On Cooperatively Distributed Ciphering and Hashing

Cunsheng Ding
Arto Salomaa

Turku Centre for Computer Science

TUCS Technical Reports

No 13, May 1996

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.abo.fi

May 1996

ISBN 951-650-755-7
ISSN 1239-1891

Abstract

In this paper we first bridge some problems of formal language theory to those of cryptography. In doing so, new problems in both fields are proposed. Motivated by the CD grammar systems [1], we then describe a general cooperatively distributed ciphering system and hashing system. These CD ciphering and hashing systems could be much more powerful than conventional ones if they are properly designed.

Keywords: Formal languages, grammar systems, ciphers, hashing, authentication.

TUCS Research Group
"Mathematical Structures of Computer Science"

1 Introduction

At a first look cryptography and formal languages seem to be quite different. Yet they are related. Formal languages originate from natural and programming languages, and ciphers have been built over human languages for thousands of years. The root of these two fields is natural language. The root will serve as a bridge between the two fields. It is interesting to note that cryptographic transformations of block ciphers correspond to language morphisms, but those of stream ciphers are not language morphisms. This suggests the study of new types of language morphisms, as outlined in this paper. Hashing is an important topic of cryptography, and one of the main issues of cryptography is to find secure hash functions. It is interesting to see that a hash function is actually a mapping from a formal language to another one. And authentication codes are the images of language morphisms. Since the frequency distributions of letters are important in cryptography, it could be interesting to study formal languages over alphabets that have a probability assignment, if the symbols of formal languages correspond to the words of natural languages.

It often happens that an idea in one field leads to an idea in another field. The idea of cooperation and distribution in grammar systems has already been implemented [1]. We will show that this idea is also quite useful in cryptography.

In this paper we first bridge some relations between formal languages and cryptography. Motivated by the CD grammar systems, we then describe a cooperatively distributed ciphering and hashing system. We will show that the CD ciphering and CD hashing systems could be much more powerful if they are properly designed.

2 Some Relations between the two Fields

We begin with some basics of formal languages. An *alphabet* Σ is a nonempty finite set of symbols, for example, $\Sigma = \{0, 1\}$. A *word* w over an alphabet is a sequence of symbols over the alphabet. For an alphabet Σ we use Σ^* (resp. Σ^+) to denote the set of all finite (resp. finite nonempty) words over Σ. A *formal language* over Σ is a subset of Σ^*. An important operation for words is the *catenation*. Let $w_1 = a_1 a_2 \cdots a_m$ and $w_2 = b_1 b_2 \cdots b_n$ be two words over an alphabet Σ, the catenation of w_1 and w_2, denoted by $w_1 \cdot w_2$ or $w_1 w_2$, is defined to be the word $a_1 a_2 \cdots a_m b_1 b_2 \cdots b_n$. Thus, $(\Sigma^*, \cdot)$ (resp. $(\Sigma^+, \cdot)$) is a monoid (resp. semigroup). Let Σ and Δ be two alphabets. A mapping h from Σ^* to Δ^* is called a *morphism* if $h(uv) = h(u)h(v)$ holds for any $u, v \in \Sigma^*$.

2.1 Between Formal Languages and Block Ciphers

A *message* or *plaintext* m over an alphabet Σ is a sequence of symbols from the alphabet. A *cipher system* is a construct $(\Sigma, \mathbf{P}, \Delta, \mathbf{C}, \mathbf{K}, e_k, \mathbf{K}', d_{k'})$, where

1. Σ is the plaintext alphabet, and $\mathbf{P}$ is the *plaintext space* which is a subset of Σ^*;

2. Δ is the ciphertext alphabet, and $\mathbf{C}$ is the *ciphertext space* which is a subset of Δ^*;

3. $\mathbf{K}$ is the encryption key space, and e_k is a bijection from $\mathbf{P}$ to $\mathbf{C}$ specified by an encryption key $k \in \mathbf{K}$, which is called the encryption transformation; and

4. $\mathbf{K}'$ is the decryption key space, and $d_{k'}$ is a bijection from $\mathbf{C}$ to $\mathbf{P}$, specified by a decryption key $k' \in \mathbf{K}'$, which is the inverse of some e_k and is called the decryption transformation.

In a cipher system the plaintext space is usually associated with an a priori probability assignment $\{p(m)|m \in \mathbf{P}\}$, and the encryption key space $\mathbf{K}$ with a probability assignment $\{p(k)|k \in \mathbf{K}\}$. In a cipher system there may exist a number of decryption keys k' with respect to an encryption key k. If any encryption key k is a decryption key with respect to itself, the cipher system is called a *one-key* or *private-key* cryptosystem, and a *two-key* or *public-key* cryptosystem otherwise. In a cipher system a plaintext m is encrypted as $c = e_k(m)$, and decryption is done as $m = d_{k'}(c) = d_{k'}(e_k(m))$.

Ciphers are generally classified into two classes: block and stream ciphers. A block cipher breaks a message $m \in \mathbf{P}$ into successive blocks $m = m_1 m_2 \cdots m_l$, where each m_i consists of a fixed number n of symbols from Σ, and the encryption and decryption are carried out as

$$\begin{aligned} c &= e_k(m) = e_k(m_1)e_k(m_2)\cdots e_k(m_l) = c_1 c_2 \cdots c_l, \\ m &= d_{k'}(m) = d_{k'}(c_1)d_{k'}(c_2)\cdots d_{k'}(c_l) = m_1 m_2 \cdots m_l, \end{aligned}$$

where $c_i = e_k(m_i)$ for each i which is a word of a fixed length. The above parameter n is called the *block length* of the block cipher.

The relation between formal languages and block ciphers is described by the following proposition.

Proposition 1 *Let* $(\Sigma, \mathbf{P}, \Delta, \mathbf{C}, \mathbf{K}, e_k, \mathbf{K}', d_{k'})$ *be a block cipher with plaintext and ciphertext block lengths* n_1 *and* n_2 *respectively. Then each encryption (resp. decryption) transformation is a morphism from* $(\Sigma^{n_1})^*$ *(resp.* $(\Delta^{n_2})^*$*) to* $(\Delta^{n_2})^*$ *(resp.* $(\Sigma^{n_1})^*$*). Conversely, an injective morphism from* $(\Sigma^{n_1})^*$ *to* $(\Delta^{n_2})^*$ *can be viewed as an encryption transformation of a block cipher.*

Proof: Let u and v be two plaintexts. By assumption the two plaintexts can be written as $u = u_1u_2 \cdots u_{l_1}$ and $v = v_1v_2 \cdots v_{l_2}$, where u_i and v_i are blocks of length n_1. Thus, $uv = u_1u_2 \cdots u_{l_1}v_1v_2 \cdots v_{l_2}$, and by the definition of block ciphers

$$e_k(uv) = e_k(u_1) \cdots e_k(u_{l_1})e_k(v_1) \cdots e_k(v_{l_2}) = e_k(u)e_k(v).$$

It follows that each e_k is a language morphism on $(\Sigma^{n_1})^*$. The inverse mapping is similarly proved to be a language morphism. The remaining part of the proposition is straightforward. $\square$

Given a plaintext space $\mathbf{P}$ over a plaintext alphabet Σ and a ciphertext space $\mathbf{C}$ over a ciphertext alphabet Δ, the design of a private-key block cipher of block length n over these spaces consists of the choices of a key space $\mathbf{K}$ and a class of invertible language morphisms e_k indexed by $k \in \mathbf{K}$ from $\mathbf{P}$ to $\mathbf{C}$ such that

1. there is a class of morphisms d_k from $\mathbf{C}$ to $\mathbf{P}$ such that the d_k is the inverse of e_k for each possible $k \in \mathbf{K}$;

2. it is "easy" to compute both $e_k(m)$ and $d_k(c)$ for all $k \in \mathbf{K}$, $m \in \mathbf{P}$, and $c \in \mathbf{C}$;

3. Given a number of plaintext-ciphertext pairs, it is both "hard" to recover the key k or an equivalent key k^*, by which we mean that $d_k = d_{k^*}$.

Given a plaintext space $\mathbf{P}$ over a plaintext alphabet Σ and a ciphertext space $\mathbf{C}$ over a ciphertext alphabet Δ, the design of a public-key block cipher of block length n over these spaces consists of the choices of an encryption key space $\mathbf{K}$, a decryption key space $\mathbf{K}'$, and a class of invertible language morphisms e_k indexed by $k \in \mathbf{K}$ from $\mathbf{P}$ to $\mathbf{C}$ such that

1. there is a class of morphisms $d_{k'}$ from $\mathbf{C}$ to $\mathbf{P}$ such that for each possible $k \in \mathbf{K}$, d_k is not the inverse of e_k if $k \in \mathbf{K}'$ and there is a decryption key k' such that $d_{k'}$ is the inverse of e_k;

2. there is a trapdoor such that given the encryption k and the encryption algorithm together with the trapdoor, it is "easy" to compute a decryption key, but "hard" to do so without knowing the trapdoor;

3. it is "easy" to compute both $e_k(m)$ and $d_{k'}(c)$ for all possible $k \in \mathbf{K}$, $k' \in \mathbf{K}$, $m \in \mathbf{P}$, and $c \in \mathbf{C}$.

Thus, the design and analysis of both private-key and public-key block ciphers are those of a class of indexed invertible morphisms from a formal language to another one such that the mentioned properties hold.

Language morphisms are *memoryless* in the sense that for any word $w = w_1 w_2 \cdots w_l$ we have by definition that $h(w) = h(w_1)h(w_2)\cdots h(w_l)$, and thus the part $h(w_i)$ is independent of its previous parts $h(w_1), \cdots, h(w_{l-1})$. Thus, block ciphers are memoryless and time-invariable. By time-invariability we mean that a morphism or encryption transformation is always the same regardless of time. Time-invariable and memoryless language morphisms are only useful in designing block (both private-key and public-key) ciphers, but not in designing stream ciphers.

2.2 Between Formal Languages and Stream Ciphers

We have just described time-invariable and memoryless language morphisms and showed their relations with both private-key and public-key ciphers. We now introduce time-varying language morphisms and language morphisms with memory. Then we show their applications in cryptography.

Let $\mathbf{T}$ be the time space. A class of mappings h_t from Σ^* to Δ^* indexed by the time variable t satisfying

$$h_t(uv) = h_t(u)h_{t+|u|}(v) \quad \text{for all } u, v \in \mathbf{P},$$

is called a family of *linear time-varying morphisms*, where $|u|$ denotes the length of u, and where $t \in \mathbf{T}$. By linearity we mean that the index $t + |u|$ depends on both t and $|u|$ linearly. This kind of morphism families is only dependent on the time memory.

It is well known that many of the families investigated in language theory are closed under morphisms and inverse morphisms. In case of linear time-varying morphisms the closure depends on how the index t influences the morphism h_t and on specialities of the $\mathbf{T}$. A time space $\mathbf{T}$ is called *finite* if has only a finite number of elements. For instance, the set of the days of a year is a finite time space. To study the closure problem of formal languages under linear time-varying morphisms, it may be useful to classify time spaces into the two classes.

A synchronous stream cipher breaks a message $m \in \mathbf{P}$ into successive characters $m = m_1 m_2 \cdots m_l$, where each character could be a symbol of the alphabet Σ or a fixed number of symbols from Σ. To encrypt the message, an encryption keystream sequence $k_1 k_2 \cdots k_l$ from an encryption key $k \in \mathbf{K}$ is generated, and the encryption is carried out as

$$c = e_k(m) = e_{k_1}(m_1) e_{k_2}(m_2) \cdots e_{k_l}(m_l) = c_1 c_2 \cdots c_l,$$

where each e_{k_i} is invertible, and $c_i = e_{k_i}(m_i)$. To decrypt the ciphertext c, with a decryption key $k' \in \mathbf{K}'$ a decryption keystream $k'_1 k'_2 \cdots k'_l$ is produced, and the decryption is:

$$m = d_{k'}(c) = d_{k'_1}(c_1) d_{k'_2}(c_2) \cdots d_{k'_l}(c_l).$$

Similarly, we have private-key and public-key synchronous stream ciphers.

It is not hard to see that linear time-varying language morphisms can be used to design private-key synchronous stream ciphers. Let $\mathbf{P}$ and $\mathbf{C}$ be the plaintext space and ciphertext space over alphabets Σ and Δ respectively, $\mathbf{T}$ be a time space. We assume that Σ and Δ have the same number of elements. Let e_t be a linear time-varying morphism with respect to time space $\mathbf{T}$, and d_t be the inverse of e_t which is a linear time-varying morphism from $\mathbf{C}$ to $\mathbf{P}$. Assume that each e_t maps a symbol of Σ into a symbol of Δ. Taking $\mathbf{T}$ as the (both encryption and decryption) key space, we have a cipher system in which the encryption and decryption are carried out as follows. Any message $m \in \mathbf{P}$ is first encrypted as

$$c = e_t(m) = e_{t+1}(m_1) e_{t+2}(m_2) \cdots = c_1 c_2 \cdots,$$

where each m_i is a symbol of Σ and $c_i = e_{t+i}(m_i)$. The decryption is then done as

$$m = d_t(c) = d_{t+1}(c_1) d_{t+2}(c_2) \cdots,$$

where d_t is the inverse of e_t for each t. Similarly, nonlinear time-varying language morphism may be employed to design both private-key and public-key synchronous stream ciphers.

One motivation behind the study of language morphisms could be the translation of a language into another one. Language translators have some similar properties with language morphisms, but they have local memory. Let l be a translator from an natural language to another one. For two complete sentences u and v

$$l(uv) = l(u) l(v).$$

But this could be false if u and v are two words. Thus, natural language translators could have local memory, but they may allow the translation of complete sentences in parallel. A language translator may be regarded as a language morphism with local memory.

In a self-synchronous stream cipher, each keystream character is derived from a fixed number n of preceding cipher characters. Thus, the keystream generator of a self-synchronous stream cipher is a language morphism with memory.

3 Cooperatively Distributed Cryptosystems

A general purpose of cooperation and distribution is to increase the "power" of each individual component. However, the actual aim of cooperation and distribution differs from system to system. In designing a cooperatively distributed (CD) system the problems concerning the choices of the components and the design of the protocol for cooperation and distribution are fundamental. It is important to choose "good partners" for a CD system, but very hard to know which of the possible components are "good" partners. Naturally the definition for "a good partner" differs from context to context, and is often difficult to make precise.

It is possible to increase the power of the system by choosing more components (this is not always the case), but more components will lead to a larger complexity of the protocol for coordinating the cooperation and distribution.

Motivated by the power of cooperative distributed grammar systems [1], we first describe a cooperatively distributed (CD) hash system, then we describe a cooperative (CD) ciphering system.

3.1 A Cooperatively Distributed Hashing System

A hash function h hashes a message X of an arbitrary length into $h(X)$ of a fixed length. Let h_1 and h_2 be two hash functions. We now describe a cooperatively distributed (CD) hash system based on two hash functions and a binary sequence generator (SG).

Let S^∞ denote the semi-infinite output sequence of the SG. In the first step of the CD hash system, messages are divided into blocks and then distributed to the two hash functions. To this end, we choose a distribution block length, say of d bits. Then a message M in bits is divided into blocks as

$$M = M_{l-1}M_{l-2}...M_1M_0,$$

where the last block is extended into a block of length d by some padding procedure if the number of bits of M is not a multiple of d. Consecutive

l bits of the control sequence, say $s_{l-1}s_{l-2}...s_1s_0$, are used to distribute the message blocks. Let

$$I_1 := \{i : s_i = 1, 0 \leq i \leq l-1\} = \{i_1 < i_2 < ... < i_t\}$$
$$\{j_1 < j_2 < ... < j_{l-t}\} := \{0, 1, ..., l-1\} \setminus I_1.$$

Then the message $M_{i_t}M_{i_{t-1}} \cdots M_{i_2}M_{i_1}$ is distributed to the first hash function as the input message, and $M_{j_{l-t}}M_{j_{l-t-1}}...M_{j_2}M_{j_1}$ to the second hash function as the input message. The hash value of the CD hash system is the concatenation of the two hashed values, that is,

$$h(M) = h_2(M_{j_{l-t}}M_{j_{l-t-1}}...M_{j_2}M_{j_1})h_1(M_{i_t}M_{i_{t-1}} \cdots M_{i_2}M_{i_1}).$$

The cryptographic checksum algorithm based on stream ciphers proposed in [4] can be viewed as a special case of this CD hash system. There are several important features in the CD hash system. One is that the CD hash system works on blocks of messages. Another is that complete hash functions are employed as subhash functions in the CD hash system.

Cryptographic hash functions are divided into two classes: message authentication code (MAC), where a secret key is used, and the manipulation detection code (MDC), where there is no secret parameter. The above CD hash system is suggested for constructing MACs. For an MAC the following attacks have been distinguished [6]: known plaintext attack, where the attacker is able to examine some plaintexts and their corresponding MAC; chosen plaintext attack, where an attacker can select a set of plaintexts, and subsequently obtain the corresponding MACs; adaptive chosen plaintext attack, where an attacker can choose a plaintext and immediately receive the corresponding MAC, and the choice of the next plaintext can depend on the outcome of previous questions.

Whenever the CD hash system is used for constructing an MAC, the secret parameter of the control SG should be part of the key of the MAC. In this case, known plaintext attack, chosen plaintext attack and adaptive chosen plaintext attack on the CD MAC do not lead to the same attack for the underlying subhash functions, since the actual input to each underlying subhash function is unknown due to the secret control sequence. Thus, even if the underlying subhash functions have weakness with respect to these attacks, the CD MAC could still be secure against these attacks, provided that the control SG is well designed. Thus, the CD hash system is suitable for constructing MACs. In this case the power of cooperation and distribution can be fully demonstrated.

Many proposed hash functions belong to the so-called iterative hash functions or hash functions with compression functions [6]. With such a hash function the message sequence is divided into t n-bit blocks $M_1, ..., M_t$. If the total number of bits is not a multiple of n, a padding procedure has to be specified [6].

An iterated hash function h with the round function f is described by

$$\begin{aligned}
&H_0 = IV, \\
&H_i = f(M_i, H_{i-1}) \quad i = 1, 2, \cdots, t, \\
&h(X) = H_t,
\end{aligned}$$

where H_i are intermediate variables ranging over blocks of m, M_i are n-bit blocks, IV is the initial block of length m, and $h(x)$ is the hash value. Thus, the round function is a mapping from $GF(2)^n \times GF(2)^m$ to $GF(2)^m$.

If we use two iterative hash functions h_1 and h_2 with two round functions

$$\begin{aligned}
&f_1 : GF(2)^n \times GF(2)^{m_1} \to GF(2)^{m_1}, \\
&f_2 : GF(2)^n \times GF(2)^{m_2} \to GF(2)^{m_2},
\end{aligned}$$

and choose the distribution block length $d = n$ in the CD hashing system, we have a special iterative CD hash system.

3.2 A Cooperative Distributed Ciphering System

There are advantages and disadvantages in both block and stream ciphering. Additive synchronous stream ciphers have the disadvantage that a ciphertext-plaintext character pair gives out immediately the corresponding keystream character under which the plaintext character is encrypted. This makes possible various kinds of key-recovering attacks such as correlation attacks and collision attacks, equivalent-machine attacks such as the attack based on the Berlekamp-Massey algorithm, approximate-machine attacks such as attacks based on linear approximations. One of their advantages is that the keystream is time-varying, which ensures that the same plaintext character usually corresponds to different ciphertext characters. This usually conceals some statistical properties of the plaintext. Block ciphers used in Electronic Codebook Mode (ECB) have the disadvantage that their keys cannot be changed very frequently due to the problem of key management. In addition the same block of a message corresponds always to the same ciphertext block if one key is selected and fixed. This may make many attacks such as differential attacks on a block cipher applicable. One of their advantages is that the detection of the modification of messages may be possible because messages are encrypted block by block.

Our motivation for investigating CD cipher systems stems from the CD grammar systems, whose power has been demonstrated in [1]. We now describe a CD ciphering system and show that it could be much more powerful than both conventional block cipher and additive stream cipher systems in rendering attacks. This demonstrates the power of cooperation and distribution in cipher systems.

Our cooperatively distributed cipher system consists of s components: s conventional block ciphers of the same block length, and a control device which is a sequence generator with internal memory, SG for short, which produces sequences over the alphabet $Z_s = \{0, 1, \cdots, s-1\}$.

Let $k_0, \cdots, k_{s-1}$ be the keys respectively; $E_0(k_0, \cdot), \cdots, E_{s-1}(k_{s-1}, \cdot)$ the encryption transformations specified by the keys; $D_0(k_0, \cdot), \cdots, D_{s-1}(k_{s-1}, \cdot)$ the decryption transformations specified by the keys respectively. Let k_{sg} be the key of the sequence generator, z_i be the output sequence of the SG at time i. The key of the CD cipher system is $k = (k_{sg}, k_0, \cdots, k_{s-1})$. At each time unit only one of the block ciphers is active, i.e., doing the encryption (respectively decryption). So we have

$$c_i = E_{z_i}(k_{z_i}, m_i),$$

where m_i and c_i are the ith plaintext block and ciphertext block. Similarly, the decryption is defined by

$$m_i = D_{z_i}(k_{z_i}, c_i).$$

In our CD cipher system the SG determines the action of each component block cipher.

In the CD ciphering system it is possible for the encryption algorithms $E_0, \cdots, E_{s-1}$ to be the same, but in this case the keys $k_0, k_1, \cdots, k_{s-1}$ should be pairwise different.

Among the four modes for block ciphers the Cipher Feedback Chaining (CFB) mode and the Output Feedback Chaining (OFB) mode are stream ones (see [6]). The former is a self-synchronous additive stream ciphering approach, and the latter is a synchronous additive one. The distinguishing feature between the CD ciphering system and these two modes of block ciphers are that in the two modes block ciphers are used to generate keystreams, while in the CD ciphering system several block ciphers are used alternatively to encrypt plaintext blocks. While the two modes are additive stream ones, the CD ciphering system is usually non-additive.

The security of the system can be supported as follows. First we consider attacks on block ciphers. All the attacks on block ciphers are done under the assumption that the key is fixed and there is only one encryption (respectively decryption) algorithm. Among such attacks are differential attacks and linear attacks. All of those attacks don't apply to our CD cipher system, since we have at least two different encryption (resp. decryption) algorithms or at least two different keys for the underlying block ciphers.

Second, though there are a number of attacks on stream ciphers, most of them apply only to additive ones, and consequently to those keystream generators for additive stream ciphers. If our CD cipher system is designed properly, those attacks don't apply either.

Our CD cipher system is a stream ciphering one, though it is a combination of block and stream ciphers, since a message usually corresponds to different ciphertexts at different times. The purpose of cooperation and distribution is to make infeasible as many known attacks on both block and additive stream ciphers as possible. Given a peace of ciphertext, it is usually difficult for the enemy to know how many times a component block cipher has contributed and where it has contributed.

If the system is designed properly, it is possible to get a very strong cipher by choosing some very weak block ciphers and a weak sequence generator. This shows again the power of cooperation and distribution.

It is easy to see that arbitrarily choosing some components and putting them together does not usually result in a more powerful CD system. This is also the case for CD cipher systems. The components and the control device should be chosen properly. In what follows we consider the system consisting of two component block ciphers.

Let K_0 and K_1 be the key spaces of the two block ciphers respectively. Assume that each key of K_0 (resp. K_1) is equally likely. Let $p_0 = p(z = 0)$, $p_1 = p(z = 1)$ and

$$n_i(m, c) = |\{k_i \in K_i | E_i(k_i, m) = c\}|, \ i = 0, 1.$$

Also Let $p(m, c)$ denote the probability that c is a corresponding ciphertext block of the plaintext block m. Then it is not difficult to see

$$p(m, c) = p_0 \frac{n_0(m, c)}{|K_0|} + p_1 \frac{n_1(m, c)}{|K_1|},$$
$$p(z = i; (m, c)) = p_i \frac{n_i(m, c)}{|K_0|}, \ i = 0, 1.$$

It follows that

$$p(z = 0 | (m, c)) = \frac{|K_1| p_0 n_0(m, c)}{|K_1| p_0 n_0(m, c) + |K_0| p_1 n_1(m, c)}$$
$$p(z = 1 | (m, c)) = \frac{|K_0| p_1 n_1(m, c)}{|K_1| p_0 n_0(m, c) + |K_0| p_1 n_1(m, c)}$$

Hence, we have got the following expression for the average mutual information

$$\begin{aligned} I(z;(m,c)) \;=\; & -\frac{|K_1|p_0 n_0(m,c)}{|K_1|p_0 n_0(m,c)+|K_0|p_1 n_1(m,c)} \log \frac{|K_1|p_0 n_0(m,c)}{|K_1|p_0 n_0(m,c)+|K_0|p_1 n_1(m,c)} \\ & -\frac{|K_0|p_1 n_1(m,c)}{|K_1|p_0 n_0(m,c)+|K_0|p_1 n_1(m,c)} \log \frac{|K_0|p_1 n_1(m,c)}{|K_1|p_0 n_0(m,c)+|K_0|p_1 n_1(m,c)}. \end{aligned}$$

To minimize the above average mutual information, we have to ensure that

$$p_0 \frac{n_0(m,c)}{|K_0|} = p_1 \frac{n_1(m,c)}{|K_1|}. \tag{1}$$

Note that

$$\sum_{c \in C} \frac{n_0(m,c)}{|K_0|} = \sum_{c \in C} \frac{n_1(m,c)}{|K_1|} = 1.$$

It follows that

$$p_0 = \sum_{c \in C} p_0 \frac{n_0(m,c)}{|K_0|} = \sum_{c \in C} p_1 \frac{n_1(m,c)}{|K_1|} = p_1.$$

Hence, we have $p_0 = p_1 = 1/2$, and furthermore

$$\frac{n_0(m,c)}{|K_0|} = \frac{n_1(m,c)}{|K_1|}. \tag{2}$$

Based on the above analysis, we have obtained the following design principle:

Design Principle 1 *For our CD cipher system with two component block ciphers, the parameters should be chosen such that*

1. $p_0 \approx \frac{1}{2}$;
2. $\frac{n_0(m,c)}{|K_0|} \approx \frac{n_1(m,c)}{|K_1|}$, *and if one of* $n_0(m,c)$ *and* $n_1(m,c)$ *is zero, so must be the other.*

Apparently, a cipher is secure against ciphertext-only attacks if it is secure against known plaintext attacks. Given a peace of plaintext-ciphertext block pairs, a cryptanalyst may first try to get a peace of keystream and then try to recover the key of the SG or to construct a generator which produces the same control sequence, by analyzing the parameters $n_0(m,c)$ and $n_1(m,c)$ of the two block ciphers for the given plaintext-ciphertext pairs. If the two block ciphers are not well designed, and the cryptanalyst gets to know $n_0(m, c) = 0$, then he/she knows immediately that the control digit under which a block cipher is selected is 1. If an attack on the SG is successful, then it remains only to attack the two block ciphers in the usual sense. At this stage the meaning of cooperation is lost. The above design principle is for making infeasible this kind of divide-and-conquer attacks.

On the other hand, the SG should be designed so that its output sequences have good pattern distributions. If the control sequence is $111 \cdots 1000 \cdots 0$, then the cooperation is obviously very bad. Whether it is necessary to require the output sequence of the SG to have a large linear complexity and ideal linear complexity stability depends on the strength of the underlying two block ciphers. If the two block ciphers are properly designed, it is unnecessary to consider the linear complexity aspect.

The designing strength of the block ciphers depends also on that of the SG. If the SG is well designed, some weak block ciphers can also be employed. It is important that the two block ciphers should have many similarities, just like "twins". This indicates that using only a two-key cooperation within one well designed algorithm seems better, but in this approach one has to guarantee that the two keys do not specify the same encryption transformation; otherwise there is no cooperation within the system.

4 Concluding Remarks

We have described some relations between formal languages and cryptography. Some other relations and cryptosystems based on formal language theory can be found in [5, 9, 8, 10]. Deeper results usually exist: the topic has so far been little investigated. For details about formal languages we refer to [2, 7], and for details about cryptography we refer to [3, 10].

The CD ciphering and hash systems described in the paper are very general and unspecific. By choosing the underlying components it is possible to design various specific CD ciphers and CD hash functions, where specific requirements are needed.

ACKNOWLEDGEMENT: The authors would like to thank Gheorghe Păun for helpful discussions.

References

[1] CSUHAJ-VARJU, E.—DASSOW, J.—KELEMEN, J.—PĂUN, G.:

Grammar Systems: A Grammatical Approach to Distribution and Co-operation. London, Gordon and Breach Science Publishers, 1994.

[2] Dassow, J.—Păun, G.: Regulated Rewriting in Formal Language Theory. Heidelberg, Springer-Verlag, 1989.

[3] Ding, C.—Xiao, G.—Shan, W.: The Stability Theory of Stream Ciphers. Heidelberg, Springer-Verlag, 1991.

[4] Lai, X.—Reuppel, R. A.—Wollven, J.: A fast cryptographic checksum algorithm based on stream ciphers. In Advances in Cryptology, Proc. Auscrypt'92, LNCS 658, Springer-Verlag 1993, pp. 55-70.

[5] Niemi, V.: Cryptographic issues in language theory. In: G. Rozenberg and A. Salomaa (ed.) Handbook of Formal Languages. Springer-Verlag, in preparation.

[6] Preneel, B.: Cryptographic Hash Functions. Kluwer Academic Publishers, 1995.

[7] Salomaa, A.: Formal Languages. New York, Academic Press, 1973.

[8] Salomaa, A.: A public-key cryptosystem based on language theory. Computers and Security, Vol. 7, 1988, pp. 83-87.

[9] Salomaa, A.—Yu, S.: On a public-key cryptosystem based on iterated morphisms and substitutions. Theoretical Computer Science, Vol. 48, 1986, pp. 283-296.

[10] Salomaa, A.: Public-key Cryptography. New York, Berlin, Heidelberg, Springer-Verlag, 1990, pp. 166-177.

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.abo.fi