



Gröbner Bases in SAGE

Martin Albrecht (malb@informatik.uni-bremen.de)

January 1, 2007



- Talk covers commutative algebra only.
 - Singular team provides Plural – the non-commutative version of Singular.
- Talk also strongly biased towards polynomial rings over finite fields.
- Let $k[x_0, \dots, x_{n-1}]$ be our ring then $x_0^{\alpha_0} \dots x_{n-1}^{\alpha_{n-1}}$ is a monomial and $c_j x_0^{\alpha_0} \dots x_{n-1}^{\alpha_{n-1}}$ a term with $c \in k$.
- $LT(f)$ is the leading term of f and $LM(f)$ is the leading monomial of f .
- Semantic used in **SAGE** and Singular



Outline

SAGE

- 1 Gröbner Bases
- 2 Detour: Algebraic Attacks on Block Ciphers
- 3 Gröbner Bases in SAGE
 - SAGE as a Tool to Compute Gröbner Bases
 - SAGE as a Tool to Develop Gröbner Basis Algorithms
- 4 Benchmarks
- 5 Todo



A Gröbner basis is a finite subset $G = \{g_0, \dots, g_{n-1}\}$ of the ideal I satisfying:

$$\langle LT(g_0), \dots, LT(g_{n-1}) \rangle = LT(I)$$

Properties:

- Gröbner bases are computable.
- Reduced Gröbner bases are unique with respect to the monomial order.
- Reduction mod G is unique; Gröbner bases allow solving the ideal member problem.
- Lexicographical Gröbner bases allow computing the variety $V(I)$.

Applications include robotics, automatic geometric reasoning, and solving multivariate polynomial systems.



Outline

- 1 Gröbner Bases
- 2 Detour: Algebraic Attacks on Block Ciphers
- 3 Gröbner Bases in SAGE
 - SAGE as a Tool to Compute Gröbner Bases
 - SAGE as a Tool to Develop Gröbner Basis Algorithms
- 4 Benchmarks
- 5 Todo



Algebraic Attacks: The Idea I

- Given a block cipher with a linear layer, key schedule, key addition, and S-boxes (XSL-cipher).
- Express this cipher as a multivariate (quadratic) equation system; usually over \mathbb{F}_{2^n} with $n \geq 1$.
 - Easy for linear layer and key addition: linear equations.
 - Find as many linear-independent equations for S-boxes as possible either by brute force or use Gröbner bases.
 - If key schedule features S-boxes do the same, else use linear equations.
- Solve this equation system, the solution to the key variables is the desired key.



Algebraic Attacks: The Idea II

- Hope/Fear that these equation systems may be solved faster than the general (NP-hard) case.
 - Very few solutions (often only one).
 - Equations over (very small) finite fields.
 - May exclude any solutions from the algebraic closure.
 - Equation systems often **overdefined** yet **sparse**.



Algebraic Attacks: Equation Systems

AES (128-bit, 10 rounds) 8000 equations in 1600 variables over \mathbb{F}_2 or 5248 equations in 3968 variables over \mathbb{F}_{2^8} .

CTC (255-bit, 6 rounds) 11985 equations in 6375 variables over \mathbb{F}_2 .

XSL $Bs \cdot Nr$ linear equations for the linear layer,
 $Bs \cdot (Nr + 1)$ equations for the key addition,
 $u \cdot Bs \cdot Nr$ equations for the S-boxes,
 $v \cdot Nr$ equations for the key schedule. (u, v vary depending on cipher)



Algebraic Attacks: Applicability

- Algebraic attacks on stream ciphers have broken several stream ciphers (e.g. Toyocrypt).
- Successful algebraic attack on HFE crypto challenge with 96-bit and degree $n \leq 4$ (MAGMA).
- Nicolas Courtois broke CTC with 255-bit and 6 rounds in under one hour using an unpublished “Fast Algebraic Attack Against Block Ciphers”. (which he claims to be a new way of computing a Gröbner basis for block cipher ideals)



Computing Gröbner Bases I

- Buchberger's algorithm is well known algorithm to compute Gröbner bases.
- But its complexity is exponential in general and it leaves a lot of freedom to implement.
- Choices influence performance dramatically: Term order, order in which the critical pairs are considered, criterion to discard useless reductions to zero.
- It's difficult to know the *best* Gröbner basis algorithm for a given ideal *a priori*
- Accurate estimations of the runtime of Gröbner basis algorithms is hard.



Computing Gröbner Bases II

- Several other algorithms have been developed for Gröbner basis computation:

Gebauer Möller Installation a powerful criterion to avoid useless reductions to zero

F_4 reduction by linear algebra

F_5 avoids reduction to zero, tricky to implement,
not suitable for the general case

SlimGB Singular's answer to F_4 , keeps polynomials and
coefficients slim during computation

FGLM and Gröbner Walk Efficient algorithms to walk from
one (zero-dimensional) Gröbner Basis to another
Gröbner Basis in another term order.

Hilbert series driven algorithms



Outline

- 1 Gröbner Bases
- 2 Detour: Algebraic Attacks on Block Ciphers
- 3 Gröbner Bases in SAGE
 - SAGE as a Tool to Compute Gröbner Bases
 - SAGE as a Tool to Develop Gröbner Basis Algorithms
- 4 Benchmarks
- 5 Todo



Algorithms and their Implementations

Singular Buchberger's algorithm with GM-Installation, slimgb,
Hilbert driven algorithm, good heuristic to choose an
algorithm

Macaulay2 Buchberger's algorithm (with GM-Installation?)

MAGMA very fast F_4 , Buchberger's algorithm with
GM-Installation, Hilbert-driven algorithm

SAGE all of the above (if available), F_4 over finite fields
(not included, slow)

ipa-smw/horatu F_4 over $\mathbb{F}_2[x_0, \dots, x_{127}] / \langle x_0^2 + x_0, \dots, x_{127}^2 + x_{127} \rangle$.



A F_4 Implementation for SAGE

far from prime-time

- Straight forward implementation of Faugère's "improved F_4 "
- Provides Gebauer & Möller installation and normal selection strategy
- User may provide own update strategy and selection strategy
- Only over \mathbb{F}_p (p prime)
- Specialized implementation over $\mathbb{F}_2[x_0, \dots, x_{n-1}] / \langle \text{FieldIdeal} \rangle$ available
 - Many ideas from ipa-smw/horatu F_4 implementation, but more flexible (arbitrary number of variables, choice of term order, interface with SAGE)
- Embarrassingly slow, most time spent in update pairs



Supported Rings for Gröbner Basis Calculation

SAGE supports Gröbner basis computation over

- $\mathbb{C}[x_0, \dots, x_{n-1}]$,
- $\mathbb{R}[x_0, \dots, x_{n-1}]$,
- $\mathbb{Q}[x_0, \dots, x_{n-1}]$, and
- $\mathbb{F}_{p^n}[x_0, \dots, x_{n-1}]$ (p prime, $n \geq 1$).
- Additionally quotient rings over these rings are also supported.

Possible additional rings include

- arbitrary transcendental and algebraic extensions over \mathbb{Q} and \mathbb{F}_p as provided by Singular.
- $\mathbb{Z}[x_0, \dots, x_{n-1}]$ if Macaulay2 is installed.
- Singular also supports GBs over \mathbb{Z} by calculating in $\mathbb{Q}[x_0, \dots, x_{n-1}]$ with option `intStrategy`.



Supported Term Orders

SAGE supports

- lexicographical (*lex*),
- reversed lexicographical (*revlex*)
- degree lexicographical (*deglex*), and
- degree reversed lexicographical (*degrevlex*),

term orders natively: `__cmp__()` and `lm()` obey these term orders.

Singular supports matrix term orders which we don't use (yet).



Calculating Gröbner Bases in SAGE

Singular's, Macaulay2's, and MAGMA's Gröbner basis algorithms are accessible from the **Ideal** class:

```
sage: R.<s,a,g,e> = MPolynomialRing(GF(2),4,order="degrevlex")
sage: I = sage.rings.ideal.Cyclic(R,4)
sage: gb1 = Ideal(I.groebner_basis()).reduced_basis() # default: singular:groebner
sage: gb2 = Ideal(I.groebner_basis('magma:GroebnerBasis')).reduced_basis() #MAGMA
sage: gb3 = Ideal(I.groebner_basis('macaulay2:gb')).reduced_basis() #Macaulay2
sage: sorted(gb1) == sorted(gb2) == sorted(gb3)
True
```

This also works in **QuotientRings** but only for Singular:

```
sage: R.<a,b,c,d,e,f,g,h,i,j> = MPolynomialRing(GF(2),10,order="degrevlex")
sage: FI = sage.rings.ideal.FieldIdeal(R)
sage: Q = R/FI
sage: I = sage.rings.ideal.Cyclic(Q,4)
sage: gb1 = Ideal(I.groebner_basis()).reduced_basis() # default: singular:groebner
sage: gb2 = Ideal(I.groebner_basis('singular:slimgb')).reduced_basis() #SlimGB
sage: sorted(gb1) == sorted(gb2)
True
```



Benchmark Ideals I

```
sage: R.<a,b,c,d> = PolynomialRing(GF(127),4,order="lex")
sage: sage.rings.ideal.Cyclic(R,4)

Ideal (d + c + b + a, c*d + b*c + a*d + a*b, b*c*d + a*c*d + a*b*d + a*b*c, \
126 + a*b*c*d) of Polynomial Ring in a, b, c, d over Finite Field of size 127

sage: sage.rings.ideal.Katsura(R,4)

Ideal (126 + 2*d + 2*c + 2*b + a, 126*c + 2*b*d + b^2 + 2*a*c, \
2*c*d + 126*b + 2*b*c + 2*a*b, 2*d^2 + 2*c^2 + 2*b^2 + 126*a + a^2) \
of Polynomial Ring in a, b, c, d over Finite Field of size 127
```

To restrict the variety to the base ring (excluding the algebraic closure) we may construct the field ideal:

```
sage: R.<a,b,c,d> = PolynomialRing(GF(127),4,order="lex")
sage: sage.rings.ideal.FieldIdeal(R)

Ideal (126*d + d^127, 126*c + c^127, 126*b + b^127, 126*a + a^127) \
of Polynomial Ring in a, b, c, d over Finite Field of size 127
```



I would like to add CTC, SR and all SymbolicData ideals to the list of benchmark ideals:

SR ideals corresponding to small scale variants of the Advanced Encryption Standard (AES) over \mathbb{F}_{2^e} with $e \in \{4, 8\}$.

CTC ideals corresponding to the Courtois Toy Cipher over \mathbb{F}_2 .

S.D. 350 Polynomial Systems, 297 Proof Schemes, 16 Testsets of benchmark ideals available at <http://www.SymbolicData.org> in some XML format.



Gröbner Basis Tests

Gröbner Basis testing is slightly intelligent by forming the syzygy module of $LT(g_0), \dots, LT(g_{n-1})$ and trying to lift it to the syzygy module of g_0, \dots, g_{n-1} . This is faster than testing every S-polynomial of g_0, \dots, g_{n-1} .

```
sage: R.<s,a,g,e> = PolynomialRing(GF(127),4,order="degrevlex")
sage: I = sage.rings.ideal.Cyclic(R,4)
sage: I.is_groebner()
False

sage: Igb = Ideal(I.groebner_basis())
sage: Igb.is_groebner()
True

sage: Ideal(f4.attack(I)).is_groebner()
True
```



Gröbner Basis Conversions

Reduced Gröbner bases:

```
sage: I = sage.rings.ideal.Cyclic(R,4)
sage: Igb = Ideal(I.groebner_basis())
sage: I.reduced_basis()
[e + g + a + s,
 e^2 + 2*a*e + a^2,
 126*e^3 + g^2*e + 126*a*e^2 + a*g^2,
 126 + 126*e^4 + g*e^3 + g^2*e^2 + 126*a*e^3 + a*g*e^2]
```

Transformed bases (using FGLM):

```
sage: I = sage.rings.ideal.Cyclic(R,4)
sage: Igb = Ideal(Ideal(I.groebner_basis()).reduced_basis())
sage: Igb.transformed_basis()
[1 + 126*e^4 + 126*g^2*e^2 + g^2*e^6,
 126*e + 126*g + g^2*e^3 + g^3*e^2,
 126*e + e^5 + 126*a + a*e^4,
 125*e^2 + g*e + g^2*e^4 + 126*a*e + a*g,
 e^2 + 2*a*e + a^2,
 e + g + a + s]
```



SAGE contains the program `gfan` to compute “Gröbner Fans” of given ideals. That is it computes an object which represents every possible reduced Gröbner basis of a given ideal with respect to every possible monomial order.

```
sage: R.<s,a,g,e> = PolynomialRing(linbox.GFq(127),4)
sage: I = sage.rings.ideal.Katsura(R,4)
sage: gf = I.groebner_fan(); gf
sage: gbs = gf.reduced_groebner_bases() #all of them!
sage: len(gbs)
115
```

Only supported over \mathbb{F}_p (p prime and ≤ 32749) and \mathbb{Q} so far up to 52 variables.



Computation Protocols

Singular offers a computation protocol:

```
sage: singular.option("prot")
sage: R.<s,a,g,e> = MPolynomialRing(GF(2),4,order="degrevlex")
sage: I = sage.rings.ideal.Cyclic(R,4)
sage: print singular.eval("slimgb(%s)"%I._singular_().name())
2M[1,1](2)3M[1,1](2)4M[2,e1](2)5M[2,2](3)6M[3,1](2)7M[2,0](0)calculated 11 NFs
applied 8 product crit, 0 extended_product crit
-[1]=s+a+g+e
-[2]=a^2+e^2
-[3]=a*g^2+g^2*e+a*e^2+e^3
-[4]=a*g*e^2+g^2*e^2+a*e^3+g*e^3+e^4+1
-[5]=a*e^4+e^5+a+e
-[6]=g^3*e^2+g^2*e^3+g+e
-[7]=g^2*e^4+a*g+a*e+g*e
```

which may turn out quite useful for the development of improved or specialized Gröbner basis algorithms. So wrapping it would be nice.

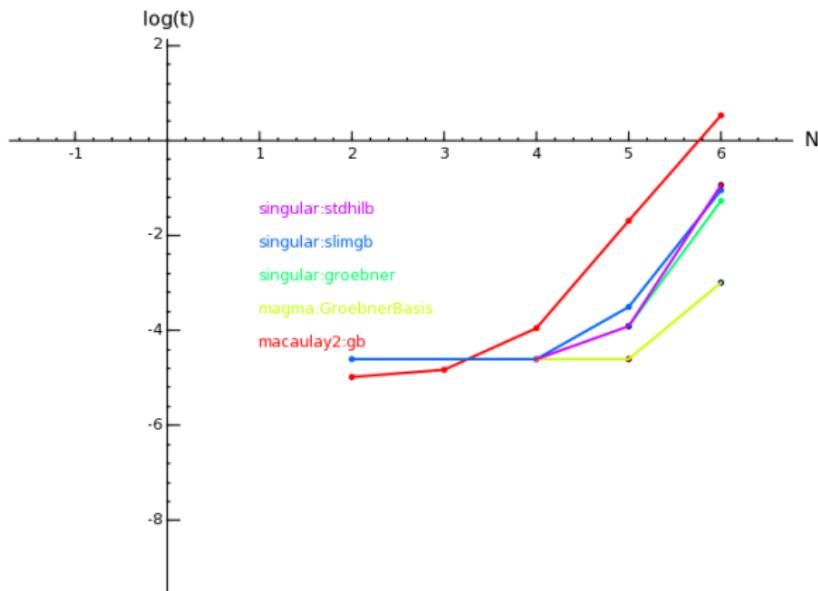


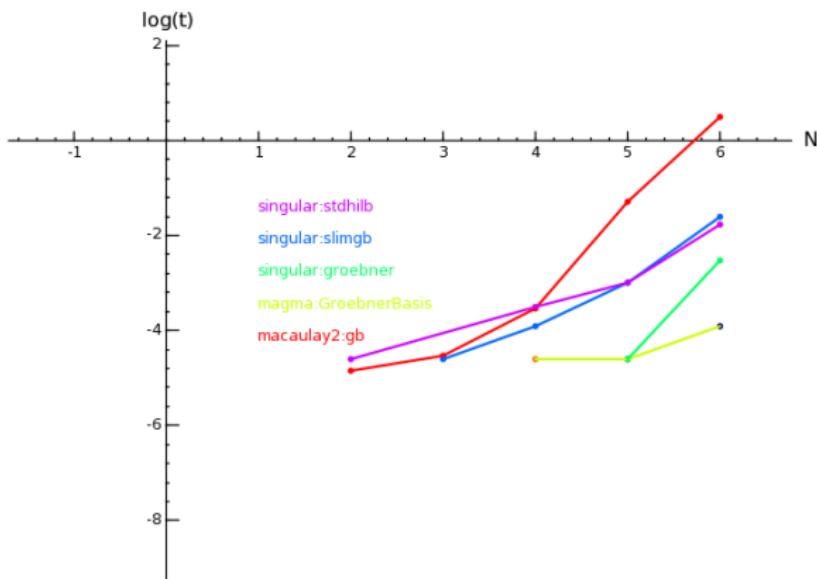
Outline

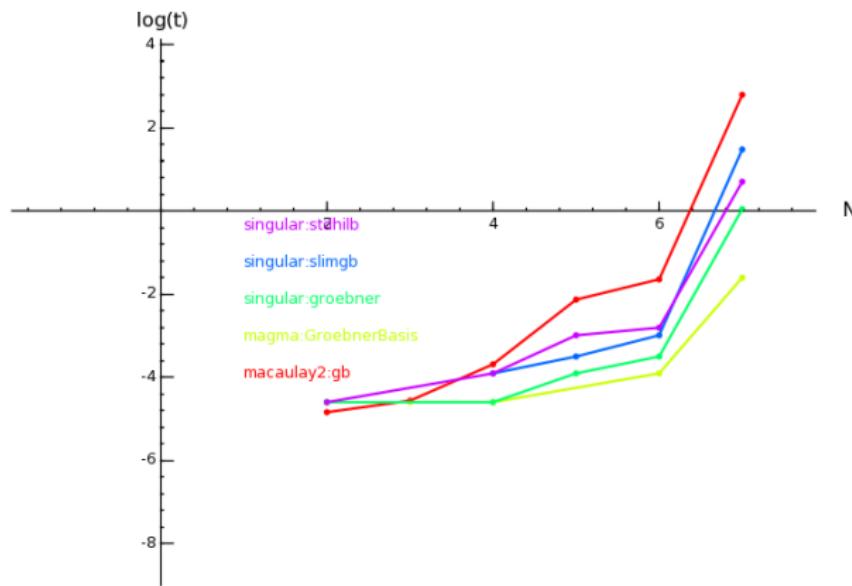
- 1 Gröbner Bases
- 2 Detour: Algebraic Attacks on Block Ciphers
- 3 Gröbner Bases in SAGE
 - SAGE as a Tool to Compute Gröbner Bases
 - SAGE as a Tool to Develop Gröbner Basis Algorithms
- 4 Benchmarks
- 5 Todo

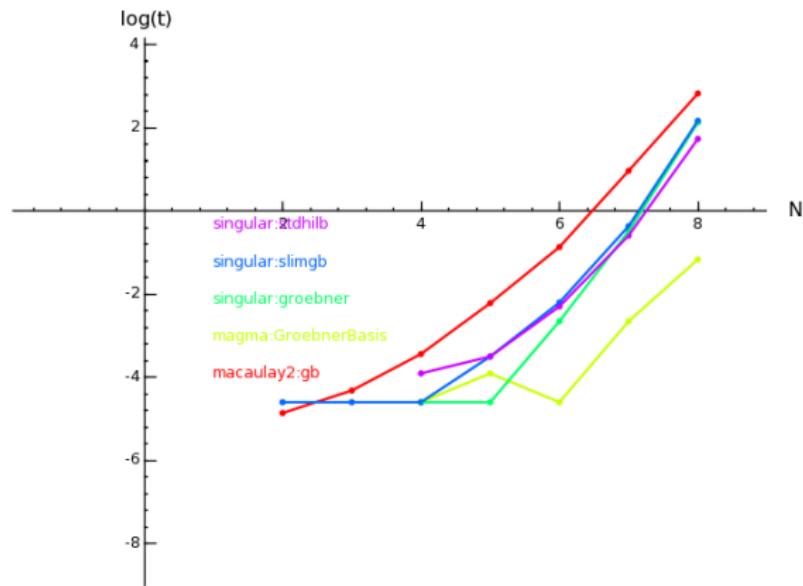


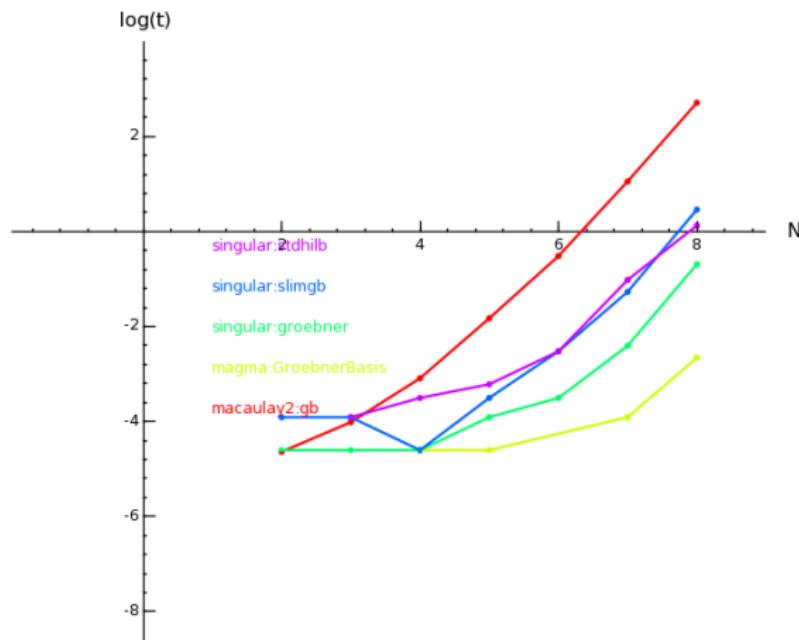
- Comparison certainly unfair: Macaulay2: `walltime()`, Singular & MAGMA: `cputime()`
- **SAGE** interface to Singular faster than the interface to Macaulay2 or MAGMA
- I'm much more familiar with Singular than with Macaulay2 or MAGMA
- Benchmarks performed on `sage.math.washington.edu` (8 Dual-Core Opterons, 64GB RAM)
- Versions: MAGMA V2.12-20, Singular 3-0-2, Macaulay2 0.9.8, **SAGE** 1.3.6.4

Cyclic, degrevlex, \mathbb{Q} 

Cyclic, degrevlex, \mathbb{F}_{127} 

Cyclic, degrevlex, \mathbb{F}_2 

Katsura, degrevlex, \mathbb{Q} 

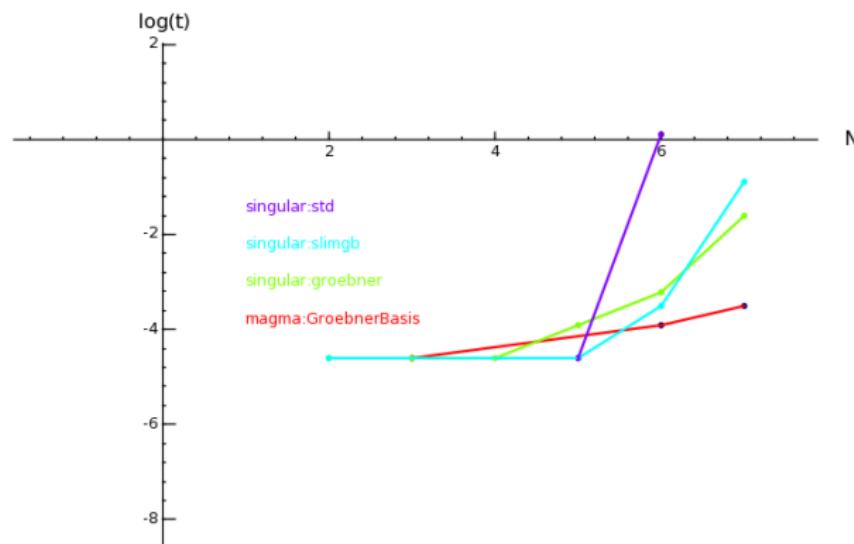
Katsura, degrevlex, \mathbb{F}_{127} 



Results

... take these with a grain of salt

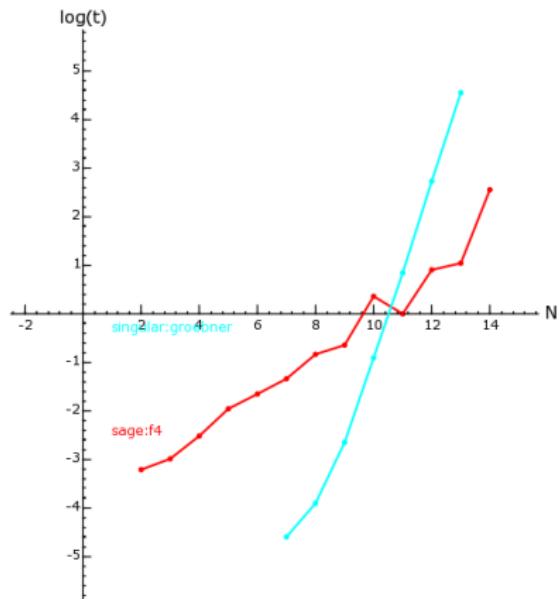
- Macaulay2 seems much slower than Singular
- MAGMA seems always faster than Singular
- Singular's groebner seems to have a good heuristic for the choice of the algorithm
- slimgb – Singular's F_4 variant – is not suited for degrevlex.

Katsura, lex, \mathbb{F}_{32003} 



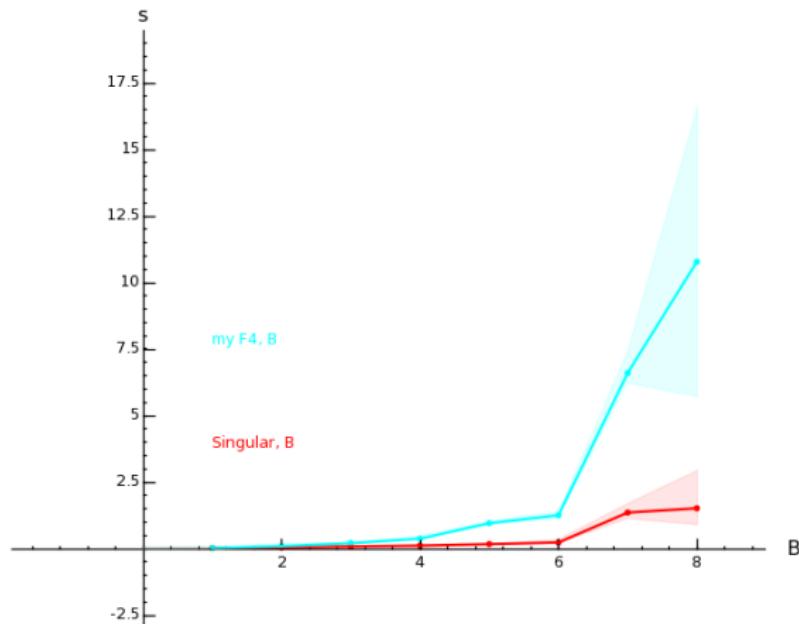
Cyclic, degrevlex, $\mathbb{F}_2[x_0, \dots, x_{n-1}] / \langle FieldIdeal \rangle$

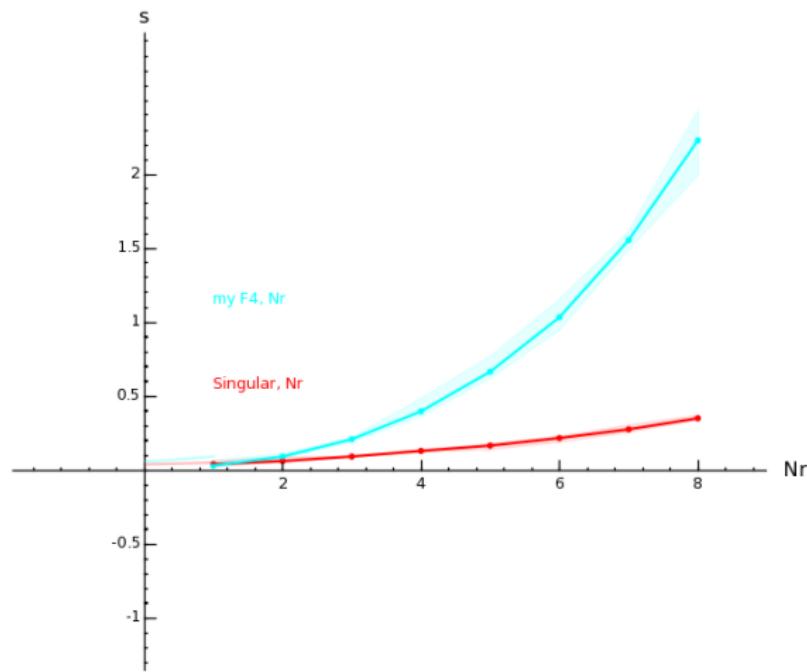
alpha code warning





CTC, Nr = 1, degrevlex

 $\mathbb{F}_2[x_0, \dots, x_{n-1}] / \langle FieldIdeal \rangle$ 

CTC, $B = 1$, degrevlex $\mathbb{F}_2[x_0, \dots, x_{n-1}] / \langle FieldIdeal \rangle$ 



Results

- `slimgb` still seems slower than MAGMA
- I can produce benchmarks where my F_4 implementation outperforms Singular over $\mathbb{F}_2[x_0, \dots, x_{n-1}] / \langle FieldIdeal \rangle$.
 - But it's much slower in general also over this ring.
 - It's embarrassingly slow over \mathbb{F}_p (p prime).



Outline

- 1 Gröbner Bases
- 2 Detour: Algebraic Attacks on Block Ciphers
- 3 Gröbner Bases in SAGE
 - SAGE as a Tool to Compute Gröbner Bases
 - SAGE as a Tool to Develop Gröbner Basis Algorithms
- 4 Benchmarks
- 5 Todo



A Short ToDo List

- Way faster multivariate polynomial arithmetic
 - ETuples: Pyrex → C.
 - Faster finite field arithmetic (e.g. Givaro) (nearly done)

```
sage: R.<s,a,g,e> = PolynomialRing(linbox.GFq(127),4)
sage: time for i in range(10000): _ = s*a + g*e
CPU times: user 3.03 s, sys: 0.02 s, total: 3.04 s
Wall time: 3.12
```

```
sage: v = singular(R.gens(),"list")
sage: singular.eval("poly p=0")
sage: cmd = "for(int i=1;i<10000;i=i+1){p=%s*%s+%s*%s;};p"%(v[1],v[2],v[3],v[4])
sage: time _ = singular.eval(cmd)
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 0.55
```

- Fast sparse linear algebra – mostly (reduced) row echelon form – over any field.
- For crypto: Efficient implementation of $\mathbb{F}_2[x_0, \dots, x_{n-1}] / \langle \text{FieldIdeal} \rangle$ (nearly done)
- A useable but flexible (extensible) F_4 implementation in SAGE
- F_5 would be very cool.



Questions?

Thank You!