# Teaching Cryptography with Open-Source Software

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

## ABSTRACT

Cryptography has become an important topic in undergraduate curricula in mathematics and computer science, not just for its intrinsic interest—"about the most fun you can have with mathematics"[7], but for its current standing as the basis for almost all computer security. From wireless networking to secure email to password protection, cryptographic methods are used to secure information, to protect users, and to protect data.

At Victoria University, cryptography has been taught as part of a mathematics and computer science degree for several years. The students all have had at least a year of tertiary mathematics, and some exposure to a computer algebra system (Maple). However, the cost of Maple, and the current licensing agreement, means that students are unable to experiment with the software away from the computer laboratories at the University. For this reason we have decided to investigate the use of open-source computer algebra systems Maxima and Axiom. Although not as full-featured and powerful as the commercial systems Maple and Mathematica, we show they are in fact admirably suited for a subject such as cryptography. In some ways Maxima and Axiom even surpass Maple and Mathematica. Student response to the introduction of these systems has been very positive.

## Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption; G.4 [**Mathematics of Computing**]: Mathematical Software; K.3.2 [**Computers and Education**]: Computer and Information Science Education

## General Terms

Experimentation, performance

## Keywords

cryptography, computer science education, discrete mathematics, open-source software, exploratory learning


*SIGCSE'08,* March 12–15, 2008, Portland, Oregon, USA.


## 1. INTRODUCTION

Cryptography can be taught from several different perspectives: the mathematics can be the main element, or cryptographic practice, or there can be a mixture of the two. However, most current courses [14] require the necessary mathematics to be mastered as part of the curriculum; courses which de-emphasize the mathematics [9, 17] seem more properly to be generic *security* courses, rather than formal *cryptography* courses.

There is no fully agreed syllabus for an undergraduate cryptography course, however the SIGCSE recommendations for computer science curricula [3] include the topics:

- Historical overview of cryptography
- Private-key cryptography and the key-exchange problem
- Public-key cryptography
- Digital signatures
- Security protocols
- Applications (zero-knowledge proofs, authentication, and so on)

In this article we describe the use of open-source software for the teaching of a formal cryptography subject at an Australian university. We use the term *cryptography* to mean the study of the design and use of cryptosystems, and *cryptanalysis* for the study of breaking cryptosystems.

## 2. OUR SUBJECT AND STUDENTS

At Victoria University, there are 12 teaching weeks in a semester, of which (in this subject), 10 are used for teaching, and 2 for revision and assessment. Table 1 shows the week-by-week breakdown of our topics and their relationship to the SIGCSE recommendations above.

Although there has been some criticism about the "just in time" approach to mathematics in cryptography courses [14]—teaching just as much mathematics as needed when it is needed—we find that in fact it works very well. This approach has also been used by some major textbooks [18].

Students enrolled in the subject are in the third year of their course, and all have done at least one year of tertiary mathematics, which in most cases is discrete mathematics and linear algebra. Although discussions with students indicates some nervousness as to the degree of mathematics in the subject, in fact almost all students rise to the challenge, and enjoy mastering the mathematics required, and learning the cryptographic techniques.

| Our topics | SIGCSE |
|---|---|
| Principles, terminology, basic protocols | History, protocols, applications |
| Classical number theory | Public-key |
| Classical ciphers (affine, Vigenère, Hill) | History |
| Public key cryptosystems (RSA, Rabin, El Gamal) | Public-key |
| Digital signatures | Digital signatures |
| Knapsack cryptosystems | History, public-key |
| Block ciphers and hash functions | Private-key, applications |
| Data Encryption Standard | History, Private-key |
| Finite fields | Private-key |
| Rijndael, the Advanced Encryption Standard | Private-key |

**Table 1: Correspondence between our topics and SIGCSE's topics**

# 3. COMPUTER ALGEBRA SYSTEMS

A *computer algebra system* (CAS) is software which can perform symbolic as well as numerical calculations. CAS's have been used for teaching since their inception some decades ago, and their is now a considerable literature on their use. More recently, there has been some investigation into the use of such systems for teaching cryptography [5], and their are several textbooks which incorporate them into their material [10, 19]. There are also are many online resources, for example [6, 11].

For use in a cryptography subject, a CAS may be required to perform any or all of the following tasks:

- Arithmetic with arbitrarily large integers,
- String and character manipulation, including determining ASCII codes,
- Number theory: modular arithmetic, primitive roots, prime testing and factorization, discrete logarithms,
- Linear algebra,
- Computations over finite fields, including matrix manipulation.

Fundamentally, our approach is very similar to that of May [12], where the CAS is used in a way which doesn't require expertise from the students. The worksheet exercises are very carefully scaffolded, starting from the very simple (copy in a command, press the enter key), to more complex exercises which require the students to create their own commands, using previous exercises as guides.

In the past we have used Maple, which supports all the above cryptography tasks, with the possible exception of finite fields—see section 4.4. But a major problem with Maple is the difficulty of access. Its cost and local license mean that not only are students unable to install it at home, but they are also unable to use it remotely. This has been the cause of much concern from the students, who have often asked for software which they can access away from the University.

In this article, we are concerned with the systems Maxima [2] and Axiom [1]. Both systems derive from commercial products: Maxima is based on a 1990's version of Macsyma, which was initially developed at MIT, and Axiom is a recently GPL-released version of software of the same name.

Both Maxima and Axiom support all the tasks listed above for a cryptography subject, or if not, it is easy to build them in. Both are provided with very powerful languages for programming mathematical algorithms.

## 3.1 Maxima

Maxima provides what might be considered "standard" CAS functionality [20], and so it is not hard to transfer to Maxima from another system. Under windows Maxima has the elegant "wxMaxima" interface of which an input-output pair is shown in figure 1. This interface also provides a complete menu system, and an interface to Maxima's help.

## 3.2 Axiom

Axiom is heavily based around the concept of a "domain", which is a sort of mathematical object in which all operations return values in the domain of their operands. So, for example, if $x$ and $y$ are matrices, then the product operation $x \times y$ will return another matrix (assuming the matrices are conformable for multiplication). But if $x$ and $y$ are members of a finite field, then $x \times y$ will return a member of that field. The domains concept was borrowed extensively for the CAS MuPAD, and has also been investigated for Maple.

Axiom has a primitive interface; under Windows at least there is only a console interface with no graphical typesetting of mathematical output, no colour differentiation of input and output, and only a single PDF file for help. (Under Linux, Axiom provides HyperDoc, which is a full-featured hypertext help browser). At the time of writing, building an interface for Windows, and porting the help browser to Windows, are both active projects in the Axiom developers community.

Axiom is strongly typed in the sense that every output belongs to a type: for example the integers, or the residue class modulo an integer, or a matrix over a finite field. There are many types, and types can be nested. Since we do not expect our students to become experts in Axiom, our worksheets are carefully designed so that students need not be concerned with types and conversion between different types. All the students are required to do is to copy, modify slightly, and run given commands. They may also be expected to turn commands into functions.

# 4. CAS USE IN THE SUBJECT

Students studying on-campus are provided with the software on a local drive, with instructions for setup (for windows, this simply requires running a single installer program for each system). For students studying the subject off-campus, a web page is provided which includes links to the software, and again, necessary instructions. Students have found no problem downloading and installing the software. Throughout the semester, worksheets are provided for both systems Maxima and Axiom, and the students decide for themselves which one they are most comfortable using.

## 4.1 Strings, characters and classical ciphers

Although classical ciphers have no place in modern cryptography—they are all far to simple to break—they are extremely useful for teaching some basic cryptographic principles, and for introducing some elementary techniques in

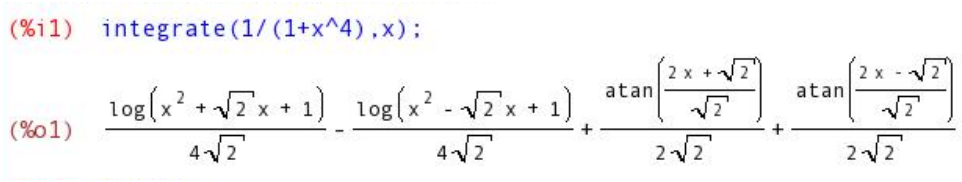


**Figure 1: wxMaxima in operation**

cryptanalysis. In our subject, students experiment with some simple classical ciphers: Caesar, affine, Vigenère, and investigate a *brute force attack* by trying every possible shift on a shift cipher, which is very easy to do as there are only 26 possible shifts. A *known plaintext attack* is also demonstrated using the Hill cipher, which is much harder to break using a ciphertext only attack (when only the ciphertexts are known), but it falls easily to an attack where both the ciphertext and some corresponding plaintext are known. For the classical cryptosystems, all plaintexts and ciphertexts are strings of capital letters with no spaces or punctuation.

As an example of the use of the systems, here are the Maple, Maxima and Axiom commands for implementing a Vigenère cipher, where `pl`, `kl` and `cl` represent the lists of numerical equivalents to the plaintext, keyword and ciphertext, and `pn`, `kn` are the lengths of the plaintext and keyword:

**Maple:**

```
cl:=[seq(pl[i]+kl[i-1 mod kn +1] mod 26 + 65,
  i=1..pn)];
```

**Maxima:**

```
cl:makelist(mod(pl[i]+kl[mod(i-1,kn)+1],26),
  i,1,pn);
```

**Axiom:**

```
cl:=[(pl.i+kl.((i-1) rem kn+1))::ZMOD 26
  for i in 1..pn]
```

Note that the differences are mainly syntactical. This makes it easy to create worksheets in either system, and to modify worksheets created using Maple to either Maxima or Axiom.

To demonstrate a known plaintext attack, the students are provided with a corresponding plaintext and ciphertext, for example

plaintext: `TIMETOGETSOMEWORKDONE`
ciphertext: `EWJTINUDMKSIMWEGVRDMP`

created with the Hill cipher using a $3 \times 3$ matrix. If $P$ is the plaintext matrix, and $M$ the encrypting matrix, then the ciphertext matrix $C$ is obtained with $C = MP$. This means that if $PP$ and $CC$ are $3 \times 3$ blocks of plaintext and corresponding ciphertext, and if $PP$ is invertible, then $M$ can be obtained with $M = CC \times PP^{-1}$.

Suppose that $P$ and $C$ are given. All that is needed is to extract any three columns of plaintext which make an invertible matrix; this requires the determinant to be relatively prime to 26. Suppose we take the first three columns of $P$ (in Axiom):

```
PP:=subMatrix(P,1,3,1,3)
gcd(determinant(PP),26)
```

This last returns a value of 1, which means we can obtain the inverse of $PP$ using the adjoint determinant method, and multiply by the corresponding three columns of $C$:

```
adj:=adjoint(PP)
PPinv:=adj.adjMat*recip(adj.detMat::ZMOD 26)
CC:=subMatrix(C,1,3,1,3)
CC*PPinv
```

The commands for applying this in Maxima are very similar.

## 4.2 Public Key Cryptosystems

The RSA and El Gamal systems all treat plaintexts and ciphertexts as very large integers. Many authors transform a text of capital letters to a single number by simply concatenating their values 1 to 26, (that is: A = 01, B = 02, Z = 26) so that for example, "CODEWORD" becomes 315040523151803. However, a method which better comports with the functionality of a CAS is to first replace each character by its ASCII value, and then treat the list of values as "digits" in base 256. In this scheme the ASCII values of "CODEWORD" are

$$[67, 79, 68, 69, 87, 79, 82, 68]$$

and the corresponding number is

$$\begin{aligned} &4850170388309561924 \\ &\quad = \quad 67(256^7) + 79(256^6) + \cdots + 82(256) + 68. \end{aligned}$$

Although the numbers produced by this method are slightly larger than numbers produced by concatenation, the advantage is that plaintexts are not arbitrarily restricted to certain characters. The students are provided with files which contain commands which convert to and from a text string and a large number. A strength of Axiom is a number of "Radix" commands which makes converting between numbers of arbitrary bases very simple.

The RSA cryptosystem requires choosing large primes — this is easily done as both systems provide commands for determining next and previous primes from a given integer, and commands for choosing arbitrarily large random integers. Modular powers and inverses are of course available.

The El Gamal system shows some of the power of Axiom, in that all operations are performed in the finite field of residues modulo a large prime $p$. Assuming that a plaintext has been converted to a large integer $pn$, the El Gamal system first requires a primitive root $g$ of $p$:

```
g:=primitiveElement()$PF(p)
```

The command `primitiveElement()` is a completely general command for finding a primitive element in any finite field, and here is used for the field generated by a prime number $p$, where `PF` is an abbreviation for `PrimeField`.

Having defined `g` as a member of this field, Axiom's domain structure ensures that any arithmetic operation including `g` will be returned as a member of this field; there is thus no need to invoke any modulo operations.

Then all the necessary arithmetic is just handled in a textbook style: first choosing a private key (randomly) and determining the public key:

```
a:=random(p)
A:=g^a
```

next encrypting the message:

```
k:=random(p)
K:=A^k
C:=[g^k,K*pn]
```

and finally decrypting:

```
K:=C.1^a
pn:=C.2/K
```

This is far simpler than is possible in almost any other CAS.

Digital signatures, leaving aside any discussion of hash functions for the moment, are easily treated using the number theoretical facility of either system. Students are encouraged to send each other messages with signatures, and verify the signatures.

## 4.3 Hash functions

Although it would be possible to build an implementation of any of the standard hash functions, of the SHA, RIPEM or other families, there is not much pedagogical value in doing so. However, the *idea* of a hash function can be introduced by easily implementing a simplified version of the modular hash function MASH [4, 13]. Our simplified MASH is this:

1. Pick two large primes $p$ and $q$, and form their product $n = pq$.
2. Express the text to be hashed as a large integer $N$, and express $N$ as a list $[n_i, i = 1 \ldots k]$ whose elements are the "digits" of $N$ in base $n$.
3. Define $h_0$ to be the previous prime to $n$, and for each $i$ between 1 and $k$ define $h_i = (h_{i-1} + n_i)^2 \bmod (n)$.
4. Then the hash value is defined to be $h_k$.

As an example, suppose take the text: "Now is the winter of our discontent, made glorious summer by this sun of York." Let $p$ be the next prime after $2^{32}$ and $q$ be the next prime after $3^{20}$, both of which are ten digit primes: $p = 4294967311$, $q = 3486784409$.

The text can be transformed into a large value $N$ using the method described at the beginning of section 4.2, which produces an integer 188 digits long. From this large value, a list of values $n_i$ for which $0 \leq n_i \leq n$ is produced. This is not hard to do in either system, and the students are supplied with such functions in separate files.

Given the list of "digits" in base n, the hash value can be calculated. In Maxima:

```
h:prev_prime(n);
for i:1 thru length(tmp) do h:mod(h+tmp[i])^2,n);
```

and in Axiom:

```
h:=prevPrime(n)
for i in 1..#tmp repeat
   h:=((h+tmp.i)^2)::IntegerMod(n)
```

In each case the final value of $h$ is the hash, which given the choices of $p$ and $q$ is 7020750617173787218. The exploration of hash functions finishes with a discussion of their use for digital signatures in signing a hash of the message, rather than the message itself. As with digital signatures, the students create a large text message, hash it, and sign the hash. Then they get another student to verify the signature.

## 4.4 Finite fields

Finite fields occupy a pivotal position in modern cryptography; they provide the theoretical underpinning of the Advanced Encryption Standard (AES); they are used for some public key cryptosystems, in particular those using elliptic curves [18], and the Chor-Rivest knapsack system. Thus students studying cryptography need to acquire a good "handle" on finite fields and the manipulation of their elements.

Axiom's implementation of finite fields is a great strength of this system. As seen above, in the discussion of the El Gamal cryptosystem, finite fields of residues modulo a prime $p$ is handled very neatly. But finite fields of any sort are equally well handled. Here, for example, is how arithmetic over the "Rijndael field" might be investigated:

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

First, define the field, and a generator of it:

```
F8:=FFP(PF(2),x^8+x^4+x^3+x+1)
a:=generator()$F8
```

This generator `a` can be used to enter elements of the field, for example:

```
p:=a^7+a^3
q:=a^6+a^4+a^2+1
```

Then arithmetic is straightforward:

```
p/q,  q^65,  r:=discreteLog(p,q)
```

It is then easy for students to experiment with finite fields; for example to generate tables of powers. Also, Axiom allows matrices over finite fields; for example:

```
M:=matrix([[1+a,a^6+a^2],[a^7,a^3+a^2+1]])
determinant(M)
inverse(M)
```

Given this, students can experiment with the "MixColumns" step of the AES.

Maxima has no inbuilt commands for arithmetic on finite fields, but the author has created a suite of commands allowing the necessary arithmetic, based on previous work developing similar routines for the commercial Macsyma [15]. First a field is defined, then it can be used:

```
gf_set(2,8,x^8+x^4+x^3+x+1);
p:x^7+x^3;
q:x^6+x^4+x^2+1;
```

And all necessary arithmetic is supported, for example:

```
gf_div(p,q);  gf_exp(q,65);  r:gf_log(q,p);
```

A slight disadvantage of this approach is that it is only possible to deal with one finite field at a time. However, in practice this is not usually a problem. The finite fields package also includes commands for matrices operations over finite fields.

In comparison to both Maxima and Axiom, note that Maple's implementation of finite fields is extremely clumsy. For example, to create the Rijndael field and multiply two elements in it requires the following:

```
GR:=GF(2,8,x^8+x^4+x^3+1);
p:=GR[ConvertIn](x^7+x^3);
q:=GR[ConvertIn](x^6+x^4+x^2+1);
GR[`*`](p,q);
```

Students in the past have complained vigorously about this implementation of finite fields.

## 4.5 DES and AES

It is not necessary to go through the rigmarole of setting up entire DES or AES encryption/decryption routines. Rather, to allow the students a feel for the workings of such cryptosystems, we introduce them to simplified versions. There are many such simplified versions available; we have chosen versions [8, 16] which can be easily implemented by hand, but also, once programmed in either system, gives the student insight into how such a system might be built up from its components.

Each cryptosystem can be implemented with binary lists; with the simplified AES there is the added interest of matrix manipulation over a finite field.

# 5. PEDAGOGY

Students are provided with the opportunity to experiment with both Maxima and Axiom to see which they like best; laboratory exercise sheets are provided for both systems through the semester. We have found that the students are roughly equally divided in their use; the convenience and pleasant interface of Maxima counterbalanced by the greater power of Axiom.

Students have commented favorably about their systems; of Maxima (with the wxMaxima interface):

"User friendly, and easy to use."
"Help function."

Of Axiom:

"Advanced and fast."
"Intuitive."
"Syntax is easy to remember."
"Easy to write functions."

Students also commented critically of the confusing nature of Maxima's functions, in particular unnamed functions, and on Axiom's primitive interface.

Because the students are now able to work away from the lab—at home if they like; some students decided to work mainly from home, and contact me only if there was a problem or difficulty. This worked very well, and helped to ease the pressure of a crowded laboratory.

# 6. CONCLUSIONS

The use of a CAS can make a vital difference to the teaching of mathematics or of a mathematically based discipline—such as cryptography. However, the current costs of commercial systems puts severe restrictions on their use. An alternative is to use one of the open-source systems, and as we have shown, this is quite feasible for a cryptography subject. There seems to be no decrease in student satisfaction, or with student learning, between using Maple (as we have done in the past), or using either Maxima or Axiom. Of the 32 students enrolled last semester, all of them, both in formal questionnaires and informal discussion, have enjoyed the freedom of the use of the software, and commented that it helped their learning. It would seem that students in fact prefer the use of software with an unrestricted license.

# 7. ACKNOWLEDGMENTS

It is my pleasure to acknowledge the help of my colleague, Anne Venables, for her close reading of an original draft of this paper, and for her many valuable suggestions for improvement.

# 8. REFERENCES

[1] MathAction and Axiom. `http://wiki.axiom-developer.org/FrontPage`.
[2] Maxima: a GPL CAS based on DOE Macsyma. `http://maxima.sourceforge.net/`.
[3] Overview of the CS Body of Knowledge,. `http://www.sigcse.org/cc2001/AL.html#AL-Cryptography`.
[4] Hash-functions using modular arithmetic. ISO Standard ISO/IEC 10118-4, 1998.
[5] A. Baliga and S. Boztas. Cryptography in the classroom using Maple. In *6th Asian Technology Conference in Mathematics*, `http://epatcm.any2any.us/EP/EP2001/ATCMP216/fullpaper.pdf`.
[6] R. Buchanan. Math 478—cryptography. `http://banach.millersville.edu/~bob/math478/`.
[7] N. Ferguson and B. Schneier. *Practical Cryptography*. John Wiley & Sons, 2003.
[8] J. Holte. Instructions for using simplified Rijndael (AES). `http://homepages.gac.edu/~holte/courses/mcs150-J01/documents/Rijndaelin%structions.html`.
[9] W.-J. Hsin. Teaching cryptography to undergraduate students in small liberal arts schools. In *InfoSecCD '05*, pages 38–42, 2005.
[10] R. E. Klima, N. P. Sigmon, and E. L. Stitzinger. *Applications of Abstract Algebra with Maple and Matlab*. Chapman & Hall/CRC, 2007.
[11] M. May. Maple worksheets for cryptography. `http://euler.slu.edu/courseware/CryptoSubmissionSet/Cryptography.html`.
[12] M. May. Designing courseware with maple. In *12th Annual International Conference on Technology in Collegiate Mathematics*, `http://archives.math.utk.edu/ICTCM/VOL12/P013/paper.pdf`, 1999.
[13] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
[14] P. D. Palma, C. Frank, S. Gladfelter, and J. Holden. Cryptography and computer security for undergraduates. In *ACM SIGCSE '04 Proceedings*, pages 94–95, 2004.
[15] K. T. Rowney and R. D. Silverman. Finite field manipulations in Macsyma. *ACM SIGSAM Bulletin*, 23(1):39–48, 1989.
[16] E. F. Schaefer. A simplified Data Encryption Standard algorithm. *Cryptologia*, 20(1):77–84, 1996.
[17] R. Schlesinger. A cryptography course for non-mathematicians. In *InfoSecCD '04*, pages 94–98, 2004.
[18] D. Stinson. *Cryptography: Theory and Practice*. Chapman & Hall/CRC, 3rd edition, 2005.
[19] W. Trappe and L. C. Washington. *Introduction to Cryptography with Coding Theory*. Prentice-Hall, 2nd edition, 2005.
[20] M. J. Wester, editor. *Computer Algebra Systems: A Practical Guide*. John Wiley & Sons, 1999.