Introduction to SageMath

Benjamin Pring

University of Bath

November 18th$^{\text{th}}$, 2016

Outline

What we'll cover today

- Basic introduction to SageMath.
- Pollard-Rho factorisation algorithm.
- Baby-Step Giant Step algorithm.
- Index Calculus algorithm (if time).

What is SageMath?

- Computer Algebra System (CAS)

What is SageMath?

- Computer Algebra System (CAS)
- Open-Source

What is SageMath?

- Computer Algebra System (CAS)
- Open-Source
- Free!

What is SageMath?

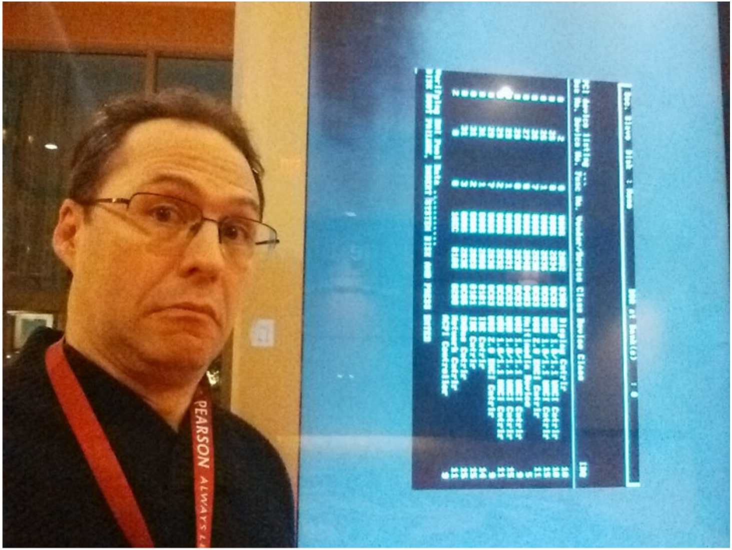
- Computer Algebra System (CAS)
- Open-Source
- Free!
- Useful for research...

A brief summary

- Originated in 2005, by William Stein

A brief summary

- Originated in 2005, by William Stein. Still running in 2015.



A brief summary

- Originated in 2005, by William Stein. Still running in 2015.
- Started as a python interface to GAP (group theory) and PARI (number theory). Has kept growing.

A brief summary

- Originated in 2005, by William Stein. Still running in 2015.
- Started as a python interface to GAP (group theory) and PARI (number theory). Has kept growing.
- Mission statement:
  "Create a viable free open source alternative to Magma, Maple, Mathematica and Matlab."

A brief summary

- Originated in 2005, by William Stein. Still running in 2015.
- Started as a python interface to GAP (group theory) and PARI (number theory). Has kept growing.
- Mission statement:
  "Create a viable free open source alternative to Magma, Maple, Mathematica and Matlab."
- **S**oftware for **A**rithmetic **G**eometry **E**xperimentation

A brief summary

- Originated in 2005, by William Stein. Still running in 2015.
- Started as a python interface to GAP (group theory) and PARI (number theory). Has kept growing.
- Mission statement:
  "Create a viable free open source alternative to Magma, Maple, Mathematica and Matlab."
- ~~**S**oftware for **A**rithmetic **G**eometry **E**xperimentation~~
  Software for **A**lgebraic **G**eometry **E**xperimentation

A brief summary

- Originated in 2005, by William Stein. Still running in 2015.
- Started as a python interface to GAP (group theory) and PARI (number theory). Has kept growing.
- Mission statement:
  "Create a viable free open source alternative to Magma, Maple, Mathematica and Matlab."
- ~~**S**oftware for **A**rithmetic **G**eometry **E**xperimentation~~
  ~~**S**oftware for **Al**gebraic **G**eometry **E**xperimentation~~
  Software for **Al**gebra and **G**eometry **E**xperimentation

A brief summary

- Originated in 2005, by William Stein. Still running in 2015.
- Started as a python interface to GAP (group theory) and PARI (number theory). Has kept growing.
- Mission statement:
  "Create a viable free open source alternative to Magma, Maple, Mathematica and Matlab."

- ~~**S**oftware for **A**rithmetic **G**eometry **E**xperimentation~~
  ~~**S**oftware for **Al**gebraic **G**eometry **E**xperimentation~~
  ~~**S**oftware for **Al**gebra and **G**eometry **E**xperimentation~~
  Sage

A brief summary

- Originated in 2005, by William Stein. Still running in 2015.
- Started as a python interface to GAP (group theory) and PARI (number theory). Has kept growing.
- Mission statement:
  "Create a viable free open source alternative to Magma, Maple, Mathematica and Matlab."
- Sage — Sometimes Acronyms Get Eliminated

A brief summary

- Originated in 2005, by William Stein. Still running in 2015.
- Started as a python interface to GAP (group theory) and PARI (number theory). Has kept growing.
- Mission statement:
  "Create a viable free open source alternative to Magma, Maple, Mathematica and Matlab."
- Sage — Sometimes Acronyms Get Eliminated
- Now consists of 90-odd packages tied together in a Python interface with arbitrary precision arithmetic

A brief summary

- Originated in 2005, by William Stein. Still running in 2015.
- Started as a python interface to GAP (group theory) and PARI (number theory). Has kept growing.
- Mission statement:
  "Create a viable free open source alternative to Magma, Maple, Mathematica and Matlab."
- Sage — Sometimes Acronyms Get Eliminated
- Now consists of 90-odd packages tied together in a Python interface with arbitrary precision arithmetic
- Number theory: PARI, FLINT, NTL
- Combinatorics: Symmetrica
- Numerical linear algebra: ATLAS, BLAS, LAPACK, NumPy
- Plotting: matplotlib
- Algebra: GAP, Maxima
- Statistics: R, SciPy
- ...and more

Why SageMath?

- Computer Algebra System (CAS)
- Provides a method of using different software systems together - great for cryptography!
- Easy syntax owing to python
- Hides implementation details — allows you to focus on the mathematics
- The most complicated calculator you could possibly own...
- Allows quick prototyping and experimentation

Methods of using Sage

- SageMath Cloud — what we'll be using.
- SageMath cell server — embed interactive code in websites[1]
- Unfortunately require either linux or a virtualbox environment for anything more...
- SageMath notebook mode — similar to cloud environment, but run locally.
- SageMath terminal mode. Great emacs integration available.

[1] see `https://wiki.sagemath.org/interact/graphics` for examples.

Language features

- Scripting language (can be run interactively)
- Open source (all code is freely available for inspection)
- Makes use of already mature Computer Algebra Systems and mathematical coding system through interfaces
- Allows use of closed-source systems, such as Magma, Maple or Mathematica if you have them installed locally (or on HPC machines).
- All bundled together in an easy to use python interface with arbitrary precision arithmetic
- Advantages: use the best parts of many different languages.
- Disadvantages: an efficiency cost for converting between objects from different software packages.

Getting help

- SageMathCloud — inbuilt using "tab" between function brackets.
- Terminal — Help: functionName?
Display code: functionName??
- Webhelp — help pages online are directly compiled from code. SageMath functions for inclusion are designed to be auto-documenting with examples of use
- Forums — (advantages of open-source community spirit...)

Sagemath basics — Python

- Sagemath is built upon Python, where indentation is key for defining structures such as loops, conditionals and functions.
- Basic printing commands are easily available

```
sage: print "Hello world"
Hello world
sage: print "This is the 1st line"
This is the 1st line
sage: print "This is the " + str(2) + "nd line."
This is the 2nd line.
sage: print "Pi rounded to 5 places is:", \
round(pi,5)
Pi rounded to 5 places is: 3.14159
sage: print "Command 1"; print "Command 2"
Command 1
Command 2
sage: # We comment out code with the hash symbol
sage: # print "Commented out"
sage:
```

Sagemath basics — variables, equality

```
sage: a = 1
sage: print a
1
sage: a == 1
True
sage: 3 * 4
12
sage: 3**4
81
sage: 3^4
81
sage: 12 / 5
12/5
sage: 12/5.n()
2.40000000000000
sage: 12 // 5
2
sage: floor(1.6)
1
sage: ceil(1.6)
2
```

Python basics — lists

```
sage: v = range(10)
sage: v
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
sage: w = range(5,10)
sage: w
[5, 6, 7, 8, 9]
sage: u = [i**2 % 5 for i in xrange(5)]
sage: u
[0, 1, 4, 4, 1]
sage: u[0:3]
[0,1,4]
sage: u[2:4]
[4,4]
sage: u[-1]
1
sage: u[2] = 99; u
[0, 1, 99, 4, 1]
sage: len(u)
5
```

Python basics — sets

```
sage: A = {1,2,3,4,4}
sage: A
{1, 2, 3, 4}
sage: B = {2,4,6,8}
sage: B
{2, 4, 6, 8}
sage: A.union(B)
{1, 2, 3, 4, 6, 8}
sage: A.intersection(B)
{2, 4}
sage: 2 in A
True
sage: 999 in A
False
sage: A.remove(2)
sage: A
{1, 3, 4}
```

Dictionaries

```
sage: D = {"Alice": 55, "Bob" : 55, "Eve" : "???"}
sage: D.update({99 : "flake"})
sage: D["Alice"]
55
sage: D["Eve"]
'???'
sage: D[99]
'flake'
```

Python basics — if

```
if condition:
    statements
else:
    statements
```

Note that pass will count as "Do nothing"

```
sage: x = 2**40
if x % 3 == 0:
   print x, "is a multiple of 3"
elif x % 3 == 1:
   print x, "- 1 is divisible by 3"
else:
   print x, "+ 1 is divisible by 3"
sage: 1099511627776 -1 is divisible by 3
```

Python basics — for

```
for item in iterable:
    statements #may reference the current item
```

```
sage: for i in range(10):
....:     print i
....:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9

sage: for i in range(20,0,-2):
....:     print i
....:
20, 18, 16, 14, 12, 10, 8, 6, 4, 2

sage: # valid for iterables,ie. sets,dictionaries
sage: D = {"Alice": "Honest", "Bob": "Honest",\
  "Eve": "Dishonest"}
sage: for i in D:
          print i, D[i]
....:
Bob Honest
Alice Honest
Eve Dishonest
```

Python basics — while

```
while condition:
    statements
```

```
sage: x = 2
sage: while x < 10:
    x += 3
    print x
....:
5
8
11
```

Python basics — functions

```
def functionName(var1,var2,...,varN):
    statements
    return VALUE
```

Functions may also take default input by placing an equality sign next to the variable. All such variables with default values must be left-most from the standard variables.

```
def foo(a,b,c=0):
    temp = a**b + b**a
    return temp + c
sage: foo(2,3,0)
17
sage: foo(2,3,10)
27
sage: foo(2,3)
17
```

Python basics — commenting

```
sage: # Single line commenting with hash symbols
sage: def test():
...     """Documentation is placed within functions
            via triple-quotes and may be
            auto-extracted by the sphinx document
            generator.
            Sage documentation online is extracted
            by this method."""
...     pass
...
sage: test()
sage:
sage: """This will result in a string in terminal"""
'This will result in a string in terminal'
sage:
```

```
"""Triple quote commenting may also be used in
scripting files"""
```

SageMath — Rings and Fields I

- $\mathbb{Z}$ — ZZ
- $\mathbb{Q}$ — QQ
- $\mathbb{R}$ — RR
- $\mathbb{C}$ — CC

```
sage: a = 1
sage: a.parent()
Integer Ring
sage: b = 3/4
sage: b.parent()
Rational Field
sage: c = 1.1
sage: c.parent()
Real Field with 53 bits of precision
sage: d = 1 + CC(i)
sage: d.parent()
Complex Field with 53 bits of precision
sage: a = RR(a)
sage: a.parent()
Real Field with 53 bits of precision
```

SageMath — Rings and Field II

$GF(p^k)$ — GF(p**k, "x")

$R[X]$ — R["X"]

```
sage: F = GF(5**3,"x")
sage: F.gen()
x
sage: F.random_element()
x^2 + 4*x
sage: F.modulus()
x^3 + 3*x + 3
sage: F.base()
Finite Field of size 5
sage: F.degree()
3
sage: F = GF(5**3, name="a",\
modulus = x**3 + x**2 + 3*x**1 + 1)
sage: F
Finite Field in a of size 5^3
sage: F.gen()
a
sage: FX = F["X"]
sage: FX.random_element(degree=2)
2*a*X^2 + (2*a^2 + 4*a + 4)*X + 4*a^2 + 3
```

SageMath — Matrices I

```
sage: M = matrix(ZZ,3,3,1)
sage: M
[1 0 0]
[0 1 0]
[0 0 1]
sage: N = matrix(QQ,[[1,2,3],[4,5,6],[7,8,9],[10,11,1
sage: N
[ 1  2  3]
[ 4  5  6]
[ 7  8  9]
[10 11 12]
sage: M.stack(N)
[ 1  0  0]
[ 0  1  0]
[ 0  0  1]
[ 1  2  3]
[ 4  5  6]
[ 7  8  9]
[10 11 12]
sage: N = N.transpose(); N
[ 1  4  7 10]
[ 2  5  8 11]
[ 3  6  9 12]
```

SageMath — Matrices II

```
sage: A = matrix(QQ,[[1,2,3],[4,5,6],[7,8,9]]); A
[1 2 3]
[4 5 6]
[7 8 9]
sage: b = vector(QQ,[1,2,3])
sage: A.solve_right(b)
(-1/3, 2/3, 0)
sage: A.solve_left(b)
(1, 0, 0)
sage: A.echelon_form()
[ 1  0 -1]
[ 0  1  2]
[ 0  0  0]
sage: A.determinant()
0
sage: A.change_ring
A.change_ring
sage: A.parent()
Full MatrixSpace of 3 by 3 dense matrices over
Rational Field
sage: A = A.change_ring(IntegerModRing(11))
sage: A.parent()
Full MatrixSpace of 3 by 3 dense matrices over
Ring of integers modulo 11
```

SageMath — Polynomials

```
sage: R = PolynomialRing(ZZ,"X");R
Univariate Polynomial Ring in X over Integer Ring
sage: f = R.random_element(degree=3);f
X^3 - 4*X^2 - X + 6
sage: Rf = R.quotient_ring(f)
sage: g = R.random_element(degree=10);g
-2*X^10 - 6*X^8 - 3*X^7 - 57*X^6 + X^5 + X^4 + X^2 -
sage: Rf(g)
-154397*Xbar^2 + 22175*Xbar + 239965
sage: h = Rf(g);h
-154397*Xbar^2 + 22175*Xbar + 239965
sage: h.lift()
-154397*X^2 + 22175*X + 239965
sage: h.lift().parent()
Univariate Polynomial Ring in X over Integer Ring
```

SageMath — Primes, number theory and random numbers

```
sage: p = random_prime(2**11,lbound=2**10)
sage: p
1997
sage: p.binary()
'11111001101'
sage: p.is_prime()
True
sage: (p+1).is_prime()
False
sage: ZZ.random_element(40,50)
47
sage: [ZZ.random_element(40,50) for _ in xrange(10)]
[44, 43, 40, 45, 48, 42, 43, 42, 47, 42]
sage: CC.random_element()
0.731798085535406 - 0.0337554359009657*I
sage: FX.random_element()
(4*x^2 + 4*x + 3)*X^2 + (4*x^2 + 2*x)*X + 4*x^2 + x +
```

Some examples of crypto with SageMath

Block Ciphers
`http://tinyurl.com/OX-SAGE-BLOCKC`
ElGamal
`http://tinyurl.com/OX-SAGE-ELGAMAL`

Timing your code

```
sage:timeit('factor(2**200+1)')
5 loops, best of 3: 78.3 ms per loop
sage: %time factor(2**200+1)
CPU times: user 136 ms, sys: 0 ns, total: 136 ms
Wall time: 135 ms
257 * 1601 * 25601 * 82471201 * 4278255361 * 43236320
sage: %prun factor(2**200+1)
         8 function calls in 0.134 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename
        1    0.134    0.134    0.134    0.134 {method
        1    0.000    0.000    0.134    0.134 <string
        1    0.000    0.000    0.134    0.134 arith.p
        2    0.000    0.000    0.000    0.000 {isinsta
        1    0.000    0.000    0.000    0.000 factori
        1    0.000    0.000    0.000    0.000 proof.p
        1    0.000    0.000    0.000    0.000 {method
```

Features to be aware about

- Easy parallel processing support - viable for quick HPC code.
- Cython — optimise specific portions of code in C for speedup.
- Good emacs support.

$\LaTeX$ integration

- SageTex — include Sage code results/graphs directly into your documents

LaTeX integration

- SageTex — include Sage code results/graphs directly into your documents

$$2^2 \cdot 3^3 \cdot 13$$

```
\begin{equation*}
  \sage{factor(1404)}
\end{equation*}
```

LaTeX integration

- SageTex — include Sage code results/graphs directly into your documents

$$2^2 \cdot 3^3 \cdot 13$$

```
\begin{equation*}
  \sage{factor(1404)}
\end{equation*}
```

- Creation of LaTeX code from within Sagemath

LaTeXintegration

- SageTex — include Sage code results/graphs directly into your documents

$$2^2 \cdot 3^3 \cdot 13$$

```
\begin{equation*}
  \sage{factor(1404)}
\end{equation*}
```

- Creation of LaTeXcode from within Sagemath

```
sage: var('a b c x y z')
(a, b, c, x, y, z)
sage: f = a*sin(b*x + y) + exp(15*x*y)**z
sage: latex(f)
a \sin\left(b x + y\right) + \left(e^{\left(15 \, x y
```

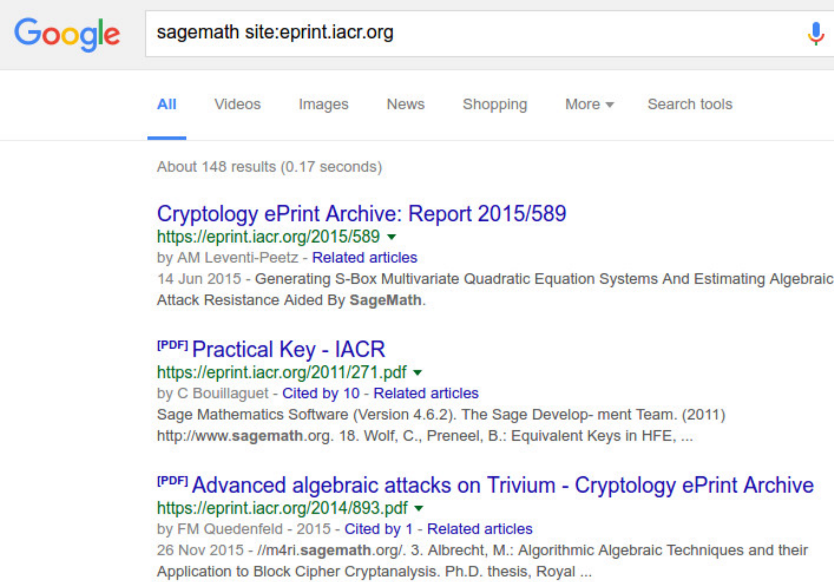
$$a \sin\left(bx + y\right) + \left(e^{\left(15\, xy\right)}\right)^z$$

Other useful features of Cryptographers

- Pycrypto
- hashlib
- Toy ciphers
- Statistics through the R interface.

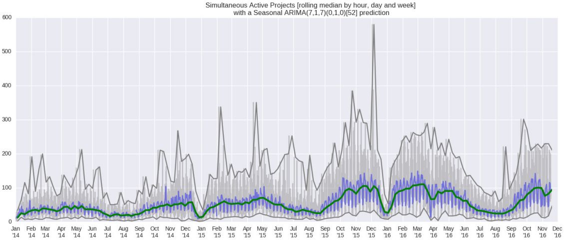
Popular usage

- Popularity in crypto (and other Math...) continues to grow



Popular usage

- Popularity in crypto (and other Math...) continues to grow
- SageMathCloud usage hitting new records each year



Popular usage

- Popularity in crypto (and other Math...) continues to grow
- SageMathCloud usage hitting new records each year
- Twitter / community activity: A break of the GGH13 Multilinear Map in 2015 scheme really started to get attention when a toy version of the break was published online using a SageCell interface by Martin Albrecht.

Use in research culture

- Popularity in crypto (and other Math...) continues to grow
- SageMathCloud usage hitting new records each year
- Recent break of the GGH13 Multilinear Map scheme really started to get attention when a toy version of the break was published online using a SageCell interface by Martin Albrecht.
- SageMath citations growing each year

Use in research culture

- Popularity in crypto (and other Math...) continues to grow
- SageMathCloud usage hitting new records each year
- Recent break of the GGH13 Multilinear Map scheme really started to get attention when a toy version of the break was published online using a SageCell interface by Martin Albrecht.
- SageMath citations growing each year
- Popular uses include prototyping and quick programs for parameter estimation — see papers on choosing parameters for cryptosystems based on hardness of LWE: `https://eprint.iacr.org/2015/046`

Use in research culture

- Popularity in crypto (and other Math...) continues to grow
- SageMathCloud usage hitting new records each year
- Recent break of the GGH13 Multilinear Map scheme really started to get attention when a toy version of the break was published online using a SageCell interface by Martin Albrecht.
- SageMath citations growing each year
- Popular uses include prototyping and quick programs for parameter estimation — see papers on choosing parameters for cryptosystems based on hardness of LWE: `https://eprint.iacr.org/2015/046`
- Provides an alternate way to think about problems via experimentation to build and confirm intution.

Implementation — Why bother?

- Helps with your intution

Implementation — Why bother?

- Helps with your intution
- *Ensures* you understand the algorithm

Implementation — Why bother?

- Helps with your intution
- *Ensures* you understand the algorithm
- Can throw out suprising results

Implementation — Why bother?

- Helps with your intution
- *Ensures* you understand the algorithm
- Can throw out suprising results
- Allows easy testing of counter-examples and conjectures

Implementation — Why bother?

- Helps with your intution
- *Ensures* you understand the algorithm
- Can throw out suprising results
- Allows easy testing of counter-examples and conjectures
- Is is the scheme practical? Asymptotically yes, but what about in the real world?

Implementation — Why bother?

- Helps with your intution
- *Ensures* you understand the algorithm
- Can throw out suprising results
- Allows easy testing of counter-examples and conjectures
- Is is the scheme practical? Asymptotically yes, but what about in the real world?
- It's vaguely satisfying...

Implementation — Why bother?

- Helps with your intution
- *Ensures* you understand the algorithm
- Can throw out suprising results
- Allows easy testing of counter-examples and conjectures
- Is is the scheme practical? Asymptotically yes, but what about in the real world?
- It's vaguely satisfying...
- Afternoon's goal: Implement a discrete logarithm solver and factorization method. Experiment.

Tools

- SageMathCloud
- SageMath help
- People around you
- Me

Pollard's ρ factorization method

- Input: N — an integer to be factored
- Input: g — a polynomial in x, usually $g(x) := x^2 + 1$, which is always computed mod N.
- Output: A small factor of N.
- Initialization: $x = 2, y = 2, d = 1$
- Algorithm:

```
while d is 1:
    x = g(x)
    y = g(g(y))
    d = gcd(|x − y|, n)
if d is n:
    return Fail
else:
    return d
```

Pollard in Sage

`http://tinyurl.com/OX-2016-POLLARD-RHO`

Baby step, giant step (BSGS)

- Input: a cyclic group $G = \langle g \rangle$ of prime order p.
- Input: $h \in G$.
- Output: find the value of x s.t. $h = g^x$.
- Let $N' = \lceil \sqrt{|G|} \rceil$
- There exist $0 \leq i, j < N'$ such that $x = jN' + i$

$$h = g^{jN'+i} \Leftrightarrow hg^{-jN'} = g^i$$

- Compute $L_B := \{g^i | i = 0, \ldots, N' - 1\}$
- Compute $L_G := \{hg^{-jN'} | j = 0, \ldots, N' - 1\}$
- Look for the same values in each list (note that you only have to create one list and can use a loop for the other)
- Attack requires time and memory each $\mathcal{O}\left(|G|^{1/2}\right)$
- Extension: What happens when we alter the size of N'?

Implementation goals

- We consider applying the BSGS algorithm to the discrete logarithm problem in $(\mathbb{Z}_p)^*$ — that is, given $g, h \in (\mathbb{Z}_p)^*$ st. $\langle g \rangle = (\mathbb{Z}_p)^*$, find x st. $g^x = h$.
- Create a BSGS function definition which should initially be defined as:

```
def BSGS(h, g):
    pass
```

and return x, cast as an integer.

If you have time and wish to experiment, this may be expanded to

```
def BSGS(h, g, f):
    pass
```

where BSGS should use f as a method of choosing N' based upon $|G|$.

BSGS in Sage

`http://tinyurl.com/OX-2016-BSGS`

A naive index calculus algorithm for $\mathbb{F}_p^*$

- DLP: given $g, h \in \mathbb{F}_p^*$, find x such that $h = g^x$
- Factor basis made of **small primes**

$$\mathcal{F}_B := \{\text{primes } p_i \leq B\} = \{p_1, \ldots, p_k\}$$

- **Relation search**
  - Compute $g_i := g^{a_i}$ for random $a_i \in \{1, \ldots, p-1\}$
  - **If** all factors of g_i are $\leq B$, we have a relation

$$g^{a_i} = \prod_{p_j \in \mathcal{F}} p_j^{e_{i,j}} \tag{1}$$

- **Linear algebra** Once we have $\ell \geq k$ linearly independent equations similar to equations (1), we solve $(\mod (p-1))$ for $\log_g p_i$, $i = 1, \ldots, k$.
- Search for t such that $[g^t \cdot h \mod p]$ is B-smooth. Once found, solve for $\log_g h$.

Index Calculus in Sage

http://tinyurl.com/OX-SAGE-INDEXCALC

Language interfaces/Behind the scenes

```
sage: M = gap('[[1,2,3],[4,5,6],[7,8,9]]')
sage: N = gap('[[1,1,1],[2,2,2],[3,3,3]]')
sage: timeit('N*M')
625 loops, best of 3: 436 µs per loop
sage: M.parent()
Gap
sage: (N*M).parent()
Gap
sage: timeit('matrix(ZZ,N*M)')
25 loops, best of 3: 11.6 ms per loop
sage: timeit('matrix(ZZ,N)*matrix(ZZ,M)')
25 loops, best of 3: 22.3 ms per loop
sage:
sage: Ms = matrix(ZZ,M)
sage: Ns = matrix(ZZ,N)
sage: timeit('Ms*Ns')
625 loops, best of 3: 5.94 µs per loop
sage: (Ms*Ns).parent()
Full MatrixSpace of 3 by 3 dense matrices over \
 Integer Ring
sage: type(Ms*Ns)
<type 'sage.matrix.matrix_integer_dense.Matrix_\
integer_dense'>
```

Contributing to SageMath

- Sage has gained limited support from OpenDreamKit — a European research project to further software used in research computing to provide a professional full-time developer.
- Open-souce nature means that *you* can contribute, bug-fix and keep the project going — good for future employment opportunities, both in academia and industry.

Sage days

- Get-togethers to discuss the future of Sage and code
- Worldwide locations
- #71 was at Oxford last year, with a focus on $p-$adic number theory in March.
- #83 Held recently in Morroco on Combinatorics and Knot Theory.
- #82 to be held in Paris in January on Women in Sage.
- Yearly school on Computational Discrete Mathematics held with focus on GAP and SageMath (`http://www.codima.ac.uk/`)

That's all folks!

Questions?