Chapter 7
The Complexity of $\mathbb{GF}(2)$-Matrix Operations

Here, we propose a new model, counting matrix-memory operations instead of field operations, for reasons to be discussed. It turns out this model describes reality only partially—but we will explicitly discuss the circumstances in which the model is descriptive and in which it fails, see Section 7.1.4 on Page 92. The complexity expressions are summarized in Table 7.1 on Page 105. Also of interest are certain data structure choices that we made in arranging our linear algebra library, see Section 9 on Page 133. This library was used by Nicolas Courtois in his cryptographic research, as well as by the author, and now forms part of the $\mathbb{GF}(2)$ linear algebra suite of SAGE [7], an open source competitor to MAGMA [2], MATLAB [5], MAPLE [3], and MATHEMATICA[4]. These are described in Section 7.4 on Page 94.

7.1 The Cost Model

In papers on matrix operations over the real or complex numbers, the number of floating point operations is used as a measure of running time. This removes the need to account for assembly language instructions needed to manipulate index pointers, iteration counters, discussions of instruction set, and measurements of how cache coherency or branch prediction will impact running time. In this dissertation, floating point operation counts are meaningless, for matrices over $\mathbb{GF}(2)$ do not use floating point operations. Therefore, we propose that matrix entry reads and writes be tabulated, because addition (XOR) and multiplication (AND) are single instructions, and can even be aggregated (see Section 9.5.4 on Page 149) while reads and writes on rectangular arrays are much more expensive. Clearly these data structures are non-trivial in size, so memory transactions will be the bulk of the time spent.

From a computer architecture viewpoint in particular, the matrices required for cryptanalysis cannot fit in the cache of the microprocessor, so the fetches to main memory are a bottleneck. Even if exceptionally careful use of temporal and spatial locality guarantees effective caching (and it is not clear that this is even possible),
the data must still travel from memory to the processor and back. The bandwidth of buses has not increased proportionally to the rapid increase in the speeds of microprocessors. Given the relatively simple calculations done once the data is in the microprocessor’s registers (i.e. single instructions), it is extremely likely that the memory transactions are the rate-determining step.

When attempting to convert these memory operation counts into CPU cycles, one must remember that other instructions are needed to maintain loops, execute field operations, and so forth. Also, memory transactions are not one cycle each, but can be pipelined. Thus we estimate that about 4–10 CPU cycles are needed per matrix-memory operation.

7.1.1 A Word on Architecture and Cross-Over

Often, there is an asymptotically fast algorithm for some problem, and then other algorithms which are better for small and medium-sized versions of the problem. The points at which one algorithm ceases to dominate another is called the “cross-over” between those two algorithms. Calculating the cross-over point, at least approximately, is of importance, so that when presented with a specific problem instance, one knows exactly which algorithm to run on it.

Due to the variations of computer architectures, the coefficients given here may vary slightly. In particular, on some machines, $32 \text{GF}(2)$ addition operations can be a single instruction, and on others, 64. Slight variations in the coefficients might appear to be of little interest, but when comparing two algorithms (e.g. M4RM and Strassen’s Matrix Multiplication Algorithm), we must consider the cross-over time. In this case, it would be given by

$$c_1 \frac{n^3}{\log n} = c_2 n^{2.807\ldots}$$

and one can see that time variations in $c_1$ or $c_2$ are very important, because $n^{0.193}/\log n = \frac{c_2}{c_1}$, or neglecting the $\log n$, roughly $n \sim (c_2/c_1)^5$. Certainly, changes in cache sizes on different machines that are otherwise identical can also change the cross-over. For this reason, the BLAS (Basic Linear Algebra System) called ATLAS (Automatically Tuned Linear Algebra System) [229] is very exciting. It automatically computes the precise cross-over sizes exactly on the machine during a tuning stage while being installed. Therefore, the algorithms always perform optimally.

On the other hand, by deriving running times mathematically rather than experimentally, one need not worry about artifacts of particular architectures or benchmarks skewing the results.