THOUGHTS ON CHEAPER NON-SECRET ENCRYPTION

M J Williamson, 10 August 1976

1. Introduction

This note is mainly a much-delayed response to some calculations made on the speed and cost of the equipment to implement non-secret encryption. Modifications can be made to the method I suggested in ref 1 to make it cheaper and faster. One of the reasons for the delay in writing this is that I find myself in an embarrassing position; having written ref 1 I have come to doubt the whole theory of non-secret encryption. The trouble is that I have no proof that the method of ref 1 is genuinely secure, in other words that it has a guaranteed work-factor involved in breaking it. This may be no more serious than the analogous fact that there is no proof that any of our ordinary encryption methods are genuinely secure but the fact does still worry me. So the whole of what follows should be prefaced by "if the method is sound then ...".

2. Method

The method I am putting forward is very similar to that of ref 1 but gets rid of the need for the Euclidean algorithm. It is thus: both link ends (and any interceptor) know a primitive element x of a finite field F and are working with elements of the field as polynomials modulo the primitive polynomial of x and coefficients modulo a prime p. Let there be pq elements in F, then x has period pq-1.

a. The two link ends generate random numbers a,b in the range 1 to p^q-1;

b. they calculate x^a and x^b respectively;

c. they calculate $(x^b)^a = (x^a)^b$;

e. Both link ends know this x^{ab} and it is difficult for the interceptor to recover it so either link end can use it as additive key to send a message.

3. Choice of field

As several people including Dr Benjamin and Nick Patterson have pointed out, it is possible to use any finite field eg GF(p) for the field F although I was originally thinking of using GF(2^q). For implementation on a computer it is probably best to use p = 2^8+I or 2^{16}+1, Fermat primes. The reason for these choices for p are two-fold, first to make it easy to work modulo p in an ordinary computer and second to use the fourier transform methods of para 4. It may seem difficult to work modulo 2^{16} +1 in a 16-bit word computer and it does require some fiddling (see ref 3) but it is cheaper than many other primes. The choice of q is more difficult; it depends upon the degree of security needed and I have done no calculations on this but I shall assume that the order of 100 would be about right. It would be nice to choose q so that (p^q-1)/(p-1) was prime and it is essential that it is not made up from many small factors. The numbers involved are too large for brute force methods of factorising and I do not know of any modification of Lucas' Mersenne prime test for this form of number. I shall just have to assume that someone who knows more number theory than myself can find suitable values for q and that it can be chosen to be of the order of 100.

4. Fourier Transforms

Almost all the work involved in the procedure is in the calculation of y^a in the field. Let $p^q \sim N$ therefore a is an N-bit number. Now y^a can be built up by a sequence of squarings and multiplications by y. For instance to calculate y^a for a = 1101001 (binary) we

start with a = 1
square to get a = 10
xy to get a = 11
square to get a = 110
square to get a = 1100
xy to get a = 1101
square to get a = 11010
square to get a = 110100
square to get a = 1101000
xy to get a = 1101001

So the work is broken down into a sequence of multiplying pairs of field elements (squaring is just multiplication). That is multiplying two polynomials modulo another polynomial. But multiplying polynomials is exactly the same as convolving their coefficients and convolution is a well-known use for the fast fourier transform. In fact there is a type of FFT precisely suited to the convolution of two modulo p sequences where, as here, p is a Fermat prime (see ref 3). The FFT only becomes economical when convolving fairly long sequences of numbers and it is only just becoming worthwhile for 100 numbers as here. The crude method would need 100 x 100 =10000 multiplys whereas two 256 point FFT's would be required in the present case, needing about 128 x 8 x 2 = 2048 multiplys but also a more complicated algorithm and greater overheads. After the two polynomials have been multiplied they need to be reduced modulo the primitive polynomial but provided this has been chosen to be fairly sparse the work involved is small compared with the multiplication.

5. Timing

For one pass of the algorithm each link end needs to make two y^a calculations and, assuming values of $p = 2^{16}+1$, q=100, each power calculations needs about 1600 squarings and a maximum of 1600 multiplications by y. Thus one pass needs 4 x 1600 x 2048 basic FFT butterflies and assuming a time of 25ms per butterfly the whole pass would take about 5 mins. The figure of 25ms is very much guesswork but is of the right order of magnitude for a minicomputer with a hardware multiply taking say 4ms. 5 mins is a long time but is does give 1600 bits of key - about 2.5 bits per second and does mean that if special hardware were built then telegraphic speeds could be achieved quite easily. The data storage for all these algorithms is quite trivial; all the workspace needed should not be more than about 2K words.

6. Transmissions

Normally three bits need to be transmitted to pass each bit of information; two preliminary bits and one message bit. This transmitting of bits can be reduced; indeed it can be reduced asymptoticly to one bit for each message bit. The idea is that the two link-ends should generate not just one but several, say K and L, random numbers. They can then transmit the corresponding x^a's and x^b's and use any of the KL pairs of field elements, so getting a number of key bits proportional to K.L for a number proportional to K+L bits transmitted. It seems peculiar to be producing more key bits than bits are transmitted and I feel that there should be a flaw in the security of the method. But I cannot find anything wrong with it and would be grateful if anyone else can.

7. Conclusions

I feel that non-secret encryption could be implemented with current technology but that work still needs to be done on the theory of the method. In open literature there is a certain amount of work published on the complexity of functions. That is assessing the minimum number of electronic components or computer operations needed to calculate a function. This clearly bears directly on the digital methods of NSE and I regret that I do not have at present the necessary background knowledge to understand ref 4 or probably ref 5 when it is published.

References

1. "Non Secret Encryption Using a finite field", 21 January 1974, M J Williamson.
2. "The Art of Computer Programming", Donald E Knuth, Vol 2 p 11.
3. "Theory and Application of Digital Signal Processing", Rabiner and Gold p 419.
4. "Computational Complexity over Finite Fields" V. Strassen, Siam J Comput., Vol 5 No 2 June 1976 p 324-331.
5. "Computational Complexity of Algebraic and Numeric Problems", American Elsevier, New York, to appear.