Sequential and Parallel Cascaded Convolutional Encryption with Local Propagation: Toward Future Directions in Symmetric Cryptography

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

Abstract

Worldwide symmetric encryption standards such as DES (Data Encryption Standard), AES (Advanced Encryption Standard), and EES (Escrowed Encryption Standard), have been – and some of them still are – extensively used to solve the problem of communication over an insecure channel, but with today's advanced technologies, they seem to not be as secure as one would like.

In this paper, we propose efficient alternatives based on special classes of globally invertible cascaded convolutional transducers. The proposed symmetric encryption techniques have at least four advantages over traditional schemes based on Feistel ciphers. First, the secret key of a cascaded convolutional cryptosystem is usually much more easier to generate. Second, the encryption and decryption procedures are much simpler, and consequentially, much faster. Third, the desired security level can be obtained by just setting appropriate values for the parameters of the convolutional cryptosystem. Finally, they are much more parallelizable than symmetric encryption standards based on Feistel ciphers.

Keywords: Block ciphers; Convolutional codes; Parallel algorithms; Symmetric encryption

1 Introduction

Symmetric cryptography has been – and still is – extensively used to solve the traditional problem of communication over an insecure channel. Well-known symmetric encryption standards such as DES [5], AES [1], and EES [7] have been designed using Feistel ciphers, but they seem to not be as practical as one would like.

DES was developed at IBM in 1977, and in the same year was adopted by NIST as the standard data encryption algorithm. It was recertified every five years until 1999, when the key size of 56 bits was considered too vulnerable to attacks. In 2001, DES was replaced by AES, which became effective May 26, 2002. AES supports key sizes of 128 bits, 192 bits, and 256 bits, in contrast to the 56-bit key offered by DES, and is intended to remain secure well into the 21st century.

The algorithm at the core of AES, called Rijndael, has been designed to have very strong resistance against the classical approximation attacks such as linear cryptanalysis and differential cryptanalysis, but recent advances in applied cryptography seem to reduce significantly its lifetime. For example, in a recent paper [4] presented at Asiacrypt 2002, Courtois and Pieprzyk describe an attack that might break AES. Their attack, called XSL, is shown to be faster than the exhaustive search, and able to break Rijndael 256 bits and Serpent [3] for key lengths 192 and 256 bits. It is likely that their method will be studied further, and probably many other methods for breaking AES will appear in the near future. Thus, it is unlikely that AES will remain secure as originally conceived, and consequentially, there is a growing need of more efficient symmetric cryptosystems.

Moreover, even with today's technology, block ciphers such as DES and AES are not as fast in software as one would like; this is another reason to seek alternatives for fast software encryption.

In this paper, we propose efficient alternatives based on special classes of globally invertible cascaded convolutional transducers. Our symmetric encryption techniques have at least four advantages over traditional schemes based on Feistel ciphers. First, the secret key of a cascaded convolutional cryptosystem is usually much more easier to generate. Second, the encryption and decryption procedures are much simpler, and consequentially, much faster. Third, the desired security level can be obtained by just setting appropriate values for the parameters of the convolutional cryptosystem. Finally, they are much more parallelizable than symmetric encryption standards based on Feistel ciphers.

The presentation is organized as follows. First, we provide a brief introduction to convolutional codes and show how to use a globally invertible convolutional transducer for data encryption and decryption (section 2). Second, we propose a new class of cascaded convolutional transducers, called *cascaded convolutional transducers with propagation*, and describe a parallel version of the corresponding encryption procedure using the well-known shared-memory model of computation (section 3). Then, in section 4, we discuss the security of our cryptosystems and provide experimental results. As we will see, our cryptosystems are much more faster than standard AES implementations. Finally, in the last section, we provide some conclusions and future work directions.

2 Convolutional transducers

Convolutional codes [6, 10, 11] 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* [12].

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. (Note that there is no previous work on this subject.) The aim of this section is to provide the reader with a brief introduction to convolutional codes.

As stated in [6], 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. Note that $u[q,-] \in \mathcal{B}_{1\times j}$ and $u[-,q] \in \mathcal{B}_{i\times 1}$. 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 $[\, u[i] \dots u[j] \,]$. If $u_1, \dots, u_h$ are binary vectors, then we denote by $vect(u_1, \dots, u_h)$ the vector consisting of the components of $u_1, \dots, u_h$, in the same order. For example, $vect([\,0\ 0\,],[\,1\ 1\,]) = [\,0\ 0\ 1\ 1\,]$.

Definition 2.1 *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

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

is an element of $\mathcal{B}_{kp\times(pn+mn)}$, $G_{t,i} \in \mathcal{B}_{k\times n}$ *for all* $i \in \{0, 1, \ldots, m\}$, *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 2.2 *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 2.3 *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* $\boldsymbol{G}_t$ *associated with* t.

For a more detailed introduction to convolutional codes, we refer the reader to [6]. Let us take an example.

Example 2.1 *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* $G_{t,0} = [\ 0\ 1\ ]$, $G_{t,1} = [\ 1\ 0\ ]$, *and* $G_{t,2} = [\ 1\ 1\ ]$. *Thus, 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 Fig. 1. Note that the two output bits at each step are serialized using a multiplexer.*

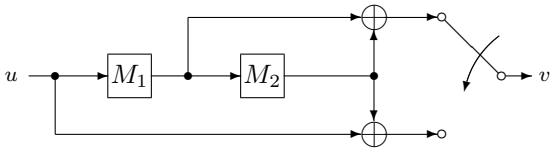

Figure 1: A $(2, 1, 2)$ convolutional encoder

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*

$$bG_{t,0}[1] + b_1G_{t,1}[1] + b_2G_{t,2}[1] = b_1G_{t,1}[1] + b_2G_{t,2}[1],$$

whereas the second output bit is

$$bG_{t,0}[2] + b_1G_{t,1}[2] + b_2G_{t,2}[2] = bG_{t,0}[2] + b_2G_{t,2}[2].$$

After both output bits have been 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 an input vector. Then, one can verify that the output vector is* $t(u) = u\boldsymbol{G}_{t,4} = [\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ ]$, *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* $pn + mn = 8 + 4 = 12$.

Throughout this paper, we will 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 2.2 *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 Fig. 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.

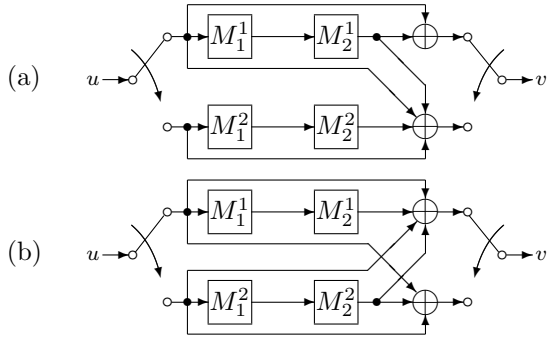

Figure 2: (a) A globally invertible $(2,2,2)$ convolutional encoder, and (b) a $(2,2,2)$ convolutional encoder that is not globally invertible

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 $t(u) = u\boldsymbol{G}_{t_1,kp} = [\,0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\,]$. We can encrypt u by $v = [\,0\ 1\ 1\ 1\,]$, i.e., the first $kp = 4$ bits of $t(u)$. Given that t_1 is globally invertible, we can uniquely decrypt v into u.

Definition 2.4 *Let k, m, and q be nonzero natural numbers. A (k,k,m) q-*cascaded convolutional transducer *is a function $t : \cup_{i=1}^{\infty}\mathcal{B}_{1\times ki} \mapsto \cup_{i=1}^{\infty}\mathcal{B}_{1\times ki}$ given by*

$$t(u) = u\boldsymbol{G}_{t,kp}^1\boldsymbol{G}_{t,kp+km}^2 \ldots \boldsymbol{G}_{t,kp+(q-1)km}^q, \tag{2}$$

for all $u \in \mathcal{B}_{1\times kp}$, where $\boldsymbol{G}_{t,kp}^i \in \mathcal{B}_{kp\times(kp+mk)}$ for all $i \in \{1,2,\ldots,q\}$.

3 Sequential and parallel cascaded convolutional encryption with local propagation

In the previous section, we have described some symmetric encryption schemes based on globally invertible (k,k,m) (q-cascaded) convolutional transducers. In this section we propose a new class of convolutional transducers, called *cascaded convolutional transducers with local propagation.* Unlike most (if not all) of the symmetric cryptosystems that have been proposed and studied in the literature, our cryptosystems are *dynamic*, i.e., their structure changes during the encryption

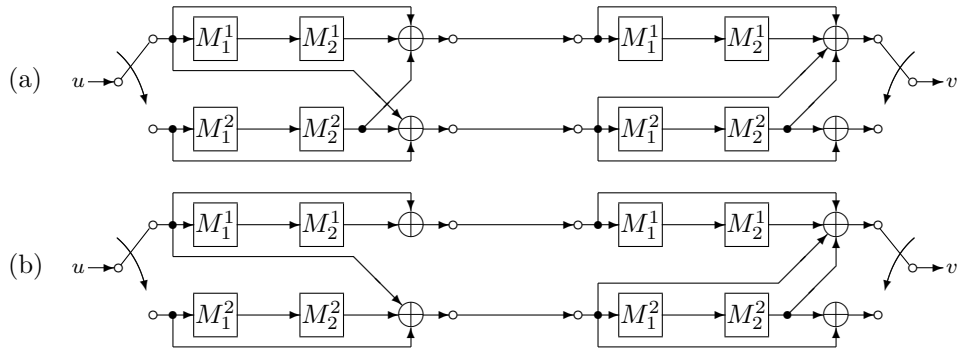

Figure 3: A $(2, 2, 2)$ 2-cascaded convolutional encoder with propagation: (a) the initial structure of the cascaded encoder, and (b) the structure after encoding the first block [1 0]

procedure. The purpose of adding such properties to a cascaded convolutional transducer is thus to increase the complexity of linear and differential cryptanalysis attacks (or even to completely eliminate the possibility of successfully applying such attacks that have been primarily developed in the context of *static* cryptosystems). After introducing the new concept, we will describe a parallel algorithm for cascaded convolutional cryptosystems with local propagation, using the well-known shared-memory model of computation [8]. As we will see, a cascaded convolutional cryptosystem with local propagation is highly parallelizable.

Definition 3.1 *Let k and m be nonzero natural numbers. A (k, k, m) q-*cascaded convolutional transducer with propagation *is an $(q + 1)$-tuple $(t, \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_q)$, where t is a function given by*

$$t(u) = u\boldsymbol{H}^1_{t,kp}(v_0)\boldsymbol{H}^2_{t,kp}(v_1)\ldots\boldsymbol{H}^q_{t,kp}(v_{q-1}), \tag{3}$$

for all $u \in \mathcal{B}_{1\times kp}$, where $v_0 = u$, $v_i = v_{i-1}H^i_{t,kp}(v_{i-1})$ for all $i \in \{1, 2, \ldots, q\}$, $\boldsymbol{H}^i_{t,kp}(w)$ is the restriction of

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

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

$$\begin{aligned} \mathcal{S}_i \;=\; & \{G^i_{t,j}(0) \mid j \in \{0, 1, \ldots, m\}\} \cup \\ & \{G^i_{t,j}(1) \mid j \in \{0, 1, \ldots, m\}\} \end{aligned}$$

is the set of state matrices *corresponding to the i-th transducer of the cascade, $i = 1, \ldots, q$. As usual, all the operations are performed over the binary field $GF(2)$.*

Remark 3.1 *For convenience, a q-cascaded convolutional transducer with propagation will be denoted by the encryption function t.*

Example 3.1 *Let* $t : \cup_{i=1}^{\infty} \mathcal{B}_{1\times 2i} \mapsto \cup_{i=1}^{\infty} \mathcal{B}_{1\times 2i}$ *be a* $(2, 2, 2)$ 2*-cascaded convolutional transducer with propagation, where*

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

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

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

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

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

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

The initial structure of the corresponding cascaded convolutional encoder can be represented graphically as in Fig. 3 (a). By examining the matrices $G^i_{t,j}(b)$*, one can remark that the cascaded encoder is globally invertible whatever its structure would be at the current step. Let* $u = [\ 1\ 0\ 0\ 1\ ]$ *be an input vector. The structure of the encoder after encoding the first block* $[\ 1\ 0\ ]$ *is given in Fig. 3 (b). More precisely, the structure of the first encoder in the cascade changes after encoding the first input block, since the number of* 1*'s in the first input block is odd. Regarding the second encoder in the cascade, its structure changes after encoding the first input block, since the number of* 1*'s in the first input block is even. (Note that the input to the second encoder in the cascade is the output of the first encoder.) It is easy to verify that*

$$t(u) = u\boldsymbol{H}^1_{t,4}(v_0)\boldsymbol{H}^2_{t,4}(v_1) = [\ 0\ 1\ 1\ 1\ ].$$

Thus, we encrypt u *by* $t(u)$*, and given that* t *is globally invertible, we can uniquely decrypt* $t(u)$ *into* u*.*

As we have already pointed out, we will show that cascaded convolutional transducers with propagation are highly parallelizable. More precisely, we will describe a parallel algorithm for cascaded convolutional cryptosystems with propagation using the shared-memory model of computation under the asynchronous mode of operation. (For a good introduction to the shared-memory model of computation, the reader is referred to [8].)

Let t be a globally invertible (k, k, m) q-cascaded convolutional transducer with propagation. Suppose that we have k processors, denoted $p_1, p_2, \ldots, p_k$, and consider a binary vector $u \in \mathcal{B}_{1\times kp}$. The processors work as follows. First, each processor copies (concurrently) the input vector u and the matrices $G^i_{t,j}(b)$ from the global memory into its local memory. Second, for all $l \in \{1, 2, \ldots, q\}$, each processor p_j computes the columns

$$\boldsymbol{H}^l_{t,kp}(u)[-, j], \ldots, \boldsymbol{H}^l_{t,kp}(u)[-, j + (p-1)k],$$

then computes the products

$$u\boldsymbol{H}^l_{t,kp}(u)[-, j], \ldots, u\boldsymbol{H}^l_{t,kp}(u)[-, j + (p-1)k],$$

and finally stores the corresponding output bits into the shared location v. The shared variable *Var* ensures that, at any time, each processor either works on the current vector-matrix product or stays idle. In other words, we do not allow the processors to work on different vector-matrix products. In line 20, each processor p_j copies (concurrently) the current output vector v into its local memory, since at the next iteration v becomes the new input vector. When the main loop

Input: Two binary row vectors u, v of size kp each, and the set of matrices $\{G^i_{t,j}(b) \mid i \in \{1,\ldots,q\}, j \in \{0,\ldots,m\}, b \in \{0,1\}\}$ stored in the global memory; also, consider a shared variable *Var* initially set to 0.
Output: The product $u\boldsymbol{H}^1_{t,kp}(v_0)\boldsymbol{H}^2_{t,kp}(v_1)\ldots\boldsymbol{H}^q_{t,kp}(v_{q-1})$ stored in the shared location v.

```
1:  X_j ← u   {concurrent READ}
2:  For each i_1 ∈ {1,2,...,q} do
3:     For each i_2 ∈ {0,1,...,m} do
4:        For each i_3 ∈ {0,1} do
5:           Local^j_{i_1,i_2,i_3} ← G^{i_1}_{t,i_2}(i_3)   {concurrent READ}
6:  For l ← 1 to q do
7:     For i ← 1 to p do
8:        Compute the (j + (i − 1)k)-th column of H^l_{t,kp}(X_j)
9:        using the matrices Local^j_{i_1,i_2,i_3} and store it in the local
10:       variable Y_j.
11:       Z_j ← X_j Y_j
12:       v[j + (i − 1)k] ← Z_j[1]
13:    Endfor
14:    Var ← Var + 1   {concurrent WRITE}
15:    If Var = k then   {concurrent READ}
16:       Var ← 0
17:    Else
18:       WAIT until Var = 0   {concurrent READ}
19:    Endif
20:    X_j ← v   {concurrent READ}
21: Endfor
```

Figure 4: Asynchronous parallel cascaded convolutional encryption with propagation

(starting at line 6) is finished, the product $u\boldsymbol{H}^1_{t,kp}(v_0)\ldots\boldsymbol{H}^q_{t,kp}(v_{q-1})$ lies in the shared location v. A complete description of the parallel encryption algorithm (for processor p_j) is provided in Fig. 4. The parallel decryption algorithm is esentially the same sequence of operations, but in reverse order.

Lines 1,6,14,15,18, and 20 are executed *concurrently*, whereas the other lines are executed independently by each processor. Let us denote by W_1 the *maximum* amount of time spent by each processor to read (concurrently) the vector u (line 1, and also line 20 for v), by W_2 the amount of time spent by each processor to read the matrices $G^i_{t,j}(b)$ from the global memory (lines 2,3,4,5), by W_3 the *maximum* amount of time spent by each processor to execute (concurrently) the lines 14–19 at the current iteration, and by T_{seq} the runtime of the sequential algorithm (esentially, the time spent to compute equation (3)). Then, the parallel running time is at most

$$W_1 + W_2 + q(T_{seq}/qk + W_3 + W_1),$$

since each processor spends at most W_1 time to execute line 1, at most W_2 time to execute the lines 2–5, and at each of the q iterations starting at line 6, each processor spends exactly T_{seq}/qk time to execute the lines 7–13, at most W_3 time to execute the lines 14 through 19, and at most W_1 time to execute line 20.

4 Security and performance

The desired security level can be obtained by just setting appropriate values for the parameters of the convolutional cryptosystem. More precisely, we have $2q(m+1)+3$ parameters: k, m, q, $G^1_{t,0}(0)$,$G^1_{t,0}(1)$,...,$G^1_{t,m}(0)$,$G^1_{t,m}(1)$,...,$G^q_{t,0}(0)$,$G^q_{t,0}(1)$, ...,$G^q_{t,m}(0)$,$G^q_{t,m}(1)$. The highest level of security is obtained when all the parameters are kept secret, since this increases the complexity of any cryptanalytic attack. If the input vector has length pk, and if $d(pk)$ denotes the number of

Table 1: Comparisons between a cascaded convolutional cryptosystem with propagation (CCCP) and six well-known AES implementations

	Method	Encryption (bytes/second)	Decryption (bytes/second)
Intel Xeon CPU @ 2.8GHz	CCCP	1,521,737,039	1,501,954,457
	devine	83,886,080	78,033,563
	gladman	101,680,097	95,869,806
	gnupg	45,343,827	44,150,568
	libtc	98,689,506	101,264,158
	mkshen	69,905,067	68,478,433
	openssl	108,240,103	39,016,781
	Method	Encryption (bytes/second)	Decryption (bytes/second)
Intel Pentium 4 CPU @ 2.4GHz	CCCP	1,116,326,531	1,051,579,592
	devine	93,206,756	101,680,097
	gladman	50,840,048	55,924,053
	gnupg	37,701,609	37,282,702
	libtc	74,565,404	90,687,654
	mkshen	52,428,800	59,918,629
	openssl	98,689,506	95,869,806

divisors of pk that are less than or equal to k, then the maximum number of decoding attempts is

$$\sum_{i=1}^{d(pk)} \sum_{j=1}^{m} \sum_{l=1}^{q} [2^{i^2 2l(j+1)+1}],$$

since the maximum number of trials for k is $d(pk)$ and we have $2q(m+1)$ binary matrices of size k^2 each. Note that for each of the $2^{i^2 2l(j+1)}$ maximum trials for fixed values of i, j, and l, we have maximum two decoding attempts: one attempt for the case in which the number of 1's in the first input block is even, and another attempt for the case in which the number of 1's in the first input block is odd.

Regarding the runtime, we have made several comparisons between a globally invertible $(16, 16, 1)$ 2-cascaded convolutional transducer t with propagation and the well-known AES Encryption Benchmark [2], which consists of six AES implementations (some of them have been certified by the National Institute of Standards and Technology [9]): devine, gladman, gnupg, libtc, mkshen, and openssl. The runtimes shown in Table 1 have been obtained using a single Intel Xeon CPU on a 2.8GHz Linux server, and then a single Intel Pentium CPU on a 2.4GHz Linux server; the implementation of the cascaded convolutional cryptosystem has been compiled with gcc, v. 3.3.5. Regarding the cascaded convolutional transducer t, we have to mention that $k = 16$, $m = 1$, and $q = 2$. If these three parameters are kept public, then for an input vector of length 64 the maximum number of decoding attempts for our cryptosystem is $2^{[d(pk)]^2 2q(m+1)+1} = 2^{393}$. The six AES implementations have been tested with keys of 256 bits. Thus, our cryptosystem is much more complex than standard AES implementations, and also much faster. The parallel version (as described in Fig. 4) of our cryptosystem has been tested on the same platforms mentioned above, using 2 (and then 4) processors. As one can see, the runtime of the parallel version is reduced approximately x times, where x is the number of processors used. Thus, we conclude that our cryptosystems are faster and much more complex than standard AES implementations. Unlike *static* cryptosystems like DES, 3DES, and AES, our cryptosystems are *dynamic*, which means that standard cryptanalytic attacks such as linear and differential cryptanalysis (which have been developed in the context of *static* cryptosystems) are almost impossible to apply in this case.

Table 2: Parallel cascaded convolutional encryption/decryption with propagation: runtimes obtained using 2 (respectively 4) CPUs of a multiprocessor Linux server

No. of Intel Xeon CPUs @ 2.8GHz	Parallel CCCP Encryption (bytes/second)	Parallel CCCP Decryption (bytes/second)
2	2,602,170,337	2,493,244,399
4	4,839,123,784	4,731,156,540
No. of Intel Pentium 4 CPUs @ 2.4GHz	**Parallel CCCP Encryption (bytes/second)**	**Parallel CCCP Decryption (bytes/second)**
2	1,913,076,391	1,866,841,471
4	3,715,240,974	3,573,255,179

5 Conclusions and future work

We have proposed symmetric encryption schemes based on special classes of globally invertible cascaded convolutional transducers. The proposed encryption techniques have at least four advantages over traditional schemes based on Feistel ciphers. First, the secret key of a convolutional cryptosystem is usually much more easier to generate (just generate the matrices $G^i_{t,j}(b)$ such that the cascaded encoder is globally invertible whatever its structure would be at the current step). Second, the encryption and decryption procedures are much simpler, and consequentially, much faster. Third, the desired security level can be obtained by just setting appropriate values for the parameters of the convolutional cryptosystem. Finally, they are much more parallelizable than symmetric encryption standards based on Feistel ciphers. There are a lot of interesting research directions which can be exploited further, and we mention three of them. First, we plan to develop special cryptanalytic methods for *dynamic* cryptosystems. Second, it is interesting to continue the study of sophisticated convolutional cryptosystems obtained by combining different classes of globally invertible convolutional transducers. Third, we plan to explore some classes of cascaded convolutional transducers augmented with error detection capabilities.

References

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

[2] AES Encryption Benchmark: `http://www.cr0. net:8040/code/crypto/aesbench/`

[3] Biham, E., Anderson, R.J., Knudsen, L.R.: Serpent: A New Block Cipher Proposal. In: Proceedings of the 5th International Workshop on Fast Software Encryption, Paris, France, LNCS 1372, pages 222–238. Springer-Verlag

[4] Courtois, N., Pieprzyk, J.: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In: Proceedings of Asiacrypt 2002, LNCS 2501, pages 267–287. Springer-Verlag

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

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

[7] Escrowed Encryption Standard: `http://csrc.nist.gov/publications/fips/fips185/fips185.txt`

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

[9] National Institute of Standards and Technology: `http://www.nist.gov/`

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

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

[12] 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)