Chapter 3
Selected Design and Analysis Techniques for Contemporary Symmetric Encryption

Vasily Mikhalev, Miodrag J. Mihaljević, Orhun Kara, and Frederik Armknecht

Abstract In this chapter we provide an overview of selected methods for the design and analysis of symmetric encryption algorithms that have recently been published. We start by discussing the practical advantages, limitations and security of the keystream generators with keyed update functions which were proposed for reducing the area cost of stream ciphers. Then we present an approach to enhancing the security of certain encryption schemes by employing a universal homophonic coding and randomized encryption paradigm.

3.1 Introduction

The concept of ubiquitous computing brings new challenges to the designers of cryptographic algorithms by introducing scenarios where classical crypto primitives are infeasible due to their costs (such as hardware price, computational time, and power and energy consumption). In this chapter we provide an overview of selected recent approaches which deal with such challenges.

An approach [27] which allows one to realize secure stream ciphers with state size beyond the bound which was previously considered to be the minimum is summarized in Sect. 3.2. The main idea is to use so-called keystream generators with keyed update functions (KSGs with KUF) where the secret key is involved not only in the initialization phase (as is common practice) but during the entire encryption process. After explaining the advantages [27] of KSGs with KUF in resisting Time Memory Data Tradeoff (TMDTO) attacks [47, 237] together with

V. Mikhalev (✉) · F. Armknecht
University of Mannheim, Mannheim, Germany
e-mail: mikhalev@uni-mannheim.de

M. J. Mihaljević
Mathematical Institute, Serbian Academy of Sciences and Arts, Belgrade, Serbia

O. Kara
Department of Mathematics, IZTECH Izmir Institute of Technology, Izmir, Turkey

G. Avoine, J. Hernandez-Castro (eds.), *Security of Ubiquitous Computing Systems*,
https://doi.org/10.1007/978-3-030-10591-4_3

practical issues and limitations on their implementation in hardware[421], we describe the stream cipher Sprout which was designed in order to demonstrate the feasibility of the approach [27], and its improvement Plantlet where the security weaknesses of Sprout were fixed [421]. In Sect. 3.3 we present a generic attack [314] against such KSGs that implies a design criterion. Section 3.4 presents an approach to security enhancement of certain encryption schemes employing universal homophonic coding [397] and a randomized encryption paradigm [503]. The approach summarized in this section has been reported and discussed in a number of references including [413, 418, 452] and [420]. A security evaluation of this encryption scheme has been reported in [452] from an information-theoretic point of view, and a computational-complexity evaluation approach is given in [420].

3.2 Keystream Generators with Keyed Update Functions

3.2.1 Design Approach

Stream ciphers usually provide a higher throughput than block ciphers. However, due to the existence of certain TMDTO [47, 91, 237] attacks, the area size required to implement secure stream ciphers is often higher. The reason is the following. The effort of TMDTO attacks against stream ciphers is $O(2^{\sigma/2})$, where σ is the internal state size. Therefore, a rule of thumb says that to achieve κ-bit security level, the state size should be at least $\sigma = 2 \cdot \kappa$. This results in the fact that a stream cipher requires at least $2 \cdot \kappa$ memory gates which are the most costly hardware elements in terms of area and power-consumption. In this section we discuss an extension [27, 421] of the common design principle, which allows for secure lightweight stream ciphers with internal state size below this bound.

We start the description of the new approach for stream ciphers design by giving the definition of the KSG with KUF [27]:

Definition 1 (Keystream Generator with Keyed Update Function) A keystream generator with a *keyed update function* comprises three sets, namely the key space $\mathscr{K} = \mathrm{GF}(2)^{\kappa}$, the IV space $\mathscr{IV} = \mathrm{GF}(2)^{v}$, and the variable state space $\mathscr{S} = \mathrm{GF}(2)^{\sigma}$. Moreover, it uses the following three functions

- an initialization function $\mathsf{Init} : \mathscr{IV} \times \mathscr{K} \to \mathscr{S}$
- an update function $\mathsf{Upd} : \mathscr{K} \times \mathscr{S} \to \mathscr{S}$ such that $\mathsf{Upd}_k : \mathscr{S} \to \mathscr{S}$, $\mathsf{Upd}_k(st) := \mathsf{Upd}(k, st)$, is bijective for any $k \in \mathscr{K}$, and
- an output function $\mathsf{Out} : \mathscr{S} \to \mathrm{GF}(2)$.

The internal state ST is composed of a variable part $st \in \mathscr{S}$ and a fixed part $k \in \mathscr{K}$. A KSG operates in two phases. In the *initialization phase*, the KSG takes as input a secret key k and a public IV iv and sets the internal state to

$st_0 := \mathsf{Init}(iv, k) \in \mathscr{S}$. Afterwards, the keystream generation phase executes the following operations repeatedly (for $t \geq 0$):

1. Output the next keystream bit $z_t = \mathsf{Out}(st_t)$
2. Update the internal state st_t to $st_{t+1} := \mathsf{Upd}(k, st_t)$

The main difference between KSGs with KUF and the KSGs traditionally used as a core of stream ciphers is that the next state is now computed not only from the current variable state st_t (as is commonly done) but also from the fixed key k.

We now explain why stream ciphers built based on the KSGs with KUF have advantages in resisting TMDTO attacks over classical KSGs. The goal of the TMDTO attacker is the following: given a function $F : \mathscr{N} \to \mathscr{N}$ and D images $y_1, \ldots, y_D$ of F, find a preimage for any of these points, i.e., determine a value $x_i \in \mathscr{N}$ such that $F(x_i) = y_i$. Typically, these attacks consist of two phases: a precomputation (offline) phase, and a real-time (online) phase. At first the attacker precomputes a large table using the function F (offline phase). In the online phase the attacker gets D outputs of F and checks if any of these values is included in the precomputed table. In the case of success, a preimage has been found. Obviously, an attacker can increase the success probability by either precomputing more values in the offline phase or collecting more data in the online phase where the optimal trade-off is usually given as $|D| = \sqrt{|\mathscr{N}|}$.

The goal of a TMDTO attack in the context of KSGs is to recover one internal state as this allows us to compute the complete keystream. To this end, let $F_{\mathsf{Out}} : \mathrm{GF}(2)^\sigma \to \mathrm{GF}(2)^\sigma$ be the function that takes the internal state $st_t \in \mathrm{GF}(2)^\sigma$ at some clock t as input and outputs the σ keystream bits $z_t, \ldots, z_{t+\sigma-1}$. Then, the attack translates to finding a preimage of F_{Out} for a given keystream segment with the search space being $\mathscr{N} = \mathscr{S}$ and an effort of at least $\sqrt{|\mathscr{S}|} = 2^{\sigma/2}$. This implies the above-mentioned rule of selecting $\sigma \geq 2\kappa$.

To understand the motivation behind the design principle given in Definition 1, we introduce the notion of keystream-equivalent states which is important for analyzing the effectiveness of a TMDTO attack. Let $F^{\text{compl.}}_{\mathsf{Out}}$ be the function that takes as input the initial state and outputs the maximum number of keystream bits. If no bound is given by the designers, we assume that the maximum period of 2^σ keystream bits is produced. An attacker is interested in any internal state that allows the keystream to be computed:

Definition 2 (Keystream-Equivalent States) Consider a KSG with a function $F^{\text{compl.}}_{\mathsf{Out}}$ that outputs the complete keystream. Two states $st, st' \in \mathscr{S}$ are said to be *keystream-equivalent* (in short $st \equiv_{\text{kse}} st'$) if there exists an integer $r \geq 0$ such that $F^{\text{compl.}}_{\mathsf{Out}}(\mathsf{Upd}^r(st)) = F^{\text{compl.}}_{\mathsf{Out}}(st')$. Here, Upd^r means the r-times application of Upd.

For any state $st \in \mathscr{S}$, we denote by $[st]$ its equivalence class, that is $[st] = \{st' \in \mathscr{S} | st \equiv_{\text{kse}} st'\}$.

Now, let us consider an arbitrary KSG with state space $\mathscr{S}$. As any state is a member of exactly one equivalence class, the state space can be divided into ℓ distinct equivalence classes:

$$\mathscr{S} = \left[st^{(1)}\right] \dot{\cup} \ldots \dot{\cup} \left[st^{(\ell)}\right] \tag{3.1}$$

Assume a TMDTO attacker who is given some keystream (z_t), based on an unknown initial state st_0. In this case if none of the precomputations are done for values in $[st_0]$, the attack will fail. This implies that the attack effort is at least linear in the number ℓ of equivalence classes. Hence we can see that if we design a cipher such that $\ell \geq 2^\kappa$, such a cipher will have the required security level against trade-off attacks.

Let us now take a look at the minimum time effort for a TMDTO attack against a KSG with a KUF. We make in the following the assumption that any two different states $ST = (st, k)$ and $ST' = (st', k')$ with $k \neq k'$ never produce the same keystream, that is $F_{\mathsf{Out}}^{\text{compl.}}(ST) \not\equiv_{\text{kse}} F_{\mathsf{Out}}^{\text{compl.}}(ST')$. Hence, we have at least 2^κ different equivalence classes. As the effort grows linearly with the number of equivalence classes, we assume in favor of the attacker that we have exactly 2^κ equivalence classes. This gives a minimum time complexity of 2^κ. This means that, in theory, it is possible to design a cipher with a security level of κ regardless of the length σ of its variable state.

3.2.2 On Continuously Accessing the Key

In most cases the workflow of ciphers looks as follows. After the encryption or decryption process is started, the key is loaded from some non-volatile memory NVM into some registers, i.e., into some volatile memory VM. We call the value in VM a *volatile value* as it usually changes during the encryption/decryption process and the value stored in NVM, the *non-volatile value* or *non-volatile key* which remains fixed. It holds for most designs that after the key has been loaded from NVM into VM, the NVM is usually not involved anymore (unless the key schedule or the initialization process needs to be restarted). But the design approach discussed in Sect. 3.2.1 requires that the key which is stored on the device has to be accessed not only for initialization of the registers but continuously in the encryption/decryption process. The feasibility of this approach for different scenarios was investigated in [421].

It has been argued there that continuously accessing the key can impact the achievable throughput. To this end, two different cases need to be considered. The first one is when the key is set once and is never changed and the second one is when it is possible to rewrite the key. The types of NVM (e.g., MROM and PROM) which can be used in the first case, allow for efficient implementations where accessing the key bits induces no overhead. However, the key management is very difficult here.

In the second case, i.e., for those types of NVM which allow the key to be rewritable (such as EEPROM and Flash), severe timing limitations for accessing the NVM may occur. In particular, it depends on how the key stored in NVM needs to be accessed. In some cases, implementation of ciphers which require continuous access to the randomly selected bits of the key stored in rewritable types of NVM may lead to a reduction of the throughput up to a factor of 50 [421]. However, designs that require sequential access to the key bits are almost unaffected, irrespective of the underlying NVM type.

With respect to area size, there is no big difference whether the key has to be read only once or continuously during encryption, since the logic for reading the key (at least once) has to be implemented anyway. Small extra logic may be needed for synchronization with an NVM cipher which should not be clocked unless key material is read from NVM.

3.2.3 The Stream Ciphers Sprout and Plantlet

The design principles discussed in Sect. 3.2.1 have been demonstrated by two concrete keystream generators with keyed update function, namely Sprout and Plantlet. Both ciphers have a similar structure (see Fig. 3.1) which was inspired by Grain-128a [14]. The differences are the following:

1. Sprout and Plantlet have shorter internal state size compared to any of the Grain family ciphers [265]
2. They use the round key function to make the state update key-dependent

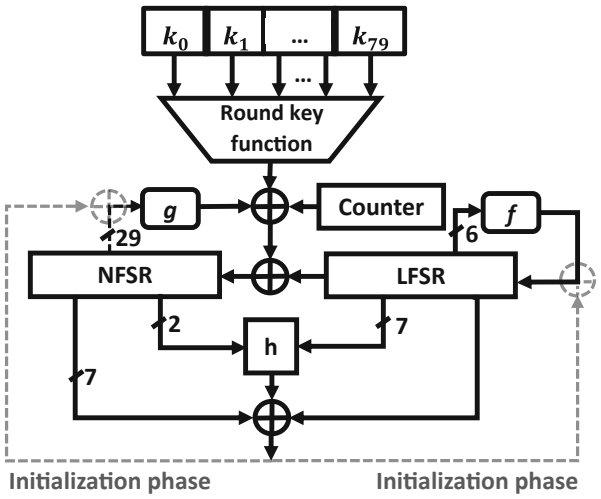

Fig. 3.1 Overall structure of the Sprout and Plantlet ciphers

3. The counter is used in the state update in order to avoid situations where shifted keys result in shifted keystreams

The design of Plantlet actually builds on Sprout but included some changes to fix several weaknesses [50, 203, 355, 387]. The main differences between Plantlet and Sprout are the following:

1. Plantlet has a larger internal state size compared to Sprout. The difference was introduced in order to increase the period of the output sequences and to increase resistance against guess-and-determine attacks
2. In both ciphers, the round key function cyclically steps through the key bits, which is well aligned with the performance of different types of NVM as mentioned before. However, in Sprout the key bits are only included in the NFSR update with a probability of 0.5, i.e., only if the linear combination of several state bits is equal to 1. This has been exploited by several attacks so in Plantlet the next key bit is added unconditionally at every clock-cycle.
3. Plantlet uses a so-called double-layer LFSR which allows for high period and at the same time avoids the LFSR being initialized with all-zeroes

For full specifications we refer the reader to [27, 422].

Implementation Results We used the Cadence RTL Compiler[1] for synthesis and simulation, and the technology library UMCL18G212T3 (UMC 0.18 μm process). The clock frequency was set to 100 kHz. For different memory types Sprout requires from 692 to 813 GEs, whereas Plantlet needs from 807 to 928 GEs. Note that the smallest KSG which follows the classical design approach needs at least 1162 GEs if the same tools are used for implementation [421].

Security As already mentioned, several serious weaknesses [50, 203, 355, 387] were shown to exist in Sprout, whereas Plantlet, to the best of our knowledge, remains secure for the moment.

3.3 A Generic Attack Against Certain Keystream Generators with Keyed Update Functions

In this section, we describe a generic attack against the following type of Keystream Generators with a Keyed Update Function (Definition 1):

Definition 3 (KSG with Boolean Keyed Feedback Function) Consider a KSG with a KUF as in Definition 1. Let Upd_i denote the Boolean component functions of the update function Upd, that is $\mathsf{Upd}(k, st) = (\mathsf{Upd}_i(k, st))_i$. We call this a KSG with a Boolean KFF (Keyed Feedback Function) if only one component function depends on the secret key. That is, there is an index i^* such that all other component

[1] See http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx.

functions with index $i \neq i^*$ can be rewritten as $\mathsf{Upd}_i(k, st) = \mathsf{Upd}_i(st)$. We call $\mathsf{Upd}_{i^*}(k, st)$ the keyed feedback function and denote it by $f_{\mathsf{Upd}}(k, st)$.

When we say the "feedback value", we mean the output of the keyed feedback function $f_{\mathsf{Upd}}(k, st)$. The most prominent examples of KSGs with a Boolean KUF in the literature are Sprout [27] and its successor Plantlet [421] (see Sect. 3.2.3). Even though several attacks against the cipher Sprout have been published [50, 203, 355, 387, 593], only little is known about the security of the underlying approach (see Sect. 3.2.1) in general. In the following, we explain the only existing generic attack [314] that implies a design criterion for this type of ciphers.

The attack is a guess-and-determine attack that is based on guessing internal states from the observed output. Its efficiency heavily relies on the guess capacity, which we define next:

Definition 4 For a given KSG with a Boolean KFF having a σ-bit internal state, a κ-bit key, and f_{Upd} as its Boolean keyed feedback function, we define the *average guess capacity* as

$$\mathrm{Pr}_g = \frac{1}{2} + 2^{-\sigma} \sum_{st} \left| \frac{\#\{k : f_{\mathsf{Upd}}(k, st) = 0\}}{2^{\kappa}} - \frac{1}{2} \right|.$$

The average guess capacity simply indicates how accurately we can guess the feedback value $f_{\mathsf{Upd}}(k, st)$ when we know the internal state but we do not know the key. The following attack [314] applies to the case of $\mathrm{Pr}_g > 1/2$ which eventually allows us to formulate a necessary design criterion.

The core of the attack is an internal state recovery algorithm (see Algorithm 1). It tests, for a given internal state whether it can consistently continue producing the observed output bits. To this end, it produces the feedback values (the outputs of the Boolean keyed feedback function) for the next states by either determining them from the output if that is possible or first checking and then guessing them. It consists of two parts: determining the feedback value is done by Algorithm 2 and checking the candidate state and then guessing the feedback value if the state survives, is achieved by Algorithm 3. It is obvious that Algorithm 2 produces only one feedback value for each clock. Similarly, Algorithm 3 first checks if a candidate state can produce the output. So, it survives with a probability of one half and the surviving states will have two successors. Hence, neither Algorithm 2 nor Algorithm 3 will propagate the total number of states to be checked.

Each candidate state has successors for consecutive clocks and a set of feedback values produced by Algorithm 1. On the other hand, we count the number of mismatches for each feedback value. We say that a feedback value is a mismatch if it is not the suggested value obtained through its internal state. If the probability that the feedback value is equal to 0 (or 1) for a given state is higher than one half, then 0 (or 1) will be the suggested value of that state.

Assume we clock the generator α_{ter} steps to check each state. Then, we expect roughly $\alpha_{ter}/2$ mismatches for a wrong state and $\alpha_{ter}(1 - \mathrm{Pr}_g)$ mismatches for

Algorithm 1 Internal state recovery

Input: Non-empty set of internal state candidates, $\mathbb{S}$; keystream $\{z_{t+1+\theta_f}, \ldots, z_{t+\theta_f+\alpha_{ter}}\}$; the maximum number of clocks for each test, α_{ter}; average guess capacity, $\Pr_g$; miss event probability ϵ
Set $\epsilon_{ter} = \sqrt{\frac{-\ln \epsilon}{2\alpha_{ter}}}$ and $\alpha_{thr} = \alpha_{ter}(1 - \Pr_g + \epsilon_{ter})$
Initialize CUR and NEW as two empty sets and load all the states in $\mathbb{S}$ into CUR
Set $\#MM(st)$ to 0 for each state st in CUR and make a copy of CUR as the $roots$
for each clock i from t to $(t + \alpha_{ter} - 1)$ **do**
    for each state st in CUR **do**
        Compute $\Pr_g(st)_f$
        if $\Pr_g(st)_f = 0.5$ **then**
            Set $fb_{sugg} = 0$
        else
            Set fb_{sugg} as the feedback value of st suggested through f_{Upd}
        end if
        if $f_{\mathsf{Upd}}(k, st)$ of st can be determined from the output bit $z_{i+1+\theta_f}$ **then**
            Run *Determine Procedure* (Algorithm 2)
        else
            Run *Check-and-guess Procedure* (Algorithm 3)
        end if
    end for
    Terminate if NEW is empty and give no output
    Copy NEW to CUR
    Empty NEW
end for
Output: the roots in CUR as the candidates for the correct state at clock t

Algorithm 2 Determine procedure

Determine the feedback value as $f_{\mathsf{Upd}}(k, st)$ from the corresponding output bit
Update st by clocking it with the feedback value fb_{det}
if $fb_{sugg} \neq fb_{det}$ (it is a mismatch) **then**
    Increment $\#MM(st)$ by one
end if
if $\#MM(st) \leq \alpha_{thr}$ **then**
    Add updated st with $\#MM(st)$ and its root to NEW
end if

a correct state. This provides us with a distinguisher to recover the correct state without knowing the key. We set a threshold value α_{thr}, between $\alpha_{ter}(1 - \Pr_g)$ and $\alpha_{ter}/2$ and simply eliminate the states whose number of mismatches exceeds α_{thr}. We expect all the wrong internal states to be eliminated for a well-chosen pair $(\alpha_{ter}, \alpha_{thr})$ and only the correct state is expected to survive. Theorem 1 suggests appropriate values for α_{ter} so as to obtain a given success rate. Then we fix the threshold value accordingly, in Algorithm 1 in its third line.

The performance of Algorithm 1 depends heavily on how many clocks we should go further to eliminate all the wrong states without missing the correct state. This is

Algorithm 3 Check-and-guess procedure

if the output of st is equal to the actual output at the corresponding clock **then**
  Make two copies st_0, st_1 of st with $\#MM(st_0) = \#MM(st_1) := \#MM(st)$
  Set the feedback value to 0 for st_0 and update st_0 and 1 for st_1 and update st_1
  if $fb_{sugg} = 0$ **then**
    Increment $\#MM(st_1)$ by one
  else
    Increment $\#MM(st_0)$ by one
  end if
  if $\#MM(st_0) \leq \alpha_{thr}$ **then**
    Add st_0 along with $\#MM(st_0)$ to NEW and set the root of S as its root
  end if
  if $\#MM(st_1) \leq \alpha_{thr}$ **then**
    Add st_1 along with $\#MM(st_1)$ to NEW and set the root of st as its root
  end if
end if

determined by the success rate of the algorithm which in turn is dominated by the guess capacity (Definition 4) as stated in the following Theorem 1 [314]:

Theorem 1 *Let* $\Pr_g > 1/2$ *be the guess capacity of a given KSG with Boolean KFF having internal state size* σ*. For a given* $0 < \epsilon < 1$*, if* α_{ter} *is greater than or equal to*

$$\frac{1}{(2\Pr_g - 1)^2}\left(\sqrt{-2\ln\epsilon} + \sqrt{2\ln 2 \cdot (\sigma - 1)}\right)^2,$$

then the success rate of the attack in Algorithm 1 is at least $1 - \epsilon$ *and the number of false alarms is less than one in total.*

The average guess capacity of Sprout is 0.75. Hence it is possible to recover its correct state without knowing the key by eliminating a wrong state in roughly 122 clocks [314]. Checking roughly 2^{40} states (which are called "weak states" and loaded into a table in the precomputation phase), one can recover the key in roughly 2^{38} encryptions [314]. On the other hand, Algorithm 1 becomes infeasible when $\Pr_g$ approaches $1/2$. Plantlet (Sect. 3.2.3) has a guess capacity of $1/2$, so Algorithm 1 is not applicable to Plantlet. Concluding, the attack above implies a new security criterion: the guess capacity of the feedback function of a KSG with Boolean KFF should be one half in order to avoid state recovery attacks that bypass the key.

3.4 Randomized Encryption Employing Homophonic Coding

3.4.1 Background

In [503], several approaches to including randomness in encryption techniques are discussed, mainly in the context of block and stream ciphers. Randomized encryption is described [503] as a procedure which enciphers a message by randomly choosing a ciphertext from a set of ciphertexts corresponding to the message under the current encryption key.

Homophonic coding was introduced in [249] as a source coding technique which transforms a stream of message symbols with an arbitrary frequency distribution into a uniquely decodable stream of symbols which all have the same frequency. The universal homophonic coding approach [397] is based on an invertible transformation of the source information vector with embedded random bits, and this approach does not require knowledge of the source statistics. The source information vector can be recovered from the homophonic coder output without knowledge of the random bits by passing the codeword to the decoder (inverter) and then discarding the random bits.

A number of randomized encryption techniques have been reported: In [234], a probabilistic private-key encryption scheme named LPN-C whose security can be reduced to the hardness of the LPN problem was proposed and analysed. An approach for the design of stream ciphers employing error-correction coding and certain additive noise degradation of the keystream was reported in [201]. A message is encoded before the encryption so that the decoding, after mod 2 addition of the noiseless keystream sequence and the ciphertext, provides its correct recovery. Resistance of this approach against a number of general techniques for cryptanalysis, was also considered in [201]. Joint employment of randomness and dedicated coding has been studied for enhancing the security of the following block-by-block encryption schemes: (1) in [418], where the basic keystream generator security is enhanced by employing a particular homophonic coding based on embedding of random bits; (2) in [413, 419] and [414] randomness and dedicated coding were employed for enhancing the security of the compact generators of pseudorandom vectors; (3) in [322] and [577] channel coding was employed to increase the security of a DES block cipher operating in the ciphertext feedback (CFB) mode. Also, certain issues of randomized encryption were considered in [321, 570] and [313].

3.4.2 Encryption and Decryption

The ciphering technique given in this section originates from the schemes reported in [322, 414, 418], and it corresponds to the randomized encryption schemes proposed and discussed in [452]. The design assumes the availability of a source of pure randomness (for example, as an efficient hardware module) and that a suitable

error-correcting coding (ECC) technique is available. This availability means that the implementation complexities of the source of randomness and the ECC do not imply a heavy implementation overhead in suitable scenarios.

The scheme employs a homophonic approach for a purpose different from the ones this coding techniques were designed for. The main purpose is not just randomization of the source message vectors (the goal of homophonic coding) nor secrecy without a secret key (the goal of wire-tap channel coding) but enhancing the cryptographic security of certain encryption schemes by employing the underlying features of homophonic or wire-tap channel coding. The goal is the security enhancement of a cryptographic keystream generator for encryption by employing a dedicated coding scheme where the codewords provide additional "masking" of the keystream vectors employed for encryption. The encryption scheme in Fig. 3.2 performs modulo 2 addition of the outputs of the encoding block and the keystream generator which can be considered not only as "masking" the message vector with a vector generated by a secret key, but also as masking the keystream vector by a randomized mapping of the information vector.

We assume that the encryption from Fig. 3.2 employs concatenation of the following coding algorithms: (1) a universal homophonic coding [397] which performs the following mapping $\{0,1\}^{\ell} \rightarrow \{0,1\}^{m}$, $\ell < m$, and (2) a linear block error-correction code which performs $\{0,1\}^{m} \rightarrow \{0,1\}^{n}$, $m < n$, and which provides reliable communication over a binary symmetric channel with a known probability of bit complementation. Please note that any suitable binary

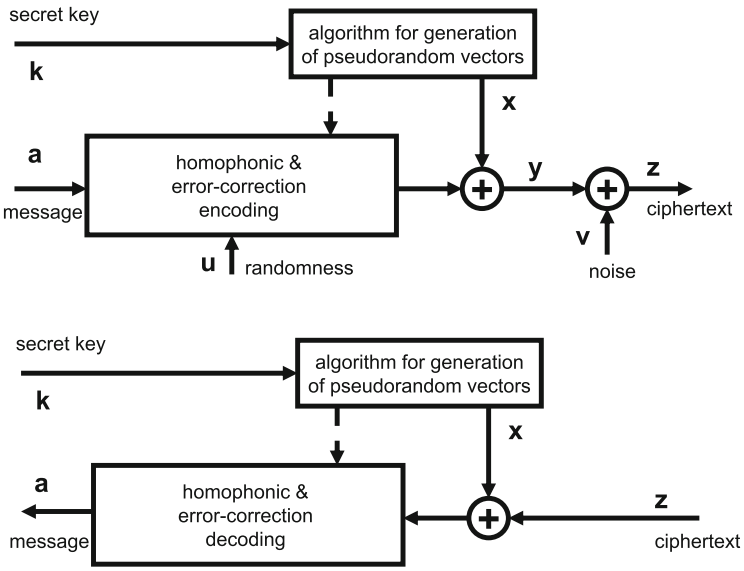

Fig. 3.2 Model of a security enhanced randomized encryption within the encoding-encryption paradigm: the upper part shows the transmitter, the lower part—the receiver [452]

linear block code designed to work over a binary symmetric channel with crossover probability p could be employed. There are a lot of these coding schemes reported in the literature and one which best fits into a particular implementation scenario (hardware or software oriented) could be selected. We consider a communication system displayed in Fig. 3.2 where some message $\mathbf{a} = [a_i]_{i=1}^{l} \in \{0,1\}^l$ is sent to a transmitter over a noisy channel and the following operations at the transmitter and receiver.

At the Transmitter To ensure reliable communication, a linear error-correcting encoder $C_{ECC}(\cdot)$ is used, that maps an m-bit message to a codeword of $n > m$ bits, using an $m \times n$ binary code generator matrix $\mathbf{G}_{ECC}$. A homophonic encoder $C_H(\cdot)$ is added prior to $C_{ECC}(\cdot)$, which requires the use of a vector $\mathbf{u} = [u_i]_{i=1}^{m-l} \in \{0,1\}^{m-l}$ of pure randomness, i.e., each u_i is the realization of a random variable U_i with distribution $\Pr(U_i = 1) = \Pr(U_i = 0) = 1/2$. The encoding $C_H(\mathbf{a}||\mathbf{u})$ may be described by an $m \times m$ binary matrix $\mathbf{G}_H$ such that

$$C_H(\mathbf{a}||\mathbf{u}) = [\mathbf{a}||\mathbf{u}]\mathbf{G}_H,\ \mathbf{G}_H = \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_l \\ \mathbf{G}^C \end{bmatrix} \tag{3.2}$$

where $\mathbf{G}^C$ is an $(m-l) \times m$ generator matrix for an $(m, m-l)$ linear error-correction code C, and $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_l$ are l linearly independent row vectors from $\{0,1\}^m \backslash C$.

We get a joint encoding $\mathbf{a} \in \{0,1\}^l \mapsto C_{ECC}(C_H(\mathbf{a}||\mathbf{u})) \in \{0,1\}^n$, which may alternatively be written as

$$C_{ECC}(C_H(\mathbf{a}||\mathbf{u})) = C_{ECC}([\mathbf{a}||\mathbf{u}]\mathbf{G}_H) = [\mathbf{a}||\mathbf{u}]\mathbf{G}_H\mathbf{G}_{ECC} = [\mathbf{a}||\mathbf{u}]\mathbf{G} \tag{3.3}$$

where $\mathbf{G} = \mathbf{G}_H\mathbf{G}_{ECC}$ is an $m \times n$ binary matrix containing the two successive encoders at the transmitter.

The codeword sent is finally an encrypted version $\mathbf{y}$ of $C_{ECC}(C_H(\mathbf{a}||\mathbf{u}))$ given by $\mathbf{y} = \mathbf{y}(\mathbf{k}) = C_{ECC}(C_H(\mathbf{a}||\mathbf{u})) \oplus \mathbf{x}$ where $\mathbf{x} = \mathbf{x}(\mathbf{k}) = [x_i]_{i=1}^{n} \in \{0,1\}^n$ is a pseudorandom vector needed for encryption, which is generated by either a keystream generator, or by a block cipher working in the cipher feedback mode (CFB) as in [322] and [577]. Notice the important dependency of $\mathbf{x} = \mathbf{x}(\mathbf{k})$ in the secret key $\mathbf{k}$. Also note that, for simplicity of the exposition, the data employed for generation of the pseudorandom vectors $\mathbf{x}$, which are publicly known (like a public seed and a synchronization parameter) are not explicitly shown. Finally, the model includes the assumption that the concatenation of the binary vectors $\mathbf{x}$ appears as a pseudorandom binary sequences and from a statistical point of view is indistinguishable from a random binary sequence.

At the Receiver The noisy communication channel is modeled by the addition of a noise vector $\mathbf{v} = [v_i]_{i=1}^{n} \in \{0,1\}^n$, where each v_i is the realization of a random variable V_i with $\Pr(V_i = 1) = p$ and $\Pr(V_i = 0) = 1 - p$. The

receiver obtains $\mathbf{z} = \mathbf{z}(\mathbf{k}) = \mathbf{y} \oplus \mathbf{v} = C_{ECC}(C_H(\mathbf{a}||\mathbf{u})) \oplus \mathbf{x} \oplus \mathbf{v}$ and starts by decrypting $\mathbf{y} = (C_{ECC}(C_H(\mathbf{a}||\mathbf{u})) \oplus \mathbf{x} \oplus \mathbf{v}) \oplus \mathbf{x} = C_{ECC}(C_H(\mathbf{a}||\mathbf{u})) \oplus \mathbf{v}$. He then first decodes $C_H(\mathbf{a}||\mathbf{u})$. In the case of a successful decoding, he computes $\mathbf{a}$ using C_H^{-1} and informs the transmitter he could decode. Otherwise he asks the transmitter for a retransmission. This assumes noiseless feedback between the receiver and the transmitter.

3.4.3 Security Evaluation

Information-Theoretic Security In [452], the above model of randomized encryption schemes was studied from an information-theoretic point of view. The goal was to analyze the security enhancement provided by the wiretap encoding, in terms of secret key $\mathbf{k}$ equivocation, that is, the uncertainty that an adversary faces about the secret key, given all the information he could collect during passive or active attacks. This analysis demonstrated a gain in unconditional security, and thus confirmed the security benefit of the additional wiretap encoder, through tight lower bounds (Lemmas 1 and 2 in [452]) and asymptotic values (Theorems 1 and 2 in [452]) of the secret key equivocation. The cost of this enhanced security is only a slight-to-moderate increase in the implementation complexity and the communications overhead. However, it also revealed that if the same secret key is used for too long, the adversary may gather large enough samples for offline cryptanalysis. The uncertainty then decreases to zero. Then starts a regime in which a computational security analysis is needed to estimate the resistance against secret key recovery, which motivated the current paper.

Computational Complexity Security Mihaljević and Oggier [420] presents a security evaluation of the considered technique in a chosen plaintext attack scenario, which shows that the computational complexity security is lower bounded by the related LPN (Learning from Parity with Noise) complexity in both the average and worst cases. This gives guidelines for constructing a dedicated homophonic encoder which maximizes the complexity of the underlying LPN problem for a given encoding overhead.

Note Recall that in a chosen plaintext attack (CPA) scenario, the claim that a scheme is secure in an information-theoretic sense means that even an attacker with unlimited resources for recovering the secret key, in the considered evaluation scenario, faces complete uncertainty about the secret key employed for encryption, i.e., a set of equally probable candidates for the true secret key will exist. On the other hand, a claim that an encryption scheme is secure in a computational-complexity sense means the following: Although the secret-key could be recovered in a CPA scenario, and so it is not possible to claim information-theoretic security, the computational complexity of this recovery is as hard as solving a problem which belongs to a class of proven hard problems, as the LPN problem is.

3.5 Conclusion and Future Directions

We have presented some advances in the design and security evaluations of some contemporary symmetric encryption techniques which provide a good trade-off between the implementation/execution complexity and the required security.

In one direction, we have demonstrated the use of keystream generators with keyed update functions to provide the same security level at much smaller hardware area costs. In particular, we have shown that the security limitations which were believed to be imposed by the size of the state can be improved to offer a much better trade-off between hardware requirements and security. In the other direction, we have described the use of homophonic encoding for security enhancement of certain randomized symmetric encryption schemes.

Also, we have discussed certain generic approaches for security evaluation of the considered encryption schemes. The encryption schemes based on keyed update functions were evaluated against a dedicated guess-and-determine attack. The randomized encryption schemes were evaluated based on generic information-theoretic and computational-complexity approaches. We believe that there is plenty of room for further work in this area, and other innovative schemes should be investigated. We have found that employment of keyed update functions and results from coding theory are particularly promising ideas for the design of advanced encryption schemes and we plan to explore them further in the near future.