Cryptanalysis of the Knapsack Generator

Simon Knellwolf and Willi Meier

FHNW, Switzerland

Abstract. The knapsack generator was introduced in 1985 by Rueppel and Massey as a novel LFSR-based stream cipher construction. Its output sequence attains close to maximum linear complexity and its relation to the knapsack problem suggests strong security. In this paper we analyze the security of practically relevant instances of this generator as they are recommended for the use in RFID systems, for example. We describe a surprisingly effective guess and determine strategy, which leads to practical attacks on small instances and shows that the security margin of larger instances is smaller than expected. We also briefly discuss a variant of the knapsack generator recently proposed by von zur Gathen and Shparlinski and show that this variant should not be used for cryptographic applications.

Keywords: knapsack, stream cipher, pseudorandom generator

1 Introduction

Let $w_0, \ldots, w_{n-1}$ be n k-bit integers, and let $u_0, u_1, \ldots$ be a sequence of bits generated by a linear feedback shift register (LFSR) of order n over $\mathbb{F}_2$. At step i the *knapsack generator* computes

$$v_i = \sum_{j=0}^{n-1} u_{i+j} w_j \mod 2^k, \tag{1}$$

discards the ℓ least significant bits of v_i and outputs the remaining $k - \ell$ bits as part of the keystream. We call $u_0, u_1, \ldots$ the *control bits*, v_i the i-th *sum* and $w_0, \ldots, w_{n-1}$ the *weights* of the generator. The entire generator is defined by $n(2 + k)$ bits: n bits for the connection polynomial of the LFSR, n bits for the initial control bits (corresponding to the initial state of the LFSR) and kn bits for the weights. The connection polynomial should be primitive in order to achieve maximum period in the control sequence. Due to this special choice, it is natural to consider the connection polynomial as a public parameter. The remaining $n(1 + k)$ bits form the key of the generator. As a concrete example, a generator with $n = k = 64$ has a key length of 4160 bits.

Rueppel and Massey [16] introduced this generator in 1985. They addressed one of the main issues in the design of LFSR-based cryptosystems, which consists in breaking the linearity of the LFSR. The knapsack generator achieves this by the use of integer addition modulo 2^k which is a highly nonlinear operation when considered over $\mathbb{F}_2^k$, see [16,20] for a systematic analysis. Therewith it provides an interesting alternative to the use of nonlinear boolean filtering and combining functions. It avoids the tradeoff between high linear complexity and high correlation immunity which is inherent to nonlinear boolean functions, as it was shown in [19]. Besides the knapsack generator, Rueppel [14] also introduced the summation generator to avoid this tradeoff, but it turned out to be vulnerable to correlation attacks [5,11], algebraic attacks [10] and to attacks based on feedback with carry shift registers [9]. Compared to the summation generator, little cryptanalytic work has been published concerning the knapsack generator. To our knowledge it resisted the well known attack strategies for stream ciphers. Due to this absence of known security flaws and due to its ease of implementation, the authors of [2] recommend the knapsack generator for the use in RFID networks.

The name comes from the close relation to the knapsack problem (also known as the subset sum problem), which consists in finding a subset of given weights that add up to a given sum. The decisional version of this problem is known to be NP-complete and several attempts have been made to use it in cryptography. A prominent example is the Merkle-Hellman knapsack cryptosystem [13] which was broken by Shamir [17]. In a different direction, Impagliazzo and Naor [8] constructed provably secure pseudorandom generators and universal hash functions based on the knapsack problem.

Other than the above cryptosystems, the security of the knapsack generator is not directly related to the hardness of the knapsack problem. In the context of the knapsack problem the weights are known, whereas in the context of the knapsack generator they are not. For comparison, Howgrave-Graham and Joux [7] presented new generic algorithms for hard knapsack problems which allowed them to solve instances with $n = k = 96$ in practical time. These results have no implications on the security of the knapsack generator, and knapsack generators of much smaller size are not a priori insecure (even for $n = k = 32$, the key consists of 1056 bits).

Throughout the literature, for example in [2,12,15], it is recommended to choose $k = n$. We also focus on these cases and we do not always mention k explicitly in the following. The cases $n = 32, 64$ are of particular interest because they are favorable for software implementation. Besides n, the knapsack generator has an additional security parameter ℓ, which is the number of discarded bits per output. Intuitively, if ℓ is small, the output reveals more information about the control sequence,whereas if ℓ is large, the throughput of the generator gets low.

1.1 Previous Cryptanalytic Results

Rueppel [15] provided a first extensive analysis of the knapsack generator. He showed that the $\lceil \log n \rceil$ least significant bits of the sums do not achieve high linear complexity, and he provided some evidence that the other bits indeed do. This let him recommend to choose $\ell = \lceil \log n \rceil$. He further estimated the number of different boolean functions $\{0,1\}^n \to \{0,1\}$ mapping n control bits to the i-th bit of a sum as in (1), and he found that for $\lfloor \log n \rfloor \leq i < n$ at least $2^{n(\lfloor \log n \rfloor - 1)}$

such functions can be specified by n weights. He stated this as a lower bound on the effective key length of the generator.

Von zur Gathen and Shparlinski [3] considered scenarios where either the control bits or the weights are known. In both cases they translate the task of finding the unknown parts of the key into a short vector problem which they solve by LLL lattice basis reduction algorithms. In the known control bit scenario, they can predict the generator if ℓ is not too small using about $n^2 - n$ outputs. It is difficult to estimate the practical time complexity of their strategy when extended to a guess and determine attack, and no empirical results are provided.

1.2 Contribution of This Paper

We describe a novel guess and determine attack which needs only a few more outputs than the number of weights n. Our analytical and empirical results show that the security level of the knapsack generator is not significantly higher than n bits. For a generator with $n = 32$ we implemented the full attack on a Desktop Computer.

Further, we analyze the faster variant of the knapsack generator recently proposed in [4], and show that it should not be used in cryptographic applications.

1.3 Road Map

In Section 2 we describe the knapsack generator as a system of modular equations and we introduce the notion of an approximation matrix, which is the basic concept of our analysis. In Section 3 we explain how to find good approximation matrices. In Section 4 we describe the full attack and illustrate its performance by empirical results, including a practical attack for $n = 32$. In Section 5 we briefly analyze the fast variant of the knapsack generator proposed in [4].

2 Problem Formalization

In this section we address the following problem: Given the control bits and s outputs of the knapsack generator, predict some bits of subsequent outputs with probability higher than 1/2. Later, in Section 4, we extend this to a guess and determine attack when the control bits are not known.

We first formulate the knapsack generator as a system of modular equations and fix some notation.

2.1 A System of Modular Equations

In order to produce s outputs, the knapsack generator computes s sums according to (1). This can be written as

$$\mathbf{v} = U\mathbf{w} \mod 2^n, \tag{2}$$

where $\mathbf{v} = (v_0, \ldots, v_{s-1})$ are the sums, $\mathbf{w} = (w_0, \ldots, w_{n-1})$ are the weights and U is a $s \times n$ matrix whose coefficients are given by the control bits. We call U the *control matrix*. Its rows are the consecutive states of a binary LFSR, and the control matrix is entirely determined by one of its rows. We write $\mathbf{u}_i$ for the i-th row of U and, more generally, for the i-th state of the LFSR generating the control sequence. It is shown in [3] that n consecutive row vectors $\mathbf{u}_i$ are always linearly independent over the integers modulo 2^n. Hence, if U is known and $s \geq n$, the system described by (2) can be easily solved for $\mathbf{w}$. The challenge is to deal with the discarded bits. An attacker can only observe the $n - \ell$ most significant bits of each component of $\mathbf{v}$. Guessing the discarded bits is too expensive, since at least $n\ell$ bits would have to be guessed. The idea is to recover only the significant bits of each weight which might be sufficient to make a prediction.

2.2 Weight Approximation Matrices

We write the outputs as a vector $\mathbf{z} = (z_0, \ldots, z_{s-1})$ such that $z_i = v_i \gg \ell$ for $0 \leq i < s$. Here, $\gg$ denotes a right shift of n-bit integers, left shift is denoted by $\ll$, and when used for vectors, the shifting is applied componentwise. Since U has full rank modulo 2^n, there always exists a $n \times s$ matrix T with integer coefficients such that $TU = I_n \mod 2^n$, where I_n denotes the $n \times n$ identity matrix. We call such a T an *approximation matrix*. The name is motivated by the fact that

$$\mathbf{w} = T(\mathbf{z} \ll \ell) + T\mathbf{d} \mod 2^n$$

for some unknown vector $\mathbf{d} = (d_0, \ldots, d_{s-1})$ with $0 \leq d_i < 2^\ell$ for $0 \leq i < s$ (the d_i correspond to the discarded bits), which lets us hope to obtain *approximate weights* $\tilde{\mathbf{w}}$ by ignoring the discarded bits, that is, by computing

$$\tilde{\mathbf{w}} = T(\mathbf{z} \ll \ell) \mod 2^n. \tag{3}$$

The matrix T will be derived only from U (independently from $\mathbf{z}$). As soon as $s > n$, the choice of T is not unique. In the next paragraph we obtain a criterion for making a good choice.

2.3 Prediction With Approximate Weights

In order to predict $z_s = v_s \gg \ell$, we compute $\tilde{v}_s = \mathbf{u}_s \tilde{\mathbf{w}} \mod 2^n$, where $\mathbf{u}_s = (u_s, \ldots, u_{s+n-1})$ are the corresponding control bits. Substituting $\tilde{\mathbf{w}}$, we get $\tilde{v}_s = \mathbf{u}_s T(\mathbf{z} \ll \ell) \mod 2^n$. The generator actually computes

$$v_s = \mathbf{u}_s T(\mathbf{z} \ll \ell) + \mathbf{u}_s T\mathbf{d} \mod 2^n.$$

Intuitively, the significant bits of $\tilde{v}_s$ are likely to be correct if the integer summand $\mathbf{u}_s T\mathbf{d}$ is small in absolute value. We denote by p_λ the probability that at least λ significant bits of $\tilde{v}_s$ are correct,

$$p_\lambda = \Pr\left[(v_s \oplus \tilde{v}_s) \gg (n - \lambda) = 0\right].$$

The intuition is then formalized by the following lemma.

Lemma 1. *Let m be the smallest integer such that $|\mathbf{u}_s T\mathbf{d}| < 2^m$. Then, we have*

$$p_\lambda > 1 - \frac{1}{2^{\lambda-m}}$$

for all λ with $m \le \lambda < n$.

Proof. For shorter notation we set $a = \mathbf{u}_s T(\mathbf{z} \lll \ell)$ and $b = \mathbf{u}_s T\mathbf{d}$. The difference $v_s \oplus \tilde{v}_s$ then writes as $(a+b) \oplus a$. Let's first assume that $b \ge 0$. Then, the sum $a+b$ can be recursively described by $(a+b)_j = a_j \oplus b_j \oplus c_{j-1}$, $c_j = a_j b_j \oplus a_j c_{j-1} \oplus b_j c_{j-1}$, where c_j denotes the carry bit, and $c_{-1} = 0$. For $j \ge m$ we have $b_j = 0$, and thus, $(a+b)_m \oplus a_m = c_{m-1}$ and for $j > m$, $(a+b)_j \oplus a_j = c_{m-1} \prod_{i=m}^{j-1} a_i$. The bound follows immediately under the assumption that the values of the bits a_i are independent and uniformly distributed for $m \le i < n$. The case of $b < 0$ is very similar (using the recursive description of $a - b$ with a borrow bit instead of the carry bit).

Lemma 1 guarantees that we can correctly predict at least one bit per output with probability higher than 1/2 if $m < n - 1$. Smaller m give more predictable bits. Hence, we are interested in an upper bound on m. Since the coefficients of $\mathbf{u}_s$ are restricted to be 0 or 1 and the coefficients of $\mathbf{d}$ are strictly smaller than 2^ℓ, we have $|\mathbf{u}_s T\mathbf{d}| < \|T\| 2^\ell$, where we use $\|T\| = \sum_{i,j} |t_{ij}|$ as the *norm* of T. By the definition of m, this gives

$$m < \lceil \log \|T\| \rceil + \ell.$$

It follows that $\lceil \log \|T\| \rceil \le n - \ell - 1$ is a sufficient condition to predict at least one bit. In the next section we describe a method that finds approximation matrices with much lower norms than needed for typical values of ℓ.

3 Finding Good Approximation Matrices

The success of our attack essentially depends on the ability to find good approximation matrices, that is, matrices with small coefficients in absolute value. To compute such a matrix T we proceed row by row. We search for n row vectors $\mathbf{t}_i$ with small norm and such that $\mathbf{t}_i U = \mathbf{e}_i$, where $\mathbf{e}_i$ is the i-th unit vector of the standard basis of $\mathbb{F}_2^n$. This is a special case of the following problem:

Problem 1. Given an $s \times n$ integer matrix A and an integer column vector $\mathbf{b}$ of dimension n, find an integer row vector $\mathbf{x}$ such that $\mathbf{x}A = \mathbf{b}$ and such that the coefficients of $\mathbf{x}$ are small in absolute value.

Typically, in our scenario, there are many solutions to the equation $\mathbf{x}A = \mathbf{b}$ and it is not difficult to find one of them by linear algebra techniques. The difficult part is to find one with a small norm. We use an approach which is implemented in Victor Shoup's NTL [18]. The idea is the following: Given an arbitrary solution $\mathbf{x}'$, search for a vector $\mathbf{x}''$ in the kernel of A such that $\mathbf{x} = \mathbf{x}' - \mathbf{x}''$ has a small norm. This essentially corresponds to the approximate closest vector problem in the lattice spanned by the kernel vectors. Using Babai's algorithm [1] together with a LLL reduced kernel basis we can find very close vectors $\mathbf{x}''$ in practice. A nice introduction to Babai's algorithm and its use in combination with LLL can be found in [6].

In our specific application, the matrix A $(= U)$ has only 0 and 1 as coefficients and its rows are the successive states of an LFSR. It turns out that the small coefficients are favorable, that is, the average norm of the returned solution is smaller than for more general integer matrices. The particular structure of the control matrix (successive states of an LFSR) has no significant influence, the results are about the same as for random matrices with binary coefficients. In particular, the choice of the connection polynomial seems not to be important (as long as it is primitive).

Not surprisingly, the average norm of the returned solutions depends on s (it basically determines the kernel dimension of A). Figure 1 illustrates the performance of the method in function of s for $n = 64$. The graph indicates the average logarithmic norm as well as the lower and the upper quartile for samples of 100 approximation matrices obtained for $s = 68, 70, \ldots, 96$. Recall that s is the number of outputs used to compute the approximate weights.

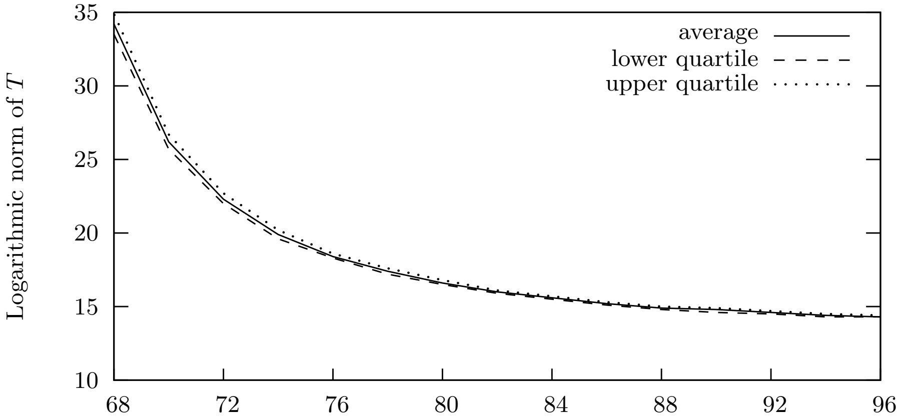

Fig. 1. Average logarithmic norm of T for $n = 64$ in function of s.

4 Description of the Attack and Empirical Results

So far, we can predict the generator if the control bits are known. The empirical results in this section illustrate the effectiveness of the approach. But first, we describe the extension of our technique to an attack where the control bits are not known.

4.1 Description of the Attack

We assume a scenario where the attacker does not know the control bits nor the weights. Since the whole control sequence is determined by only n bits (typically, the initial state of the LFSR), the above approach naturally extends to a guess and determine attack:

1. Guess $u_0, \ldots, u_{n-1}$ and derive the $s \times n$ control matrix U.
2. Find an approximation matrix T based on U.
3. Use T and $z_0, \ldots, z_{s-1}$ to compute $\tilde{\mathbf{w}}$ as in (3).
4. Compute t predictions and check their λ most significant bits. If almost all of them are correct, the control bits have been guessed correctly. Otherwise, go back to step 1.

The parameters t and λ must be chosen such that the checking is reliable. At least, t should be chosen such that $n \geq t\lambda p_\lambda$. This is not a problem, because the norms of the approximation matrices are very low, and λ can be chosen such that p_λ is almost one. The attack then needs $s + t$ outputs: s outputs to approximate the weights and t outputs to check the predictions. The most expensive part of the attack is at step 2, where the approximation matrices are computed. Instead of computing 2^n such matrices, we can check several guesses by the same matrix T. Using $z_1, \ldots, z_s$ to compute $\tilde{\mathbf{w}}$ at step 3, we can check if $u_0, \ldots, u_{n-1}$ was the state of the LFSR after one clock and we can easily compute the initial state. In general, if $r \geq s + t$ outputs are available, only $2^n/(r - s)$ approximation matrices must be computed. Since this computation is independent of the observed outputs, it can even be done offline.

4.2 Practical Attack for $n = 32$

For $n = 32$ the above attack is practical on a desktop computer. In our experiments we assumed that 552 outputs could be observed. We used control matrices with $s = 40$ rows and the parameters for the checking part were $t = 20$ and $\lambda = 5$. Hence, only 2^{23} approximation matrices had to computed. A guess was accepted when less than 20 of the 100 predicted bits where wrong. On a Intel Core 2 Duo E8400 3.0 GHz Processor with 4 GB of RAM it took about three days to identify the correct initial control bits and about 870 bits of the weights. This allows an attacker to predict more than 22 bits per output (we used $\ell = 5$, hence an output has 27 bits).

4.3 Empirical Results for Larger n

For larger n the attack is not practical on a desktop computer, since we could not circumvent the guessing of the n bits. Hence, we assume in this paragraph that the control bits are known and that we have observed s outputs. This corresponds to a known control bits attack or to the checking part of our guess and determine attack. We are interested in the average number of significant bits per output that we can correctly predict. To be precise, let λ^* be the largest λ such that $(v_s \oplus \tilde{v}_s) \gg (n - \lambda) = 0$. We analyze the average size of λ^*. Note that if $\tilde{v}_s$ would be obtained by coin tossing, the expectation of λ^* would be $\sum_{i=1}^{n} i/2^{i+1} \approx 1$ bits per output. Table 1 contains the results for $n = 32, 64, 128$ with $\ell = \log n$ and different values of s. The samples were taken randomly from the key space of the generator (recall that the key consists of $n(1 + n)$ bits).

Table 1. Average number of correctly predicted bits per output (λ^*).

$s - n$	$n = 32$	$n = 64$	$n = 128$	$n = 256$
8	20.6	42.9	85.3	164.6
16	22.2	48.7	100.9	203.4
24	22.6	50.3	105.9	216.4
32	22.7	50.8	108.1	222.4

The results in Table 1 show that even for large n the security of the knapsack generator is not significantly higher than n bits. Computing an approximation matrix which allows to predict about 108 bits per output of a generator with $n = 128$ takes a few seconds and needs no more than 160 outputs.

5 Analysis of the Fast Knapsack Generator

In [4], von zur Gathen and Shparlinski describe a variant of the knapsack generator which achieves faster output generation. We call it the *fast knapsack generator*. They consider a slightly more general setting as we did in this paper by taking the weights in an arbitrary ring R (in this paper we just considered $R = \mathbb{Z}/m\mathbb{Z}$ with $m = 2^n$). The speedup is achieved by a special choice of the weights. Instead of randomly choosing each of the n weights, it is proposed to choose two elements $a, b \in R$ and to compute $w_j = ab^{n-j}$ for $0 \leq j \leq n - 1$. With these weights, the $(i + 1)$-th sum can be computed recursively from the i-th sum by

$$v_{i+1} = bv_i - ab^{n+1}u_i + abu_{i+n}, \text{ for } i \geq 0.$$

Hence, only one multiplication and two additions are needed for generating one output. If R is a prime field, the sequence (v_i) has provable properties concerning the uniformity of its distribution (see Theorem 3.5 in [4]). However, it was left open whether this specialization affects the cryptographic security of the generator. We show that the sequence does not provide cryptographic security if R is a prime field and we believe that the security is highly questionable if $R = \mathbb{Z}/m\mathbb{Z}$ for $m = 2^n$. Our attack is a guess and determine attack whose complexity essentially depends on the number of discarded bits (and not on n as in the case of the original generator). It specifically exploits the strong relation between the weights.

5.1 The Fast Generator Over Prime Fields

Assume that R is a prime field, i.e. $R = \mathbb{F}_p$ for p prime. We think of the elements of $\mathbb{F}_p$ as $\lceil \log p \rceil$-bit integers and as above we denote by ℓ the number of discarded bits. Let's first suppose that the control bits are known. Then a and b can be determined as follows (here, all operations are modulo p):

1. Find i_0 such that $u_{i_0} = 0$ and $u_{i_0+n} = 0$
2. Guess the 2ℓ discarded bits of v_{i_0} and v_{i_0+1}
3. Compute $b = v_{i_0+1}/v_{i_0}$ and $a = v_{i_0}/\sum_{j=0}^{n-1} u_{i_0+j} b^{n-j}$
4. For some $i \neq i_0$ compute $\sum_{j=0}^{n-1} u_{i+j} ab^{n-j}$ and check if the significant bits agree with those of v_i.

The cost of this attack is about the cost of $2^{2\ell}$ times computing two modular inverses and two sums with n summands. If we drop the assumption that the control bits are known, we can not choose i_0 suitably in the first step. We have to guess it and hope that u_{i_0} and u_{i_0+n} are zero. Then, at the third step, we miss $n-1$ control bits for computing a. Instead of guessing these control bits, we speculate on $u_{i_0+1} = 0$ and $u_{i_0+n+1} = 1$ such that $v_{i_0+2} = bv_{i_0+1} + ab$. So, we only have to guess the discarded bits of v_{i_0+2} for obtaining a. In order to check a and b we try to find $u_{i_0+2}, \ldots, u_{2n-1}$ such that the significant bits of $bv_i - ab^{n+1}u_i + abu_{i+n}$ agree with those of v_{i+1} for $i_0 + 2 \leq i < n$. If such control bits can be found, our guess is correct with high reliability. In average we need about 2^4 trials for finding a suitable i_0 (satisfying the conditions for the first and the third step) and for each trial we have to guess $2^{3\ell}$ discarded bits. Checking a guess costs at most $4n$ additions of three summands. The attack works with about $2^4 + n$ outputs and its total cost is about $2^{4+3\ell}$ times the cost of computing two modular inverses and at most $4n + 1$ additions.

5.2 The Fast Generator Modulo 2^n

The attack of the prime field case does not directly translate to the case $R = \mathbb{Z}/m\mathbb{Z}$ for $m = 2^n$ (or to other rings). The problem is that, in general, the elements of a ring do not have a unique inverse. Hence, the divisions at the third step are not well defined. Instead, we have to find a and b such that

$$\begin{aligned} bv_{i_0} &= v_{i_0+1}, \\ v_{i_0+2} &= bv_{i_0+1} + ab. \end{aligned}$$

For some guesses, no such a and b exist. These guesses can be easily ruled out. But for other guesses, many choices for a and b are possible and the checking of them will be more costly.

6 Discussion

It was already noticed by von zur Gathen and Shparlinski in [3] that the security of the knapsack generator is smaller than suggested by a naive estimate based on its key length. The results in this paper show that it is no more than n bits. Our approach applies to all relevant parameters ℓ, including the following:

n	32	64	128
ℓ	≤ 25	≤ 42	≤ 98

.

In particular, it applies to $\ell = \lceil \log n \rceil$. The full attack works with only a few more outputs than the number of weights and it is not difficult to translate our analysis to the cases where k and n are not equal.

However, we could not circumvent the guessing part of our attack, for example, using ideas from fast correlation attacks. Due to the high nonlinearity of integer addition with multiple inputs, it seems very unlikely to find approximate linear relations between outputs or correlations to the state of the LFSR.

So far, a knapsack generator of size n provides n-bit security. To guarantee this security level it needs a $n(n+1)$ bit secret key, and the example of the fast knapsack generator shows that it is delicate to reduce the potential entropy of the weights. This has to be taken into account when evaluating the knapsack generator as an alternative to nonlinear boolean filtering and combining functions, or when using it in RFID applications.

Acknowledgements. We thank the anonymous reviewers for helpful comments. Especially, we thank the reviewer who pointed us to the use of LLL for computing the approximation matrices. This work was partially supported by European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II and by the Hasler Foundation `www.haslerfoundation.ch` under project number 08065.

References

1. Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica 6(1), 1–13 (1986)
2. Cole, P.H., Ranasinghe, D.C.: Networked RFID systems and lightweight cryptography: raising barriers to product counterfeiting. Springer-Verlag (2007)
3. von zur Gathen, J., Shparlinski, I.: Predicting Subset Sum Pseudorandom Generators. In: Handschuh, H., Hasan, M.A. (eds.) SAC. LNCS, vol. 3357, pp. 241–251. Springer (2004)
4. von zur Gathen, J., Shparlinski, I.: Subset sum pseudorandom numbers: fast generation and distribution. J. Math. Crypt. 3, 149–163 (2009)
5. Golic, J.D., Salmasizadeh, M., Dawson, E.: Fast Correlation Attacks on the Summation Generator. J. Cryptology 13(2), 245–262 (2000)
6. Hoffstein, J., Pipher, J., Silverman, J.H.: An introduction to mathematical cryptography. Springer (2008)

7. Howgrave-Graham, N., Joux, A.: New Generic Algorithms for Hard Knapsacks. In: Gilbert, H. (ed.) EUROCRYPT. LNCS, vol. 6110, pp. 235–256. Springer (2010)
8. Impagliazzo, R., Naor, M.: Efficient Cryptographic Schemes Provably as Secure as Subset Sum. J. Cryptology 9(4), 199–216 (1996)
9. Klapper, A., Goresky, M.: Feedback Shift Registers, 2-Adic Span, and Combiners with Memory. J. Cryptology 10(2), 111–147 (1997)
10. Lee, D.H., Kim, J., Hong, J., Han, J.W., Moon, D.: Algebraic Attacks on Summation Generators. In: Roy, B.K., Meier, W. (eds.) FSE. LNCS, vol. 3017, pp. 34–48. Springer (2004)
11. Meier, W., Staffelbach, O.: Correlation Properties of Combiners with Memory in Stream Ciphers. In: EUROCRYPT. pp. 204–213 (1990)
12. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (2001)
13. Merkle, R., Hellman, M.: Hiding information and signatures in trapdoor knapsacks. IEEE Transactions Information Theory 24(5), 525–530 (1978)
14. Rueppel, R.A.: Correlation Immunity and the Summation Generator. In: CRYPTO. LNCS, vol. 218, pp. 260–272. Springer (1985)
15. Rueppel, R.A.: Analysis and Design of Stream Ciphers. Springer (1986)
16. Rueppel, R.A., Massey, J.L.: Knapsack as a nonlinear function. IEEE Intern. Symp. of Inform. Theory 46 (1985)
17. Shamir, A.: A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem. In: CRYPTO. pp. 279–288 (1982)
18. Shoup, V.: NTL: A Library for doing Number Theory. `www.shoup.net/ntl`
19. Siegenthaler, T.: Correlation-immunity of nonlinear combining functions for cryptographic applications. IEEE Transactions on Information Theory 30(5), 776–780 (1984)
20. Staffelbach, O., Meier, W.: Cryptographic Significance of the Carry for Ciphers Based on Integer Addition. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO. LNCS, vol. 537, pp. 601–614. Springer (1990)