In a first phase, NESSIE developed tools for testing submissions that were not algorithm specific. In a second phase, tools which are specific for individual submissions are developed. The need for such tools must be identified in the course of the security evaluation of the submissions.
Introduction

In a first phase, NESSIE developed tools for testing submissions, which were not algorithm specific. In a second phase, tools which are specific for individual submissions are developed. The need for such tools must be identified in the course of the security evaluation of the submissions.

List of primitive specific tools

Name of the tool: Leviathan-32-bits-comparison
NESSIE submission for which the tool is specific: Leviathan
Programming language used: C

Property which is tested by the tool:

P. Crowley and S. Lucks showed in their paper “Bias in the Leviathan stream cipher” presented at FSE 2001 that subsequent 32-bit-words in the output of Leviathan are equal with a probability of $2^{-31}$. For a good stream cipher this probability should be $2^{-32}$. They show this problem of Leviathan theoretically, and also give experimental results for a reduced version of Leviathan with a word size of 16 bits.

In order to verify this problem for the full 32-bit-version of Leviathan, software which counts the frequency of equal subsequent 32 bit words is needed.

Name of the tool: HFE
NESSIE submission for which the tool is specific: Quartz
Programming language used: C++

Property which is tested by the tool:

It's really easy to find two different signatures for a same message. This property can be used to find a lot of equations according to the coefficients of the secret key. In order to verify if this property can be used to recover a part or all the secret key, software for general HFE schemes is needed. We can vary the parameters to study the efficiency of the attack depending of this choice.

Name of the tool: SC2000diff.c
NESSIE submission for which the tool is specific: SC2000
Programming language used: C

Property which is tested by the tool:

A program that 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 what the designers of SC2000 found.
Name of the tool: SC2000test.c
NESSIE submission for which the tool is specific: SC2000
Programming language used: C
Property which is tested by the tool:
A program that checks that the best differential characteristic found using SC2000diff.c actually has the probability found in the analysis of that characteristic, that is \(2^{-18}\).

Name of the tool: Noekeonchar32.c
NESSIE submission for which the tool is specific: Noekeon
Programming language used: C
Property which is tested by the tool:
A program that checks that the number of good pairs following a specific differential characteristic is roughly what to 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 higher probability than usual in a differential characteristic over several rounds. 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.

Name of the tool: Noekeonchar64.c
NESSIE submission for which the tool is specific: Noekeon
Programming language used: C
Property which is tested by the tool:
This program does the same as Noekeonchar32.c, 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 to be expected when assuming independent round keys.

Name of the tool: Noekeonsquare.c
NESSIE submission for which the tool is specific: Noekeon
Programming language used: C
Property which is tested by the tool:
This program records how often \(p[i] \text{ AND } p[j] = c[k]\), where \(p[i]\) and \(p[j]\) are the \(i\)'th and \(j\)'th bits in the plaintext block, and \(c[k]\) is the \(k\)'th bit in the ciphertext block. For each plaintext/ciphertext pair, this relation is tested for each \(i, j, k\), \(0 < i, j, k < 128\) 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.
Name of the tool: Noekeonbox.c
NESSIE submission for which the tool is specific: Noekeon
Programming language used: C
Property which is tested by the tool:
This program 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 10,000 S-boxes fulfilling the differential and linear requirements, of which 86% were vulnerable to a differential or linear attack.

Name of the tool: Anubisintegral.c
NESSIE submission for which the tool is specific: Anubis
Programming language used: C
Property which is tested by the tool:
This program 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.

Name of the tool: Anubislintest
NESSIE submission for which the tool is specific: Anubis
Programming language used: C
Property which is tested by the tool:
A program that 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.

Name of the tool: RC4tool
Algorithm which the tool is specific: RC4
Programming language used: C
Property which is tested by the tool:
Following Mantin and Shamir's attack presented at FSE'01 (I. Mantin and A. Shamir, A practical attack on broadcast RC4, FSE'01, Yokohama, LNCS, Springer, 2001), which showed a bias in the second byte output of RC4, and which proposed a conjecture on this, a tool has been written to study this behaviour of the RC4 stream cipher.
The internal state of RC4 consist 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:
\[
i++; \ j+=S[i]; \ S[i] \leftrightarrow S[j]; \ \text{output } S[S[i]+S[j]].
\]
The second byte of the stream output appears to have some bias, because all but one initial states having a 0 in $S[2]$ output 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 the paper of Mantin and Shamir states that $b$-predictive $a$-states exist 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.

**Further primitive specific tools**

This list will be extended as soon as NESSIE identifies the need for further primitive specific tools.