Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents*

Christof Paar
christof@ece.wpi.edu

Peter Fleischmann
P.Fleischmann@ukc.ac.uk

Pedro Soria-Rodriguez
sorrodp@alum.wpi.edu

ECE Department
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609, USA

Institute of Mathematics & Statistics
University of Kent
Canterbury, CT2 7NF, UK

IEEE Transactions on Computers, October 1999, vol. 48, no. 10, pp. 1025-1034

Abstract

This contribution describes a new class of arithmetic architectures for Galois fields $GF(2^k)$. The main applications of the architecture are public-key systems which are based on the discrete logarithm problem for elliptic curves. The architectures use a representation of the field $GF(2^k)$ as $GF((2^n)^m)$, where $k = n \cdot m$. The approach explores bit parallel arithmetic in the subfield $GF(2^n)$, and serial processing for the extension field arithmetic. This mixed parallel-serial (hybrid) approach can lead to fast implementations. As the core module, a hybrid multiplier is introduced and several

*This paper is an extension of [1]. The bit parallel squarer architectures have been completely revised.

optimizations are discussed. We provide two different approaches to squaring. We develop exact expressions for the complexity of parallel squarers in composite fields which can have a surprisingly low complexity.

The hybrid architectures are capable of exploring the time-space trade-off paradigm in a flexible manner. In particular, the number of clock cycles for one field multiplication, which is the atomic operation in most public-key schemes, can be reduced by a factor of n compared to other known realizations. The acceleration is achieved at the cost of an increased computational complexity. We describe a proof-of-concept implementation of an ASIC for multiplication and squaring in $GF((2^n)^m)$, m variable.

Keywords: Galois field, multiplication, squaring, VLSI, implementation, cryptography, elliptic curves.

1 Introduction

Finite fields play an important role in public-key cryptography. Many public-key algorithms are either based on arithmetic in prime fields or on extension fields of $GF(2)$, denoted by $GF(2^k)$. Examples of schemes which can be based on Galois fields of characteristic two are discrete logarithms schemes (see, e.g., [2]), elliptic curve schemes [3], and systems based on hyperelliptic curves [4]. This contribution focuses on computer architectures for the latter two families of public-key algorithms. Schemes based on the assumed difficulty of the discrete logarithm (DL) problem over (non-supersingular) elliptic curves should have extension degrees of $k \geq 160$. These long word lengths required for public-key algorithms lead to relatively low performance which is widely recognized as a major shortcoming in practical applications. The provision of fast hardware architectures for arithmetic in Galois fields $GF(2^k)$ is thus of great interest.

In the case of algorithms over $GF(2^k)$, addition can be realized with k bitwise exclusive OR operations. Addition is, thus, a fast and relatively inexpensive operation. The other field operation, multiplication, on the other hand is very costly in terms of gate count and delay. Multipliers can be classified into bit parallel and bit serial architectures. The former

ones compute a result in one clock cycle but have generally an area requirement of $\mathcal{O}(k^2)$. Bit serial multipliers compute a product in k clock cycles but have an area requirement of $\mathcal{O}(k)$. The two types of architectures are a typical example of the space-time trade-off paradigm. The main idea of this contribution is the introduction of a new class of Galois field arithmetic architectures which are faster than bit serial ones but with an area complexity which is considerably below the k^2 bound of bit parallel ones. It appears that avoiding the two extreme choices provided by bit parallel and bit serial architectures (very fast and large versus relatively slow and small) can lead to architectures with more optimized performance/cost characteristic for many applications. We will refer to the new arithmetic schemes as *hybrid architectures*. The principle of the architecture was first introduced in [5, Chapter 6]. The reference, however, only describes a hybrid multiplier and does not address optimizations, hybrid and parallel squaring, exponentiation, and applications to cryptography as it is done here. The main application of the new architecture are systems based on elliptic and hyperelliptic curves. Note that cryptosystems based on the discrete logarithm in $GF(2^k)$ might have weaknesses if k is composite, so that hybrid architectures are not applicable.

The outline of the remaining paper is as follows. Section 2 summarizes previous approaches of finite field architectures in general and of public-key architectures in particular. Section 3 introduces the general structure of the new multiplier architecture together with several optimizations. Section 4 describes two architecture options for hybrid squaring which enables the design of fast arithmetic units. Section 5 shows an architecture option which allows for a variable extension degree m. Section 6 shows the design and results of a proof-of-concept ASIC implementation of an arithmetic unit. Section 7 concludes with a summary of results and a description of areas of application.

2 Previous Work

Important theoretical results on the complexity of finite field multiplication are given in [6, 7]. The use of composite fields $GF((2^n)^m)$ for public-key schemes, more specifically for elliptic curve systems, is described in [8, 9, 10]. All three references deal with software imple-

mentations which explore table look-up for subfield arithmetic. Neither reference mentions the application to hardware architectures.

Computer architectures for finite field arithmetic have drawn considerable attention over the past decade. The majority of publications have concentrated so far on finite field architectures for relatively small fields, thus being mainly relevant for the implementation of channel codes. The focus in the research literature has been on architectures for the arithmetic operations multiplication (see, e.g., [11, 12, 13]), inversion (see, e.g., [14, 15, 16]), and exponentiation (see, e.g., [17, 18, 19]). Multiplication in $GF(2^k)$ is usually considered the crucial operation which determines the speed or throughput of a cryptosystem. Finite field architectures can be classified into bit serial (one output bit per clock cycle) and bit parallel ones (all output bits are computed within one clock cycle.) The vast majority of the proposed schemes are based on either of these two types. Architectures which are of hybrid-type (partially serial, and partially parallel), as proposed here, have only be mentioned in the dissertation [5, Chapter 6] and in our earlier work [1]. A remotely related architecture is the digit-serial multiplier proposed in [20].

Another classification of Galois field architectures is possible with respect to the basis representation of field elements. The most popular representations are standard (or polynomial or canonical), normal basis, and dual basis. Each basis representation has certain advantages; polynomial and dual basis representations are well suited for bit parallel multipliers, whereas normal basis representation allows for very efficient exponentiation. There have have been a few attempts to compare different types of arithmetic architectures for Galois fields. In [21, 22, 23] multipliers in different basis representations are compared. The focus is mainly on relatively small fields. Reference [17] compares normal and standard basis exponentiation architectures which are relevant for public-key algorithms.

There is a relatively small number of published work on Galois field architectures especially designed for cryptographic applications. Many of the bit serial architectures mentioned above, however, also extend to cryptographic applications. It should be noted that the $\mathcal{O}(k^2)$ complexity bound of most parallel multiplier architectures would result in unrealistically large arithmetic units for most public-key algorithms. So far, polynomial basis and normal

basis representation have been used for cryptographic applications.

There are three reported implementations which gain their security from the discrete logarithm in finite fields. Reference [24] contains a detailed description of an implementation of an exponentiation unit in the field $GF(2^{593})$, using an optimal normal basis representation of field elements. Reference [25] deals with various aspects of bit serial architectures in Galois fields for cryptographic applications. An implementation of an exponentiation unit in $GF(2^{333})$ using polynomial basis representation is described. In addition, there is the early description of an implementation of a cryptosystem over $GF(2^{127})$ [26].

There have also been publications about successful implementations of elliptic curve systems in hardware. Reference [27] describes the realization of a non super-singular elliptic curve system over $GF(2^{155})$. Field elements are represented with respect to an optimal normal basis.

3 Hybrid Multipliers

3.1 General Architecture

This subsection describes the general structure of a hybrid multiplier architecture for Galois fields in standard basis. The critical operation in terms of system performance of almost all public-key algorithms is multiplication. Both exponentiation (in schemes based on the DL in finite fields) and the group operation for elliptic curves (in schemes based on the DL over elliptic curves) rely on finite field multiplication as elementary function. The new class of architecture for arithmetic in $GF(2^k)$ will be based on the following two principles:

1. Representation of the field $GF(2^k)$ as $GF((2^n)^m)$, where $n\,m = k$;

2. Application of bit parallel architectures to arithmetic in the subfield $GF(2^n)$ and of a bit serial structure to arithmetic in the extension field $GF((2^n)^m)$. The goal is to obtain an acceleration by reducing the number of clock cycles required for a field multiplication.

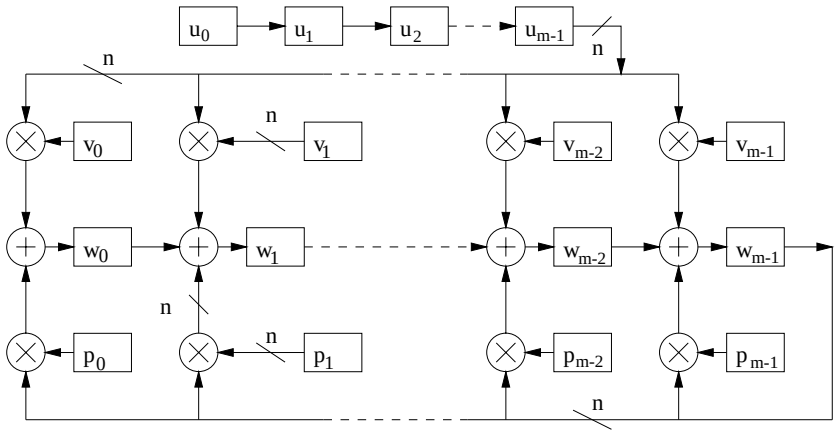

Figure 1: General structure of a hybrid multiplier in $GF((2^n)^m)$

We consider arithmetic in an extension field of $GF(2^n)$. The extension degree is denoted by m, so that the field can be denoted by $GF((2^n)^m)$. This field is isomorphic to $GF(2^n)[x]/(P(x))$, where $P(x)$ is an irreducible polynomial of degree m over $GF(2^n)$. In the following, a residue class will be identified with the polynomial of least degree in this class. For a standard basis multiplier we consider two field elements U, V:

$$\begin{aligned} U(x) &= u_{m-1}x^{m-1} + \cdots + u_1 x + u_0, \\ V(x) &= v_{m-1}x^{m-1} + \cdots + v_1 x + v_0, \end{aligned}$$

where $u_i, v_i \in GF(2^n)$. Field multiplication with the two elements is performed by the operation $W(x) = U(x) \times V(x) \bmod P(x)$, with W being the product element, and $P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i, p_i \in GF(2^n)$, is a monic irreducible polynomial. A possible hardware realization for this operation, polynomial multiplication modulo the field polynomial, is shown in Figure 1. At the kernel of the architecture is a linear feedback shift register (LFSR) of width n and length m. The registers of the LFSR hold the w_i coefficients. The coefficients p_i of the field polynomial are the feedback coefficients of the the LFSR.

If $n = 1$, the structure degenerates into the classical bit-serial *MSB-first* architecture for multiplication in the field $GF(2^m)$ (see, e.g., [28, 29]). In this case all lines are one-bit connections. The U operand is fed into the architectures in a bit serial manner, most significant bit first. The product coefficients w_i are available after m clock cycles, i.e.,

	binary ($n = 1$)	hybrid ($n > 1$)	hybrid ($n > 1$), $P(x)$ binary	hybrid ($n > 1$), $P(x)$ binary, comp. subfield
# AND	$2k$	$2nk$	nk	$3/4nk$
# XOR	k	$k(2n + 1 - 2/n)$	$k(n + 1 - 1/n)$	$k(3/4n + 3 - 3/n)$
# REG	$3k$	$3k$	$k(2 + 1/n)$	$k(2 + 1/n)$
# CLK	k	$k/n = m$	$k/n = m$	$k/n = m$

Table 1: Gate, register, and clock requirements for multipliers in $GF((2^n)^m)$

multiplication of m bit operands requires m clock cycles. All hardware implementations of public-key cryptosystems we are aware of are designed with $n = 1$, i.e., $m = k$. For the large m occurring in public-key algorithms, the resulting processing time can be considerable. The complexity of the classical binary architecture with $n = 1$ is given in the second column of Table 1 where we consider the number of $GF(2)$ multiplications (AND), additions (XOR), registers (in bits), and number of clock cycles for one field multiplication, respectively.

However, if the field $GF(2^k)$ needed in a given cryptographic application allows a composite field extension $k = n\,m$, $n > 1$, application of the same principal structure leads to the new architecture. In that case, all connections are n bit wide buses and all arithmetic is performed in the subfield $GF(2^n)$. Assuming bit parallel architectures for the subfield multiplication and addition in the LFSR, the result is now computed in m clock cycles. We name this architecture a *hybrid* multiplier. The hybrid architecture reduces the number of clock cycles for one multiplication by a factor of $n = k/m$.

One attractive feature of the hybrid architecture is that it is still highly regular and modular which are very desirable features for VLSI realizations [30]. The multiplier can be built from m identical modules to which we will refer as "slices". Each slice consists of two subfield multipliers, one subfield adder, and three n-bit registers. The only global communication required is an n-bit feedback path which is common to all slices. The architecture allows also full flexibility with respect to the field polynomial $P(x)$. Any monic m degree polynomial over $GF(2^n)$ can be loaded into the architecture. The field polynomial can be changed

during operation after each multiplication if desired. The complexity of the general hybrid architecture is given in the third column of Table 1, where a bit parallel subfield multiplier with a complexity of n^2 AND gates and $n^2 - 1$ XOR gates [11] is assumed. It can be seen that the number of logic gates increases by roughly a factor of $2n$ compared to the traditional approach, whereas the number of registers is the same. The major advantage of the hybrid architecture is that the number clock cycles for one multiplication is reduced by a factor of n. The hybrid multiplier explores thus the time-space trade-off paradigm, where the degree of the trade-off (performance versus complexity) is determined by the field decomposition $n \cdot m$. The following section describes two optimizations of the general architecture which result in considerably reduced gate counts.

3.2 Optimizations

3.2.1 Binary Field Polynomials

In many public-key algorithms, in particular for elliptic curves schemes, the extension degree m can be chosen such that $\gcd(n, m) = 1$. In this case a field polynomial $P(x)$ which is irreducible over $GF(2)$ is also irreducible over $GF(2^n)$ [31]. In particular, we can now choose a $P(x)$ with coefficients from $GF(2)$. Field polynomials with binary coefficients result in a hybrid multiplier with drastically improved complexity. A block diagram of the improved multiplier is shown in Figure 2.

In slice i, the signal from the feedback path is either passed through (coefficient $p_i = 1$) or not processed (coefficient $p_i = 0$). Hence, in each slice, the general multiplier with the polynomial coefficient p_i is now replaced by a binary n-bit switch. A switch can be realized efficiently in digital hardware. In a simple realization the switch can be built by n AND gates, but more efficient realizations, e.g., through transmission gates [30], are also possible. If we neglect the switch complexity relatively to the other components, the over-all complexity is given in the fourth column of Table 1.

The architecture for binary field polynomials reduces the gate complexity roughly by half and the number of register bits by about one third, compared to the general hybrid multiplier

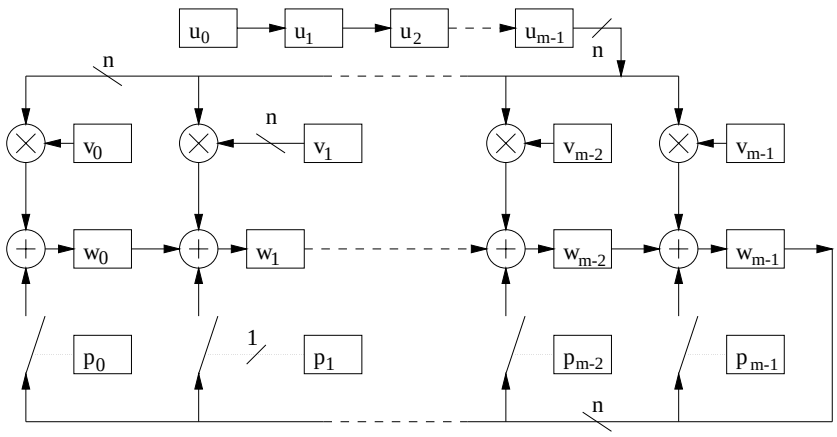

Figure 2: Hybrid multiplier in $GF((2^n)^m)$ with binary field polynomial

from Section 3.1. It should be noted that the architecture still allows flexibility with respect to the field polynomial. Any irreducible polynomial with coefficients from $GF(2)$ of degree m can be loaded into the architecture and serve as the field polynomial. Also, the high degree of modularity and regularity is preserved with the optimization.

A further optimization is possible if the field polynomial is fixed. In this case switches are replaced by a connection or no connection. The optimum field polynomials for this option are sparse ones, in particular trinomials and pentanomials. Note that an irreducible trinomial (a polynomial of the form $x^m + x^t + 1$) or pentanomial (a polynomial of the form $x^m + x^{t_3} + x^{t_2} + x^{t_1} + 1$) exists for all values of m, $2 \leq m \leq 10000$ [32].

3.2.2 Composite Subfield Multipliers

The gate complexities of the hybrid multiplier is now mainly determined by the bit parallel subfield multiplier. Applying a more efficient architecture to the subfield multiplication can thus be relevant for reducing the over-all system complexity.

So far we assumed a subfield architecture with a complexity of n^2 AND gates and $n^2 - 1$ XOR gates. The vast majority of bit parallel architectures has at least these gate counts. However, the complexity can be further reduced by applying the bit parallel architecture described in [33, 34]. The multipliers are based on a representation of the subfield $GF(2^n)$

through another field decomposition $GF((2^o)^p)$, where $n = o \cdot p$. In particular, for values of $n = 6, \ldots, 14$, n even, a representation of $GF(2^n) \cong GF((2^{n/2})^2)$ will lead to highly efficient architectures. This range of values for n appears to be attractive for practical applications such as elliptic curve cryptosystems. Elements of the subfield are now represented as polynomials with a maximum degree of one with coefficients from $GF(2^{n/2})$. The complexity of one subfield multiplication can be reduced to $\#\text{AND} = 3/4n^2$ and $\#\text{XOR} \approx 3/4n^2 + 2n - 3$ [34].

It should be noted that for the specific value of $n = 8$, it has been shown [23] that a multiplier based on the decomposition $GF((2^4)^2)$ requires in fact a considerably smaller number of gate equivalences in an ASIC implementation than architectures based on $GF(2^8)$. At the same time, the former architecture was found to be faster than the latter ones which is an attractive feature since the subfield multiplier is in the critical path of the architecture.

The structure of a hybrid multiplier with a decomposed subfield $GF((2^{n/2})^2)$ is still given by Figure 2 but the complexity is reduced to the expressions shown in the rightmost column of Table 1. The introduction of the subfield decomposition has thus reduced the gate complexity by roughly 25%. If the complexity after the two optimization is compared with the one of the bit serial architecture given in the second column it can be seen that the time performance (clock cycles) improves by a factor of n, whereas the gate complexities increase only by a factor of $3/8n$ (AND) and $3/4n$ (XOR), respectively.

3.3 Comparison to Normal Base Multipliers

Some implementations of public-key schemes over fields $GF(2^k)$ use a normal basis (NB) representation of field elements [24, 27]. The computational complexity for one multiplication depends heavily on the specific field polynomial [35]. A lower complexity bound, however, is given for irreducible polynomials which have a corresponding optimum normal basis [36]. Assuming an optimum normal basis for non-composite fields $GF(2^k)$, the complexity is given by $\#\text{AND} = k$, $\#\text{XOR} = 2k - 2$, and $\#\text{CLK} = k$. The NB multiplier requires thus n times as many clock cycles as the hybrid multiplier but has a lower gate count. Other advantages of the hybrid architectures are that the field polynomial can be changed and that the field

extension m can be alterable, as will be explained in Section 5. However, a major advantage of a NB is that squaring can be accomplished through a simple cyclic shift whereas squaring with the hybrid architectures is more costly as will be described in the following section.

4 Squaring and Exponentiation

Besides multiplication, another arithmetic operation of importance for the implementation of public-key algorithms is squaring. Systems based on the DL problem for non-supersingular elliptic curves require multiplications, squarings and one inversion, with respect to time-critical arithmetic, per group operation if affine coordinates are used. A popular method for inversion in hardware is based on Fermat's little theorem, according to which $A^{-1} = A^{2^k-2},\ \forall A \in GF(2^k),\ A \neq 0$. Although the extended Euclidean algorithm has a better theoretical performance, it requires more operands which in turn need more registers and control logic. It can thus be less attractive for hardware implementations. An attractive option for implementing Fermat's little theorem is the use of the standard square-and-multiply algorithm [37]. It should be noted that its derivation such as the k-ary or sliding window method require additional register and control logic which is a drawback for hardware implementations. If the inputs to the square-and-multiply algorithm are denoted by A and e and the output is the value A^e, each iteration stage of the algorithm performs one of the two operations:

1. Multiply result of previous iteration with A.
2. Square result of previous iteration.

The hybrid multiplier architecture from the previous section can be applied to the first operation. In this section, two architectures for squaring a result of a preceding multiplication (or squaring) will be developed which dovetail with the multiplier architecture.

4.1 Hybrid Squaring

The first architecture is based on the application of the general multiplier from Section 3 to squaring. We assume that the variable to be squared is contained in the registers $w_i, i = 0, 1, \ldots, m-1$, of a hybrid multiplier as a result of a preceding multiplication or squaring. In order to square this variable we must assure that its coefficients are available as both inputs of the multiplier. This is achieved by the following two operations:

Preparation of operands: Before start of squaring, load values from register w_i into input register v_i for $i = 0, 1, \ldots, m-1$. This can be performed simultaneously in all slices.

Squaring: Perform regular multiplication in m clock cycles. In clock cycle i, connect variable v_i as global input coefficient to all slices.

A corresponding hardware architecture is shown in Figure 3. It can be seen that the squaring functionality can be added to the hybrid multiplier with a modest amount of additional hardware. In every slice two switches must be added. Globally, a single control unit must be added to the system. The switches sp perform the initial parallel loading of the v_i registers. The switches ss allow for the generation of the global input coefficients during the multiplication cycles. The control logic assures that switch ss_{m-1-i} (and only switch ss_{m-1-i}) is closed during cycle i. The control logic can be realized as a counter with $\lceil \log_2 m \rceil$ bits and a $(\lceil \log_2 m \rceil)$–to–m decoder.

The squaring functionality can be added to all three multiplier options discussed in Section 3. As stated earlier, switches can be realized very efficiently in ASIC implementations. The computational complexity of the expanded architecture is thus essentially the same as the hybrid multiplier complexity. It should be noted that a single squaring requires the same time as a general multiplication, namely m clock cycles. This is a major drawback compared to normal basis architectures which realize squaring in a single clock cycle by means of a cyclic shift. The following section introduces a much faster but more costly approach to squaring.

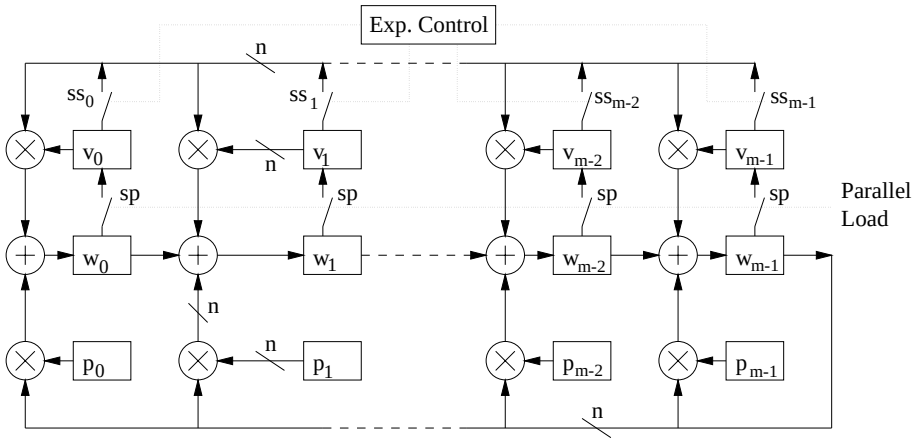

Figure 3: Structure of a hybrid squarer for $GF((2^n)^m)$

4.2 Parallel Squaring

Exponentiation with a k bit exponent requires $k-1$ squarings and on average not more than $(k-1)/2$ multiplications. Hence, a squaring architecture which requires fewer clock cycles than the one from the previous section can greatly improve the performance of a public-key system involving exponentiation. In the following we assume again that the input operand for the squaring is being held in the w registers of the hybrid multiplier. The architecture computes the result $T(x) = W^2(x)$ in one clock cycle, and puts the t_i coefficients in the w_i registers. For the development of a parallel squarer, i.e., squaring within one clock cycle, we note that [31]

$$T(x) = W^2(x) = \left(\sum_{i=0}^{m-1} w_i x^i\right)^2 = \sum_{i=0}^{m-1} w_i^2 x^{2i} \equiv \sum_{i=0}^{m-1} t_i x^i \pmod{P(x)}. \quad (1)$$

A parallel realization of this operation must provide the following two extensions to a hybrid multiplier:

Subfield squaring: In every slice, compute w_i^2.

Shift of coefficients and modulo reduction: Shift and summation of the squared coefficients w_i^2 yield the resulting coefficients t_i.

4.2.1 Complexities

In order to provide complexities for the two operations described above, we introduce the following theorem. Lemmas 1 and 2 will then allow us to develop gate counts for both operations under the assumption that the two field polynomials $P(x)$ and $Q(y)$ are trinomials. $Q(y)$ is an irreducible polynomial of degree n over $GF(2)$. $P(x)$ is an irreducible polynomial over $GF(2^n)$ of degree m, with binary coefficients, as described in Subsection 3.2.1.

Theorem 1 *Let*

$$A(z) = a'_{d-1}z^{2(d-1)} + \cdots + a'_1 z^2 + a'_0 = \sum_{i=0}^{d-1} a'_i z^{2i}$$

be a polynomial with coefficients from $GF(2^d)$, $d \geq 1$, and let $R(z)$ denote the irreducible trinomial over $GF(2)$

$$R(z) = z^d + z^t + 1$$

with $d \geq 2t$, and d and t not both even. $A(z)$ can be reduced modulo $R(z)$ with the following number of additions in $GF(2^d)$:

$$\#add = \begin{cases} (d-1)/2 & \text{if } d \text{ and } t \text{ both odd} \\ (d-t+1)/2 & \text{if } d = 2t \\ (d+t-1)/2 & \text{otherwise} \end{cases}$$

The proof is given in the appendix.

Please note that the conditions on $R(z)$ aren't too restrictive: If $d \leq 2t$ we can use the reciprocal polynomial to obtain $d \geq 2t$; if d and t are even, then $R(z)$ is a square and not irreducible.

From the above theorem we can derive the following lemma which gives the complexity for squaring an element in the subfield $GF(2^n)$ which is the first operation in a realization of (1).

Lemma 1 *Let*

$$w_i(y) = \sum_{j=0}^{n-1} w_{i,j}\, y^j$$

represent an element of $GF(2^n)$*, where* $w_{i,j} \in GF(2)$*. Let* $Q(y) = y^n + y^t + 1$*,* $n \geq 2t$*, be the irreducible polynomial of the field. The squaring* $w_i^2(y)$ *can be computed with not more than the following number of modulo 2 additions (*XOR*):*

$$\#\text{XOR} = \begin{cases} (n-1)/2 & \text{if } n \text{ and } t \text{ both odd} \\ (n-t+1)/2 & \text{if } n = 2t \\ (n+t-1)/2 & \text{otherwise} \end{cases}$$

Proof. Since the field characteristic is two and $w_i^2 = w_i$ in $GF(2)$, it holds that

$$w^2(y) = \sum_{j=0}^{n-1} w_j^2 y^{2j} = \sum_{j=0}^{n-1} w_j y^{2j}$$

from which the proof follows directly through Theorem 1. Note that an addition in $GF(2)$ is an XOR. $\diamond$

For the second operation required for a parallel realization of (1) the following lemma provides complexity expressions.

Lemma 2 *Let*

$$W(x) = \sum_{i=0}^{m-1} w_i x^i$$

represent an element of $GF((2^n)^m)$*, where* $w_i \in GF(2^n)$*. Let* $P(x) = x^m + x^\tau + 1$*, where* $m \geq 2\tau$*, be the irreducible polynomial of the field* $GF((2^n)^m)$*.* $W^2(x)$ *can then be computed with not more than* m *squarings in* $GF(2^n)$ *and the following number of modulo 2 additions (*XOR*):*

$$\#\text{XOR} = \begin{cases} n(m-1)/2 & \text{if } m \text{ and } \tau \text{ are both odd} \\ n(m-\tau+1)/2 & \text{if } m = 2\tau \\ n(m+\tau-1)/2 & \text{otherwise} \end{cases}$$

Proof. The number of subfield squarings follows from (1). The number of additions follows directly from Theorem 1 together with (1) if it is taken into consideration that each subfield addition can be realized with n XOR gates. $\diamond$

4.2.2 Example

As an example for a parallel squaring architecture, let's consider the field $GF((2^{15})^{11}) \cong GF(2^{165})$. This field order is highly interesting for elliptic curve cryptosystems such as suggested, e.g., in the upcoming IEEE P1363 standard. The field polynomials are $Q(y) = y^{15} + y + 1$ and $P(x) = x^{11} + x^2 + 1$. A parallel squaring architecture for the element $W \in GF((2^{15})^{11})$:

$$T(x) = W^2(x) = \sum_{i=0}^{10} w_i^2 x^{2i} \equiv \sum_{i=0}^{m-1} t_i x^i \pmod{P(x)}.$$

consists of the basic cells shown in Figure 4. Each cell consists of an n-bit register which contains the w_i variable and a parallel subfield squarer. For $n = 15$, $t = 1$, each parallel subfield squarer can be realized with $(n-1)/2 = 7$ XOR according to Lemma 1.

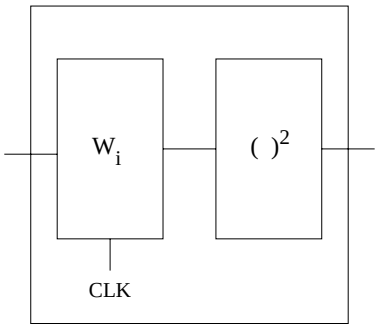

Figure 4: Cell for parallel subfield squaring

From Lemma 2 we see that in order to perform the reduction modulo $P(x)$, we need $(m + \tau - 1)/2 = (11 + 2 - 1)/2 = 6$ additions in $GF(2^{15})$ which can be realized with $6 \cdot 15 = 90$ XOR. The required interconnections for the parallel squarer in $GF((2^n)^{11})$, including the 6 adders, is shown in Figure 5. Each block represents a cell for parallel subfield squaring as shown in Figure 4. Note that each line denotes an n-bit wide bus. Although the interconnection is irregular we would like to stress that the gate complexity is relatively low. Parallel squaring in $GF((2^{15})^{11})$ can be realized with $11 \cdot 7 + 90 = 167$ XOR. This gate count can be neglected compared to the gate costs of a multiplier for this field.

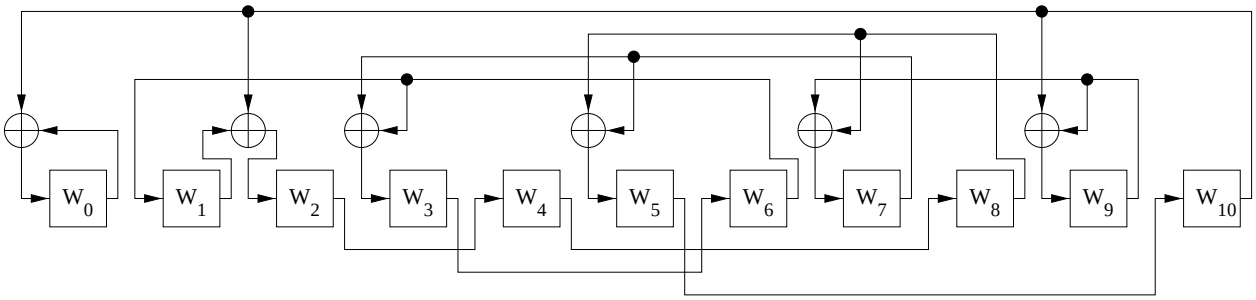

Figure 5: Parallel squarer in $GF((2^n)^{11})$

5 Variable Field Order

In some cryptographic applications it is desirable to allow for an alterable order of the underlying finite field. For instance, in elliptic curve systems, the order of the point group — which is central for determining the complexity of solving the discrete logarithm problem — is approximately equal to the order of the underlying finite field due to Hasse's Theorem [38]. An implementation with variable field order allows thus variable security levels depending on the application.

If we impose the restriction that the subfield order 2^n is fixed, architectures with variable extension degree m can be designed from the hybrid multiplier and the hybrid squarer. We can essentially apply the architecture in Figure 1 to a design that allows for variable m. With a modest amount of additional hardware, an exponentiation architecture with m slices can be programmed to use s slices, where $s \leq m$. A corresponding slice is shown in Figure 6. In order to perform arithmetic in the field $GF((2^n)^s)$ with the m-slice architecture, the connection between slice $s-1$ and slice s is open, and the output of register w_{s-1} is redirected to the feedback data bus. This can be done with one switch st (see Figure 6) which connects the output of slice $s-1$ (i in the figure) to either the next slice or to the feedback loop, but not both. Since the feedback happens now between slices $s-1$ and s, only s slices are used to perform a multiplication. Slices s and above are unused because there is no communication between them and the lower s slices.

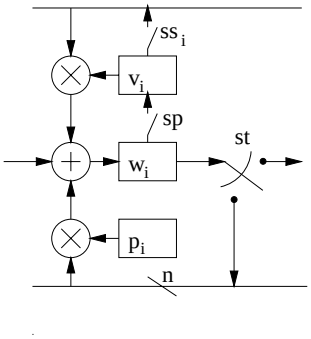

Figure 6: General slice structure for variable extension degree m

6 Proof-of-Concept Implementation

In order to gain further experience with the new class of finite field architectures, we performed a proof-of-concept hardware implementation. We implemented the most general multiplier architecture described in Section 3.1 and the hybrid squarer from Section 4.1, with variable m and $n = 8$ [39]. One slice of our implementation has thus the structure shown in Figure 6. We applied a full-custom design approach in CMOS technology using the MOSIS design tools. The fabrication facilities available for this research project enabled us to use 2μm technology. Obviously, this imposes a serious speed limitation on the prototype compared to current technologies. However, our goal was not to compete with commercial or semi-commercial implementations but rather to demonstrate the principle feasibility of the architectures, and to obtain reliable resource and timing estimations for this technology. Similarly we did not implement an entire public-key system. Our main interest was to study the underlying arithmetic architectures.

For the two subfield multipliers in each design we used the architecture of reference [11]. The implementation uses transmission gates (X-gates) to discern whether slice i is the final one in the chain or not. The transmission gates offer better switching characteristic than a pass transistor implementation [30], but still introduce a delay in passive mode. The coefficients of $V(x), W(x)$, and $P(x)$ are stored in 8-bit latches. This kind of memory requires less area than an SRAM implementation. While other types of memory elements can operate faster than a simple latch, they also consume more area.

The test ASICs that we implemented contains four slices, each of which requires 3040 transistors or 760 gate equivalences. We estimated that by using the optimized architecture in Section 3.2.1, the gate count would roughly be reduced by a factor of 1/2 and only 400 gates would be needed per slice. This estimate indicates that hybrid architectures with a subfield of $n = 8$ are feasible even for relatively large field orders. An example of a field order which is well suited for an elliptic curve cryptosystem is $k = 152 = 8 \cdot 19$. For this field order we obtain an estimate of about 8000 gate equivalences for a multiplier and (hybrid) squaring architecture which already includes I/O registers. These would even allow for a realization with reprogrammable logic (FPGA, EPLD).

Our implementation allows a clock rate of 3.5 Mhz. This clock rate was obtained by measuring the maximum path delay of the chips that were fabricated using a Tektronix VLSI tester. A multiplication in $GF(2^k)$ with $k = 152 = 8 \cdot 19$ takes thus 5.4μsec. Although we did not implement an elliptic curve system we can derive the following rough estimates: Using projective coordinates a point addition requires 14 multiplications or 75.6μsec, and a point doubling 12 multiplications or 64.8μsec [40]. A point multiplication with a 152-bit integer would then take about 11 msec on average, using the standard double-and-add algorithm. This estimate does not take any control overhead or additions into account, but also ignores possible improvement of the double-and-add algorithm (k-ary, sliding window). We would like to stress that the implementation is by no means speed optimized (but area optimized) and that we used relatively slow technology. We expect that the use of the parallel squaring architecture from Section 4.2, state-of-the-art technology (e.g., 0.35μm or smaller), and application of the faster subfield multiplier from Section 3.2.2 would lead to a very competitive performance of the arithmetic unit.

7 Conclusions and Applications

We developed new types of multipliers and squarers for Galois fields $GF(2^k)$. The multiplication and squaring architectures are designed such that exponentiation units can be built which are of central interest for public-key cryptosystems. The underlying idea is to

represent the field $GF(2^k)$ by $GF((2^n)^m)$, $k = n \cdot m$, and to apply bit parallel architectures to arithmetic in the subfield and a serial approach to the extension field arithmetic. The main feature of the new hybrid architectures is that they have the potential of being considerably faster than previously reported public-key architectures for finite field arithmetic. Hybrid multiplication requires only m clock cycles as opposed to k in traditional fully bit serial approaches. The principal feasibility of the approach was demonstrated in an ASIC implementation for the field $GF((2^8)^m)$, m variable. It appears that hybrid architectures could result in improved performance for several important public-key schemes.

The most attractive public-key schemes for the new architecture are those based on elliptic curves. Some of the reported implementation of elliptic curve systems over Galois fields $GF(2^k)$ already use a composite field extension k (e.g., $k = 155$ [27], or $k = 176$ [9, 10]), although not all of them explore subfield arithmetic. This situation is ideally suited for hybrid architectures. Also, since values of $k = 150 \ldots 250$ provide high security against currently known attacks, hybrid architectures for elliptic curves can be realized with a moderate gate complexity.

Another type of cryptosystem which can be used in conjunction with our architecture are those based on hyperelliptic curves [4]. Secure one-way functions can be built with $k < 100$. This range of field orders seems also very well suited for hybrid architectures.

Finally we would like to stress that the DL in finite fields might be insecure for composite Galois fields $GF((2^n)^m)$ [41]. Hence it is not recommended to apply the hybrid architecture to such schemes, which include, for instance, the classical Diffie-Hellman key exchange protocol.

A Proof for Theorem 1

Proof. We have $z^d = z^t + 1$, hence $z^{d+i} = z^{t+i} + z^i$ for $0 \leq i < d - t$ and $z^i + z^{i-d+2t} + z^{i-d+t}$ for $d - t \leq i < d$. From this we get

$$a^2 = \sum_{0 \leq 2i < d} a_i z^{2i} + \sum_{d \leq 2j < 2d-t} a_j (z^{t+2j-d} + z^{2j-d}) + \sum_{2d-t \leq 2j \leq 2d-2} a_j (z^{2j-d} + z^{2j-2d+2t} + z^{2j-2d+t}).$$

Hence for $0 \leq \ell < d$, the coefficient of z^ℓ is

$$a_{\frac{\ell}{2}} + a_{\frac{\ell-t+d}{2}} \text{ (if } \ell \in [t,d)) + a_{\frac{d+\ell}{2}} + a_{\frac{2d+\ell-2t}{2}} \text{ (if } \ell \in [t,2t-2]) + a_{\frac{2d+\ell-t}{2}} \text{ (if } \ell \in [0,t-2]).$$

Here $a_r := 0$ if r is not an integer.

Let $\ell \in [0, t-2]$ then this term is $a_{\frac{\ell}{2}} + a_{\frac{d+\ell}{2}} + a_{\frac{2d+\ell-t}{2}}$. If d and t are odd, then we need exactly one addition for each odd ℓ. If d is odd and t is even, then we need exactly one addition for each even ℓ. If d is even (and t is odd), then for each even ℓ one addition is needed. Hence in this sub-case, the total number of additions is $\frac{t-1}{2}$ if t is odd and $\frac{t}{2}$ if t is even.

Let $\ell = t-1$ then the term is $a_{\frac{\ell}{2}} + a_{\frac{d+\ell}{2}}$ and we need one addition if and only if d is even.

Let $\ell \in [t, \ldots, 2t-2]$ then the term is $a_{\frac{\ell}{2}} + a_{\frac{\ell-t+d}{2}} + a_{\frac{d+\ell}{2}} + a_{\frac{2d+\ell-2t}{2}}$. If d and t are odd, we do not need any addition if ℓ is odd and we need two additions for each even ℓ, which makes $t-1$ additions. If d is odd and t is even we need one addition for each even ℓ and if d is even, t is odd and $d > 2t$, we need two additions for even ℓ and none for odd ℓ. In all these cases we need a total of $t-1$ additions. If $d = 2t$, t odd, the last two summands in the above term coincide and no addition is needed.

Let $\ell \in [2t-1, d-1]$ the term is $a_{\frac{\ell}{2}} + a_{\frac{\ell-t+d}{2}} + a_{\frac{d+\ell}{2}}$. If d and t are odd, we do not need any addition for ℓ odd and one addition for ℓ even. If d is odd and t is even it is exactly the other way round. In both cases the total is $\frac{d-2t+1}{2}$. If d is even and t is odd, we need one addition only if ℓ is even and the total number is $\frac{d-2t}{2}$.

Summing all this together we get a total of $(\frac{d+t-1}{2})$ additions. Some of these additions are redundant if and only if a pair of indices appears twice. Let's see how this can happen. First we look at the case where d and t are both odd. In this case there are three types of additions occurring:
α: $\ell \in [0, t-2]$: the index pairs of the additions are
$\{(\frac{d+\ell}{2}, \frac{2d+\ell-t}{2}) \mid \ell \text{ odd}\}$ with difference $\delta := \frac{d-t}{2}$.
β: $\ell \in [t, 2t-2]$: the index pairs of the additions are

$\{(\frac{\ell}{2}, \frac{\ell-t+d}{2}), (\frac{\ell}{2}, \frac{2d+\ell-2t}{2}),\ (\frac{\ell-t+d}{2}, \frac{2d+\ell-2t}{2}) |\ \ell$ even$\}$ with difference $\delta := \frac{d-t}{2},\ d-t$ and $\frac{d-t}{2}$ respectively.

γ: $\ell \in [2t-1, d-1]$: $\{(\frac{\ell}{2}, \frac{\ell-t+d}{2})\}$ with $\delta = \frac{d-t}{2}$.

Possible intersections of type $\alpha \cap \beta$ imply either $d + k = \ell \leq 2t-2 \ \Rightarrow k \leq 2t-2-d < 0$, a contradiction; or $d + k = \ell - t + d \ \Rightarrow \ell = t + k$, which can happen. Type $\alpha \cap \gamma$ yields $d + k = \ell \leq d - 1 \ \Rightarrow k \leq -1$, a contradiction; $\beta \cap \beta$ yields $t \leq k = \ell - t + d \leq 2t - 2 \ \Rightarrow 2t - d \leq \ell \leq 3t - d - 2 \leq 3t - 2t - 2 = t - 2$, a contradiction; $\beta \cap \gamma$ yields $k - t + d = \ell \leq d - 1 \Rightarrow k \leq t - 1$, a contradiction. Hence exactly $\frac{t-1}{2}$ additions for even $\ell \in [t, 2t-2]$ are redundant, which reduces the total in this case from $\frac{d+t-2}{2}$ to $\frac{d-1}{2}$.

A similar analysis shows that no redundancies occur for $d \not\equiv t \ mod\ 2$. We give only the relevant index pairs:

d odd t even: α: $\ell \in [0, t-2]$: $\{(\frac{\ell}{2}, \frac{2d+\ell-t}{2}) \mid \ell$ even$\}$, $\delta = \frac{2d-t}{2}$

β: $\ell \in [t, 2t-2]$: $\{(\frac{\ell}{2}, \frac{2d+\ell-2t}{2}), (\frac{\ell-t+d}{2}, \frac{d+\ell}{2})\}$, $\delta = d-t, \frac{t}{2}$, respectively.

γ: $\ell \in [2t-1, d-1]$: $\{(\frac{\ell-t+d}{2}, \frac{d+\ell}{2})\},\ \delta = \frac{t}{2}$.

d even t odd: α: $\ell \in [0, t-2]$: $\{(\frac{\ell}{2}, \frac{d+\ell}{2})\}$, $\delta = \frac{d}{2}$.

α': $\ell = t-1$: $\{(\frac{t-1}{2}, \frac{d+t-1}{2})\}$, $\delta = \frac{d}{2}$.

β: $\ell \in [t, 2t-2]$: $\{(\frac{\ell}{2}, \frac{d+\ell}{2}), (\frac{\ell}{2}, \frac{2d+\ell-2t}{2}), (\frac{d+\ell}{2}, \frac{2d+\ell-2t}{2})\}$, $\delta = \frac{d}{2}, d-t, \frac{d-2t}{2}$, respectively.

γ: $\ell \in [2t-1, d-1]$: $\{(\frac{\ell}{2}, \frac{d+\ell}{2})\}$, $\delta = \frac{d}{2}$. $\diamond$

References

[1] C. Paar and P. S. Rodriguez, "Fast arithmetic architectures for public-key algorithms over Galois fields $GF((2^n)^m)$," in *Advances in Cryptography — EUROCRYPT '97* (W. Fumy, ed.), vol. LNCS 1233, pp. 363–378, Springer-Verlag, 1997.

[2] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[3] V. Miller, "Uses of elliptic curves in cryptography," in *Lecture Notes in Computer Science 218: Advances in Cryptology — CRYPTO '85* (H. Williams, ed.), vol. 218, pp. 417–426, Springer-Verlag, Berlin, 1986.

[4] N. Koblitz, "Hyperelliptic cryptosystems," *Journal of Cryptology*, vol. 1, no. 3, pp. 129–150, 1989.

[5] E. Mastrovito, *VLSI Architectures for Computation in Galois Fields*. PhD thesis, Linköping University, Dept. Electr. Eng., Linköping, Sweden, 1991.

[6] A. Schönhage and V. Strassen, "Schnelle Multiplikation großer Zahlen," *Computing*, vol. 7, pp. 281–292, 1971.

[7] D. Cantor and E. Kaltofen, "On fast multiplication of polynomials over arbitrary algebras," *Acta. Inform.*, vol. 28, pp. 693–701, 1991.

[8] G. Harper, A. Menezes, and S. Vanstone, "Public-key cryptosystems with very small key lengths," in *Advances in Cryptology — EUROCRYPT '92* (R. Rueppel, ed.), vol. LNCS 658, pp. 163–173, May 1992.

[9] E. DeWin, A. Bosselaers, S. Vandenberghe, P. D. Gersem, and J. Vandewalle, "A fast software implementation for arithmetic operations in $GF(2^n)$," in *Asiacrypt '96*, vol. LNCS 1233, pp. 65–76, Springer Verlag, 1996.

[10] J. Guajardo and C. Paar, "Efficient algorithms for elliptic curve cryptosystems," in *Advances in Cryptography — CRYPTO '97* (B. Kaliski, ed.), vol. LNCS 1294, pp. 342–356, Springer-Verlag, 1997.

[11] E. Mastrovito, "VLSI design for multiplication over finite fields $GF(2^m)$," in *Lecture Notes in Computer Science 357*, pp. 297–309, Springer-Verlag, Berlin, March 1989.

[12] M. Hasan, M. Wang, and V. Bhargava, "Modular construction of low complexity parallel multipliers for a class of finite fields $GF(2^m)$," *IEEE Transactions on Computers*, vol. 41, pp. 962–971, August 1992.

[13] S. Fenn, M. Benaissa, and D. Taylor, "$GF(2^m)$ multiplication and division over the dual base," *IEEE Transactions on Computers*, vol. 45, pp. 319–327, March 1996.

[14] G. Feng, "A VLSI architecture for fast inversion in $GF(2^m)$," *IEEE Transactions on Computers*, vol. C-38, p. 1989, Oct 1989.

[15] M. Morii and M. Kasahara, "Efficient construction of gate circuit for computing multiplicative inverses over $GF(2^m)$," *Transactions of the IEICE*, vol. E 72, pp. 37–42, January 1989.

[16] S. Fenn, M. Benaissa, and D. Taylor, "Finite field inversion over the dual base," *IEEE Transactions on VLSI Systems*, vol. 4, pp. 134–136, March 1996.

[17] W. Geiselmann and D. Gollmann, "VLSI design for exponentiation in $GF(2^n)$," in *Advances in Cryptology — AUSCRYPT '90* (J. Seberry and J. Pieprzyk, eds.), vol. LNCS 453, (Sydney, Australia), pp. 398–405, Springer-Verlag, January 1990.

[18] C. Wang and D. Pei, "A VLSI design for computing exponentiation in $GF(2^m)$ and its application to generate pseudorandom number sequences," *IEEE Transactions on Computers*, vol. C-39, pp. 258–262, February 1990.

[19] M. Hasan and V. Bhargava, "Low complexity architecure for exponentiation in $GF(2^m)$," *Electronics Letters*, vol. 28, pp. 1984–86, October 1992.

[20] L. Song and K. K. Parhi, "Low energy digit-serial/parallel finite field multipliers," *Journal of VLSI Signal Processing*, vol. 19, pp. 149–166, June 1998.

[21] I. Hsu, T. Truong, L. Deutsch, and I. Reed, "A comparison of VLSI architecture of finite field multipliers using dual-, normal-, or standard bases," *IEEE Transactions on Computers*, vol. 37, pp. 735–739, June 1988.

[22] Y. Jeong and W. Burleson, "Choosing VLSI algorithms for finite field arithmetic," in *IEEE Symposium on Circuits and Systems, ISCAS 92*, pp. 799–802, 1992.

[23] C. Paar and N. Lange, "A comparative VLSI synthesis of finite field multipliers," in *3rd International Symposium on Communication Theory and its Applications*, (Lake District, UK), July 10–14 1995.

[24] G. Agnew, R. Mullin, I. Onyschuk, and S. Vanstone, "An implemenation for a fast public-key cryptosystem," *Journal of Cryptography*, vol. 3, 1991.

[25] W. Gollmann, "Algorithmenentwurf in der Kryptographie." Habilitation, Fakultät für Informatik, Universität Karlsruhe, Germany, August 1990.

[26] K. Yiu and K. . Peterson, "A single-chip VLSI implemenation of the discrete exponential public-key distribution system," *IBM Systems Journal*, vol. 15, no. 1, pp. 102–116, 1982.

[27] G. Agnew, R. Mullin, and S. Vanstone, "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$," *IEEE Journal on Selected areas in Communications*, vol. 11, pp. 804–813, June 1993.

[28] S. Lin and D. Costello, *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[29] T. Beth and D. Gollmann, "Algorithm engineering for public key algorithms," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 4, pp. 458–466, 1989.

[30] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, A Systems Perspective*. Addison-Wesley Publishing Company, second ed., 1992.

[31] R. Lidl and H. Niederreiter, *Finite Fields*, vol. 20 of *Encyclopedia of Mathematics and its Applications*. Reading, Massachusetts: Addison-Wesley, 1983.

[32] G. Seroussi, "Table of low-weight binary irreducible polynomials," Tech. Rep. HPL–98–135, HP Labs, 1998.

[33] V. Afanasyev, "On the complexity of finite field arithmetic," in *5th Joint Soviet-Swedish Intern. Workshop on Information Theory*, (Moscow, USSR), pp. 9–12, January 1991.

[34] C. Paar, "A new architecture for a parallel finite field multiplier with low complexity based on composite fields," *IEEE Transactions on Computers*, vol. 45, pp. 856–861, July 1996.

[35] W. Geiselmann, *Algebraische Algorithmenentwicklung am Beispiel der Arithmetik in Endlichen Körpern*. PhD thesis, Universität Karlsruhe, Fakultät für Informatik, Institut für Algorithmen und Kognitive Systeme, Karlsruhe, Germany, 1993.

[36] R. Mullin, I. Onyszchuk, S. Vanstone, and R. Wilson, "Optimal normal bases in $GF(p^n)$," *Discrete Applied Mathematics, North Holland*, vol. 22, pp. 149–161, 1988/89.

[37] D. Knuth, *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*. Reading, Massachusetts: Addison-Wesley, 2nd ed., 1981.

[38] A. Menezes, *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[39] M. Lehky, M. Nappi, and P. Soria-Rodriguez, "Coprocessor board for cryptographic applications." Major Qualifying Project (Senior Thesis), May 1996. ECE Dept., Worcester Polytechnic Institute, Worcester, USA.

[40] M. Rosner, "Elliptic curve cryptosystems on reconfigurable hardware," Master's thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, USA, May 1999.

[41] L. Adleman and J. DeMarrais, "A subexponential algorithm for discrete logarithms over all finite fields," in *Advances in Cryptography — CRYPTO '93* (D. Stinson, ed.), vol. 773, pp. 147–158, Springer-Verlag, 1993.