Inverting the Pseudo Exponentiation

Fritz Bauspieß Hans-Joachim Knobloch
Peer Wichmann

Universität Karlsruhe
European Institute for System Security
Kaiserstr. 8
D-7500 Karlsruhe 1
FR Germany

Abstract

At Eurocrypt'89 W. J. Jaburek suggested an algorithm, which he called pseudo exponentiation, for use in generalized El-Gamal type public key cryptosystems. This pseudo exponentiation uses a modified form of binary addition in the place of multiplication in an ordinary exponentiation.

In this paper we show that the pseudo exponentiation on the set $GF(2)^k$ of bitstrings of length k has a considerable amount of mathematical structure. Using this structure we present an algorithm for inverting pseudo exponentiation that has a running time polynomial in k.

I.B. Damgard (Ed.): Advances in Cryptology - EUROCRYPT '90, LNCS 473, pp. 344-351, 1991.

1 Introduction

In their famous work [2] Diffie and Hellman presented a protocol for public key exchange based on the discrete log problem. Then, in 1985 El-Gamal presented a public key cryptosystem [3] based on this probloem. Some years later Beth found a corresponding zero-knowledge identification scheme [1].

These cryptosystems are based on an exponential structure over a set $\mathbf{G}$, usually a group. There a generalized exponentiation x^e is the application of the $(e-1)$–fold composition of an associative function $f : \mathbf{G} \times \mathbf{G} \to \mathbf{G}$ on $x \in \mathbf{G}$, namely

$$x^e = \underbrace{f(\ldots f}_{e-1 \text{ times}}\underbrace{(x,x),\ldots x)}_{e \text{ times}}$$

The cryptosystems mentioned above make use of the associativity of f:

$$f \text{ is associative} \Rightarrow \begin{cases} (x^a)^b = x^{ab} = (x^b)^a \\ f(x^a, x^b) = x^{a+b} \end{cases}$$

Furthermore if f is associative (and can be computed in polynomial time), x^e can be efficiently computed in polynomial time (with complexity parameter $\varepsilon = \log e$) using the square and multiply algorithm, whereas it is hoped that in general computing its inverse is much harder.

Now consider the order $< x >$ of any element $x \in \mathbf{G}$, defined as

$$< x >= \max\{n \in \mathbf{N} \ : \ \forall \nu : 1 < \nu \leq n \Rightarrow x^\nu \neq x\}$$

Because $\mathbf{G}$ is finite the repeated application of f on x will lead into a cycle of f-images. Therefore $< x >$ is either infinite or $< x >\leq |\mathbf{G}|$. From a cryptographical point of view, given $x, y \in \mathbf{G}$, as much as possible uncertainty about the integer e with $x^e = y$ is required. In terms of the order of elements this means that most of the elements of $\mathbf{G}$ have an order of roughly $|\mathbf{G}|$. For complexity theoretic analysis this allows for the simplification that $e = O(|\mathbf{G}|)$.

Some well-known f-functions are the multiplication in the multiplicative group of a finite field GF(q) and the addition of points on an elliptic curve ([5]).

2 Reviewing the Pseudo Exponentiation

To achieve efficient implementations of public key cryptosystems, in [4] Jaburek suggests a modified binary addition as as suitable function f, there called *pseudo addition.* It is defined over the vector space GF(2)^k of bitstrings of length k. The resulting exponential structure is called *pseudo exponentiation.* For the rest of the paper we shall denote pseudo addition with $\dagger$ and pseudo exponentiation with $\ddagger$.

Let A be a $k \times k$ matrix over GF(2), satisfying the conditions

$$a_{m,m} = 0 \tag{1}$$

$$a_{l,m} = 1 \ \Rightarrow \ \forall \lambda \neq l : a_{\lambda,m} = 0 \tag{2}$$

Pseudo addition of two vectors x and y is performed using Algorithm 1.

Algorithm 1 (Pseudo addition)

```
Variables are a, b, c
a ← x
b ← y
While b ≠ (0,...,0) Do
        c ← (a_1 · b_1, ..., a_k · b_k)
        a ← a + b
        b ← c · A
EndWhile
x † y ← a
```

The pseudo addition so defined is associative and commutative. Note that

$$x \dagger x = \sum_{\substack{n=0 \\ x_n=1}}^{k-1} a_n \tag{3}$$

where a_n denotes the n-th row vector of A, so that together with (2) in particular

$$(\forall n : x_n = 1 \Rightarrow a_n = 0) \;\Leftrightarrow\; x\ddagger^2 = 0 \tag{4}$$

Furthermore it is worth noting that the $k \times k$ matix

$$Z_k = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & & & \ddots & 1 \\ 1 & 0 & \cdots & \cdots & 0 \end{pmatrix} \tag{5}$$

describes the usual addition in $\mathbf{Z}_{2^k-1}$.

From a technical viewpoint, pseudo addition can be looked upon as a series of k full adders with widely scattered carry lines. Each carry output can be connected to one or more carry inputs or can be ignored. Each carry input can be connected to one carry output or to zero. These connections are given by the matrix A:

$$a_{l,m} = \begin{cases} 1 & \text{if the carry output of the } l\text{-th full adder is connected} \\ & \text{to the carry input of the } m\text{-th full adder} \\ 0 & \text{otherwise} \end{cases}$$

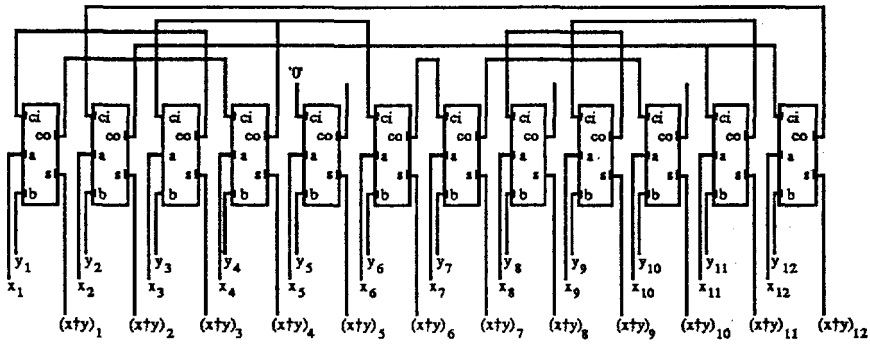

Figure 1: Full adder representation of the pseudo addition defined by Å

An example of this hardware representation for the matrix

$$\mathring{A} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

is given in fig. 1.

3 Identifying the Mathematical Structure

To see the mathematical properties of pseudo addition and exponentiation more clearly we introduce the notion of a *cluster*. A cluster $\mathcal{C}_i$ is a subset of the vector space $\mathrm{GF}(2)^k$ that is defined as follows:

$$\mathcal{C}_i = \{x \in \mathrm{GF}(2)^k : x\ddagger^{2^{i-1}} \neq 0\ ,\ x\ddagger^{2^i} = 0\} \qquad i = 1, 2, \ldots, m \tag{6}$$

$$\mathcal{C}_\infty = \{x \in \mathrm{GF}(2)^k : x\ddagger^{2^i} \neq 0\ ,\ i = 1, 2, \ldots\} \tag{7}$$

for some integer $0 \leq m \leq k$. We call i the order of cluster $\mathcal{C}_i$.

Equation (4) directly implies that any member x of $\mathcal{C}_1$ is nonzero at most in those coordinates x_n, for which the corresponding row vectors a_n of matrix A are zero. We will identify these coordinates of x and corresponding rows and columns of A with $\mathcal{C}_1$. In an analog way we will identify those row vectors a_n of A (and their corresponding columns of

A and vector coordinates x_n) with cluster $\mathcal{C}_i$, $i = 2, 3, \ldots, m$, who are themselves members of cluster $\mathcal{C}_{i-1}$.

For ease of notation we will assume from now on that the coordinates of x and hence the rows and columns of A are ordered in a way that all bits belonging to the same cluster are adjacent and the clusters are sorted according to their order, so that x_0 belongs to the cluster with highest order and x_k to the cluster with lowest order. Such an ordering can easily be obtained by applying a suitable permutation matrix P at the beginning and the inverse matrix P^{-1} at the end of the algorithm. In the hardware representation of the pseudo addition this is equivalent to permuting the order of the full adders.

According to this definition the matrix $A' = PA$ shows the following structure:

$$A' = \begin{pmatrix} A'_\infty & ? & \cdots & \cdots & ? \\ 0 & A'_m & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & A'_2 & ? \\ 0 & \cdots & \cdots & 0 & A'_1 \end{pmatrix} \tag{8}$$

where the $A'_\infty, A'_m \ldots, A'_1$ are square sub-matrices, not necessarily equal in size associated with clusters $\mathcal{C}_\infty, \mathcal{C}_m, \ldots, \mathcal{C}_1$ respectively. Note that this is the most general case and that either $A_1, \ldots, A_m$ or A_∞ need not exist for some of the possible matrices definig a pseudo addition.

Considering the definition of a cluster above it is obvious, that

$$A'_1 = A'_2 = \cdots = A'_m = 0. \tag{9}$$

The sub-matrix A'_∞ must contain at most one 1 per column (because of the definition of A, cf. equation (2)). On the other hand it must contain at least one 1 per row, else the specific row would belong either to one of the existing clusters $\mathcal{C}_m, \ldots, \mathcal{C}_1$ or to a new cluster $\mathcal{C}_{m+1}$. Therefore A'_∞ has exactly one 1 per row and column (i. e. A'_∞ is a permutation matrix), none of them on the main diagonal. So A'_∞ can be permuted to a form

$$A''_\infty = \begin{pmatrix} Z_j & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & Z_i \end{pmatrix} \tag{10}$$

with cyclic permutation matrices $Z_i, \ldots, Z_j$ as defined in equation (5). For simplicity we will assume that P is suitably chosen so that $A'_\infty = A''_\infty$

To obatin such an order for our example matrix Å, we must exchange the rows and

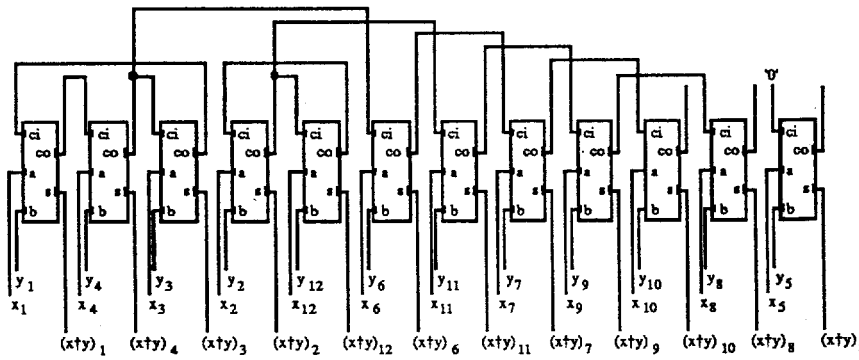

Figure 2: Sorted full adder representation of Å

columns 5 with 12, 8 with 11, 7 with 8 and 2 with 4 giving:

$$\mathring{A}' = \left(\begin{array}{ccc|cc||cc||cc||ccc} 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ \hline 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ \hline\hline 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0\\ \hline\hline 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&0\\ \hline 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0 \end{array}\right)$$

The example matrix thereby divides into four clusters $\mathcal{C}_\infty, \mathcal{C}_3, \mathcal{C}_2$ and $\mathcal{C}_1$. The sub-matrix $\mathring{A}'_\infty$ divides into two cyclic matrices Z_3 and Z_2. The corresponding sorted full adder chain is shown in fig. 2.

4 Computing the Pseudo Logarithm

We will call a solution e of the equation

$$y = x \ddagger^e \tag{11}$$

the pseudo logarithm of y (with respect to the base x).

The pseudo logarithm can be computed by first finding an exponent that yields a correct result for those bits belonging to cluster $\mathcal{C}_\infty$ using Chinese Remainder techniques. Then this exponent will then be successively adjusted so that it yields also the correct result for those bits belonging to clusters $\mathcal{C}_m, \ldots, \mathcal{C}_1$.

To explain this in more detail, we will denite by $x|_i$ the projection of x on cluster C_i, i.e. x with all coordinates belonging to clusters other than C_i set to 0.

The first goal is to solve the equation

$$y|_\infty = (x|_\infty \ddagger^{e_\infty})|_\infty \quad . \tag{12}$$

This can easily be done by first performing separate divisions in $\mathbf{Z}_{2^{i}-1}, \ldots, \mathbf{Z}_{2^{j}-1}$, yielding $e_\infty \bmod 2^i - 1, \ldots, e_\infty \bmod 2^j - 1$, and then combining the results via (the inverse of) the Chinese Remainder Theorem. If the moduli $2^i - 1, \ldots, 2^j - 1$ are not coprime, the Chinese Remainder Theorem can be applied to coprime factors of $z_\infty = \mathrm{lcm}(2^i - 1, \ldots, 2^j - 1)$. The solution e_∞ satisfies $e_\infty = e \bmod z_\infty$, i.e.

$$e = e_\infty + f z_\infty \tag{13}$$

for some $f \in \mathbf{Z}$.

Multiples of z_∞ may now be used as correction terms to adjust the other bits of $x \ddagger^{e_\infty}$, because a direct consequence of the definition of z_∞ is that $(x \ddagger^{z_\infty})|_\infty = \mathbf{0}$.

There are, however, two distinct i-bit representations of $0 \in \mathbf{Z}_{2^{i}-1}$, namely i zeros and i ones. So, in general, $(x \ddagger^{z_\infty})|_\infty$ will consist of several runs of zeros and ones, all representing 0 in one of $\mathbf{Z}_{2^{i}-1}, \ldots, \mathbf{Z}_{2^{j}-1}$. It can nevertheless been shown that this fact has no influence on the algorithm for computing pseudo logarithms.

Thus the remaining task of determining f in equation (13) can be completely done by evaluating the clusters of finite order. Algorithm 2 computes f bit by bit, starting with the least significant bit. Each cluster contributes exactly one bit to f, because definition (6) implies that $x \ddagger^{2^i z_\infty}|_m = \cdots = x \ddagger^{2^i z_\infty}|_{m-i+1} = 0$.

Algorithm 2 (Pseudo logarithm for clusters of finite order)

Variables are g, i, r, s, a, b
$g \leftarrow 0$
$i \leftarrow 0$
Repeat
  $r \leftarrow x \ddagger^{e_\infty} \dagger x \ddagger^{g z_\infty}$
  $s \leftarrow x \ddagger^{e_\infty} \dagger x \ddagger^{(2^i+g) z_\infty}$
  $a \leftarrow \max\{j \ : \ r|_j \neq y|_j\}$ (or 0 if the set is empty)
  $b \leftarrow \max\{j \ : \ s|_j \neq y|_j\}$ (or 0 if the set is empty)
  If $a > b$ Then
    $g \leftarrow 2^i + g$
  EndIf
  $i \leftarrow i + 1$
Until $a = b$
$f \leftarrow g$

If y was the result of a pseudo exponentiation, not an arbitrary vector, algorithm 2 will terminate with a = b = 0 and yield a correct result. In the case that C_∞ does not exist for a given A, set e to 0 and z_∞ to 1 in order to compute the pseudo logarithm.

5 Computational Complexity

The permutation matrix P can be found in at most $O(k^3)$ steps. The same holds for the permuation from A_∞ to A'_∞.

Division in Z_{2^i-1} can be done using the extended Euclidean algorithm. Such a division is needed at most $O(k)$ times. Finding coprime factors of z_∞, if necessary, can be done by applying the Euclidean algorithm on the moduli $2^i - 1, \ldots, 2^j - 1$ at most $O(k^2)$ times. The inverse of the Chinese Remainder Theorem is needed only once. Both the Euclidean algorithm and the inverse Chinese Remainder Theorem are well known polynomial time algorithms.

The subsequent computation of f consists of at most k steps, whereby the computational cost of each step is dominated by a pseudo exponentiation.

Thus the presented algorithm prohibits the cryptographic application of the pseudo exponentiation.

6 Remarks

The interested reader may find that in the original publication [4] the matrix A is described with the main diagonal from top right to bottom left.

An implementation of pseudo exponentiation and pseudo logarithm for values of $k \approx 200$ showed that computing the pseudo log is almost as fast as the pseudo exponentiation.

It is also feasible to solve equation (11) for x given y and e to compute 'pseudo roots'.

References

[1] T. Beth, *Efficient Zero-Knowledge Identification Scheme for Smart Cards*, Adv. in Cryptology - EUROCRYPT '88, Springer, Berlin 1988, pp. 77-84.

[2] W. Diffie, M. Hellman, *New Directions in Cryptography*, IEEE Trans. on Information Theory 22, 1976, pp. 644-654.

[3] T. El-Gamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Trans. on Information Theory 31, 1985, pp. 469-472.

[4] W. J. Jaburek, *A generalization of El-Gamal's public key cryptposystem*, Adv. in Cryptology - EUROCRYPT '89, to appear.

[5] N. Koblitz, *Elliptic Curve Cryptosystems*, Math. of Computation 48, 1985.