Evaluation of SOSEMANUK With Regard to Guess-and-Determine Attacks

Yukiyasu Tsunoo[1], Teruo Saito[2], Maki Shigeri[2], Tomoyasu Suzaki[2], Hadi Ahmadi[3], Taraneh Eghlidos[4], and Shahram Khazaei[5]

[1] NEC Corporation
1753 Shimonumabe, Nakahara-Ku, Kawasaki, Kanagawa 211-8666, Japan
tsunoo@BL.jp.nec.com
[2] NEC Software Hokuriku Ltd.
1 Anyoji, Hakusan, Ishikawa 920-2141, Japan
{t-saito@qh, m-shigeri@pb, t-suzaki@pd}.jp.nec.com
[3] School of Electrical Engineering, Sharif University of Technology, Tehran, Iran.
hadimc@mehr.sharif.edu
[4] Electronics Research Center, Sharif University of Technology, Tehran, Iran.
teghlidos@sharif.edu
[5] Zaeim Electronic Industries Company, P.O. BOX 14155-1434, Tehran, Iran.
khazaei@zaeim.com

Abstract. This paper describes the attack on SOSEMANUK, one of the stream ciphers proposed at eSTREAM (the ECRYPT Stream Cipher Project) in 2005. The cipher features the variable secret key length from 128-bit up to 256-bit and 128-bit initial vector. The basic operation of the cipher is performed in a unit of 32 bits i.e. "word", and each word generates keystream.
This paper shows the result of guess-and-determine attack made on SOSEMANUK. The attack method enables to determine all of 384-bit internal state just after the initialization, using only 2^4-word keystream. This attack needs about 2^{224} computations. Thus, when secret key length is longer than 224-bit, it needs less computational effort than an exhaustive key search, to break SOSEMANUK. The results show that the cipher has still the 128-bit security as claimed by its designers.
Key words: SOSEMANUK, ECRYPT, eSTREAM, stream cipher, pseudo-random number generator, guess-and-determine attack

1 Introduction

Everywhere, cipher standardization project has been encouraged vigorously. It is exemplified by the Advanced Encryption Standard (AES) [1], or the New European Schemes for Signatures, Integrity, and Encryption (NESSIE) project whose goal is to establish European standard cipher [3]. NESSIE project aims to choose secure cipher primitives, and in fact, they chose a stream cipher. However, many attacks against the stream ciphers proposed for NESSIE project were proposed during the 3-year evaluation phase, and finally, no stream cipher candidate remained. Thus, widespread attention is focused on stream cipher design and attacks against them.

In February 2004, European Network of Excellence for Cryptology (ECRYPT) was established. Its goal is to encourage the cooperation among European researchers on information security. In 2005, Symmetric Techniques Virtual Lab (STVL), a working group for ECRYPT established the ECRYPT Stream Cipher Project (eSTREAM), to call for papers on new stream ciphers [2]. Finally, 34 candidates were submitted to eSTREAM, which will complete 2 evaluation phases for those candidates by January 2008.

Stream ciphers submitted to eSTREAM include SOSEMANUK, which is proposed by Berbain et al. and features variable secret key length from 128-bit up to 256-bit and 128-bit initial vector [4]. The cipher allows faster software implementation, since its basic operation is performed in a unit of 32 bits i.e. "word", to generate keystreams. The structure and the name of SOSEMANUK are based on SNOW 2.0 stream cipher [6] and Serpent block cipher [5]. Berbain et al. assert that SOSEMANUK has overcome the vulnerability of SNOW 2.0, while reducing it in internal state size. However, they also assert that SOSEMANUK guarantees up to 128-bit security, regardless of the secret key length.

According to the evaluation made by designers of SOSEMANUK, with 2^{256} computations or less, guess-and-determine attack can not be made on the cipher. However, this paper reports that the attack can be made on it, with less computations than the necessary computational effort that those designers claim. This attack can recover all of 384 bits of internal state just after the initialization. The amount of data required for this attack is only about 2^4 words, which attackers can easily collect. The needed amount of computation is approximately 2^{224}. Thus, when secret key length is longer than 224-bit, it needs less computational effort than an exhaustive key search, to break SOSEMANUK.

Section 2 describes the structure of SOSEMANUK stream cipher and Section 3 is about how to make guess-and-determine attack against SOSEMANUK. Section 4 considers the structural vulnerability of SOSEMANUK and the countermeasure to the attack. Section 5 concludes this paper.

2 Description of SOSEMANUK

This section describes the structure of SOSEMANUK stream cipher. Since it employs the techniques originally used for Serpent, explanation on Serpent and its derivatives is given for the first, and then that on SOSEMANUK.

2.1 Serpent and Derivatives

Serpent [5] is the block cipher proposed by Biham et al. in 1998, and one of AES candidates. Serpent performs the operation called bit-slice to divide 32-bit 4-word data. Divided data are mixed and then, reunited into 32-bit 4-word data. SOSEMANUK defines two functions, *Serpent1* and *Serpent24*, as its derivatives.

Serpent1 is the round function of Serpent, with neither subkey addition by bitwise exclusive OR nor linear transformation. 8 distinct S-boxes ($S_0, \cdots, S_7$) are used for Serpent, while *Serpent1* uses S_2 only. *Serpent1* performs bit-slice to

divide 32-bit 4-word data, mix the divided data using S_2, and reunite them into 32-bit 4-word data, which is used as an output data.

Under full round, Serpent takes 32 round, and *Serpent24* is a reduced function of Serpent, which takes 24 rounds. Note that *Serpent24* inserts the 25th subkey after performing a linear transformation in the round function at the 24th round. Thus, *Serpent24* uses twenty-five 128-bit subkeys.

2.2 Keystream Generation

This subsection explains the keystream generation of SOSEMANUK, which can be grouped under roughly 3 parts; Linear Feedback Shift Register (LFSR), Finite State Machine (FSM), and Output Transformation.

LFSR consists of ten 32-bit registers, and is defined by the feedback polynomial over $GF(2^{32})$ as follows;

$$\pi(X) = \alpha X^{10} + \alpha^{-1} X^7 + X + 1$$

Here, α is a root of primitive polynomial $P(X)$ over $GF(2^8)$.

$$P(X) = X^4 + \beta^{23} X^3 + \beta^{245} X^2 + \beta^{48} X + \beta^{239}$$

β is a root of primitive polynomial $Q(X)$ over $GF(2)$.

$$Q(X) = X^8 + X^7 + X^5 + X^3 + 1$$

Since LFSR consists of primitive polynomial over $GF(2^{32})$, its 32-bit output sequence $\{s_t\}$ offers the maximal length cycle of $2^{320} - 1$.

FSM consists of two 32-bit registers. $(R1_t, R2_t)$ denotes the FSM registers at the given time t $(t \geq 1)$. With the equations described below, FSM updates registers $(R1_t, R2_t)$ and generates 32-bit output f_t. Hereafter, $\oplus$ denotes bit-wise exclusive OR, whereas $+$ and $\times$ mean addition and multiplication over mod 2^{32}.

$$\begin{aligned} R1_t &= (R2_{t-1} + mux(lsb(R1_{t-1}), s_{t+1}, s_{t+1} \oplus s_{t+8})) \\ R2_t &= Trans(R1_{t-1}) \\ f_t &= (s_{t+9} + R1_t) \oplus R2_t \end{aligned}$$

Here, $lsb(x)$ means the least significant bit of data x, and $mux(c, x, y)$ means the function where x is output, if $c = 0$, while y is output, if $c = 1$. The function $Trans(x)$ is defined as follows;

$$Trans(x) = (M \times x) \lll 7$$

Here, constant $M = 0x54655307$, and $x \lll 7$ denotes that 32-bit data x is 7-bit rotated to the left (towards the most significant bit). Figure 1 is an overview of SOSEMANUK.

Using FSM output f_t and LFSR register s_t, Output Transformation generates keystreams. z_t represents the keystream at given time t $(t \geq 1)$. Output Transformation processes 32-bit 4-word data, i.e. the data for 4 different ts at a time, and uses the equation below to generate the keystream for 4 different ts $(z_{t+3}, z_{t+2}, z_{t+1}, z_t)$;

$$(z_{t+3}, z_{t+2}, z_{t+1}, z_t) = \mathit{Serpent1}(f_{t+3}, f_{t+2}, f_{t+1}, f_t) \oplus (s_{t+3}, s_{t+2}, s_{t+1}, s_t)$$

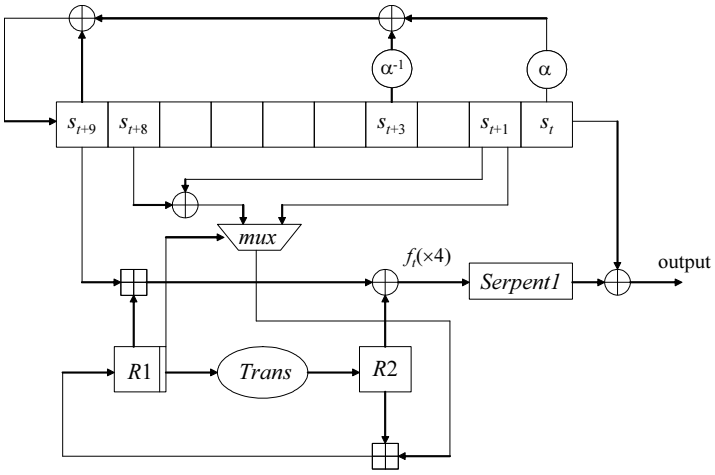

Fig. 1. An overview of SOSEMANUK

2.3 Initialization

This subsection describes the initialization of SOSEMANUK. As initialization, SOSEMANUK generates the initial value of internal state, using the key schedule of Serpent and *Serpent24*. Figure 2 illustrates the initialization of SOSEMANUK.

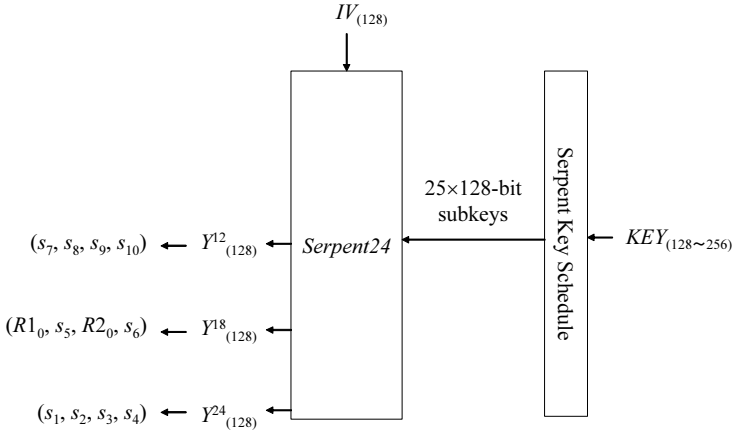

Fig. 2. Initialization of SOSEMANUK

SOSEMANUK takes secret key KEY as an input to key schedule of Serpent, to generate twenty-five 128-bit subkeys. Though the secret key of SOSEMANUK is variable, ranging from 128-bit up to 256-bit, Serpent's secret key is also variable, ranging from 1-bit from 256-bit. Thus, key scheduler can be operated in

accordance with the secret key length. After the subkey generation, initial vector IV is taken as an input to *Serpent24*. Then, intermediate data of rounds 12 and 18 of *Serpent24*, and output data of round 24 of *Serpent24* are used as initial values of internal state. Providing that Y^{12}, Y^{18}, and Y^{24} denote the outputs of rounds 12, 18, and 24, respectively, these three data are substituted in the equations below, as the initial values of their respective registers. Here, LFSR register and FSM register at the completion of initialization are represented by $(s_{10}, s_9, \cdots, s_1)$ and $(R1_0, R2_0)$, respectively.

$$\begin{aligned} s_7 \parallel s_8 \parallel s_9 \parallel s_{10} &= Y^{12} \\ R1_0 \parallel s_5 \parallel R2_0 \parallel s_6 &= Y^{18} \\ s_1 \parallel s_2 \parallel s_3 \parallel s_4 &= Y^{24} \end{aligned}$$

3 Cryptanalysis of SOSEMANUK

This subsection explains how to apply guess-and-determine attack against SOSEMANUK whose secret key size is 256-bit. Followings are the preconditions for the attack;

- Fixed secret key value during the attack.
- Attackers can obtain some quantity of keystream.

Assume that time t satisfies the assumption given below, when guess-and-determine attack is made against SOSEMANUK.

Assumption : $lsb(R1_{t-1}) = 0$

If the Assumption is satisfied, register $R1_t$ is updated with the following equation;

$$R1_t = R2_{t-1} + s_{t+1}$$

As is apparent from the equation given above, $R1_t$ is not influenced by s_{t+8}, when the Assumption is satisfied. Thus, it can be used for cryptanalysis.

Assuming that $t = 1$ satisfies the Assumption, guess the internal state just after the initialization described below.

Guess 1 : $s_1, s_2, s_3, s_4, R1_0, R2_0$

Since $lsb(R1_0) = 0$, from Assumption, 191 bits need to be guessed. Eqs. (1) and (2) are the ones to update registers $(R1_1, R2_1)$, respectively, where $t = 1$, while Eqs. (3) and (4) are used to update register s_{11}, and to generate FSM output f_1.

$$R1_1 = (R2_0 + s_2) \tag{1}$$
$$R2_1 = Trans(R1_0) \tag{2}$$
$$s_{11} = s_{10} \oplus \alpha^{-1}(s_4) \oplus \alpha(s_1) \tag{3}$$
$$f_1 = (s_{10} + R1_1) \oplus R2_1 \tag{4}$$

Based on Guess 1, the values of registers $R1_1$ and $R2_1$ can be calculated. (f_1, f_2, f_3, f_4), FSM outputs for 4 consecutive times can also be obtained, using the equation of Output Transformation, where the inverse function of *Serpent1* is represented by $Serpent1^{-1}$.

$$(f_4, f_3, f_2, f_1) = Serpent1^{-1}(z_4 \oplus s_4, z_3 \oplus s_3, z_2 \oplus s_2, z_1 \oplus s_1)$$

Thus, s_{10} and s_{11} can be determined, using Eqs. (4) and (3), respectively. In this way, guessing a part of internal state allows determining remaining undetermined part of internal state. Figure 3 shows how the cryptanalysis is performed, where $t = 1$.

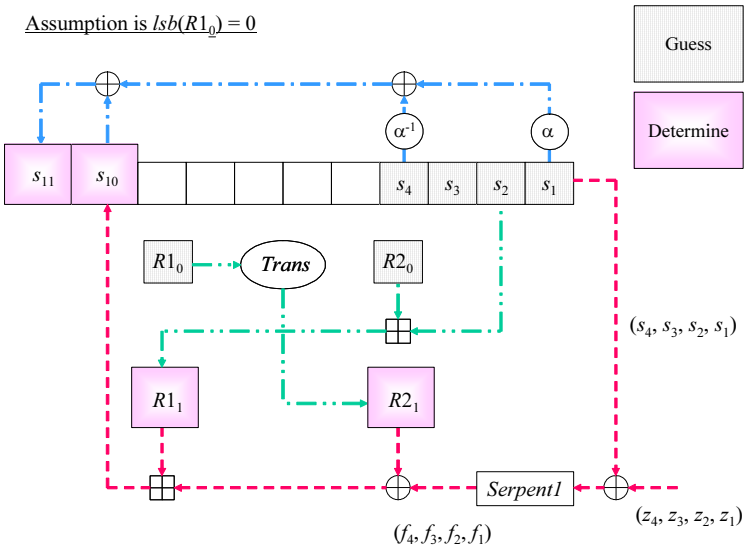

Fig. 3. Cryptanalysis steps where $t = 1$

Then, calculate $(R1_2, R2_2)$ where $t = 2$, using the equation to update FSM. Now, pay attention to the equation to generate FSM output f_2.

$$f_2 = (s_{11} + R1_2) \oplus R2_2 \tag{5}$$

Here, s_{11} is known data determined where $t = 1$. As f_2, $R1_2$, and $R2_2$ are also the data determined by Guess 1, any errors in Guess 1 must result in contradiction in Eq. (5). Thus, if any contradiction is found in Eq. (5), it means some error in Guess 1. Then, attackers can drop the candidate data. These steps narrow down the candidates for Guess 1 to about 2^{-32} of its original number.

Similarly, the data for $t = 3$ can be determined. Using the equation to update FSM, calculate $R1_3$, and $R2_3$. With equation to generate FSM output f_3, it is

possible to calculate s_{12}. Now, substituting s_{12} into equation to update LFSR, where $t = 2$, it becomes as follows;

$$s_5 = \alpha(s_{12} \oplus s_{11} \oplus \alpha(s_2))$$

Thus, s_5 can be determined, too.

Taking similar steps to above allows to determine unknown internal state where $t = 4$, and then, $R1_4$, $R2_4$, s_6, and s_{13} can be calculated. So far, 191 bits needed to be guessed, and candidates for those were narrowed down to about 2^{159}

Take similar steps to determine the internal state where $t = 5$ through 8. Since s_5, and s_6 are known data where $t = 1$ through 4, Guess 2 given below is to be done.

Guess 2 : s_7, s_8

Beside the bits determined already, 64 bits need to be guessed. With Guess 2, (f_5, f_6, f_7, and f_8) can be determined, using Output Transformation equation. Also, in the course of determining the internal state in a similar way, attackers can check to see if there is any contradiction in Guess 1 and/or Guess 2 at 3 times, using the equation to generate FSM output f_t. This step narrow down the number of candidates to around 2^{-96} of its original number. Thus, attackers have guessed 223 bits so far, and the candidates for them can be narrowed down to 2^{127}. Consecutive registers of internal state, i.e. $R1_t$, $R2_t$ ($t = 1, \ldots, 8$) and s_t ($t = 1, \ldots, 18$) can also be determined.

Take similar steps to determine the internal state where $t = 9$ through 12. Since s_9, s_{10}, s_{11}, and s_{12} are known data where $t = 1$ through 8 no more guess has to be performed. As attackers can check to see if there is any contradiction at 4 times, using the equation to generate FSM output f_t, In the course of determining the internal state, guessed candidates at Guess 1 and Guess 2 can be narrowed down to about 2^{-128} of its original number. This means that theoretically, all the 384 bits of internal state just after the initialization can be determined uniquely. Table 1 lists determined registers at each t and the number of candidates to be narrowed down at Guess 1 and Guess 2.

Finally, the amount of computation required for this attack is estimated. At Guess 1 and Guess 2, 223 bits at most have to be guessed. The success probability of Assumption is the probability that the least significant bit of register $R1_0$ becomes 0. Thus, it becomes 1/2, if initialization of SOSEMANUK offers completely random transformation. Consequently, the amount of computation required for the cryptanalysis T is determined as follows; [1]

$$T = 2^{223} \times 2^1 = 2^{224}$$

[1] Large memory is not needed for this attack, because the candidates for the registers are tried one by one.

Table 1. Registers determined at t and number of candidates at Guess 1 and Guess 2

t	Registers to Guess	Determined Registers	Contradiction Check	Number of Candidates
1	$s_1, s_2, s_3, s_4, R1_0, R2_0$	$s_{10}, s_{11}, R1_1, R2_1$		2^{191}
2		$R1_2, R2_2$	$\surd$	2^{159}
3		$s_5, s_{12}, R1_3, R2_3$		
4		$s_6, s_{13}, R1_4, R2_4$		
5	s_7, s_8	$s_{14}, s_{15}, R1_5, R2_5$	$\surd$	2^{191}
6		$R1_6, R2_6$	$\surd$	2^{159}
7		$s_9, s_{16}, s_{17}, R1_7, R2_7$		
8		$s_{18}, R1_8, R2_8$	$\surd$	2^{127}
9		$s_{19}, R1_9, R2_9$	$\surd$	2^{95}
10		$s_{20}, R1_{10}, R2_{10}$	$\surd$	2^{63}
11		$s_{21}, R1_{11}, R2_{11}$	$\surd$	2^{31}
12		$s_{22}, R1_{12}, R2_{12}$	$\surd$	1

If Assumption is not satisfied at $t = 1$, the attack will fail. However, assuming that Assumption is satisfied at $t = 5$, that is shifted from $t = 1$ by 4 times, the similar attack seems to break the cipher, successfully.

Guess-and-determine attack needs to compare the keystream that attackers obtained with keystreams output by cipher for 12 times, in order to determine candidates for Guess 1 and Guess 2, uniquely. Thus, taking the probability that Assumption is satisfied at any given t into account, only about 2^4 word of keystream is enough for success of the attack.

4 Discussion

This section discusses the vulnerability in SOSEMANUK structure and countermeasures to guess-and-determine attack. Designers of SOSEMANUK claimed that they eliminated the weakness in SNOW 2.0 structure and reduced the internal state size, in order to increase the suitability of SOSEMANUK to be implemented on a processor of any kind. They also employ reduced version of Serpent block cipher, as initialization process in SOSEMANUK, to increase its security.

However, when SOSEMANUK uses a longer than 224-bit secret key, guess-and-determine attack could be made successfully, with less amount of computation than an exhaustive key search. This indicates that the security provisions for SOSEMANUK made against existing attacks were not enough. It is considered that guess-and-determine attack was made successfully, directly because of

– smaller internal state size

– less LFSR feedback polynomial taps

To increase the suitability to be implemented, internal state size of SOSEMANUK is 384 bits, rather smaller than 576 bits, that of SNOW 2.0. It is considered that this modification allows attackers guess most part of internal state by guessing less bit-size than secret key size. Thus, the change in internal state size helps attackers to apply guess-and-determine attack. It is also considered that more registers in LFSR feedback polynomial, which are responsible for data updating, perhaps provide SOSEMANUK with more security, because they increase the required amount of bits to be guessed. Though the designers regard using reduced version of Serpent for initialization as a refinement in SOSEMANUK, it does not seem to work, as far as the attack proposed in this paper concerns, because even completely random transformation as initialization has no effect on the attack.

Hereafter, countermeasures to the attack proposed in this paper is discussed. Countermeasures described below are suggested;

– Elongate the internal state size enough.
– Increase LFSR feedback polynomial taps in number.

As described earlier, taking two countermeasures makes it difficult for attackers to make guess-and-determine attack, since those countermeasures increase amount of computation that is required for cryptanalysis. The countermeasures described above merely aim to provide the cipher with more resistance to guess-and-determine attack. We have not studied how they work on other existing attacks. When making improvements to the cipher, application of the countermeasure must be examined, so that it may not reduce the resistance to each of existing attacks.

5 Conclusion

This paper describes the attack on SOSEMANUK proposed as an improved algorithm of SNOW 2.0 in 2005. It is true that those who proposed the cipher added more security to SOSEMANUK. However, we demonstrated that guess-and-determine attack can be made on SOSEMANUK. This attack method can determine all of 384-bit internal state just after the initialization, using only 2^4-word keystream, the amount of data that attackers can easily collect. This attack needs about 2^{224} computations. Thus, when secret key length is longer than 224-bit, it needs less computational effort than an exhaustive key search, to break SOSEMANUK.

Our way of applying guess-and-determine attack proposed in this paper breaks SOSEMANUK more efficiently than the designers of SOSEMANUK expected. However, note that our method does not break the cipher whose security level is 128-bit, with 2^{128} computations or less. Since this attack method requires very large amount of computation, it can not be a practical threatening to SOSEMANUK. But, in the terms of extremely little amount of data needed for the attack, it can be said a very strong attack.

This paper discusses not only the structural vulnerability of SOSEMANUK but also countermeasures to guess-and-determine attack. Designers of SOSEMANUK pursued the advantages in implementation, to reduce the internal state size. This, however, resulted in vulnerability to guess-and-determine attack. Judging from existing attacks against stream ciphers, the size of internal state is a critical point for the security. Consequently, in terms of security as well as implementation.

Acknowledgement

The authors would like to thank the anonymous referees of SASC 2006 whose helpful remarks that improved the paper. The authors also would like to thank Shunsuke Ando for his useful comments.

References

1. AES, the Advanced Encryption Standard, NIST, FIPS-197.
   Available at http://csrc.nist.gov/CryptoToolkit/aes/
2. eSTREAM, the ECRYPT Stream Cipher Project.
   Available at http://www.ecrypt.eu.org/stream/
3. NESSIE, the New European Schemes for Signatures, Integrity, and Encryption.
   Available at https://www.cosic.esat.kuleuven.ac.be/nessie/
4. C. Berbain et al.: "SOSEMANUK, a fast software-oriented stream cipher," *eSTREAM, the ECRYPT Stream Cipher Project*, Report 2005/027, 2005.
5. E. Biham, R. Anderson, and L. Knudsen: "Serpent: A New Block Cipher Proposal," *Fast Software Encryption, FSE 1998*, LNCS 1372, pp.222-238, Springer Verlag, 1998.
6. P. Ekdahl and T. Johansson: "A New Version of the Stream Cipher SNOW," *Selected Areas in Cryptography, SAC 2002*, LNCS 2295, pp.47-61, Springer Verlag, 2002.