns of $\boldsymbol{H}_{t,256000}(u)$ can be learned with 2^{24} work each, and
- each of the columns $\boldsymbol{H}_{t,256000}(u)[-, 257], \dots, \boldsymbol{H}_{t,256000}(u)[-, 512]$ can be learned with $2^{24}2^3$ work.

Similarly, it can be easily seen that

- each of the columns $\boldsymbol{H}_{t,256000}(u)[-, 513], \dots, \boldsymbol{H}_{t,256000}(u)[-, 768]$ can be learned with $2^{24}3^3$ work,
- $\dots$
- each of the columns $\boldsymbol{H}_{t,256000}(u)[-, 7937], \dots, \boldsymbol{H}_{t,256000}(u)[-, 8192]$ can be learned with $2^{24}32^3$ work, and
- beginning with the 8193rd column, the amount of work involved is $2^{24}33^3 \approx 2^{39}$.

So, given that every column of $\boldsymbol{H}_{t,256000}(u)$ can be learned with at most 2^{39} work, we conclude that a $(256, 256, 32)$ automata-based dynamic convolutional cryptosystem is quite insecure, since finding a column of $\boldsymbol{H}_{t,256000}(u)$ is enough for breaking the cipher partially. The entire matrix $\boldsymbol{H}_{t,256000}(u)$ can be broken with $256000 \cdot 2^{39} \approx 2^{57}$ work. However, the matrix $\boldsymbol{H}_{t,256000}(u)$ reveals only partial information about the cipher, since most of the values of the function f (and possibly some of the states) remain unknown. Increasing the parameter m does not necessarily result in a very secure cryptosystem. For example, in the case of a $(256, 256, 512)$ cryptosystem and an input u of length 256000, every column of $\boldsymbol{H}_{t,256000}(u)$ can be learned with at most $2^{24}513^3 \approx 2^{51}$ work. The entire matrix $\boldsymbol{H}_{t,256000}(u)$ can be learned with $256000 \cdot 2^{51} \approx 2^{69}$ work (but again, it reveals only partial information about the cipher). On the other hand, increasing the parameter m results in a slower cryptosystem, since the amount of work involved grows significantly.

4.2 Mounting the attack

We have seen in Section 4.1 that a $(256, 256, 32)$ automata-based dynamic convolutional cryptosystem can be partially broken quite easily. It can happen that the matrix $\boldsymbol{H}_{t,256000}(u)$ does not contain all the states of the cryptosystem. In this case, we may want to consider another plaintext-cryptotext pair, say (u_1, v_1), and then find the matrix $\boldsymbol{H}_{t,256p}(u_1)$, where u_1 and v_1 are of length $256p$. It is essential that $p > m$ so that the matrix $\boldsymbol{H}_{t,256p}(u_1)$ has at least a complete state. The matrix $\boldsymbol{H}_{t,256p}(u_1)$ may reveal new states of the encoder (i.e. states that have not been revealed by $\boldsymbol{H}_{t,256000}(u)$) or not. If the matrix $\boldsymbol{H}_{t,256p}(u_1)$ does reveal all the remaining states of the encoder, we stop. Otherwise, we pick another plaintext-cryptotext pair, say (u_2, v_2), and continue the same analysis as before. In other words, our goal is to break the cipher partially by finding all the bits of the matrices $G^1_{t,0}, \ldots, G^1_{t,m}, \ldots, G^q_{t,0}, \ldots, G^q_{t,m}$ (or equivalently, by finding all the states of the cryptosystem). The attack that does just that, called Attack-I, it given in Figure 4.

Input: k, m, and q
Output: the set of states $A = \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$

1: Let $j > m$.
2: $A \leftarrow \emptyset$.
3: Let (u, v) be a randomly chosen plaintext-cryptotext pair that has not been analyzed yet, with $u, v \in \mathcal{B}_{1\times kj}$.
4: Apply a Gaussian elimination attack to the equation $u\boldsymbol{H}_{t,kj}(u) = v$ in order to find the matrix $\boldsymbol{H}_{t,kj}(u)$.
5: The last $(j-m)k$ columns of the matrix $\boldsymbol{H}_{t,kj}(u)$ define at most $j-m$ complete states of the encoder. Let $\mathcal{A}_i = (G^{i+m-1}_{t,0,z}, G^{i+m-1}_{t,1,z}, \ldots, G^{i+m-1}_{t,m,z})$, where $z = vect(u, [\underbrace{0 \ldots 0}_{mk}])$ and $i \in \{1, \ldots, j-m\}$.
6: **For** $i = 1$ to $j - m$ **do**
  If $\mathcal{A}_i \notin A$ **then** $A \leftarrow A \cup \{\mathcal{A}_i\}$.
7: **If** $|A| < q$ **then** go to step 3.
8: Stop.

Figure 4: Attack-I

Example 4 *Let $(t, f, \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ be the $(2, 2, 2)$ ADCC with three states introduced in Example 3. We would like to apply Attack-I in order to find the private keys $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$.*

Let $j = 5$, and let

$$(u_1, v_1) = ([\,0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\,], [\,0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\,])$$

be a plaintext-cryptotext pair. By applying a Gaussian elimination attack to the equation $u_1\boldsymbol{H}_{t,10}(u_1) = v_1$, one can find that

$$\boldsymbol{H}_{t,10}(u_1) = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & & & & \\ 0 & 1 & 0 & 0 & 1 & 1 & & & & \\ & & 1 & 1 & 0 & 0 & 1 & 1 & & \\ & & 0 & 1 & 0 & 0 & 1 & 1 & & \\ & & & & 1 & 1 & 0 & 0 & 1 & 0 \\ & & & & 0 & 1 & 0 & 0 & 1 & 1 \\ & & & & & & 1 & 1 & 0 & 0 \\ & & & & & & 0 & 1 & 0 & 0 \\ & & & & & & & & 1 & 1 \\ & & & & & & & & 0 & 1 \end{bmatrix}.$$

The matrix $\boldsymbol{H}_{t,10}(u_1)$ *contains two distinct states. One of the states is given by columns* 5 *and* 6 *(or equivalently, by columns* 9 *and* 10*), and the other one is given by columns* 7 *and* 8*. More precisely, the attacker finds that*

$$(\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}) \text{ and } (\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix})$$

are two of the three states of the cryptosystem. Since the cryptosystem has three states, the attacker picks then another plaintext-cryptotext pair, say

$$(u_2, v_2) = ([\,1\,1\,0\,1\,0\,1\,0\,1\,0\,0\,], [\,1\,0\,0\,1\,0\,0\,1\,0\,0\,1\,])$$

applies a Gaussian elimination attack to the equation $u_2\boldsymbol{H}_{t,10}(u_2) = v_2$*, and finds that*

$$\boldsymbol{H}_{t,10}(u_2) = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & & & & \\ 0 & 1 & 0 & 0 & 1 & 1 & & & & \\ & & 1 & 1 & 0 & 0 & 1 & 1 & & \\ & & 0 & 1 & 0 & 0 & 1 & 1 & & \\ & & & & 1 & 1 & 0 & 0 & 1 & 1 \\ & & & & 0 & 1 & 0 & 0 & 0 & 1 \\ & & & & & & 1 & 1 & 0 & 0 \\ & & & & & & 0 & 1 & 0 & 0 \\ & & & & & & & & 1 & 1 \\ & & & & & & & & 0 & 1 \end{bmatrix}.$$

The matrix $\boldsymbol{H}_{t,10}(u_2)$ *reveals the last state of the cryptosystem, namely*

$$(\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}).$$

Thus, the private parameters $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ *have been found and the cryptosystem is partially broken. We also note that even if the private parameter* f *remains unknown after applying our attack, some of its values are actually revealed. More precisely, if we denote the state*

$$(\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix})$$

by i_1*, the state*

$$(\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix})$$

by i_2*, and the state*

$$(\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix})$$

by i_3*, and then consider the equation* $u_1\boldsymbol{H}_{t,10}(u_1) = v_1$*, we conclude that* $f(i_1, [\,1\,1\,]) = i_2$ *and* $f(i_2, [\,1\,1\,]) = i_1$*, since the last three states of* $\boldsymbol{H}_{t,10}(u_1)$ *are* i_1, i_2, i_1 *(in this order). On the other hand, by considering the equation* $u_2\boldsymbol{H}_{t,10}(u_2) = v_2$*, we conclude that* $f(i_1, [\,0\,1\,]) = i_2$ *and* $f(i_1, [\,0\,1\,]) = i_3$*, since the last three states of* $\boldsymbol{H}_{t,10}(u_2)$ *are* i_1, i_2, i_3 *(in this order). Thus, four of the twelve values of the private parameter* f *are actually revealed. However, by applying our attack as described above, the remaining eight values of* f *remain unknown.*

Remark 6 *The algorithm Attack-I implicitly assumes that any state of the cryptosystem is reachable from the current state. Otherwise, the algorithm never stops, since those unreachable states would never be found.*

Remark 7 *We have seen that the algorithm Attack-I may reveal a number of values of the transition function f. Throughout the attack described in Example 4, four of the twelve values of the function f are revealed. In practice, however, it may happen that only a tiny part of the transition function is revealed, as it happened in our experiment. We had considered a $(256, 256, 32)$ automata-based dynamic convolutional cryptosystem with ten states, and then applied the algorithm Attack-I by taking $j = 200$ and choosing at random a plaintext-cryptotext pair (u, v). It had happened that the first plaintext-cryptotext pair revealed all ten states of the cryptosystem, which means that the algorithm iterated only once through the steps $3 - 7$. However, from the $10 \cdot 2^{256}$ values of the function f, only 167 were revealed (which is less than 1%).*

Let us now analyze the complexity of the algorithm Attack-I. Steps $1, 2, 3, 7$, and 8 can be done with $\mathcal{O}(1)$ work. Step 4 can be done with at most $k^4 j(m+1)^3$ work, since the matrix $\boldsymbol{H}_{t,kj}(u)$ has kj columns and each of its columns can be learned with at most $k^3(m+1)^3$ work, as we have seen in Section 4.1. In step 5, each of the tuples $\mathcal{A}_1, \ldots, \mathcal{A}_{j-m}$ can be constructed with $k^2(m+1)$ work, since each of them consists of $m+1$ matrices of size $k \times k$. Therefore, step 5 can be done with $(j-m)k^2(m+1)$ work. Step 6, clearly, can be done with $j-m$ work. If p is the number of iterations through the steps $3-7$, then the algorithm Attack-I can be applied with at most $pk^4 j(m+1)^3$ work.

Since the algorithm Attack-I may break the cipher in part only, we would be interested in designing an attack that breaks the cipher completely with probability 1. Such an algorithm could be obtained by doing some minor changes to Attack-I. For example, suppose that we impose that j is at least $m+2$ (such that any matrix $\boldsymbol{H}_{t,kj}(u)$ contains at least one value of f) and then iterate through the steps $3-7$ until all the states and all the values of f are found. The modified attack, called here Attack-II, is given in Figure 5.

`Input:` k, m, and q
`Output:` the set of states $A = \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$ and the values of f

---

`1:` Let $j > m+1$.
`2:` $A \leftarrow \emptyset$.
`3:` Let (u, v) be a randomly chosen plaintext-cryptotext pair that has not been analyzed yet, with $u, v \in \mathcal{B}_{1\times kj}$.
`4:` Apply a Gaussian elimination attack to the equation $u\boldsymbol{H}_{t,kj}(u) = v$ in order to find the matrix $\boldsymbol{H}_{t,kj}(u)$.
`5:` The last $(j-m)k$ columns of the matrix $\boldsymbol{H}_{t,kj}(u)$ define at most $j-m$ complete states of the encoder. Let $\mathcal{A}_i = (G_{t,0,z}^{i+m-1}, G_{t,1,z}^{i+m-1}, \ldots, G_{t,m,z}^{i+m-1})$, where $z = vect(u, [\underbrace{0 \ldots 0}_{mk}])$ and $i \in \{1, \ldots, j-m\}$.
`6:` `For` $i = 1$ to $j-m$ `do`
`If` $\mathcal{A}_i \notin A$ `then` $A \leftarrow A \cup \{\mathcal{A}_i\}$.
`7:` `If` $|A| < q$ `then` go to step `3`.
`8:` `While` there are unknown values of f `do`
`begin`
- Let (u, v) be a randomly chosen plaintext-cryptotext pair that has not been analyzed yet, with $u, v \in \mathcal{B}_{1\times kj}$.
- Apply a Gaussian elimination attack to the equation $u\boldsymbol{H}_{t,kj}(u) = v$ in order to find the matrix $\boldsymbol{H}_{t,kj}(u)$.
- Note those values of f revealed by $\boldsymbol{H}_{t,kj}(u)$ which were previously unknown.

`end`
`9:` Stop.

Figure 5: Attack-II

Regarding the complexity of Attack-II, we note that each matrix $\boldsymbol{H}_{t,kj}(u)$ reveals at most $j-m-1$ values of f. Since there are $q2^k$ values of f, we conclude that we have to analyze at least

$$\frac{q2^k}{j-m-1}$$

plaintext-cryptotext pairs (u,v) until the algorithm stops. Thus, the amount of work necessary to break the cipher completely is at least

$$\frac{q2^k}{j-m-1}k^4j(m+1)^3,$$

since each matrix $\boldsymbol{H}_{t,kj}(u)$ can be learned with about $k^4j(m+1)^3$ work, as we have already seen.

However, in practice, it seems that Attack-II would be almost impossible to apply due to the fact that it requires too much time. Suppose, for example, that we want to apply Attack-II to completely break a $(256,256,32)$ ADCC with ten states. Also, suppose that we choose $j=200$ and run the attack on a computer that learns each matrix $\boldsymbol{H}_{t,kj}(u)$ in 10^{-3} seconds. Since we have to learn at least

$$\frac{q2^k}{j-m-1}=\frac{10\cdot 2^{256}}{167}$$

matrices, we conclude that the attack would take at least

$$\frac{10\cdot 2^{256}}{167}10^{-3} \text{ seconds } = \ 2.2\cdot 10^{65} \text{ years.}$$

As a final conclusion to our analysis, we remind our main findings:

1. It is relatively easy to find partial information about an automata-based dynamic convolutional cryptosystem.

2. Breaking an automata-based dynamic convolutional cryptosystem completely is a very difficult task.

Preserving the second property and eliminating the first one (which is a drawback) should lead to a much more secure cryptosystem. We will deal with this security issue in the next section.

5 Cascades of automata-based dynamic convolutional cryptosystems

As we have seen in Section 3, an automata-based dynamic convolutional cryptosystem is, at least from some point of view, much more resistant to cryptanalytic attacks than previously proposed dynamic cryptosystems. However, as shown in Section 4, such a cryptosystem can be partially broken in a reasonable amount of time. A much more secure alternative is proposed and analyzed in this section.

Definition 9 *A* permutation *of the set S is a tuple consisting of the elements of S. A permutation of the set $\{1,2,\ldots,n\}$ will be called a* permutation of length n.

Definition 10 *An* S-box *(or* substitution*)* of length n *is a permutation of the set $\{0,1,\ldots,2^n-1\}$.*

Definition 11 *A* meta S-box *(or* meta substitution*)* of length n *is a tuple $(S_1,\ldots,S_p)$ of S-boxes such that the sum of the lengths of its S-boxes is n.*

Definition 12 *Let k, m, N, q_1, $\ldots$, q_N be nonzero natural numbers. A* nonlinear (k,k,m) N-cascaded automata-based dynamic convolutional transducer with $q_1 \ldots q_N$ states *is a $2N$-tuple $(e, t_1, s_1, t_2, s_2, \ldots, t_{N-1}, s_{N-1}, t_N)$, where*

$$e(u) = x_{N-1}\boldsymbol{H}_{t_N,kp}(x_{N-1}) \tag{7}$$

for all $u \in \mathcal{B}_{1\times kp}$, $x_0 = u$, $x_i = s_i(x_{i-1}\boldsymbol{H}_{t_i,kp}(x_{i-1}))$, $(t_i, f_i, \mathcal{S}_{t_i,1}, \ldots, \mathcal{S}_{t_i,q})$ is the i-th (k,k,m) ADCT of the cascade, q_i is the number of states of the i-th transducer of the cascade, and $s_i : \cup_{i=1}^{\infty}\mathcal{B}_{1\times ki} \to \cup_{i=1}^{\infty}\mathcal{B}_{1\times ki}$ is a function which, for every input of length kj, applies a meta S-box $MSub_i$ of length k and then a permutation Per_i of length k to each of the j blocks of length k.

Definition 13 *A nonlinear (k,k,m) N-cascaded ADCC is a globally invertible nonlinear (k,k,m) N-cascaded ADCT in which the transition functions, the meta S-boxes, the permutations, and the states of each cryptosystem of the cascade are kept private.*

Definition 14 *A linear (k,k,m) N-cascaded ADCT is obtained from a nonlinear (k,k,m) N-cascaded ADCT by eliminating all meta S-boxes and all permutations.*

The most fundamental property of an S-box is that it is a nonlinear mapping, i.e., the output bits cannot be represented as a linear operation on the input bits. From this reason, the possibility of applying a Gaussian elimination attack to a nonlinear N-cascaded ADCC is eliminated. Let us now take an example.

Example 5 *Let (e, t_1, s_1, t_2) be a nonlinear $(4,4,2)$ 2-cascaded automata-based dynamic convolutional cryptosystem with $q_1q_2 = 4$ states, where*

$$G^1_{t_1,0} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, G^1_{t_1,1} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, G^1_{t_1,2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$G^2_{t_1,0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, G^2_{t_1,1} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, G^2_{t_1,2} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$G^1_{t_2,0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, G^1_{t_2,1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, G^1_{t_2,2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$G^2_{t_2,0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, G^2_{t_2,1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, G^2_{t_2,2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$q_1 = 2$ is the number of states of the first transducer in the cascade, $q_2 = 2$ is the number of states of the second transducer in the cascade,

$$f_1(1, \{[\,0\;0\;0\;0\,], [\,0\;0\;0\;1\,], [\,0\;0\;1\;0\,], [\,0\;0\;1\;1\,]\}) = \{1\},$$
$$f_1(1, \{[\,0\;1\;0\;0\,], [\,0\;1\;0\;1\,], [\,0\;1\;1\;0\,], [\,0\;1\;1\;1\,]\}) = \{2\},$$
$$f_1(1, \{[\,1\;0\;0\;0\,], [\,1\;0\;0\;1\,], [\,1\;0\;1\;0\,], [\,1\;0\;1\;1\,]\}) = \{1\},$$
$$f_1(1, \{[\,1\;1\;0\;0\,], [\,1\;1\;0\;1\,], [\,1\;1\;1\;0\,], [\,1\;1\;1\;1\,]\}) = \{1\},$$

$$f_1(2,\{[\,0\;0\;0\;0\,],[\,0\;0\;0\;1\,],[\,0\;0\;1\;0\,],[\,0\;0\;1\;1\,]\}) = \{2\},$$
$$f_1(2,\{[\,0\;1\;0\;0\,],[\,0\;1\;0\;1\,],[\,0\;1\;1\;0\,],[\,0\;1\;1\;1\,]\}) = \{2\},$$
$$f_1(2,\{[\,1\;0\;0\;0\,],[\,1\;0\;0\;1\,],[\,1\;0\;1\;0\,],[\,1\;0\;1\;1\,]\}) = \{1\},$$
$$f_1(2,\{[\,1\;1\;0\;0\,],[\,1\;1\;0\;1\,],[\,1\;1\;1\;0\,],[\,1\;1\;1\;1\,]\}) = \{1\},$$
$$f_2(1,\{[\,0\;0\;0\;0\,],[\,0\;0\;0\;1\,],[\,0\;0\;1\;0\,],[\,0\;0\;1\;1\,]\}) = \{1\},$$
$$f_2(1,\{[\,0\;1\;0\;0\,],[\,0\;1\;0\;1\,],[\,0\;1\;1\;0\,],[\,0\;1\;1\;1\,]\}) = \{2\},$$
$$f_2(1,\{[\,1\;0\;0\;0\,],[\,1\;0\;0\;1\,],[\,1\;0\;1\;0\,],[\,1\;0\;1\;1\,]\}) = \{2\},$$
$$f_2(1,\{[\,1\;1\;0\;0\,],[\,1\;1\;0\;1\,],[\,1\;1\;1\;0\,],[\,1\;1\;1\;1\,]\}) = \{1\},$$
$$f_2(2,\{[\,0\;0\;0\;0\,],[\,0\;0\;0\;1\,],[\,0\;0\;1\;0\,],[\,0\;0\;1\;1\,]\}) = \{1\},$$
$$f_2(2,\{[\,0\;1\;0\;0\,],[\,0\;1\;0\;1\,],[\,0\;1\;1\;0\,],[\,0\;1\;1\;1\,]\}) = \{2\},$$
$$f_2(2,\{[\,1\;0\;0\;0\,],[\,1\;0\;0\;1\,],[\,1\;0\;1\;0\,],[\,1\;0\;1\;1\,]\}) = \{1\},$$
$$f_2(2,\{[\,1\;1\;0\;0\,],[\,1\;1\;0\;1\,],[\,1\;1\;1\;0\,],[\,1\;1\;1\;1\,]\}) = \{1\},$$

$MSub_1 = (Sub_{1,1}, Sub_{1,2}) = ((0,3,1,2),(2,3,0,1))$, $Sub_{1,1}$ is given in Table 2, $Sub_{1,2}$ is given in Table 3, and the permutation $Per_1 = (2,4,1,3)$ is given in Table 4. It can be seen that each of the S-boxes $Sub_{1,1}$ and $Sub_{1,2}$ can be easily implemented with a table lookup of four 2-bit values, indexed by the integer represented by the input block. More precisely, the integer represented by $[\,u_1\;u_2\,]$ is $2u_1 + u_2$. Regarding the permutation Per_1, the numbers in Table 4 represent bit positions in the block, with 1 being the leftmost bit and 4 being the rightmost bit.

Table 2: the S-box $Sub_{1,1}$

input	0	1	2	3
output	0	3	1	2

Table 3: the S-box $Sub_{1,2}$

input	0	1	2	3
output	2	3	0	1

Table 4: the permutation Per_1

input	1	2	3	4
output	2	4	1	3

The initial structure of the cascade is illustrated in Figure 6 (a). Let $u = [\,0\;1\;1\;0\,]$ be an input vector. The vector u is encrypted by

$$e(u) = x_1 \boldsymbol{H}_{t_2,4}(x_1) = [\,0\;1\;0\;1\,].$$

Given that $f_1(1,[\,0\;1\;1\;0\,]) = 2$, the first transducer in the cascade reaches its second state after encoding the vector u. As $x_1 = s_1(u\boldsymbol{H}_{t_1,4}(u)) = [\,0\;1\;0\;1\,]$ and $f_2(1,[\,0\;1\;0\;1\,]) = 2$, the second transducer in the cascade reaches its second state as well. This is illustrated in Figure 6 (b).

5.1 Cryptanalysis

Given that equation (7) makes use of nonlinear components, the possibility of applying a Gaussian elimination attack (such as Attack-I) in order to partially break a nonlinear N-cascaded ADCC is completely eliminated. On the other hand, we were unable to design an efficient linear or

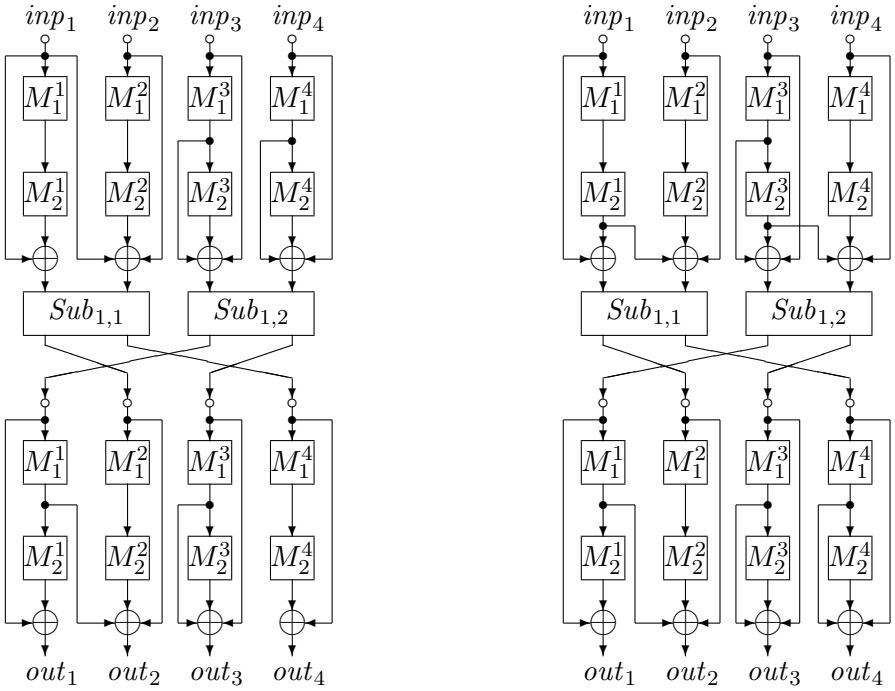

Figure 6: (a) the initial structure of the cascade, and (b) the structure of the cascade after encoding the vector [0 1 1 0]

differential cryptanalytic attack that would break a dynamic cryptosystem completely. However, linear or differential cryptanalysis are techniques that work on static ciphers, so a single linear or differential cryptanalytic attack would simply not be sufficient to break a dynamic cryptosystem completely, since a dynamic cryptosystem has multiple structures. Moreover, for a nonlinear N-cascaded automata-based dynamic convolutional cryptosystem we also have N transition functions (which have to be learned in order to break the system completely).

So, the idea would be to apply a linear or differential cryptanalytic attack multiple times until all the structures and all the values of the transition functions are found. Let us suppose that CRYPT is a linear or differential cryptanalytic attack that, for any plaintext-cryptotext pair $(u, e(u))$, where u and $e(u)$ are blocks of length k, would find that structure of the cascade that was used to encrypt u by $e(u)$. In order to find that structure of the cascade that was used to encrypt u by $e(u)$, we also have to know what bit was stored in each memory register at the time u was encrypted. In other words, we have to know the previous m encrypted blocks and the previous m structures of the cascade. We implicitly assume that such a structure is uniquely determined. Unlike a Gaussian elimination attack, which can work on plaintext-cryptotext pairs of any length[1] (multiple of k), CRYPT can work only on plaintext-cryptotext pairs of length k (since CRYPT is a linear or differential cryptanalytic attack). Such an attack that uses CRYPT multiple times until the cryptosystem is broken completely is Attack-III from Figure 7. Due to the fact that we do not know what complexity the attack CRYPT might have, we will not deal with computing the complexity of Attack-III. However, we will do compute the minimum number of plaintext-cryptotext pairs (u, v) needed to break the cryptosystem completely.

First, we remind that a cascade has $q_1 q_2 \dots q_N$ possible states. Given that each plaintext-cryptotext pair $(u_{(i-1)k+1:ik}, v_{(i-1)k+1:ik})$ of length k reveals at most one state of the cascade, we

[1]A pair (u, v) is said to be of length n if u and v are both of length n.

Input: $k, m, N, q_1, \ldots, q_N$.
Output: all $q_1 q_2 \ldots q_N$ states of the cascade, all values of the transition functions, all $N-1$ meta S-boxes, and all $N-1$ permutations.

1: Let $j \geq 2$.
2: **while** the cryptosystem is not yet broken completely **do**
begin
- Let (u, v) be a randomly chosen plaintext-cryptotext pair that has not been analyzed yet, with $u, v \in \mathcal{B}_{1 \times kj}$.
- **For** $i = 1$ to j **do**
begin
- Apply CRYPT and find that structure of the cascade that was used to encrypt $u_{(i-1)k+1:ik}$ by $v_{(i-1)k+1:ik}$. In order to do so, we have to provide the previous m structures of the cascade and the previous m plaintext-crytotext pairs of length k.
- That structure of the cascade that was used to encrypt $u_{(i-1)k+1:ik}$ by $v_{(i-1)k+1:ik}$ may reveal one new state for each of the N transducers of the cascade and, if $i \geq 2$, it may also reveal one new value for each of the N transition functions. It also reveals the meta S-boxes $MSub_1, \ldots, MSub_{N-1}$ and the permutations $Per_1, \ldots, Per_{N-1}$. Note them all.

end
end
3: Stop.

Figure 7: Attack-III

conclude that each plaintext-cryptotext pair (u, v) of length kj reveals at most j states of the cascade. Thus, we have to analyze at least

$$\lceil \frac{q_1 q_2 \ldots q_N}{j} \rceil$$

plaintext-cryptotext pairs (u, v) of length kj in order to find all the states of the cascade. On the other hand, we know that there are $q_1 2^k + \ldots + q_N 2^k$ values of the transition functions and each plaintext-cryptotext pair (u, v) of length kj reveals at most $(j-1)N$ values. So, we have to analyze at least

$$\lceil \frac{q_1 2^k + \ldots + q_N 2^k}{(j-1)N} \rceil$$

plaintext-cryptotext pairs (u, v) of length kj in order to find all the values of the transition functions. Thus, we conclude that we have to analyze at least

$$max\{\lceil \frac{q_1 q_2 \ldots q_N}{j} \rceil, \lceil \frac{q_1 2^k + \ldots + q_N 2^k}{(j-1)N} \rceil\}$$

pairs (u, v) of length kj in order to break the cryptosystem completely. For a nonlinear $(256, 256, 32)$ 3-cascaded automata-based dynamic convolutional cryptosystem with $q_1 q_2 q_3 = 10 \cdot 10 \cdot 10 = 1000$ states and $j = 10$, we have to analyze at least

$$max\{\lceil \frac{1000}{10} \rceil, \lceil \frac{30 \cdot 2^{256}}{27} \rceil\} = \frac{10}{9} 2^{256} \approx 1.28 \cdot 10^{77}$$

plaintext-cryptotext pairs (u, v) of length $kj = 2560$. If we run the attack on a computer that executes the algorithm CRYPT in 10^{-2} seconds (although this is somewhat unrealistic, since most

linear or differential cryptanalytic attacks usually take much more time to complete), then we can break our cryptosystem completely in at least

$$1.28 \cdot 10^{77} \cdot 10 \cdot 10^{-2} \text{ seconds } \approx 4.07 \cdot 10^{68} \text{ years.}$$

So, the results of our analysis can be summarized by the following two statements.

1. If $N \geq 2$, then the task of partially breaking a nonlinear N-cascaded automata-based dynamic convolutional cryptosystem becomes more difficult than in the case of automata-based dynamic convolutional cryptosystems, since the linearity is completely eliminated.

2. If $N \geq 2$, then breaking a nonlinear N-cascaded automata-based dynamic convolutional cryptosystem completely is a very difficult task. In fact, we make no mistake if we say that for well chosen values of $k, m, N, q_1, q_2, \ldots, q_N$, an attack such as Attack-III would be practically impossible to apply with the current technology.

6 Efficient implementations of nonlinear N-cascaded ADCCs in FPGAs with embedded memory

Cryptographic algorithms can be implemented in hardware using VLSI (Very Large Scale Integrated) circuits such as ASICs (application-specific integrated circuits) [6], or reconfigurable logic platforms such as FPGAs (Field-Programmable Gate Arrays) [14]. An ASIC is an integrated circuit customized for a particular use, rather than intended for general-purpose use. On the other hand, an FPGA is a user-programmable digital device that provides efficient, yet flexible, implementation of digital circuits. It consists of an array of programmable logic blocks interconnected by programmable routing resources. The flexibility of FPGAs allows them to be used for a variety of digital applications from small finite state machines to large complex systems. The choice of the platform (ASICs or FPGAs) depends on algorithm performance, cost, and flexibility. Since our cryptosystems are dynamic, it is natural to discuss their implementation in FPGAs, since such devices can be reconfigured at runtime. In fact, we will discuss their implementation in FPGAs with embedded memory, since a nonlinear N-cascaded ADCC is a cryptosystem that requires memory.

An implementation of nonlinear $(k, k, 3)$ 2-cascaded ADCCs in FPGAs with embedded memory is illustrated in Figure 8. It allows the encryption of a block of length k into the corresponding block of length k, and consists of

- ten FPGA registers, namely $A_1, A_2, A_3, A_4, B_1, B_2, B_3, B_4, M_1$, and M_2,
- two state generators, namely SG_1 and SG_2,
- two memory modules, namely $MEMORY_1$ and $MEMORY_2$,
- three concatenators, namely A, B, and X,
- two modular vector-matrix multiplicators,
- two S-boxes, namely $Sub_{1,1}$ and $Sub_{1,2}$,
- and one permutation, namely Per_1.

The rest of the circuit consists of FPGA wires and FPGA logic blocks. Among the structures mentioned above, the FPGA registers are somewhat special, since their content is released and sent out through the circuit only when a certain clock cycle occurs. Regarding the state generators

and the S-boxes, their main function is to process some data at certain clock cycles. It is also worth mentioning that only the FPGA registers and the memory modules are able to store information, since the concatenators, the permutation, and the modular vector-matrix multiplicators can be easily implemented using only FPGA wires and/or FPGA logic blocks.

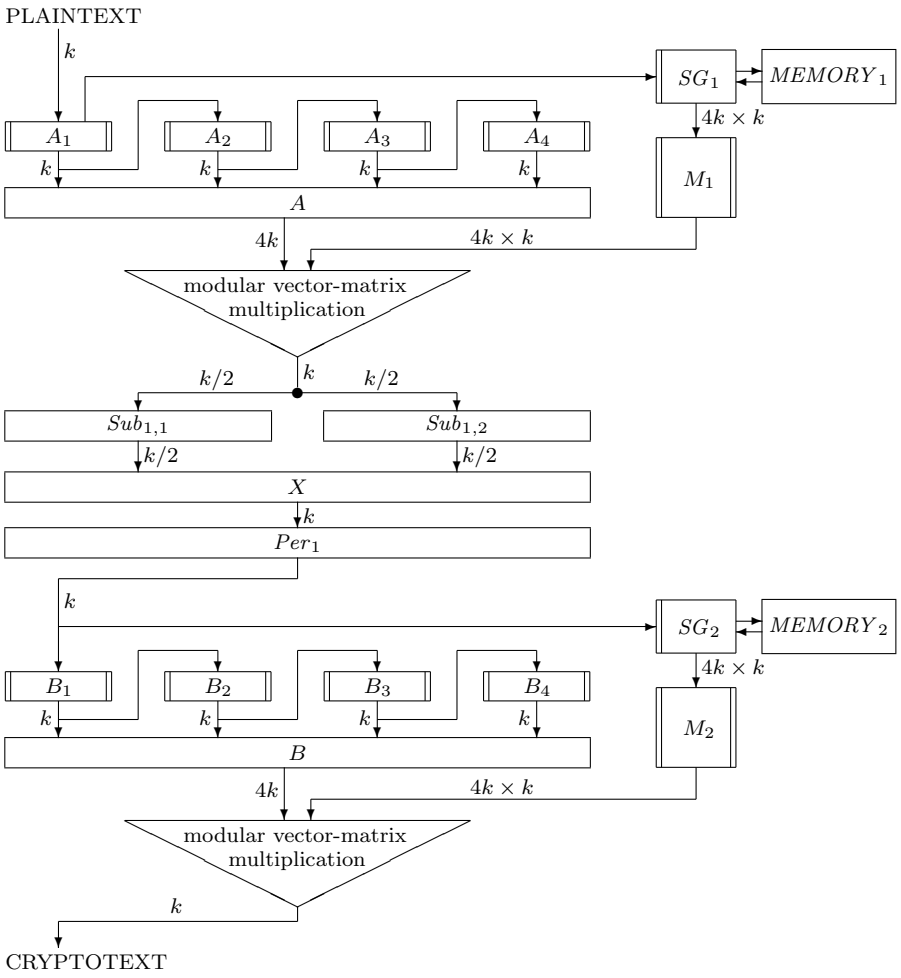

Figure 8: An FPGA implementation of a nonlinear $(k, k, 3)$ 2-cascaded ADCC

The circuit has four pins: Chip Enable (CE), Clock (CLK), input data (IN), and output data (OUT). The aim of the CE pin is to activate the circuit. The external clock CLK is the master clock for the whole circuit and is used to generate all control signals that synchronize the data flow. Initially, each of the registers A_2, A_3, A_4, B_2, B_3, and B_4 contains a block of k zeros, while A_1 and B_1 are empty. Also, M_1 stores the $4k \times k$ matrix corresponding to the initial state of the first transducer in the cascade, while M_2 stores the $4k \times k$ matrix corresponding to the initial state of the second transducer in the cascade. Regarding the memory modules, $MEMORY_1$ stores the states of the first transducer in the cascade and the transtion function f_1, while $MEMORY_2$ stores the states of the second transducer in the cascade and the transition function f_2.

When CE enables the circuit, the k-bit plaintext is stored in the FPGA register A_1 and remains there until the first clock cycle occurs. The circuit outputs the corresponding k-bit cryptotext using two clock cycles. When the first clock cycle occurs, the following actions take place:

1. Each of the FPGA registers A_1, A_2, A_3, A_4, and M_1 releases its content and sends it out through the circuit. The register A_1 becomes empty, the register A_2 stores the previous

content of A_1, the register A_3 stores the previous content of A_2, and the register A_4 stores the previous content of A_3.

2. The concatenator A takes as input the four k-bit vectors released from A_1, A_2, A_3, and A_4, and concatenates them into one single $4k$-bit vector.

3. A modular vector-matrix multiplication between the $4k$-bit row vector resulted from concatenation and the $4k \times k$ matrix released from M_1 is performed. The result is a k-bit row vector, call it v_1.

4. The first half of the vector v_1 is sent to the S-box $Sub_{1,1}$, whereas the second half of v_1 is sent to the S-box $Sub_{1,2}$.

5. Each of those two subvectors is substituted with another one of the same length. The resulted subvectors are then concatenated into one single vector of length k.

6. The k bits of the resulted vector are then permuted using Per_1. The result is stored in the FPGA register B_1 and remains there until the second clock cycle occurs.

7. When the content of A_1 is released, the state generator SG_1 receives it and then, by accessing the memory module $MEMORY_1$, computes the $4k \times k$ matrix corresponding to the next state of the the first transducer in the cascade. That $4k \times k$ matrix is then stored in M_1 and remains there.

The FPGA registers B_1, B_2, B_3, B_4, and M_2 do not release their content during the first clock cycle. When the second clock cycle occurs, the following actions take place:

1. Each of the FPGA registers B_1, B_2, B_3, B_4, and M_2 releases its content and sends it out through the circuit. The register B_1 becomes empty, the register B_2 stores the previous content of B_1, the register B_3 stores the previous content of B_2, and the register B_4 stores the previous content of B_3.

2. The concatenator B takes as input the four k-bit vectors released from B_1, B_2, B_3, and B_4, and concatenates them into one single $4k$-bit vector.

3. A modular vector-matrix multiplication between the $4k$-bit row vector resulted from concatenation and the $4k \times k$ matrix released from M_2 is performed. The result is the corresponding k-bit cryptotext.

4. When the content of B_1 is released, the state generator SG_2 receives it and then, by accessing the memory module $MEMORY_2$, computes the $4k \times k$ matrix corresponding to the next state of the the first transducer in the cascade. That $4k \times k$ matrix is then stored in M_2 and remains there.

The FPGA registers A_1, A_2, A_3, A_4, and M_1 do not release their content during the second clock cycle. After the k-bit cryptotext is obtained, the circuit is disabled. For another k-bit plaintext, the circuit has to be enabled, and then, using two clock cycles, the corresponding k-bit cryptotext is obtained. Such an implementation is very efficient, since the parallel structure of the two S-boxes and of the FPGA registers $A_1, A_2, A_3, A_4, B_1, B_2, B_3, B_4$ results in a significant reduction of the critical path for encryption.

7 Conclusions

In this paper, we have proposed and analyzed a new class of dynamic symmetric cryptosystems, called automata-based dynamic convolutional cryptosystems (ADCCs). First, we provided the reader with a brief introduction to convolutional codes. Second, we gave the definition of an ADCC, and then showed how to use such a cryptosystem for encryption/decryption. Third, we provided a thorough security analysis of ADCCs, and then discussed their practical advantages. The conclusion of our cryptanalysis is that an ADCC is very hard to break completely, but quite easy to break partially. Fourth, an extension of ADCCs, called nonlinear cascaded ADCCs, was proposed and shown to be much more secure in practice than ADCCs. Finally, an efficient implementation of nonlinear cascaded ADCCs in FPGAs with embedded memory was presented.

References

[1] Advanced Encryption Standard: `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

[2] Data Encryption Standard: `http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`

[3] Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology **4**(1):3–72, 1991.

[4] Dholakia, A.: Introduction to Convolutional Codes with Applications. Kluwer (1994)

[5] Hellman, M., Langford, S.: Differential-Linear Cryptanalysis. In *Proceedings of CRYPTO'94*, Santa Barbara (USA), August 21–25, 1994, Lecture Notes in Computer Science, volume 839, Springer-Verlag.

[6] Horspool, N., Gorman, P.: The ASIC Handbook. Prentice Hall (2001)

[7] JáJá, J.: An Introduction to Parallel Algorithms. Addison-Wesley (1992)

[8] Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In *Proceedings of EUROCRYPT'93*, Lofthus (Norway), May 23–27, 1993, pages 386–397, Lecture Notes in Computer Science, volume 765, Springer-Verlag.

[9] Piret, Ph.: Convolutional Codes: An Algebraic Approach. MIT Press, Cambridge (USA) (1988)

[10] Pless, V.S., Huffman, W.C. (Eds.): Handbook of Coding Theory (2 volumes). Elsevier (1998)

[11] Trincă, D.: Sequential and Parallel Cascaded Convolutional Encryption with Local Propagation: Toward Future Directions in Symmetric Cryptography. In *Proceedings of the 3rd International Conference on Information Technology: New Generations*, Las Vegas (USA), April 10–12, 2006, pages 464–469, IEEE Computer Society Press. Also, available as Cryptology ePrint Archive Report 2006/003.

[12] Triple DES: `http://www.itl.nist.gov/fipspubs/fip46-2.htm`

[13] Viterbi, A.J.: Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. IEEE Transactions on Information Theory **IT-13**(2):260–269, 1967.

[14] Wolf, W.: FPGA-based System Design. Prentice Hall (2004)