A Generalization of El Gamal's Public Key Cryptosystem

W. J. Jaburek, GABE Vienna

The general scheme

El Gamal's Public Key Cryptosystem (El Gamal 1985) can be generalized as follows (compare Shamir 1980) giving a public key exchange system:

The potential receiver of encrypted messages chooses a function f and publishes his public keys

s, k where k=f(s), f remains secret
G a set of functions commutative to f

The sender of message m chooses $g \in G$ and computes

$$k' = g(k) = g(f(s))$$

He uses k' as a key for a symmetric Cryptosystem such as DES or even simpler computes

$$m' = m \text{ xor } k'$$

and sends m', g(s) to the receiver. The latter computes

$$f(g(s)) = g(f(s)) = k'$$

and

$$m' \text{ xor } k' = m$$

and has received m in a safe way.

Associative Operations

(Modular) Multiplication per se does not offer a secure way of encryption. But multiplying an integer m n times by itself gives a very popular encrypting function, modular exponentiation, which has been used by El Gamal (El Gamal 1985) and by Rivest-Shamir-Adleman (Rivest 1978) as well.

The advantage modular exponentiation gives the friend against the foe is the possibility to compute f(x) in ld n steps (cf Knuth 1981, p 441) whereas the enemy nearly has to go through O(n) steps to get n by trial and error. This advantage is caused by the associativity of (modular) multiplication.

Associativity of the basic operation causes commutativity of the exponentiation, too.

$$a^{x^{y}} = a^{y^{x}}$$

J.J. Quisquater and J. Vandewalle (Eds.): Advances in Cryptology - EUROCRYPT '89, LNCS 434, pp. 23-28, 1990.

Generalisation of exponentiation

Multiplication cannot be the only possible associative operation in that respect. Perhaps there are other operations that are easier to compute and more secure in a cryptographic sense. That would imply that the resulting pseudo-exponentiation is more easily applied to real life cryptography without special hardware. Rueppel (Rueppel 1988) is following the track of considering function composition as a basis for pseudo-exponentiation. In this paper binary operations are considered.

The "pseudo-exponentiation" defined as follows - at least to the author - sounds very promising in the light of fast computation:

Let
x ... bitstring
f(x) = pa(pa(... pa(pa(x,x),x) ...),x)

the pseudoaddition as defined below applied n times to x, n integer

```
function pseudoaddition(x,y)
(x,y,acc1,acc2,carry: bitstrings of length l)
acc1:=x
acc2:=y
while acc2<>0
  carry:= acc1 and acc2
  acc1:= acc1 xor acc2
  (* Transformation of carry into acc2 *)
  acc2:= 0
  for i:=1 to l do
    if (Bit i in carry equal to 1)
    then acc2:=acc2 or tabelle[i]
  end_for
end_while
```

Tabelle[i] is a bitstring of length l, the i-th bit being zero and none or another few bits being one. For all bits in tabelle[i], i=1 .. l, the j-th bit (j=1 .. l) only once has value 1 because otherwise the or-function in the above pseudo-code must be replaced by a recursive call of pseudoaddition in order that pseudoaddition remains associative.

In general there exist l^l possible values for tabelle, as tabelle describes the mapping of l source bits into l bits, where each source bit may be used zero to l times (Variations with repetition).

The while-loop must terminate, because after the and-operation any bit of the carry has value one with probability p=0.25 and after the xor-operation any bit of acc1 has value one with p=0.5 with both probabilities clearly being smaller than one. The or-operation with a tabelle satisfying the above given conditions does not change the number of one-bits.

Example for tabelle with $l=4$

tabelle[1] = 0010
tabelle[2] = 0000
tabelle[3] = 0001
tabelle[4] = 1100

Note that tabelle[4] results in the urgently needed non-linearity!

Remark: Tabelle with values

0010
0100
1000
0001

describes binary addition modulo 15.

Lemma
Pseudoaddition is associative.

Proof
Can easily be verified by considering the similarity with addition.

Lemma
By repeating pseudoaddition a pseudo-exponentiation can be defined. Pseudo-exponentiation takes ld n (n being the number of times pseudoaddition is repeated in the trivial way of computation) pseudoadditions.

Proof
Just take the square-and-multiply-algorithm for exponentiation and substitute pseudoaddition for multiplication (cf Knuth 1981, p 441).

Example
Pseudoaddition using the above given tabelle, (2,0,1,12) when representing the bitstrings as decimal numbers, applied to 0011 or 3 gives values when repeated: 3, 13, 1, 14, 2, 12, 15, 3, ... a sequence that cannot be matched with the modular powers of 3 with any integer modulus.

Computational Complexity

Pseudoaddition takes n bit-operations in the for-loop times the number of times the while-loop is taken. The latter depends on the effect of carry-propagation. By applying the idea of a Carry Save Adder (Vgl Brickell 1982 and the literature given there) the while-loop ceases to exist (except in the case of normalizing the result of the whole operation). By using special hardware operating on all n bits at once, Pseudoaddition only takes O(1) step. Pseudoexponentiation therfore takes O(ld n) steps, which is faster than modular exponentiation by a factor of n, as the latter takes O(n . ld(n)) steps in good hardware.

Remark
By implementing tabelle in hardware n^n basic functions can be chosen, adding even more security against possible attacks. In order to prevent easy reading of the chip, it should not respond to requests with low exponents.

Security Assessment

Up to now the author did not do a concise exploration of the properties of the resulting set of binary operations. By using the following 3-bit-pseudoaddition some properties of the operations are discussed.

tabelle[1]= 010
tabelle[2]= 101
tabelle[3]= 000

results in the Cayley table for pseudoaddition

f	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111
001	001	010	011	101	101	110	111	001
010	010	011	101	110	110	111	001	010
011	011	101	110	111	111	001	010	011
100	100	101	110	111	000	001	010	011
101	101	110	111	001	001	010	011	101
110	110	111	001	010	010	011	101	110
111	111	001	010	011	011	101	110	111

Some properties can be deduced:

* There is an Identity Element: 000 and a sort of dual representation of it: 111 (Compare to addition with negative numbers represented as one's complement). The latter 11..111 could be called pseudo-identity.

* For each possible operand x there exists a value y so that pa(x,y) = 111...1111, the pseudo-identity. The **Pseudo-Inverse** of each bitstring can be calculated by applying the NOT-operation.
* Operations in general are not commutative as can be shown by using a second operation g described by tabelle 110, 000, 001 or the Cayley table:

g	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111
001	001	110	011	100	101	011	111	001
010	010	011	000	001	110	111	100	101
011	011	100	001	110	111	001	101	011
100	100	101	110	111	001	110	011	100
101	101	011	111	001	110	111	100	101
110	110	111	100	101	011	100	001	110
111	111	001	101	011	100	101	110	111

For example f(g(010,110),101) = f(100,101) = 001 and g(f(010,110),101) = g(001,101) = 011.

* Operations f with any tabelle[i] that includes two one-bits, applied ø times to one value x result in the value x itself. ø is Euler's Totient Function of the largest prime-number in the value range used. In the example given above ø = p-1 = 6 as 7 is the largest prime number representable with 3 bits.

Potential Weaknesses:

1. Pseudo-exponentiation could be represented as multiplication and therefore easily inverted, as one of the pseudoadditions is addition. Examples chosen at will show that pseudoexponentiation cannot be represented neither as addition nor as multiplications, except in the linear case of addition or permutations of addition's carry-tabelle.

2. By applying pseudoaddition repeatedly the identity-element may be produced. That is the same problem with modular exponentiation and therefore does not seem to be critical.

3. For some values in the example-f
   pseudoaddition(a,b) = pseudoaddition(a+1, b-1)

The author did not find a way how to exploit that potential weakness.

If other security threats to the system should become known it seems to be possible to expand the algorithm for pseudo-addition in a number of ways - e.g. by using a more complex transformation of carry into acc2 - without changing the run-time complexitiy of the algorithm.

Conclusion

The above given idea of creating pseudoadditions has two advantages over El Gamal's scheme:

* The basic function only takes O(1) step. El Gamal's takes O(n).

* No large primes have to be calculated for initializing the system.

The new operation pseudo-exponentiation can be applied to all cryptographic procedures using modular exponentiation as a one-way-function, e.g. the Pohlig-Hellmann Public Key Distribution System (Pohlig 1978) or the one-way encipherment of passwords in computer-systems. Thus a new set of operations worth studying for cryptographic purposes seems to emerge.

References

Brickell 1982
Brickell, E.F., A Fast Modular Multiplication Algorithm With Applications to Two Key Cryptography, in: Chaum et al(Eds), Advances in Cryptology - Proceedings of Crypto 82, Plenum 1983, 51-60

El Gamal 1985
El Gamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Trans. Inf. Theory, IT-31 (1985), 469-472

Knuth 1981
Knuth, D.E., The Art of Computer Programming[2], Vol 2: Semi-numerical Algorithms, Addison-Wesley 1981

Pohlig 1978
Pohlig - Hellmann, An Improved Algorithm for Computing Logarithms Over GF(p) and its Cryptographic Significance, IEEE Trans. Inf. Theory, IT-24 (1978), 106-110

Rivest 1978
Rivest - Shamir - Adleman, A Method of Obtaining Digital Signatures and Public Key Cryptosystems,
Communications of the ACM 1978, 120 - 126

Rueppel 1988
Rueppel, R., Key Agreements Based on Function Composition, Proc. EUROCRYPT'88, LNCS 330, Springer 1988, 3-10

Shamir 1980
Shamir, A., On the Power of Commutativity in Cryptography, in: Automata, languages, and programming, Proc ICALP 1980, Lecture Notes in Computer Science 85, Springer 1980, 582-595