Gerald's Column
by Gerald Fitton

In my article last month I introduced you to Modulo Arithmetic and showed you how it might be used to provide a simple encryption system. From the correspondence you have sent me by email and letter it would seem that some of my points need clarification. Up to now I have received no solutions to the puzzles I set—but it is early days yet.

On a totally different subject, Backward Compatibility, you have asked me what opinions I have—I shall cheat by describing a few examples and then ask you what you think!

Encryption

Last month I described how Modulo Arithmetic is used to provide "Nearly One Way Functions" of the type used in encryption algorithms. I included a simple example in which I used the function y = mod((m*x + s),256). In this formula x is the extended ASCII code of a printable character and y is the ASCII code of its encoded equivalent. On the Archive monthly disc and elsewhere I included spreadsheets in PipeDream, Fireworkz, Eureka, Excel and Lotus format so that you would be able to see and use the formulae for yourselves and hence gain familiarity with this 'Nearly one-way function' technique.

The value of m (the multiplier) has to be an odd number between 1 and 255 inclusive. The value of s (the shift) can be any integer between 0 and 255. Together this pair of parameters can be considered as the 'address' of the recipient even though they are used by the sender of the message. In current jargon, the values of m and s used by the sender are called the 'Public Key' of the intended recipient. The encryption function is made public knowledge as is the Public Key of all possible recipients. My example function has a 'Key Space' of about thirty three thousand unique Public Keys—not enough for practical use.

The recipient keeps secret their own pair of parameters called the 'Private Key'. The recipient uses the decoding function: x = mod((m*y + s),256) with their own Private Key values for m and s. This function decodes the encrypted message by converting the numerical y values into x values. The decoding function has an identical form to that of the encoding function so that the same computer program can be used for both encoding and decoding. The only difference between encoding and decoding is that the Public and Private values of m and s are different (but, I repeat, the same computer program is used).

Modulo Arithmetic

Algebra is much harder (but a more powerful mathematical tool) than is Arithmetic.

Spreadsheets don't do Algebra (it's too hard)—they do Arithmetic.

An example of a problem in Modulo Arithmetic is: "Encrypt the ASCII code for the upper case G (ASCII 71) using the function y = mod((m*x + s),256) with parameters m = 13 and s = 30." The answer involves you doing some arithmetic namely: Multiply 13 times 71, add 30 and then find the remainder when the result is divided by 256. The answer is 185.

Last month's spreadsheets demonstrate this arithmetic.

Modulo Algebra

An example of a problem in Modulo Algebra is: "Find the function which is the inverse of y = mod((m*x + s),256) where {m = 13, s = 30}. Find this inverse function in the form x = mod((m*y + s),256). The 'new' values of m and s define the inverse function."

For those of you unfamiliar with the concept of an 'inverse function' you can regard it as either a function which 'does it backwards' or as an 'undo' function. The answer, as demonstrated in last month's spreadsheet, is that the inverse of the function defined by the parameters {m = 13, s = 30} has the parameters {m = 197, s = 234}.

Let me try to ensure that you are absolutely clear what an 'inverse function' is and at the same time answer a few of the questions you have asked me.

If you choose any value of x between 0 and 255 and insert it in the encoding formulae y = mod((13*x + 30),256) you will find a y between 0 and 255. Every value of x produces a different value of y so that for every x there is a y and all possible values of y (between 0 and 255) are 'used up'. If you use an even value of m then the 256 different values of x produce only 128 different values of y. Every value of y which appears, appears twice. Half the possible values of y do not appear. The reason why even values of m can not be used for the Public Key is that the encoding of each character is not unique. The mathematical requirement is known as 'one to one correspondence' meaning that every x must 'map to' one and only one y and every y must correspond to one and only one x.

If the value of y returned by y = mod((13*x + 30),256) is entered into the inverse function, x = mod((197*y + 234),256), then the original value of x will be returned in every case.

The inverse function recovers the original value of x. It executes an 'undo' operation. It decodes the encoding of the original function.

Spreadsheets can not do Algebra of any kind. Spreadsheets can not find inverse functions except by the (usually time consuming) Trial and Error method.

However, if an arithmetic algorithm can be devised for finding the inverse parameters (the Private Key) corresponding to the parameters used during the encoding process (the Public Key), then this Arithmetical or Logical algorithm can be executed on a spreadsheet (or by a computer program). If such an algorithm can be devised then the encryption is insecure.

My Solution

Last month I said: "I have constructed a spreadsheet which finds the inverse of every multiplier." I asked you if you could produce either a spreadsheet or program which returns the Private Key when the Public Key is available. As an example question, "If the Public Key is known to be m = 13 and s = 30 then what is the corresponding Private Key?"

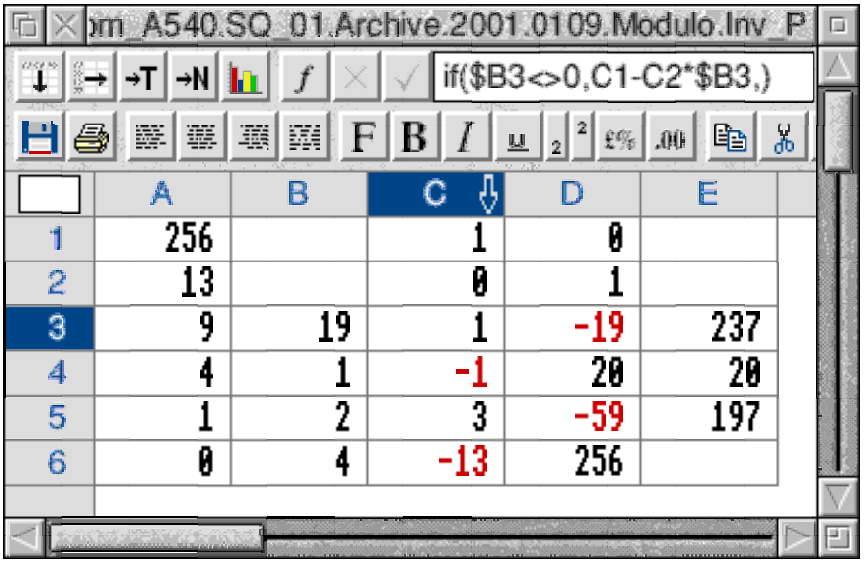
I have not yet received such a spreadsheet or program from anyone but I'm sure that I shall get one soon because the problem is one which is a relatively simple problem in Modulo Algebra and it must still be taught somewhere—but not for any A Level I know of.

My most general ‘Hint’ was meant to refer to Euclid’s algorithm for finding Highest Common Factors but, for some reason which eludes me I typed Euler instead of Euclid! I didn’t spot my ‘typo’ when I checked it. I did and do know that it was the Ancient Greek who invented the algorithm so I don’t know how it got past me!

The value of m for the Private Key is a function of the Public Key m; it is independent of the Public Key shift, s. The screenshot below shows the PipeDream version of my spreadsheet for finding the Private Key multiplier when given the Public key multiplier.

The cell A1 contains the modulus which we are using, 256. A2 contains the Public Key, I have chosen this to be 13. The values in the block C1D2 are always the same. Row three contains five formulae (in cells A3E3) which are replicated down as far as is necessary to produce the Private Key. The if(,,) function is used just to make everything look neater. Without the if(,,) function the formulae are:

A3=mod(A1/A2)
B3=int(A1/A2)
C3=C1−C2*B3
D3=D1−D2*B3
E3=mod(D3+A$1,A$1)



	A	B	C	D	E
1	256		1	0	
2	13		0	1	
3	9	19	1	-19	237
4	4	1	-1	20	20
5	1	2	3	-59	197
6	0	4	-13	256	

The Private Key is the last value to appear in column E. You will see that my spreadsheet returns the correct value, 197. I have included (on the Archive monthly disc) two files. The first is [Inv_CSV] which gives the value of m for the Private Key for every value of the Public Key m. This file will Load into almost any spreadsheet or wordprocessor so you’ll be able to lookup all the inverse multipliers. The second file is [Inv_P], the PipeDream spreadsheet which I have described above. You can change the modulus (the number in A1) and the Public Key multiplier (in A2) and the spreadsheet will return the Private Key multiplier. Do check for one to one correspondence (a 1 as the penultimate digit in column A—see below under Security) if you change the value of the modulus.

I am short of space so I shall defer my explanation of the mathematics (Modulo Algebra) which is behind the [Inv_P] spreadsheet until another day (unless Colin takes it up).

The value of the Private Key shift, s, is a function of both the Public Key multiplier, m, and the Public Key shift. When you have found the value of the Private Key m then the easiest way of finding the value of s for the Public Key is to put x = 0 into the encoding function. You'll remember that it is: y = mod((m*x + s),256). When x = 0 this reduces to y = s.

I'm sure that those of you familiar with ordinary algebra will be able to take it from there.

The answers to my other puzzles are:

The decoding multiplier corresponding to 127 is 127 and that for 99 is 75. You will be able to read these values (and many more) from the [Inv_CSV] or [Inv_P] file. The {m, s} Public Key of {127, 30} has the inverse {127, 30}. Yes! {127, 30} has an inverse which is identical to itself. The Public Key {99, 30} has the Private Key {75, 54}. As a by-the-way each Public/Private Key pair can be reversed. As an example the Public Key {75, 54} has the (inverse) Private Key {99, 30}.

Security

My simple example of an encryption technique is insecure because knowledgeable people will be able to find the inverse function with relative ease. You are now in this category because you can use my spreadsheet [Inv_P] and determine all the inverse multipliers. The Key Space can be increased to, say, a few million in order to deter a Trial and Error approach but, using the spreadsheet [Inv_P], you (and other knowledgeable people) will still be able to find the inverse multiplier. For example if the modulus is a million and the Public Key multiplier is 13 then the Private Key multiplier is 923077.

As a by-the-way, when you have chosen your modulus you can discover whether the Public Key multiplier does give the necessary unique one to one correspondence by checking that column A terminates with a 1 before the 0. Any other digit before the zero implies that the Public Key multiplier you have chosen is invalid because it does not give one to one correspondence.

I was able to use my simple example last month only because the majority of you were unlikely to know enough Modulo Algebra to be able to find the inverse function easily. Now you can all do that (using [Inv_P]) so the security has gone. Furthermore, I'm sure that many of you who couldn't do the Modulo Algebra would be able to set up a spreadsheet or program which finds all the inverse parameters by a Trial and Error method. My simple code can be cracked in less than an hour using a Trial and Error spreadsheet.

Serious encryption systems do not rely on the ineptitude of the code crackers. These days, if you want to make your name in the field of Public/Private Key encryption you have to show mathematically that nobody knows how to find the inverse function for your system using the most powerful computers in the world.

Nearly One Way

Of course there has to be an inverse function because true 'One-way functions' are useless. You might be able to encode a message but nobody, not even the intended recipient, could decode it! There has to be both a Public and a Private Key.

Modern serious encryption techniques use more sophisticated Modulo Arithmetic functions (such as raising numbers to powers) than the one I have used for my demonstration. This extra sophistication means that not even clever mathematicians with powerful computers can devise a practical general means of finding the inverse function. The famous Public Key/Private Key system falls into this category and I shall describe it on another occasion. For this famous system there is no method known to even the best mathematicians for finding the inverse function using powerful computers and having weeks, months, or even years of computer time available.

So, given a Public Key and an encoding function how do we find the corresponding Private Key? The answer is that we don't and can't! The Public and Private Keys are found as a pair. Indeed, a program is supplied which allows each user of the encryption technique to find their own Public/Private Key pair. The pairs of Keys are not issued by a central body.

The user then publishes their Public Key and they keep their Private Key secret.

Simplicity

I think that it was Mr P R Boxall who has asked how to keep (some of) his computer files secret. I suggest that for his personal system the level of security he requires could be met by something with the degree of sophistication of my example. So, if anyone has the energy to create a small (BASIC) program which will process a binary file using the algorithm of my example and Save an encoded version of the file then there might be a few people (including Mr P R Boxall) willing to try it.

I foresee the method of use being something like this. The encryption program appears as an icon on the icon bar; the secret file is dragged to it; the password, {m, s}, is entered; the processed version of the file is Saved. Different values of {m, s} are used for encoding and for decoding. Any volunteers?

Backward Compatibility

Now onto my second subject—backward compatibility.

Let me take as examples of different strategies two popular packages, Ovation Pro and PipeDream. Also I shall say a few words about a Microsoft package!

The most recent version of PipeDream is Version 4.50/23.

Version 2.64 of Ovation Pro is available for testing from:

http://www.davidpilling.net/ovationpro/upgrade/op264.zip.

PipeDream

Let's start with PipeDream. The most recent version of PipeDream has many features which you will not find in earlier versions. As an extreme example PipeDream 3 does not support charts (graphs) created from the data; PipeDream 4 and 4.5 do support charts.

There are many other minor examples such as Kerning and the addition of new functions such as monthname(date). This function returns the name of the month—for example monthname(1.1.1) returns “January”.

If you create a PipeDream file using the most recent version and send it to a friend who has only PipeDream 3 then your friend will be able to load the file you have created. If you have used none of the ‘newer’ features then they will be able to read the file exactly as you can. If you do use some of the ‘newer’ features then the file will still load but those features will be ignored. For example if you have included a chart and the function monthname(date) then they will not see the chart and the function will appear in the corresponding cell as text instead of returning the result of evaluating the function.

PipeDream is almost totally backward compatible.

Files created with the most recent version will Load into even the earliest version of PipeDream 3 running on a machine with RISC OS 2.

Features added in ‘newer’ versions will be ignored when the PipeDream file is Loaded into an earlier version which does not support those features.

Ovation Pro

Now to Ovation Pro. The most recent version of Ovation Pro has many features not found in earlier versions. The ‘big change’ was the upgrade to Version 2.60, the first version to be supplied on CD. Version 2.60 is the starting point for upgrades to later versions. For example the most recent version, 2.64, can be downloaded but used only if you have version 2.60 (or later).

If you create a file in version 2.60 and send it to a friend who has an earlier version then they will not be able to load the file you have created. There are many good reasons for this and (as a general statement from me) I do approve of the decision taken by David Pilling to dispense with Backward Compatibility between version 2.60 and earlier versions. Certainly it means that V 2.60 has many extra and improved features which earlier versions do not have. Catering for Backward Compatibility would have inhibited the inclusion of some of these new features.

What I am not as certain of is the decision to abandon backward compatibility of files created with V 2.64. One of the main improvements in 2.64 not present in earlier versions is support for variable sized spaces in text (a sort of Kerning). I have no doubt that this is a desirable new feature which many will welcome. However, files which you save from version 2.64 will not load into V 2.60.

The ‘downside’ of this Backward Incompatibility is that you can not use it if your intention is that the file should be read by people who have an earlier version. I can think of many cases where this might be a problem. Here’s just a couple. You might have several computers (a school, office or group of offices) with different versions of Ovation Pro. You might use Ovation Pro for publishing a magazine (Archive?) and want to publish the files on a CD. Which version of Ovation Pro do you use for the CD or website?

Microsoft Excel

Turning now to a Microsoft package which runs under Windows. I find it difficult to keep up with Excel. The most recent one I know the name of is Excel 2000 but I think there is an even newer version! Never mind, the version is not important; what is important is the strategy which they use to overcome backward incompatibility.

Excel 5 workbooks can not be loaded into earlier versions of Excel. For example files Saved in Excel 5 format will not load into Excel 3. However, from Excel 5 you can choose to Save in Excel 3 format. To do this you select the "Save As..." sub menu and choose 'Microsoft Excel 3.0 Worksheet (*.xls)'.

Of course, features added since V 3.0 will be excluded from the file which you Save—but at least your friend with Excel 3 will be able to load your file and look at it.

One of the 'problems' with the RISC OS platform at the moment is that there is no spreadsheet which can load or convert files which are in Excel 5 Workbook format. There are RISC OS spreadsheets which will load files which are in Excel 4 format including Eureka and Fireworkz Pro. This is the same 'sort of' backward compatibility problem that users of the Windows platform have with Saving in Excel 5 format.

The 'Best' Solution?

I would like to hear views about the desirability or otherwise of making the effort to retain backward compatibility. Is the PipeDream Strategy one which should be encouraged? Is the Ovation Pro strategy of dispensing with backward compatibility better? Is the Excel strategy (Save As... <an earlier version>) the best compromise?

The Ovation Pro strategy has the advantage that it encourages everyone to make use of the excellent new features of the later version. Sometimes a more recent version of PipeDream does contain features stated to be absent from a personal 'Wish List' of a critic! The Ovation Pro strategy would reduce the incidence of this unjust criticism.

The Excel 'Save As...'< an earlier Excel format> might be seen as a compromise between the total backward incompatibility of Ovation Pro and the (almost) total backward compatibility of the long established PipeDream.

I have not yet mentioned Hardware. Without wishing to invite too much abuse I have to draw to your attention that the average age of users of RISC OS machines increases as each year goes by. Many are still using RISC OS 3 and are using software which does everything they want. They are not tempted to upgrade to the most recent machines and certainly not tempted to fit RISC OS 4 in their ageing machines. Providing new features which will work on a (slow) RISC OS 3 machine is another sort of backward compatibility.

Finally

You can write to me at the address given in Paul's Fact File.

If you do need some help then please let me have an example file which you have created; it saves me so much work and reduces the chances of misunderstanding your problem. Return postage and a self addressed label will be appreciated.

I do give help with spreadsheets including Eureka, Schema, Fireworkz and PipeDream. I can even load files into Excel and Save them in a format which can be loaded into a RISC OS spreadsheet. A recent addition to my portfolio is Microsoft Works so I can help with these conversions too.