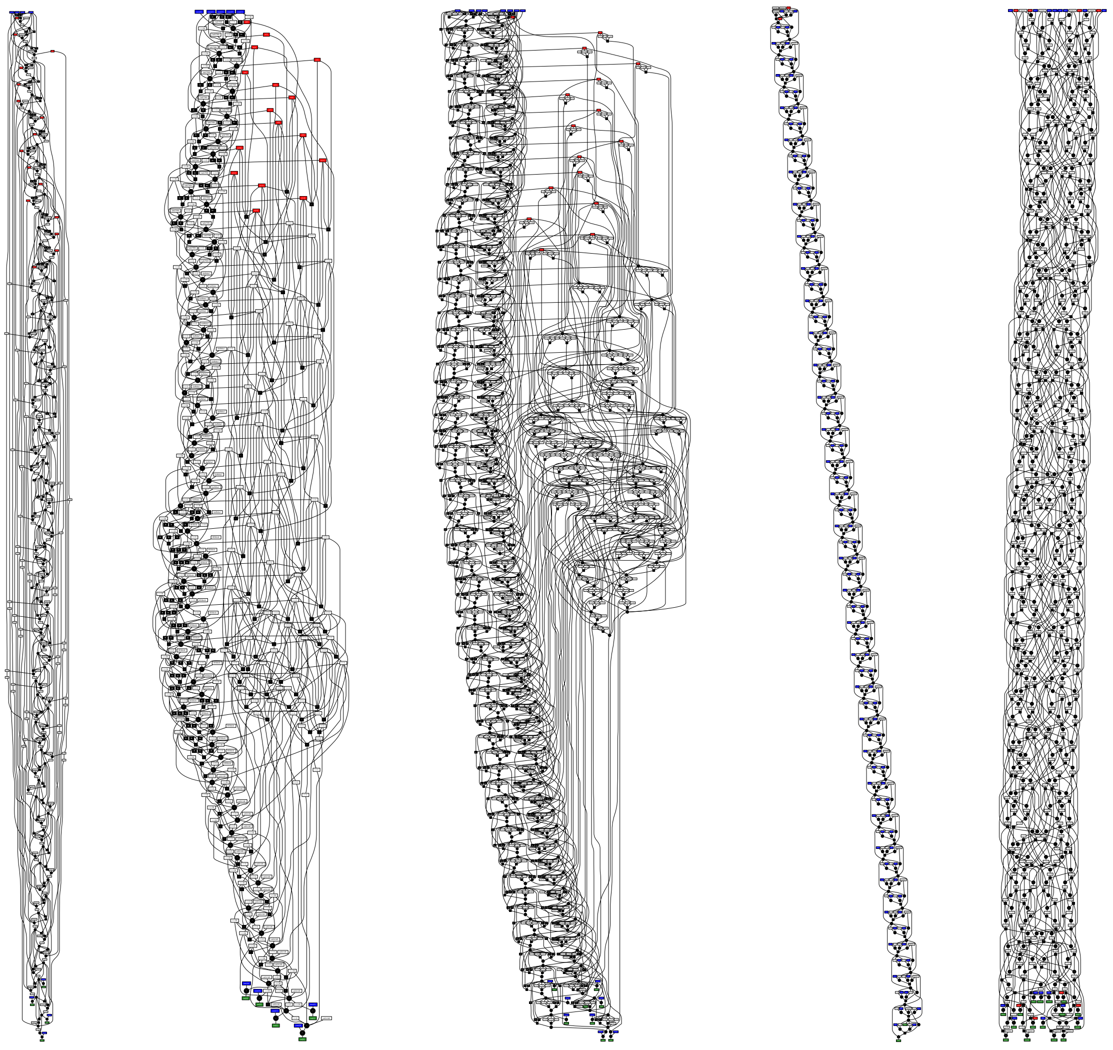
Cipher DAGs (extended abstract)

Daniel J. Bernstein

Department of Mathematics, Statistics, and Computer Science (M/C 249)
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to



[*] Permanent ID of this document: `ecf59fdc3229562c4bcf7d4214b335a4`. Date of this document: 2007.06.30. This work was carried out while the author was visiting Technische Universiteit Eindhoven.

The first graph on the previous page is a directed acyclic graph (DAG) that computes the MD5 hash function. The graph consists of 480 labelled nodes and 812 edges. Each edge carries data from a node higher in the graph to a node lower in the graph. Each node computes 32 bits: for example, a node labelled “+” adds its inputs modulo 2^{32}, a node labelled “$\lll 7$” rotates its 32-bit input left by 7 bits, a node labelled “x[0]” copies the first 32-bit word from the MD5 input, and a node labelled “out[3]” copies a 32-bit word to the end of the MD5 output.

Of course, nodes and labels and edges in a large DAG are much easier to see on a computer screen, with appropriate zooming and panning in two or three dimensions, than on a printout. A computer display also has the virtue of allowing extremely fast modifications—for example, users can experiment with input differentials to see how the differentials propagate through the DAG.

One can draw a DAG for any stream cipher, block cipher, hash function, etc. There are five examples on the previous page: DAGs for MD5, SHA-1, SHA-256, TEA, and Salsa20. (The graphs have been scaled to the same vertical height—another unfortunate consequence of being squeezed onto the printed page.) The right half of the SHA-1 graph is the SHA-1 message expansion, and the right half of the SHA-256 graph is the more complex SHA-256 message expansion; for comparison, the MD5 graph has many long edges, allowing an attacker to effortlessly pierce deep into the heart of the MD5 computation.

Objection: All five of these functions are “32-bit algorithms” built from a small selection of 32-bit operations. Response: *Every* circuit can be built from a single type of gate, namely the bit NAND. One can therefore draw every limited-time computation as a DAG consisting of bit NANDs. Of course, restricting the power of a node makes DAGs larger and more difficult to draw; my initial experiments with bit DAGs for MD5 have crashed every standard drawing tool that I’ve tried. My own drawing tools are much more careful in their use of memory.

(I haven’t tried drawing a DAG for an RC4-style stream cipher. A single write to a 256-element array at a variable index would produce an impressively large bit-level DAG, reflecting the 256 potentially new array entries. The high probability of a single array entry passing unchanged through a single write would be visible from clumps of AND gates in the DAG, but I wouldn’t expect larger-scale differentials in RC4-style stream ciphers to be visible in DAGs.)

The idea of drawing cipher DAGs certainly isn’t new; DAGs are common in cryptographic research and even more common in cryptographic education. What’s new here is the level of automation, minimizing the amount of cipher-specific effort required to build a DAG from a cipher (starting from a typical reference implementation in C or C++) and to visualize the DAG.

My tools are only prototypes at this point. I’m planning to put a `cipherdag` package online, but I haven’t done so yet, and I certainly can’t claim that the tools have saved time in cryptanalysis. But I think that the tools *will* save time in cryptanalysis, automating several tedious tasks that today are normally done by hand.