

XOR for Fun and Profit

From L1 Cache to Algebraic Cryptanalysis

Martin Albrecht



dev1, Seattle, June 20, 2008

Outline

XOR and \mathbb{F}_2

Words, Loops, Cache & SSE2

Linear and Polynomial Systems of Equations

Algebraic Cryptanalysis

Final



Outline

XOR and \mathbb{F}_2

Words, Loops, Cache & SSE2

Linear and Polynomial Systems of Equations

Algebraic Cryptanalysis

Final



\mathbb{F}_2

- ▶ Field with two elements.
- ▶ XOR is addition.
- ▶ AND is multiplication.
- ▶ It doesn't get any easier than that.
- ▶ This is why it's hard.

| | | \oplus | \odot |
|---|---|----------|---------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Before we begin

I suspect most of the stuff in this presentation to be correct, but I realised several times during this week that my theories are false and had to update these slides.

Gaussian Reduction

```
for c from 0 <= c < nc:
  for r from sr <= r < nr:
    if A[r, c]:
      A.swap_rows(r, sr)
      for i from 0 <= i < nr:
        if i != sr and A[i, c]:
          A.add_row(i, sr, c) # expensive
      sr = sr + 1
    break
```

M4RI [2]: Idea

$$\left(\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & \dots \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & \dots \\ \dots & & & & & & & & \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & \dots \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & 0 & 1 & 0 & 1 & 1 & \dots \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & \dots \\ \dots & & & & & & & & \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & 1 & 1 & 1 & 0 & 1 & \dots \end{array} \right)$$

M4RI [2]: Gray Codes

| | | | | |
|----------|----------|----------|---|---|
| | | 0 | 0 | 0 |
| | | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| | | 1 | 0 | 1 |
| | | 1 | 0 | 0 |

- ▶ Computing all possible 2^k sums, costs only 2^k additions.

M4RI [2]: Algorithm

```
while c < ncols:  
    if c+k > A.ncols():  
        k = ncols - c  
    kbar = A.gauss_submatrix(r, c, k, nrows)  
    if kbar > 0:  
        T = A.make_table(r, c, kbar)  
        # add rows from table T  
        A.add_rows(0, r, c, kbar, T)  
        A.add_rows(r+kbar, nrows, c, kbar, T)  
    r += kbar  
    c += kbar  
    if k != kbar:  
        c++
```

class M4RI(M4RM):

- ▶ M4RI inspired by “Method of the Four Russians” multiplication (M4RM) or Kronrod’s method.
- ▶ Complexity of both algorithms: $O(n^3 / \log_2(n))$ memory accesses.

$O(n^{2.807})$ matrix elimination was planned for this workshop, but we’re not there yet.

Outline

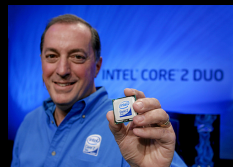
XOR and \mathbb{F}_2

Words, Loops, Cache & SSE2

Linear and Polynomial Systems of Equations

Algebraic Cryptanalysis

Final



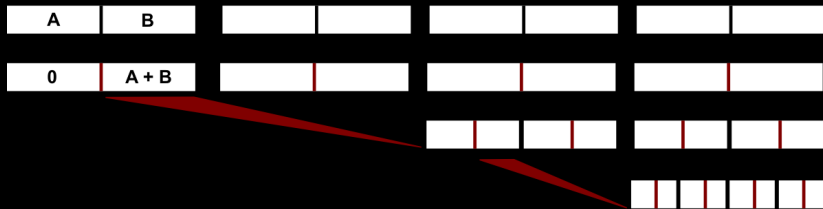
Words

- ▶ Modern CPUs have 64-bit words \rightarrow operate on 64 \mathbb{F}_2 elements in parallel.
- ▶ Every operation on single bits is a waste of time, to be avoided at all costs.
- ▶ Even our naive cubic multiplication doesn't do that, thanks to David Harvey.

Dot Product

```
for (j=RADIX*(eol-1); j>=0; j-=RADIX) {  
    for (k=RADIX-1; k>=0; k--) {  
        b = BT->values + BT->rowswap[j+k];  
        parity[k] = a[0] & b[0];  
        for (i=wide-1; i>=1; i--)  
            parity[k] ^= a[i] & b[i];  
    }  
    c[j/RADIX] ^= parity64(parity);  
}
```

parity 2^n



XOR is Cheap, Loops are Expensive

```
for (i=0; i<2048; i++) {  
    dst[i] ^= src[i];  
}
```

```
400567: 488b04d3      mov    (%rbx,%rdx,8),%rax  
40056b: 483144d500    xor    %rax,0x0(%rbp,%rdx,8)  
400570: 4883c201      add    $0x1,%rdx  
400574: 4881fa00080000  cmp    $0x800,%rdx  
40057b: 75ea         jne    400567 <main+0x2f>
```

Don't take this example too seriously, your compiler is your friend, don't try to outsmart it: It will outsmart you and unroll loops on the way.

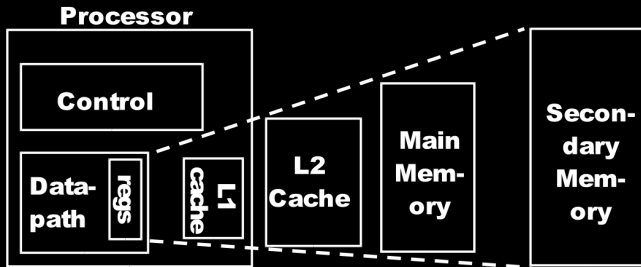
Loops cont.

If the loop unrolling of the compiler doesn't satisfy you:

- ▶ Unroll it yourself. ATLAS unrolls up to 32.
- ▶ http://en.wikipedia.org/wiki/Duff's_device
- ▶ Do more stuff per iteration, e.g.:

```
for(i=0; i<2048; i++) {  
    dst[i] ^= s0[i] ^ s1[i] ^ s2[i] ^ s3[i];  
}
```


Cache [3] I



| Memory | Regs | L1 | L2 | Ram | Swap |
|---------------|------------|-----|------|--------|----------------|
| Speed (ns) | 0.5 | 2 | 6 | 10^2 | 10^7 |
| Cost (cycles) | 1 | 4 | 14 | 200 | $2 \cdot 10^7$ |
| Size | 4 · 64-bit | 64k | 1-4M | 1G | 100G |

Cache [3] II

“Therefore, we propose that matrix entry reads and writes be tabulated, because addition (XOR) and multiplication (AND) are single instructions, while reads and writes on rectangular arrays are much more expensive. Clearly these data structures are nontrivial in size (hundreds of megabytes at the least) and so memory transactions will be the bulk of the computational burden.”

— Gregory Bard, [2]

Cache Efficiency: n Gray Code Tables (Bill Hart)

$$\begin{pmatrix} 1 & 0 & | & 0 & 0 & | & 0 & 1 & 1 & 1 & \dots \\ 0 & 1 & | & 0 & 0 & | & 1 & 1 & 1 & 0 & \dots \\ \hline 0 & 0 & | & 1 & 0 & | & 0 & 1 & 1 & 1 & \dots \\ 0 & 0 & | & 0 & 1 & | & 1 & 0 & 0 & 1 & \dots \\ \dots & & & & & & & & & & \\ 0 & 0 & | & 0 & 1 & | & 1 & 0 & 1 & 0 & \dots \\ \mathbf{1} & \mathbf{1} & | & \mathbf{0} & \mathbf{0} & | & 1 & 0 & 1 & 1 & \dots \\ 0 & 1 & | & 0 & 0 & | & 1 & 0 & 0 & 1 & \dots \\ \dots & & & & & & & & & & \\ \mathbf{1} & \mathbf{1} & | & \mathbf{0} & \mathbf{1} & | & 1 & 1 & 0 & 1 & \dots \end{pmatrix}$$

Example: M4RM

$n = 8$, $k = 5$ and size = 2048: $n \cdot 2^k \cdot 2048/8 = 64k$ (= **L1** on Opteron).

Cache Efficiency: Matrix Block'ing

- ▶ For M4RM we can block the source matrices to gain better cache efficiency

$$\begin{pmatrix} AE & AF \\ CE & CF \end{pmatrix} = \begin{pmatrix} A \\ C \end{pmatrix} \cdot (E \ F).$$

- ▶ My attempts for M4RI failed.
- ▶ However, we will inherit block'ing from M4RM eventually.
- ▶ Of course, Strassen's formula comes with block matrix decomposition.

Vectorisation I

Modern compilers (GCC 4, MSVC, SunCC) support 128-bit SSE2 integer instructions via compiler intrinsics.

```
while ( __c < eof ) {  
    xmm1 = _mm_xor_si128(* __c , * __t0 ++);  
    xmm1 = _mm_xor_si128(* __c , * __t1 ++);  
    xmm1 = _mm_xor_si128(* __c , * __t2 ++);  
    xmm1 = _mm_xor_si128(* __c , * __t3 ++);  
    * __c ++ = xmm1;  
}
```

This gives up to 25% speed improvement on the Intel Core2Duo, it seems slower on the AMD Opteron.

Vectorisation II

Alternatively, GCC has custom vectorisation support across all platforms (SSE, AltiVec, ...) I haven't tried that yet.

```
typedef int v4si __attribute__((vector_size(16)));  
v4si a, b, c;  
c = a + b;
```

<http://gcc.gnu.org/onlinedocs/gcc/Vector-Extensions.html>

Results: Multiplication I

- ▶ Strassen-Winograd $O(n^{2.807})$ multiplication with memory efficient scheduling [9].
- ▶ M4RM as base case, cutoff: $\frac{2n^2}{8} = L2$.
- ▶ 64-bit bulk operations on AMD CPUs.
- ▶ 128-bit SSE2 bulk operations on Intel CPUs.
- ▶ 8 Gray code tables.
- ▶ Matrix block decomposition in M4RM.

Even with all that, we are not there yet. But there are some ideas in the pipeline.

Results: Multiplication II

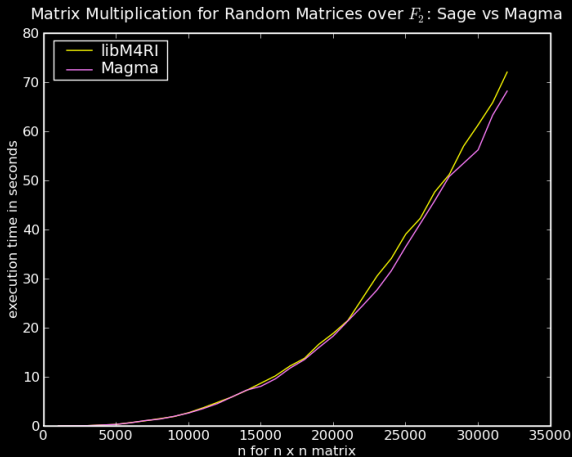


Figure: 2.6 Ghz Opteron, 18GB RAM, VMWare Virtualised

Results: Multiplication III

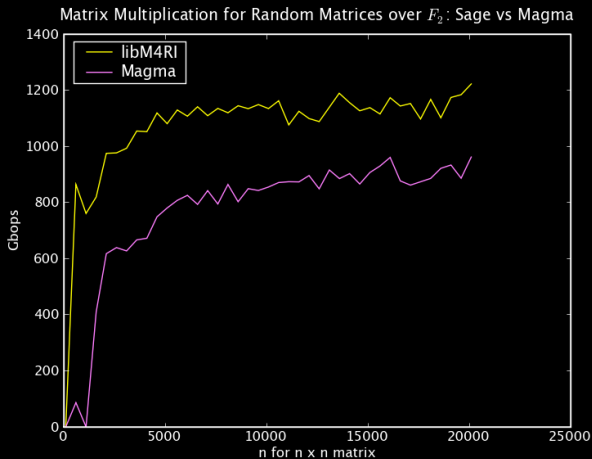


Figure: “Gbops” ($2 \cdot n^3 / (10^9 \cdot t)$) on Core2Duo

Parallelisation

```
#pragma omp parallel sections
{
#pragma omp section
{
_mzd_mul_mp(Q0, A00, B00, cut);
}
#pragma omp section
{
_mzd_mul_mp(Q1, A01, B10, cut);
}
}
```

“The **OpenMP** API supports multi-platform shared-memory parallel programming in C/C++ ... It is a portable, scalable model ... on platforms from the desktop to the supercomputer.”

Intermediate Results: OpenMP

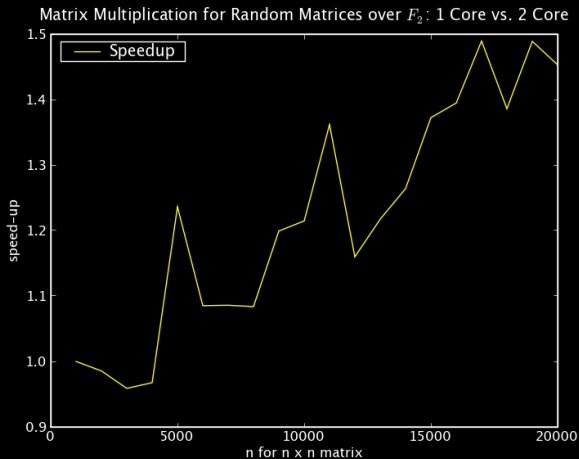


Figure: 2.33 Ghz Core2Duo, 3GB RAM

Results: Reduced Row Echelon Form I

- ▶ M4RI $O(n^3 / \log_2(n))$ reduction.
- ▶ 64-bit bulk operations.
- ▶ 4 Gray code tables.

This is not asymptotically fast!

Results: Reduced Row Echelon Form II

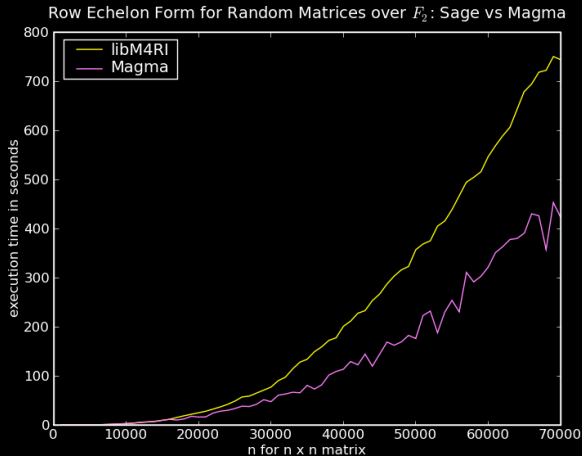


Figure: 2.6 Ghz Opteron, 18GB RAM, VMWare Virtualised

Results: Reduced Row Echelon Form III

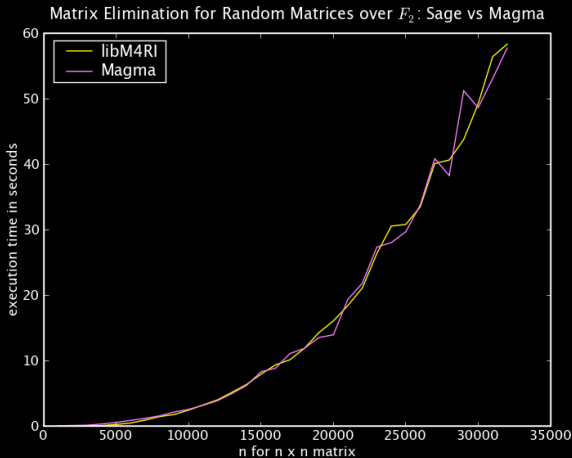


Figure: 2.33 Ghz Core2Duo, 3GB RAM

Outline

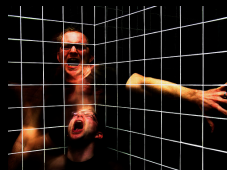
XOR and \mathbb{F}_2

Words, Loops, Cache & SSE2

Linear and Polynomial Systems of Equations

Algebraic Cryptanalysis

Final



Linear Systems

Consider for example a linear system of equations over $\mathbb{F}_{127}[x, y, z]$.

$$\begin{aligned} 0 &= 26y + 52z + 62 \\ 0 &= 54y + 119z + 55 \\ 0 &= 41x + 91z + 13 \end{aligned} \quad \left(\begin{array}{cccc} 0 & 26 & 52 & 62 \\ 0 & 54 & 119 & 55 \\ 41 & 0 & 91 & 13 \end{array} \right)$$

$$\begin{aligned} 0 &= x + 29 \\ 0 &= y + 38 \\ 0 &= z + 75 \end{aligned} \quad \left(\begin{array}{cccc} 1 & 0 & 0 & 29 \\ 0 & 1 & 0 & 38 \\ 0 & 0 & 1 & 75 \end{array} \right)$$

Polynomial Division

Now consider two polynomials in $\mathbb{F}_{127}[x, y, z]$ with term ordering **degrevlex**.

$$\begin{aligned} f &= x^2 + 2xy - 2y^2 + 14z^2 + 22z \\ g &= xy + y^2 + z^2 + x + 2z \end{aligned} \quad \begin{pmatrix} 1 & 2 & -2 & 14 & 0 & 22 \\ 0 & 1 & 1 & 1 & 1 & 2 \end{pmatrix}$$

$$\begin{aligned} f' &= x^2 - 4y^2 + 12z^2 - 2x + 18z \\ g &= xy + y^2 + z^2 + x + 2z \end{aligned} \quad \begin{pmatrix} 1 & 0 & -4 & 12 & 125 & 18 \\ 0 & 1 & 1 & 1 & 1 & 2 \end{pmatrix}$$

Polynomial Division cont. I

This approach fails for

$$f = x^2 - 2xy - 2y^2 + 14z^2,$$
$$g = y + x + 2z.$$

since y is not a monomial of f . However, it divides two monomials of f : y^2 and xy . To deal with these include multiples of $m \cdot g$ such that

$$LM(m \cdot g) = m \cdot LM(g) \in M(f).$$

Polynomial Division cont. II

$$f = x^2 - 2xy - 2y^2 + \dots$$

$$x \cdot g = x^2 + xy \dots$$

$$y \cdot g = xy + y^2 + \dots$$

$$g = x + y + 2z$$

$$\begin{pmatrix} 1 & -2 & -2 & 0 & 0 & 14 & 0 & \dots \\ 1 & 1 & 0 & 2 & 0 & 0 & 0 & \dots \\ 0 & 1 & 1 & 0 & 2 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \dots \end{pmatrix}$$

$$0 = x^2 + 4yz + 14z^2,$$

$$0 = xy + 2xz + -4yz - \dots,$$

$$0 = y^2 - 2xz + 6yz + \dots,$$

$$0 = x + y + 2z$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & \dots \\ 0 & 1 & 0 & 2 & \dots & 0 & \dots \\ 0 & 0 & 1 & -2 & \dots & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots & 1 & \dots \end{pmatrix}$$

Lets call the whole process **inter-reduction**.

Solving Polynomial Systems of Equations

- ▶ Inter-reduction alone is not sufficient for systems with degree $d > 1$.
- ▶ Iteration of multiplication & inter-reduction steps:
 - ▶ Multiplication can either be naive (XL [8] and variants) or
 - ▶ more careful (Buchberger, F_4 [10], SlimGB [4]) and
 - ▶ up to optimal in some situations (F_5 [11]).
- ▶ After a finite number of steps, we end up with an “easy” system of equations to read the solution from.
- ▶ ... I didn't use the word **Gröbner basis** once.

Expensive step is reduction \rightarrow linear algebra.

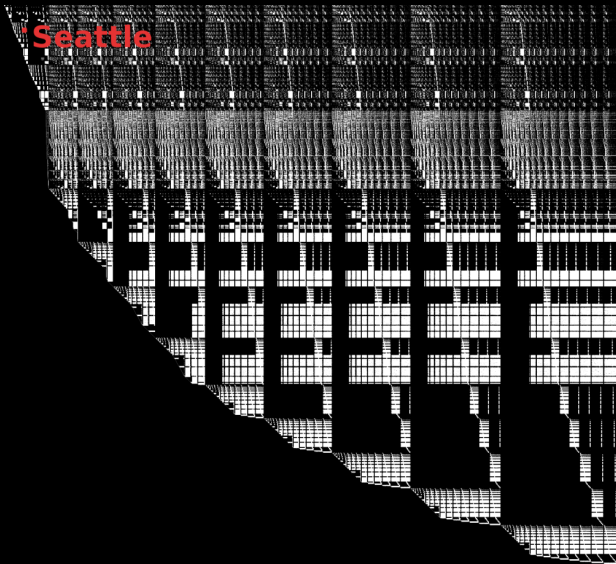


Figure: The shape of the initial matrix is “similar” to triangular.

Outline

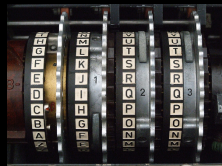
XOR and \mathbb{F}_2

Words, Loops, Cache & SSE2

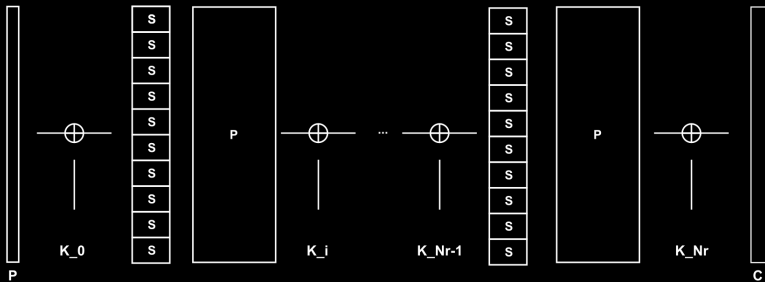
Linear and Polynomial Systems of Equations

Algebraic Cryptanalysis

Final



Substitution Permutation Network



S-Box Equations I

Consider the (exceptionally bad) S-Box

$[4, 1, 2, 3, 0, 5, 6, 7]$

as an example.

Construct the matrix on the right and perform fraction-free Gaussian elimination on it.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & x_0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & x_1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & x_2 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & y_0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & y_1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & y_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & x_0 x_1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & x_0 x_2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & x_0 y_0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & x_0 y_1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & x_0 y_2 \\ \dots & & & & & & & & \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & x_1 y_2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & x_2 y_0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & x_2 y_1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & x_2 y_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & y_0 y_1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & y_0 y_2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & y_1 y_2 \end{pmatrix}$$

S-Box Equations II

$$\left(\begin{array}{cccccccc}
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_0y_0 + y_0 \\
 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & x_0x_1 + x_0y_0 + x_0 + x_1 + y_0 + 1 \\
 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & x_0x_2 + x_0y_0 + x_0 + x_2 + y_0 + 1 \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & x_0x_1 + x_0x_2 + x_0y_0 + x_0 + x_1 + x_2 + y_0 + 1 \\
 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & x_0y_0 + x_0 \\
 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & x_0x_1 + x_0y_0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & x_0x_2 + x_0y_0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_0x_1 + x_0x_2 + x_0y_0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_1 + y_1 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_2 + y_2 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_0x_1 + x_0y_1 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_0x_2 + x_0y_2 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_1x_2 + x_0 + x_1 + x_2 + y_0 + 1 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_0x_1 + x_1y_0 \\
 \dots & & & & & & & & \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_2y_1 + x_0 + x_1 + x_2 + y_0 + 1 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_2y_2 + x_2 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_0x_1 + y_0y_1 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_0x_2 + y_0y_2 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & y_1y_2 + x_0 + x_1 + x_2 + y_0 + 1
 \end{array} \right)$$

Cipher Equations

- ▶ Define subkey variables for the subkey bits used in each round.
- ▶ Define “state” variables for S-Box input and output bits.
- ▶ Diffusion layer is linear in these variables and the subkey variables.
- ▶ Define equations for key schedule analogously.

There are many ways the above can be done. How it is done most effectively, is an open research problem.

Performance

| | PolyBoRi [5] | MRHS [12] | ElimLin [6] | MiniSat [6] |
|--------------|--------------|-----------|------------------------------|-----------------------------|
| CPU | 2.2 GHZ | ? | 1.6 Ghz | 1.6 Ghz |
| RAM | 16 GB | 1 GB | ? | ? |
| SR(4,1,1,4) | – | 0.032 s | – | – |
| SR(10,1,1,4) | 0.14 s | 0.32 s | – | – |
| SR(10,1,2,4) | 6.7 s | – | – | – |
| SR(10,2,2,4) | 1205 s | – | – | – |
| 4r DES | – | – | $2^{19} \cdot 8 \text{ s}$ | – |
| 5r DES | – | – | $2^{23} \cdot 173 \text{ s}$ | – |
| 6r DES | – | – | – | $2^{20} \cdot 68 \text{ s}$ |

Algebraic and “Conventional” Attacks

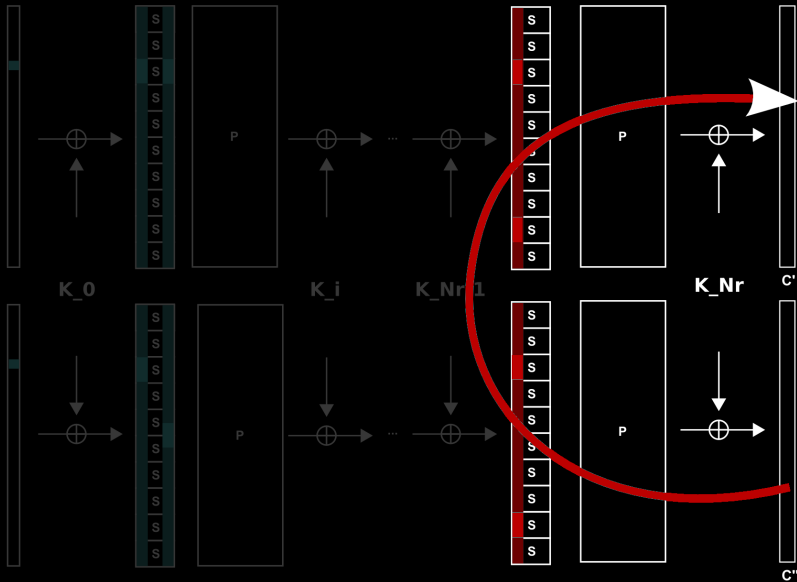
KeeLoq 32-bit blocksize, 64-bit keysize, 528 very simple rounds. Attack combining algebraic and slide attacks breaks the cipher [7]. There are better attacks though.

PRESENT 64-bit blocksize, 128-bit keysize, 31 round ciphers. Technique combining algebraic and differential (more **XOR!**) attacks breaks 18 rounds [1].

Algebraic and “Conventional” Attacks

KeeLoq 32-bit blocksize, 64-bit keysize, 528 very simple rounds. Attack combining algebraic and slide attacks breaks the cipher [7]. There are better attacks though.

PRESENT 64-bit blocksize, 128-bit keysize, 31 round ciphers. Technique combining algebraic and differential (more **XOR!**) attacks breaks 18 rounds [1].



Outline

XOR and \mathbb{F}_2

Words, Loops, Cache & SSE2

Linear and Polynomial Systems of Equations

Algebraic Cryptanalysis

Final



Having said that: However, for many problems, what we really need is reduction of sparse matrices.

Thank You!





Martin Albrecht and Carlos Cid.

Algebraic Techniques in Differential Cryptanalysis.

Cryptology ePrint Archive, Report 2008/177, 2008.

available at <http://eprint.iacr.org/2008/177.pdf>.



Gregory V. Bard.

Accelerating Cryptanalysis with the Method of Four Russians.

Cryptology ePrint Archive, Report 2006/251, 2006.

available at <http://eprint.iacr.org/2006/251.pdf>.



L.N. Bhuyan.

CS 161 Ch 7: Memory Hierarchy Lecture 22, 1999.

available at [http:](http://www.cs.ucr.edu/~bhuyan/cs161/LECTURE22.ppt)

[//www.cs.ucr.edu/~bhuyan/cs161/LECTURE22.ppt](http://www.cs.ucr.edu/~bhuyan/cs161/LECTURE22.ppt).



Michael Brickenstein.

Slimgb: Gröbner Bases with Slim Polynomials.

In *Reports On Computer Algebra 35*. Centre for Computer Algebra, University of Kaiserslautern, 2005.

available at: http://www.mathematik.uni-kl.de/~zca/Reports_on_ca/35/paper_35_full.ps.gz.






Michael Brickenstein and Alexander Dreyer.

PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials.

In *Electronic Proceedings of MEGA 2007*, 2007.

available at <http://www.ricam.oeaw.ac.at/mega2007/electronic/26.pdf>.

-  Nicolas T. Courtois and Gregory V. Bard.
Algebraic Cryptanalysis of the Data Encryption Standard.
In Steven D. Galbraith, editor, *Cryptography and Coding – 11th IMA International Conference*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169, Berlin Heidelberg New York, 2007. Springer Verlag.
available at <http://eprint.iacr.org/2006/402>.
-  Nicolas T. Courtois, Gregory V. Bard, and David Wagner.
Algebraic and slide attacks on KeeLoq.
In *Proceedings of FSE 2008*, 2008.
-  Nicolas T. Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir.
Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations.

In *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer Verlag, 2000.



Jean-Guillaume Dumas and Clement Pernet.
Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm.

available at

<http://www.citebase.org/abstract?id=oai:arXiv.org:0707.2347>, 2007.



Jean-Charles Faugère.
A New Efficient algorithm for Computing Gröbner Basis (F4), 1999.

available at [http:](http://modular.ucsd.edu/129-05/refs/faugere_f4.pdf)

[//modular.ucsd.edu/129-05/refs/faugere_f4.pdf](http://modular.ucsd.edu/129-05/refs/faugere_f4.pdf).



Jean-Charles Faugère.

A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F5).

In *Proceedings of ISSAC*, pages 75–83. ACM Press, 2002.



Håvard Raddum and Igor Semaev.

Solving MRHS linear equations.

Cryptology ePrint Archive, Report 2007/285, 2007.

available at <http://eprint.iacr.org/2007/285>.