



Project Number	IST-1999-12324
Project Title	NESSIE
Deliverable Type	Report
Security Class	Public
Deliverable Number	D19
Title of Deliverable	Algorithm Specific Toolbox
Document Reference	NES/DOC/SAG/WP2/049/3
Contractual Date of Delivery	Y3 M6
Actual Date of Delivery	Y3 M6
Editors	Marcus Schafheutle, SAG
Abstract	<p>The purpose of this document is to describe the developed algorithm specific NESSIE tools. In contrast to the statistical tests of the NESSIE test suite, each of these tools is specific for only one individual algorithm. So their reusability is rather limited. Due to this fact only short descriptions of these tools are given. The software provided with these tools is not publicly available.</p>
Keywords	NESSIE, Toolbox.

Version 1.2

April 18, 2003

NESSIE Document NES/DOC/SAG/WP2/049/3 * **

NESSIE Deliverable D19: Algorithm specific Toolbox

Marcus Schafheutle

Siemens AG, Corporate Technology, 81730 München, Germany, Email:
marcus.schafheutle@mchp.siemens.de

1 Introduction

The purpose of this document is to describe the developed algorithm specific NESSIE tools. In contrast to the statistical tests of the NESSIE test suite, each of these tools is specific for only one individual algorithm. So their reusability is rather limited. Due to this fact only short descriptions of these tools are given. The software provided with these tools is not publicly available.

2 NESSIE algorithm specific tools

2.1 Specific tools for the submission ANUBIS

- name of the tool: anubisintegral
This tool has been used to test the square attack on ANUBIS. It has also tried to extend this attack to more rounds by considering larger sets of balanced plaintexts.
The software for this tool was written in the programming language C.
- name of the tool: anubislintest
This tool searches for linear functions over $GF(2^n)$ that agree with the S-box function on as many inputs as possible. After the program had run for a week it had found a linear function that was identical to the ANUBIS S-box on 15 inputs. If a linear function that agreed in more points had been found, one might have used this function as a basis of an attack.
The software for this tool was written in the programming language C.

2.2 Specific tool for the submission LEVIATHAN

- name of the tool: leviathanbias
In [CL01] two biases in the LEVIATHAN function that distinguished it from a random function were presented. The PRF-PRF attack shows that the probability of a collision in 64 bit output words is doubled compared to a random function. The S-box matching attack exploits a weakness in the key-dependent S-box of LEVIATHAN. This weakness causes a specific xor pattern of two successive LEVIATHAN output words to be twice as likely to occur as the same pattern for a random function. The authors of [CL01] demonstrated both biases with experiments on a reduced version of the stream cipher. This tool implements the S-box matching attack for the full version of LEVIATHAN. The result of this S-box matching attack is described in [Sch01].
The software for this tool was written in the programming language C.

* The work described in this paper has been supported by the Commission of the European Communities through the IST program under contract IST-1999-12324.

** The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

2.3 Specific tools for the submission NOEKEON

- name of the tool: noekeonchar32

The aim of this tool is to check if the number of good pairs following a specific differential characteristic is roughly what would be expected when assuming independent round keys. This was done to verify that even though the same key is used in all rounds in NOEKEON, one can not expect a differential characteristic over several rounds to occur with higher than usual probability. This program uses a scaled-down implementation of NOEKEON, where the plaintext block is 32 bits. The program has tested all 2^{31} pairs of input with a fixed difference that starts a characteristic.

The software for this tool was written in the programming language C.

- name of the tool: noekeonchar64

This tool does the same as noekeonchar32, but on a version where the plaintext block is 64 bits long. The program has been run with 2^{36} pairs of input of a specific difference, to check that the number of pairs that follow the characteristic is roughly what is to be expected when assuming independent round keys.

The software for this tool was written in the programming language C.

- name of the tool: noekeonsquare

This tool records how often $p[i] \wedge p[j] = c[k]$, where $p[i]$ and $p[j]$ are the i th and j th bits in the plaintext block, \wedge is the AND-operation, and $c[k]$ is the k th bit in the ciphertext block. For each plaintext/ciphertext pair, this relation is tested for each i, j, k with $0 < i, j, k < 128$ and $i < j$. At the end, the 10 relations giving the highest bias are printed out. The program has been run on 2^{19} plaintext/ciphertext pairs.

The software for this tool was written in the programming language C.

- name of the tool: noekeonsbox

This tool generates 4-bit self-inverse S-boxes at random, and checks whether the differential and linear requirements stated in the NOEKEON submission are met. For each S-box that meets these requirements, it checks if a linear or differential attack would be applicable if this S-box was used instead of the one given in the submission. The program found 10000 S-boxes fulfilling the differential and linear requirements, of which 86% were vulnerable to a differential or linear attack.

The software for this tool was written in the programming language C.

2.4 Specific tools for RC4

- name of the tool: block counting test

The original idea was to add to the NESSIE test toolbox a test for revealing the irregular statistics in the second output byte of RC4 detected by A. Shamir [MS01]. This defect could of course be detected by the dependence test, e.g., as a side effect, but we preferred to provide a specific test for this purpose.

The new test, called the block counting test, studies functions from a finite bit string to a finite bit string. The output string is split into a number of subsequent, non-overlapping blocks of a certain size. This block size is one of the parameters of the test. For random inputs, the frequencies of the occurrences of the output blocks are recorded and evaluated statistically by means of the χ^2 test. For each position in the sequence of output blocks, the frequencies are evaluated individually.

The block counting test output contains – for each position in the sequence of output blocks – how often each possible value occurs (if the block size is less than or equal to 8), the expected value (i.e. how often each possible value should occur), the number of degrees of freedom, the χ^2 value and the corresponding percentage level of acceptance, i.e. the probability that a truly random sequence has a greater χ^2 value. To verify the RC4 problem, the test would have to be applied to the function that maps RC4 keys to a finite part of the bytes generated by RC4, e.g. the first two bytes.

It makes sense to apply the block counting test not only to stream ciphers but also to block ciphers, if the key is considered as input and the plaintext is fixed, hash functions and MACs.

And for MACs, we may even consider two cases: fixed input and variable key, and variable input and fixed key. For a detailed description of this test please refer to [Ser02].

The software for this tool was written in the programming language C.

- name of the tool: rc4tool

Following Mantin and Shamir's attack presented at FSE'01 [MS01], which showed a bias in the second byte output of RC4, and proposed a conjecture about this. This tool was written to study this behaviour of the RC4 stream cipher.

The internal state of RC4 consists of an array S of 256 bytes describing a permutation and two indexes i and j . Given a key, the initialisation of the states fills the array with an apparently random value and sets i and j to 0. Each step of the algorithm does the following operations modulo 256: $i++$; $j+=S[i]$; $S[i]\leftrightarrow S[j]$; output $S[S[i]+S[j]]$.

The second byte of the stream output appears to have some bias, because all but one of the initial states having a 0 in $S[2]$ outputs a 0 after two rounds. This is called a 1-predictive 1-state in the paper.

This tool studies predictive states for reduced versions of RC4, by exhaustive search.

The conjecture in [MS01] states that b -predictive a -states exists only for $a \geq b$. This is trivially false for a close to N (the length of the array), but an evaluation of the maximal value of b for small a can lead to generalisations of Mantin and Shamir's attack.

The software for this tool was written in the programming language C.

2.5 Specific tools for the submission SC2000

- name of the tool: sc2000diff

This tool searches for differential characteristics of low Hamming weight through the layer of the six S-boxes and the matrix multiplication found in the Feistel rounds of SC2000. For each 32-bit string s of weight four or less, it checks if there is a characteristic that starts and ends in s . If not, it counts the number of S-boxes that need a different input in order to fit this characteristic. This program has been used to find better differential characteristics than were found by the designers of SC2000.

The software for this tool was written in the programming language C.

- name of the tool: sc2000test

This tool checks whether the best differential characteristic found using the tool sc2000diff actually has the probability found in the analysis of that characteristic, that is 2^{-18} .

The software for this tool was written in the programming language C.

2.6 Specific tool for the submission SHACAL

- name of the tool: shacalchar

This tool can be used to search for characteristics over each of the 4 'rounds' of SHACAL.

It searches the subspace of characteristics that is likely to contain the characteristics with the highest probability. The subspace is defined as follows: the starting value of the characteristic is split up in 5 32-bit words, where the Hamming weight is at most 2 in each word. (The authors of SHACAL restricted this Hamming weight to 1)

The software for this tool was written in the programming language C.

2.7 Specific tool for the submission QUARTZ

- name of the tool: hfe

For the QUARTZ signature algorithm, it is possible to find two different signatures for the same message [JM]. This property can be used to find a lot of equations with coefficients that depend upon the secret key. This tool was written in order to verify whether this property can be used to recover a part or all of the secret key.

The software for this tool was written in the programming language C++.

References

- [CL01] P. Crowley and S. Lucks, *Bias in the LEVIATHAN stream cipher*, Proceedings of FSE '01, Lecture Notes in Computer Science, Springer-Verlag, 2001.
- [JM] A. Joux and G. Martinet, *Cryptanalysis of QUARTZ*, Tech. report, unpublished.
- [MS01] I. Mantin and A. Shamir, *A practical attack on broadcast RC4*, Proceedings of FSE '01, Lecture Notes in Computer Science, Springer-Verlag, 2001.
- [Sch01] M. Schafheutle, *Statistical attacks on the stream cipher LEVIATHAN*, Tech. report, NES/DOC/SAG/WP3/027/1, 2001.
- [Ser02] P. Serf, *The block counting test*, Tech. report, NES/DOC/SAG/WP2/047/1, 2002.