# Teaching finite fields with open-source CAS

*Alasdair McAndrew*
Alasdair.McAndrew@vu.edu.au
School of Engineering and Science
Victoria University
PO Box 14428, Melbourne 8001
Australia

**Abstract**

Finite fields have long been studied for their intrinsic interest, and more recently for their uses in the definition of some modern cryptographic systems. The Advanced Encryption Standard is based on the cryptosystem Rijndael [2], which makes extensive use of finite fields in its computation. At Victoria University, a cryptography course has been taught to students both locally, and transnationally using the medium of the Access Grid [1]; this course has been running for 10 years in various forms, usually with about 30–40 students each year. Many of these students have limited exposure to modern abstract algebra, and the use of a Computer Algebra System has been vital to aid their understanding and assimilation of the material. Over the years we have used Maple, Maxima, Axiom and Sage. This article concentrates on our use of the last three, and shows that for abstract algebra, the open source systems are far superior to the alternatives.

## 1 Introduction to (finite) fields

A *field* may be broadly considered as a mathematical system in which the operations of addition, subtraction, multiplication and division (except by zero) are all defined and well-behaved: addition and multiplication are commutative and associative, subtraction and division are the "inverses" of addition and multiplication respectively, and addition is distributive over multiplication.

More formally, a field is a set $X$ with two distinguished elements zero and one, and two operations addition and multiplication for which

1. For any two elements $x, y \in X$, $x + y$ and $xy$ are defined, and $x + y = y + x$, $xy = yx$, $(x + y) + z = x + (y + z)$, and $(xy)z = x(yz)$.
2. Zero and one are the identity elements for addition and multiplication respectively: for all $x \in X$, $x + 0 = x$ and $1x = x$.
3. For every $x \in X$, there exists $y = -x$ for which $x + y = 0$, and for every $0 \neq x \in X$ there exists $y = x^{-1}$ for which $xy = 1$.
4. For every $x, y, z \in X$, $x(y + z) = xy + xz$.

Examples of everyday fields are the rational numbers $\mathbb{Q}$, the real numbers $\mathbb{R}$, and the complex numbers $\mathbb{C}$. These fields have an infinite number of elements.

We are more interested in *finite* fields, being fields in which the number of elements is finite. Such fields are also called *Galois Fields* after the French mathematician Évariste Galois, who was the first to discuss them.

One simple example consists of the residues modulo 7; that is, the integers 0, 1, 2, 3, 4, 5 and 6, with addition and subtraction modulo 7.

Addition and multiplication tables are:

| + | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 0 |
| 2 | 2 | 3 | 4 | 5 | 6 | 0 | 1 |
| 3 | 3 | 4 | 5 | 6 | 0 | 1 | 2 |
| 4 | 4 | 5 | 6 | 0 | 1 | 2 | 3 |
| 5 | 5 | 6 | 0 | 1 | 2 | 3 | 4 |
| 6 | 6 | 0 | 1 | 2 | 3 | 4 | 5 |

| × | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 0 | 2 | 4 | 6 | 1 | 3 | 5 |
| 3 | 0 | 3 | 6 | 2 | 5 | 1 | 4 |
| 4 | 0 | 4 | 1 | 5 | 2 | 6 | 3 |
| 5 | 0 | 5 | 3 | 1 | 6 | 4 | 2 |
| 6 | 0 | 6 | 5 | 4 | 3 | 2 | 1 |

These tables can be seen to define a field in that all of the requirements are satisfied. Commutativity and distributivity follow from the definitions of addition and multiplication; the existence of additive inverses can be noted by observing that every row (or column) of the first table has a single zero, the existence of multiplicative inverses by every row (or column) of the second table—except for the zero row and column—having a single 1.

It is an elementary result that the residues of any prime number $p$ form a finite field; such a field is denoted $GF(p)$, or $\mathbb{Z}/p\mathbb{Z}$ or sometimes as $\mathbb{Z}_p$. The tables above thus define the field $GF(7)$.

Finite fields can be shown to exist for any order $p^n$, where $p$ is prime, and for no others. Furthermore, any two fields of the same order are isomorphic. Straightforward proofs of these results are given by McEliece [7].

To construct a field $F = GF(p^n)$ of order $p^n$, first choose an irreducible polynomial $p(x)$ of degree $n$ over $GF(p)$. Then the elements of $F$ will be all polynomials of degree $n-1$ or less; addition is performed modulo $p$, and multiplication is performed modulo $p(x)$. The irreducibility of $p(x)$ will ensure that inverses exist.

For example, the field $GF(8)$ can be constructed using the polynomial $p(x) = x^3+x+1$—this polynomial is irreducible over $GF(2)$. The addition table is:

| + | 0 | $x$ | $x^2$ | $x+1$ | $x^2+x$ | $x^2+x+1$ | $x^2+1$ | 1 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | $x$ | $x^2$ | $x+1$ | $x^2+x$ | $x^2+x+1$ | $x^2+1$ | 1 |
| $x$ | $x$ | 0 | $x^2+x$ | 1 | $x^2$ | $x^2+1$ | $x^2+x+1$ | $x+1$ |
| $x^2$ | $x^2$ | $x^2+x$ | 0 | $x^2+x+1$ | $x$ | $x+1$ | 1 | $x^2+1$ |
| $x+1$ | $x+1$ | 1 | $x^2+x+1$ | 0 | $x^2+1$ | $x^2$ | $x^2+x$ | $x$ |
| $x^2+x$ | $x^2+x$ | $x^2$ | $x$ | $x^2+1$ | 0 | 1 | $x+1$ | $x^2+x+1$ |
| $x^2+x+1$ | $x^2+x+1$ | $x^2+1$ | $x+1$ | $x^2$ | 1 | 0 | $x$ | $x^2+x$ |
| $x^2+1$ | $x^2+1$ | $x^2+x+1$ | 1 | $x^2+x$ | $x+1$ | $x$ | 0 | $x^2$ |
| 1 | 1 | $x+1$ | $x^2+1$ | $x$ | $x^2+x+1$ | $x^2+x$ | $x^2$ | 0 |

and the multiplication table is

| $\times$ | 0 | $x$ | $x^2$ | $x+1$ | $x^2+x$ | $x^2+x+1$ | $x^2+1$ | 1 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $x$ | 0 | $x^2$ | $x+1$ | $x^2+x$ | $x^2+x+1$ | $x^2+1$ | 1 | $x$ |
| $x^2$ | 0 | $x+1$ | $x^2+x$ | $x^2+x+1$ | $x^2+1$ | 1 | $x$ | $x^2$ |
| $x+1$ | 0 | $x^2+x$ | $x^2+x+1$ | $x^2+1$ | 1 | $x$ | $x^2$ | $x+1$ |
| $x^2+x$ | 0 | $x^2+x+1$ | $x^2+1$ | 1 | $x$ | $x^2$ | $x+1$ | $x^2+x$ |
| $x^2+x+1$ | 0 | $x^2+1$ | 1 | $x$ | $x^2$ | $x+1$ | $x^2+x$ | $x^2+x+1$ |
| $x^2+1$ | 0 | 1 | $x$ | $x^2$ | $x+1$ | $x^2+x$ | $x^2+x+1$ | $x^2+1$ |
| 1 | 0 | $x$ | $x^2$ | $x+1$ | $x^2+x$ | $x^2+x+1$ | $x^2+1$ | 1 |

These tables can be seen to define a field in the same way as the tables for $GF(7)$ above. This field can be denoted $\mathbb{Z}_2[x]/(x^3+x+1)$; this notation identifies the base prime (in this case 2) as well as the irreducible polynomial used to define the field.

# 2 Finite fields in Maxima, Axiom and Sage

At Victoria University, we used first Maxima and Axiom, and then switched to Sage. Maxima [5] is a descendant of the venerable CAS Macsyma, and in its current form has been fast gathering adherents for its algebraic and analytic strength, and its implementation on all modern operating systems. The author is one of the principal authors of the finite fields package for Maxima [6]. Here is an example of a simple Maxima session, which must start with loading the `gf` package:

```
(%i1) load(gf);
```

We shall define the field $GF(16) = \mathbb{Z}_2[x]/(x^4+x+1)$:

```
(%i2) gf_set(2,4,x^4+x+1);
(%o2)                          true
```

The command `gf_set` checks that $p$ is prime, and the generating polynomial (in this case $x^4+x+1$) is irreducible. If these conditions are satisfied, the command performs some background calculations, and returns true. Addition, multiplication and inversion are performed with the commands `gf_add`, `gf_mul` and `gf_inv` respectively. For example:

```
(%i3) p:x^3+x+1;
                                3
(%o3)                          x + x + 1
(%i4) q:gf_inv(p);
                                2
(%o4)                          x + 1
(%i5) gf_mul(p,q);
(%o5)                           1
```

In Maxima, it is only possible to work with one finite field at a time. However, in practice this is not a major restriction.

Matrices over the field can be defined and arithmetic performed:

```
(%i6) M:genmatrix(lambda([i,j],gf_rand()),3,3);
```

$$\text{(\%o6)} \quad \begin{pmatrix} x^3+x^2 & x^2+x & x^3+x^2+x \\ x^3+1 & x^3+x^2 & x^3+x^2+x+1 \\ x^2+1 & 1 & x^3+x \end{pmatrix}$$

```
(%i7) MI:gf_matinv(M);
```

$$\text{(\%o7)} \quad \begin{pmatrix} 1 & x^3+1 & 0 \\ x^2+1 & x^2+x & x \\ 0 & x+1 & x^2+x+1 \end{pmatrix}$$

```
(%i8) gf_matmul(M,MI);
```

$$\text{(\%o8)} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Axiom [3] is an open-source system which started life as the commercial system ScratchPad, produced by IBM. It has a more powerful approach to finite fields, but suffers from being less transparent than Maxima, and harder to use and to learn. There is a native Windows version, but it only runs in a console. Under Linux, Axiom can run in a console, or in TeXmacs, which provides for LaTeX-formatted output. Under Linux, Axiom also has a fully hypertext help browser and graphics.

Here are the above commands in Axiom:

```
(1) -> F:=FFP(PF 2,x^4+x+1)
  (1)
      FiniteFieldExtensionByPolynomial(PrimeField 2,?**4+?+1) Type:  Domain
```

Axiom requires that a "generator" of the field be defined: an element which can be used to generate all polynomials which constitute the field:

```
(2) -> x := generator()$F
  (2)  %A
                Type:  FiniteFieldExtensionByPolynomial(PrimeField 2,?**4+?+1)
```

Axiom uses a "dummy variable"—in this case $\%A$—to describe all elements of the field. Now all operations can be done as above:

```
(3) -> p := x^3+x+1
  (3)  %A^3 + %A + 1
                Type:  FiniteFieldExtensionByPolynomial(PrimeField 2,?**4+?+1)
(4) -> q := 1/p
  (4)  %A^2 + 1
                Type:  FiniteFieldExtensionByPolynomial(PrimeField 2,?**4+?+1)
```

The use of "types" (mathematical domains of operation) in Axiom means that operator overloading can be used. Since $p$ is known to be an element of the field, the notation `1/p` is automatically defined for and produces output in the field.

Matrices within the field are easy:

```
(5) -> M := matrix([[random()$F for i in 1..3] for j in 1..3])
```

$$\text{(5)} \quad \begin{bmatrix} \%A^3+\%A^2+\%A & 0 & \%A+1 \\ \%A^3+\%A^2+\%A+1 & \%A^3+\%A & \%A^3+\%A^2+\%A \\ \%A & \%A^3 & \%A^3+\%A^2+\%A+1 \end{bmatrix}$$

```
      Type:  Matrix FiniteFieldExtensionByPolynomial(PrimeField 2,?**4+?+1)
```

```
(6) -> MI := inverse(M)
```

$$(6)\quad \begin{bmatrix} \%A^3+\%A & \%A^2+1 & \%A^3+1 \\ \%A^3+\%A & \%A^2+\%A & 1 \\ \%A^3+\%A^2 & 1 & \%A^3+\%A^2 \end{bmatrix}$$

```
                Type: Union(Matrix FiniteFieldExtensionByPolynomial(PrimeField
2,?**4+?+1),...)
(7) -> M*MI
```

$$(7)\quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
              Type: Matrix FiniteFieldExtensionByPolynomial(PrimeField 2,?**4+?+1)
```

Sage [12] in many ways is the best of both worlds: it combines the ease of use of Maxima with the power of Axiom. Its main disadvantage is its reliance on Linux; there is, at the time of writing, no native version for Windows (although one is in active development). This is ameliorated somewhat by running Sage from a server, rather than locally, and accessing Sage through its browser-based "notebook" interface. Here are the above examples in Sage:

```
sage: F.<a> = GF(16,x^4+x+1)
sage: p = a^3+a+1
sage: q = 1/p; q
```

$$a^2+1$$

```
sage: M = random_matrix(F,3,3); M
```

$$\begin{bmatrix} a^3+a^2+1 & a^2+a+1 & a^2+a+1 \\ a^3+a+1 & a^3+a^2 & a^3+a^2+1 \\ 1 & a^2+1 & a^2+a+1 \end{bmatrix}$$

```
sage: MI = M.inverse(); MI
```

$$\begin{bmatrix} a^3 & a^3 & a^2 \\ a^3+a^2 & a^2+1 & a \\ a^2+a & a^3+1 & a^2 \end{bmatrix}$$

```
sage: M*MI
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Both Sage and Axiom support domains or types: every object is an element of a domain, or alternatively has a specific type. Once an object is defined, every future operation on that object will inherit the original type. This makes for highly concise and elegant operations on algebraic objects. In Maple, by comparison, there is only a rudimentary domains system, in consequence of which operations on fields are clumsy and confusing. There is no easy built-in way, for example, of inverting a matrix over a finite field, or of multiplying two matrices. Although Maple is now at version 16, from version 11 when we used it, the domains system has not been significantly enhanced. This is one of the reasons that we stopped using Maple for the cryptography subject.

# 3 The finite field syllabus, and laboratory work

The syllabus of the finite field part of the cryptography subject consisted of the following topics:

1. Basic definitions and examples.
2. Constructing finite fields of orders $p^k$, with $k \geq 2$.
3. Arithmetic in a finite field.
4. Primitive elements (whose powers generate all non-zero elements of the field).
5. Discrete logarithms in the field: for example, if $a^6 = a^3 + a^2$ in $\mathbb{Z}_2[x]/(x^4 + x + 1)$ then we can write $\log_a(a^3 + a^2) = 6$.
6. Using tables of logarithms and powers to facilitate multiplication, inversion, and in general raising to powers.
7. Application of finite fields to cryptography, in particular to the workings of the Advanced Encryption Standard (AES).

For the AES, the 128-bit plaintext is divided into 16 bytes; these are placed into a $4 \times 4$ matrix, column by column, and each byte is treated as an element of the field

$$GF(256) = \mathbb{Z}_2[x]/(x^8 + x^4 + x^3 + x + 1).$$

At the heart of the AES is an operation called **MixColumn**, which multiplies the current state of bytes by the matrix

$$M = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix}$$

Decryption uses the inverse operation, denoted **InvMixColumn** which is a multiplication by $M^{-1}$. As we have seen above, in each of Maxima, Axiom and Sage a matrix inversion over a finite field is simple to compute. In the definition of the AES this matrix is given as

$$M^{-1} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} = \begin{bmatrix} x^3+x^2+x & x^3+x+1 & x^3+x^2+1 & x^3+1 \\ x^3+1 & x^3+x^2+x & x^3+x+1 & x^3+x^2+1 \\ x^3+x^2+1 & x^3+1 & x^3+x^2+x & x^3+x+1 \\ x^3+x+1 & x^3+x^2+1 & x^3+1 & x^3+x^2+x \end{bmatrix}$$

Sage comes with commands to transfer between elements of a binary field and their decimal equivalents, so students can see for themselves how the two matrices are related. First define the field and the matrix:

```
sage: F.<a> = GF(256,x^8 + x^4 + x^3 + x + 1)
sage: M=matrix([[2,3,1,1],[1,2,3,1],[1,1,2,3],[3,1,1,2]])
```

Then turn the matrix into elements of the field:

```
sage: MF = map_threaded(lambda i: F.fetch_int(i),M)
```

Invert the matrix:

```
sage: MIF = MF.inverse()
```

and convert from field elements to integers:

```
sage: map_threaded(lambda i:ZZ(i.int_repr()),MIF)
```

$$\begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix}$$

For small fields, for example $GF(16) = \mathbb{Z}_2[x]/(x^4 + x + 1)$, multiplication and division can be implemented by a table of powers of a primitive element:

| $i$ | $x^i$ | $i$ | $x^i$ | $i$ | $x^i$ | $i$ | $x^i$ | $i$ | $x^i$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | $x$ | 4 | $x+1$ | 7 | $x^3+x+1$ | 10 | $x^2+x+1$ | 13 | $x^3+x^2+1$ |
| 2 | $x^2$ | 5 | $x^2+x$ | 8 | $x^2+1$ | 11 | $x^3+x^2+x$ | 14 | $x^3+1$ |
| 3 | $x^3$ | 6 | $x^3+x^2$ | 9 | $x^3+x$ | 12 | $x^3+x^2+x+1$ | 15 | 1 |

Since all multiplication is performed modulo $x^4 + x + 1$, it follows that $x^4 = x + 1$ in this field. This allows us to represent every power as a polynomial with degree 3 or less. Alternatively, every non-zero element of the field can be represented as a power of $x$. Then, for example:

$$(x^3 + x)(x^3 + x^2 + 1) = x^9 x^{13} = x^{22} = x^7 = x^3 + x + 1.$$

Since $x^{15} = 1$ all powers can be reduced modulo 15. Divisions, logarithms and powers can all be performed with reference to this table.

However, such a table is inefficient for large fields, and other methods, not easily performable by hand, must be used. This is where the use of a CAS is invaluable. Having experimented by hand with small fields, students can perform operations on fields large enough to be of cryptographic significance with the CAS.

Students are invited to explore two other cryptographic systems which use finite fields, the *Hill cipher* and the *Chor-Rivest knapsack cryptosystem.*

## The Hill cipher

The Hill cipher works by converting the plaintext into a matrix of elements in the field, and multiplying that matrix by a fixed "key matrix" $M$. Using the ring $\mathbb{Z}_{26}$ of integers modulo 26, with an encryption matrix

$$M = \begin{bmatrix} 14 & 19 & 16 \\ 24 & 7 & 3 \\ 21 & 14 & 0 \end{bmatrix}$$

then encrypting the plaintext `THISISMYTEXT` is done by first placing the plaintext column by column into a matrix, and replacing each letter by its value modulo 26 (so `A` $= 0$, `B` $= 1$, up to `Z` $= 25$). Then Hill encryption can be done by multiplying this matrix by the key matrix. In this case:

$$\begin{bmatrix} \texttt{T} & \texttt{S} & \texttt{M} & \texttt{E} \\ \texttt{H} & \texttt{I} & \texttt{Y} & \texttt{X} \\ \texttt{I} & \texttt{S} & \texttt{T} & \texttt{T} \end{bmatrix} \Longrightarrow \begin{bmatrix} 19 & 18 & 12 & 4 \\ 7 & 8 & 24 & 23 \\ 8 & 18 & 19 & 19 \end{bmatrix} \Longrightarrow \begin{bmatrix} 14 & 19 & 16 \\ 24 & 7 & 3 \\ 21 & 14 & 0 \end{bmatrix} \begin{bmatrix} 19 & 18 & 12 & 4 \\ 7 & 8 & 24 & 23 \\ 8 & 18 & 19 & 19 \end{bmatrix} = \begin{bmatrix} 16 & 12 & 3 & 13 \\ 23 & 0 & 8 & 6 \\ 20 & 2 & 0 & 11 \end{bmatrix}$$

The ciphertext is obtained by reading off this last matrix column by column, and converting back to letters, producing `QXUMACDIANGL`.

The conceptual simplicity of this cipher means that it is amenable to work with finite fields. For example, using the AES field $GF(2^8)$ as defined previously, instead of mapping letters to numbers in the range 0–25 as above, an ASCII character can be mapped onto the field by raising a primitive element to the ASCII value. So, for example, with the primitive element $x$, and the character '?', which has ASCII value 63, the corresponding field element is

$$x^{63} = x^7 + x^5 + 1.$$

Suppose the key matrix $M$ is

$$M = \begin{bmatrix} x^6 & x^{20} & x^{19} \\ x^{17} & x^{22} & x^{18} \\ x^{25} & x^{23} & x^{14} \end{bmatrix} = \begin{bmatrix} x^6 & x^7+x^5+x^4+x^2 & x^6+x^4+x^3+x \\ x^7+x^4+x^3 & x^7+x^6+x^5+x^3+x & x^5+x^3+x^2+1 \\ x+1 & x^7+x^6+x^3+1 & x^4+x+1 \end{bmatrix}$$

A plaintext, for example: `Two oranges?` can be encrypted as follows:

```
sage:  pl = "Two oranges?"
sage:  pn = map(lambda i:ord(i),pl) # Turn the plaintext into a list of numbers
sage:  pm = transpose(matrix(4,3,pn))
         # Turn the list into a matrix
sage:  P = map_threaded(lambda i: x^i,pm)
         # Turn every element of the matrix into an element of the field
sage:  M = matrix([[x^6, x^20, x^19],[x^17, x^22, x^18],[x^25, x^23, x^14]])
         # Make encryption matrix
sage:  C = M*P # This is the encryption step
sage:  cm = map_threaded(lambda i: ZZ(i.int_repr()),C)
         # Replace each element of $C$ with its integer value
sage:  transpose(cm).list()
```

$[179, 32, 164, 165, 89, 229, 74, 240, 108, 18, 190, 123]$

This last list is the ciphertext. It can be turned back into 8-bit ASCII characters if the display can handle them.

## The Chor-Rivest knapsack cryptosystem

Knapsack problems have long been used to create cryptosystems [9]. They all work by turning a hard problem into an easy one by some algebraic or number theoretical computation. In the Chor-Rivest system, the knapsack problem is, from a given modular sum, to determine which elements have produced the sum. This system hides everything within computations over a finite field.

Here is a slightly simplified version (the complete system includes a permutation, which we have omitted for ease of description):

1. Choose a finite field $\mathbb{Z}_p[x]/f(x)$ of order $q = p^h$, with $f(x)$ being the irreducible polynomial. Let $g(x)$ be a primitive element in the field.
2. For each $i \in \mathbb{Z}/p\mathbb{Z}$, determine the discrete logarithms $a_i = \log_{g(x)}(x + i)$.
3. Choose a random integer $d$ so that $0 \leq d \leq p^h - 2$.
4. Compute $c_i = (a_i + d) \bmod (p^h - 1)$ for all $0 \leq i \leq p - 1$.
5. Then your public key is $([c_0, c_1, \ldots, c_{p-1}], p, h)$ and your private key is $[f(x), g(x), d]$.

Messages are binary strings $m_i$ of length $p$ with exactly $h$ ones. Encryption is implemented by computing

$$C = \sum_{i=0}^{p-1} m_i c_i \bmod (p^h - 1).$$

Decryption requires the following steps:

1. Compute $r = (C - hd) \bmod (p^h - 1)$.
2. Compute $u(x) = g(x)^r \pmod{f(x)}$.
3. Compute $s(x) = u(x) + f(x)$.
4. Factor $s(x)$ into linear factors

$$s(x) = \prod_{j=1}^{h} (x + t_j)$$

   where each $t_j \in \mathbb{Z}_p$. The values of the $t_j$ in the factorization are the positions of the 1's in the message.

For further discussion of this cryptosystem, with proofs of its validity, see [8], chapter 8. Here is an example in Sage with small values:

```
sage:  p = 7
sage:  h = 4
sage:  F.<x> = GF(p^h,name='x')
sage:  f = F.modulus()
sage:  g = F.multiplicative_generator()
sage:  g.multiplicative_order()
2400
sage:  a = [discrete_log(x+i,g) for i in range(p)]
sage:  d = randint(0,p^h-2)
sage:  c = [ZZ(Mod(i+d,p^h-1)) for i in a]
```

Here's an encryption:

```
sage:  m = [0,1,1,1,0,0,1]
sage:  C = sum(x*y for x,y in zip(m,c))
```

and decryption:

```
sage:  r = ZZ(mod(C-h*d,p^h-1))
sage:  u = g^r
sage:  factor(u.polynomial()+f)
(x + 1)*(x + 2)*(x + 3)*(x + 6)
```

This is straightforward and natural, and Sage provides an elegant translation from the theoretical description to a practical example. In comparison, using Maple [4] requires a large amount of work writing necessary procedures first.

# 4 Pedagogical considerations

In this section we discus the educational theories behind our use of computer algebra systems. Our base philosophy is that of *constructivism*, which in education refers to a student-centred model of learning: students "construct" their own models of the subject, and each student moves through its various topics in whatever manner supports their construction. This has the

virtue that any given student will have the necessary background to comprehend the current topic. Conversely, *behaviourist* teaching assumes that knowledge exists outside of the student, and it is the teacher's job to transfer the knowledge to the student. The second model is of course the most popular, because it is far easier for the teacher. See Scheurman [10] for a discussion.

Behaviourist teaching is of course particularly difficult for mathematics; students must build conceptual "schema" of the concepts before they can be fully assimilated into the student's own body of knowledge. See for example, Skemp [11].

Although the uses of lectures, tutorials and computer laboratories are seemingly a fairly traditional behaviourist approach, our teaching of finite fields owes much to the constructivist model. *Understanding* any single topic may mean understanding all the theoretical underpinnings of it, but also an appreciation of the effects of a particular algorithm, and how a change in a value, or of the construction of a field, may affect the result of an algebraic operation. The use of a simple example, as provided by the Hill cipher, allows the students to gain insight into the topics, without the overhead of the complete mathematical theory of finite fields. The laboratories are carefully designed to focus not so much on the theory but on practice: what happens if you change the value of a variable—how is the result affected? We encourage the students to be subjective; this helps them construct their own models of the topics.

An example of this is our approach to discussing inversion. The theoretical background can be daunting to the student with a limited background in abstract algebra, with unfamiliar terminology, and many new equations, some of which can be confusingly similar to each other. But using an algorithmic and exploratory approach to explore and demonstrate this operation provides students with a greater insight into its definition, its properties, and its use. One insight is that inversion can be used to build an operation which mimics arithmetic division. By experimenting with different expressions in the computer labs, students can see for themselves how inversion actually works in a finite field.

Student satisfaction, as measured with formal questionnaires, informal discourse, classroom assessment tasks, and test results, has been very high. Many students enter the subject with reservations as to their ability to handle the mathematics involved. The combination of the mathematics with the use of a CAS enables the students to succeed at what initially appears to be a very difficult subject. Almost without exception students are initially concerned that the amount of mathematics—and the unfamiliarity of the topics—may hinder their progress. And they are delighted when they realize that they can make sense of the mathematics, and perform cryptographic computations. This delight is reflected strongly in their comments on the final evaluation questionnaires. These questionnaires have consistently returned scores which rank among the highest of any subject, and students have been effusive in their praise. The students' delight in mastery of the material means that the subject is always recommended to other students, and so has always been well-subscribed. We feel that this subject is in many ways an exemplar of the use of a CAS: we do not eschew rigour or explanation, but in realizing that with weak backgrounds we cannot expect our students to master the fundamentals in good time, but need the help of the CAS to perform some of the messier computations, and allow for experimentation, without getting lost and frustrated in pages of messy algebra.

# 5 Conclusions

Finite fields provide a vital building block for modern encryption systems (and also for error-correcting coding). It is vital then, that students of those disciplines obtain a clear and deep understanding of their construction, arithmetic properties and associated algorithms. However, from a teaching perspective, there is the trouble that many students, especially those from a computer science or engineering course, may have little or no exposure to modern abstract algebra. The use of a computer algebra system, such as have been demonstrated above, can be hugely helpful in enabling the students to experiment with finite fields and to gain a deep appreciation of them without wrestling with all the algebra by hand.

# References


[1] *Access Grid* at `http://www.accessgrid.org/`, retrieved on June 24, 2012

[2] Joan Daemen, Vincent Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002

[3] Tim Daly et al, "Axiom: The Scientific Computation System" at `http://axiom-developer.org/index.html`

[4] L. Hernández Encinas, J. Muñoz Masqué, and A. Queiruga Dios, "Maple Implementation of the Chor-Rivest Cryptosystem", *Computational Science—ICCS 2006*, ed V. N. Alexandrov, 2006, published as Lecture Notes in Computer Science vol. 3992, pp 438–455

[5] *Maxima, a Computer Algebra System*, Version 5.25.1 (2011) at `http://maxima.sourceforge.net/`

[6] Fabrizio Caruso, Jacopo D'Aurizio, and Alasdair McAndrew, "Efficient Finite Fields in the Maxima Computer Algebra System", *Arithmetic of Finite Fields*, 2nd International Workshop, published in Lecture Notes in Computer Science vol. 5130, pp 62–76

[7] Robert J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, Boston 1987

[8] Alfred Menezes, Paul van Oorschot and Scott Vanstone, "Handbook of Applied Cryptography", CRC Press, 5th printing, 2001

[9] Andrew Odlyzko, "The rise and fall of knapsack cryptosystems", *Cryptology and Computational Number Theory*, American Mathematical Society, 1990, pp 75–88

[10] Geoffrey Scheurman, "From Behaviorist to Constructivist Teaching", *Social Education*, Vol. 62, No. 1 pp 6–9, Jan 1998

[11] Richard Skemp, "The Psychology of Learning Mathematics", Lawrence Erlbaum, 1987

[12] William A. Stein et al. *Sage Mathematics Software (Version 4.7.1)*, The Sage Development Team, 2011, `http://www.sagemath.org`