Architectures for exponentiation in $GF(2^n)$

T.Beth^{*}, B.M.Cook^{**}, D.Gollmann^{*}

Abstract.

We investigate different data structures in $GF(2^n)$ and their correspondence to silicon architectures to examine possible hardware implementations of the Diffie-Hellman key exchange system.

1.Introduction.

We want to analyse possible MOS implementations of the Diffie-Hellman key exchange system. This system is based on exponentiation in $GF(2^n)$. Exponentiation will be performed using a square and multiply algorithm. In this paper we will point out how different algebraic structures in the representation of $GF(2^n)$ give rise to different architectures for the MOS implementation of this algorithm. We will use examples to demonstrate the relative merits of the different architectures while giving only references for their mathematical foundations. (A survey of the mathematical background is given in [1]).

^{*} University of Karlsruhe, Faculty of Computer Science

^{**} University of Surrey, Departm. of Electronic and Electrical Engineering

2. Multiplication.

2.1. Polynomial basis multiplier

This architecture is based on the standard way of representing elements in $GF(2^n)$ by polynomials. Multiplication is performed by convolution and reduction modulo an irreducible polynomial $p(x)$ of degree n (see e.g. [2],[3]). Fig.1 gives a serial input architecture for a polynomial basis multiplier.

Fig.1. A polynomial basis multiplier for $GF(2^5)$ using $p(x)=x^5+x^2+1$.

2.2. Normal basis multiplier (Massey-Omura)

In a normal basis representation squaring is a single shift in a cyclic shift register. A multiplication algorithm can be found e.g. in [4]. Fig.2 gives a parallel input architecture for a normal basis multiplier ([4]), Fig.3 a serial input architecture ([5]).

Fig.2. A parallel input normal basis multiplier for $GF(2^5)$ using $p(x)=x^5+x^4+x^2+x+1$

Fig.3. A serial input normal basis multiplier for $GF(2^5)$ using $p(x)=x^5+x^4+x^2+x+1$

2.3. Dual basis multiplier.

Dual basis multipliers for multiplication with a constant were introduced in [6]. A general dual basis multiplier is discussed in [5], in this algorithm one of the factors and the result are represented in the dual basis and the other factor in the polynomial basis. Fig.4 and Fig.5 give architectures for a parallel input and a serial input dual basis multiplier.

Fig.4. A parallel input dual basis multiplier for $GF(2^5)$ using $p(x)=x^5+x^2+1$.

Fig.5. A serial input dual basis multiplier for $GF(2^5)$ using $p(x)=x^5+x^2+1$.

2.4. Evaluation and comparison.

The proposed polynomial basis multiplier performs a multiplication in n steps. The area required grows linearly in n . The architecture does not use straightforward shift registers. This disadvantage for standard hardware implementations had been a reason to look for new architectures. Normal basis and dual basis multipliers can be built from standard shift registers (Fig.2 and Fig.4). Both perform a multiplication in n steps. In the parallel input designs there is an additional delay to the XOR-tree. It may be noted that the serial input designs avoid the XOR-tree and use 'non-standard' reduction registers similar to the polynomial basis multiplier.

The size of a normal basis multiplier is mainly determined by the size of the PLA. In general it requires $O(n^2)$ AND-gates [4]. In MOS implementations additional problems occur due to the number of crossing wires in the PLA. The regular PLA of the dual basis multiplier is more convenient for MOS implementations. In the parallel input design area requirement however is not linear in n when we consider the wiring to the XOR-tree.

We have already pointed out that the serial input design is not too different from the polynomial basis multiplier. In a full custom MOS design it is always possible to define basic cells and bit slices tailored towards a given architecture. In such an environment the above argument against the polynomial basis multiplier is no longer valid. We conclude that the polynomial basis architecture is superior to the other architectures with regard to a full custom VLSI implementation.

3. Squaring.

The normal basis representation is obviously most appropriate for squaring. Squaring in the polynomial basis could be done by a multiplier. For a particular choice of the reduction polynomial $p(x)$ we can find a special architecture for the squarer. Take an irreducible trinomial $p(x)=x^n+x^k+1$, n odd, k even, $k < n/2$. Observe for some field element u

$$u^2 = \left(\sum_{i=0}^{n-1} u_i x^i \right)^2 = \Sigma_1 + \Sigma_2 + \Sigma_3$$

with

$$\Sigma_1 = \sum_{i=0}^{(n-1)/2} u_i x^{2i},$$

$$\Sigma_2 = \sum_{i=(n+1)/2}^{n-1} u_i x^{2i-n},$$

$$\Sigma_3 = \sum_{i=(n+1)/2}^{n-1-k/2} u_i x^{2i-n+k} + \sum_{i=n-k/2}^{n-1} u_i (x^{2i-2n+k} + x^{2i-2n+2k}).$$

Σ_1 corresponds to putting coefficients $u_0, \dots, u_{(n-1)/2}$ into the even positions of a register,

Σ_2 corresponds to putting the coefficients $u_{(n+1)/2}, \dots, u_{n-1}$ into the odd positions of this register,

Σ_3 corresponds to shifting the coefficients $u_{(n+1)/2}, \dots, u_{n-1}$ by k steps before adding them into the register (again in odd positions). The top $k/2$ coefficients are added into even positions starting from position 0 and starting from position k .

Fig.6 gives the architecture of such a squarer. It computes the square of an element u in $(n+k)/2$ shifts. With a minor modification of the design this can be achieved in $(n+k+1)/2$ shifts.

Fig.6. Squarer for $GF(2^5)$ using $p(x)=x^5+x^2+1$.

4. Square and multiply.

Squaring and multiplication can be performed in parallel if square and multiply starts at the least significant bit of the exponent ("right to left"). Thus the squarer of Fig.6 can be combined with either the polynomial basis multiplier or the dual basis multiplier. Fig.7 gives the architecture for the latter case. Note that we do not need an extra register to store intermediate results. An exponentiation is performed in $n(n+k/2)$ steps.

(n^2 steps with the modified squarer). If a serial input multiplier is used we arrive at a pipeline architecture where we can load the next inputs while shifting out the result.

Fig.7. Exponentiator for GF(2⁵) using a dual basis multiplier.

If exponentiation starts at the most significant bit of the exponent (‘left to right’) multiplication and squaring have to be performed serially. In the polynomial basis or dual basis architectures we could use the multiplier alternatively for multiplication and squaring. In this case we need an extra register for storing the base of the exponentiation. Let m denote the number of 1’s in the exponent. Both implementations require $(n+m)n$ steps for an exponentiation. In a normal basis architecture squaring can be done in a single additional step. Thus we need $n+mn$ steps for one exponentiation.

5. Conclusion.

Normal basis exponentiators need less steps than polynomial basis or dual basis exponentiators though the average number of steps is still of order $O(n^2)$. They are however not very well suited to VLSI implementations especially when area constraints are considered. Dual basis exponentiators show an advantage against polynomial basis exponentiators if TTL or standard cell technology is employed. If full custom MOS design is available the use of polynomial basis exponentiators seems to be the best choice. When both area and time requirements are demanding, a bit-slice architecture based on principles of Algebraic Algorithm Engineering shows an improved behaviour, particularly as unusual geometric configurations can be generated.

Literature.

- [1] T.Beth, On the Arithmetics of Galoisfields and the Like,
Proc. AAECC3, LNCS 229, Springer, 1986
- [2] F.J.MacWilliams, N.J.A.Sloane, The Theory of Error-Correcting Codes,
North-Holland, New York, 1977
- [3] B.A.Laws, C.K.Rushforth, A cellular-array multiplier for $GF(2^m)$,
IEEE Trans.Comput., vol.C-20, pp.1573-1578, 1971
- [4] C.C.Wang et al., VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$,
IEEE Trans.Comput., vol.C-34, pp.709-717,1985
- [5] W.Fumy, Über orthogonale Transformationen und fehlerkorrigierende Codes,
Ph.D.Thesis, Erlangen, 1986
- [6] E.R.Berlekamp, Bit-Serial Reed-Solomon Encoders,
IEEE Trans.Inf. Theory, vol. IT-28, pp.869-874, 1982