Nothing better than a Python to write a Serpent

Frank Stajano
Olivetti Oracle Research Laboratory
& University of Cambridge Computer Laboratory

Serpent is a 128-bit block cipher designed by Ross Anderson, Eli Biham and Lars Knudsen as a candidate for the Advanced Encryption Standard, a competition sponsored by the National Institute of Standards and Technology to define a successor to DES. Serpent is designed to be faster than DES and more secure than triple-DES. It is in the public domain and there are no restrictions on its use: the full AES submission package, including the full specification of the cipher, a reference implementation in C, several test suites, and optimised versions in C and Java, is available from

http://www.cl.cam.ac.uk/~rja14/serpent.html

The authors of Serpent had a preliminary C implementation and wanted to cross-check it with an independently developed one.

I started from the paper defining Serpent and implemented the algorithm from scratch. My only concerns were correctness and readability, with the view that "premature optimisation is the root of all evil". For this reason I chose Python, and I represented bit strings simply as little-endian strings of the characters "0" and "1". This gave me a "virtual processor" with words of arbitrary length. I could now easily XOR, rotate, extract bit 103 and so on, as well as assigning a block or key to a variable and inspecting it (even interactively) simply by printing it out.

It must be realised that implementing a cryptographic primitive such as a block cipher from scratch is very much an open-loop operation, in that there is no obvious feedback on whether the output is correct or not. After all, the whole point of a block cipher is to transform the plaintext into something that resembles random garbage as closely as possible — so how is one supposed to know whether the garbage that comes out is "good" garbage or the outcome of some internal bugs? Compare this with the much easier closed-loop task of writing a new implementation after a known-good reference implementation is available: bugs are immediately identified by the fact that the output is different from that of the reference, and by tracing back the intermediate results one is quickly led to the point of divergence and thus to the probable cause of the bug.

Short of going all the way to formal verification methods, a good solution seems to be that of making the code as clear and simple as possible, aiming for a one-to-one correspondence between the code and the formulae describing the algorithm in its specification. A very high-level language such as Python, with its "executable pseudocode" flavour, was found to be great for this task.

The Python version was instrumental in finding a couple of bugs in the C code that, while conceptually trivial, were still of course affecting the workings of the cipher. The absolutely unoptimised Python code, while it ran at the appalling speed of about one block per second (the C code was 4 to 5 orders of magnitude faster), was the first implementation to produce the correct results. I later had to rewrite an unoptimised reference version in C, since that's what the call for algorithms required, but this task was made much more pleasant (and closed-loop) by being able to rely on the existing Python one.

If there is a moral to this story, it must be that the level of abstraction of Python makes it a good language for expressing ideas without getting too bogged down in machine-level details, and that "ideas in executable form" are a good complement to the formal specification of an algorithm.

The Python reference implementation of Serpent is freely available from

http://www.cl.cam.ac.uk/~fms27/serpent/