Hardware Architectures proposed for Cryptosystems Based on Hyperelliptic Curves

Thomas Wollinger and Christof Paar

Department of Electrical Engineering and Information Sciences
Communication Security Group (COSY)
Ruhr-Universität Bochum, Germany
Email: {wollinger, cpaar}@crypto.ruhr-uni-bochum.de

ABSTRACT

Security issues play an important role in almost all modern communication and computer networks. The foundation of IT security are cryptographic systems, for example hyperelliptic curves cryptosystems (HECC). The advantage of HECC is that they allow encryption with shorter operands and at the same time, they provide the same level of security as other public-key cryptosystems, based on the integer factorization problem (e.g. RSA) or the discrete logarithm problem in finite fields or Elliptic Curves. Shorter operands appear promising for applications in constrained environments.

This work describes hardware architectures for HECC. We present efficient architectures to implement the necessary field operations and polynomial arithmetic in hardware, including architectures for the polynomial division and the calculation of the Extended Euclidean Algorithm in the polynomial ring. All architectures are speed and area optimized. To our knowledge, this is the first work that presents hardware architectures for the implementation of a HECC.

Keywords: cryptology, hardware architecture, hyperelliptic curves

1. INTRODUCTION

It is widely recognized that data security will play a central role in future IT systems. Today's information society relies on a large number of security sensitive applications, e.g. eCommerce, digital rights management, medical records, eBanking, etc. Cryptographic algorithms are the foundation of IT security. There are two major classes of algorithms in cryptography: private-key algorithms and public-key (PK) algorithms.

PK algorithms are necessary for services such as key establishment and digital signatures. Practical PK algorithms can be divided into three families: algorithms based on the integer factorization problem (for example RSA), algorithms based on the discrete logarithm problem (DLP) in finite fields (e.g. Diffie-Hellman key exchange), and algorithms based on the DLP in Elliptic Curves.

The DLP in a general group $\mathbb{G}$ can be stated as follows: given a group $\mathbb{G}$, a generator α of $\mathbb{G}$, and an element $\beta \in \mathbb{G}$, find the smallest integer k, such that $\alpha^k = \beta$.

Despite the differences between these three algorithm families, they all perform complex operations on large numbers, except for HECC. The numbers are typically 1024–2048 bits in length for RSA and DLP based systems and 160–256 bits in length for elliptic curve cryptosystems. Long operands are a major drawback in embedded applications such as cellular phones, PDAs, etc., where memory and processing power are constrained. Thus, HECC which require only 60 to 80-bit arithmetic, seem promising for many applications.

There are relatively few reported software implementations, e.g. [10] and no hardware implementations of HECC. Advantages of hardware implementations include: better performance and increased physical security (e.g., better protection of the private-key) when compared to software solutions.

In this paper we describe the development of optimized field and polynomial architectures that are time critical for a HECC. Once efficient architectures for these operations are found, the implementation of the group operations are straight forward. The remaining of this paper is organized as follows: Section 2 summarizes the mathematics, Section 3 presents the

optimized field and polynomial architectures, and we finish with give some conclusions.

2. MATHEMATICAL BACKGROUND

In this section we present an elementary introduction to some of the theory of hyperelliptic curves over finite fields of arbitrary characteristic, restricting attention to material that is relevant for the hardware implementations. Most proofs and details about HECC can be found in [2, 3].

2.1. HECC and the Jacobian

Let $\mathbb{F}$ be a finite field, and let $\overline{\mathbb{F}}$ be the algebraic closure of $\mathbb{F}$. A hyperelliptic curve C of genus g over $\mathbb{F}$ ($g \in \mathbb{N}\backslash\{0\}$) is the set of solutions $(u, v) \in \mathbb{F} \times \mathbb{F}$ to an equation $C : v^2 + h(u)v = f(u)$ in $\mathbb{F}[u, v]$. The polynomial $h(u) \in \mathbb{F}[u]$ is of degree at most g and $f(u) \in \mathbb{F}[u]$ is a monic polynomial of degree $2g + 1$.

A divisor $D = \sum m_i P_i$ is a finite formal sum of $\overline{\mathbb{F}}$-points. If $\mathbb{K}$ is an algebraic extension of $\mathbb{F}$, we say that D is defined over $\mathbb{K}$ if for every automorphism σ of $\overline{\mathbb{F}}$ that fixes $\mathbb{K}$ one has $\sum m_i P_i^{\sigma} = D$, where P^{σ} denotes the point obtained by applying σ to the coordinate values of P.

Let $\mathbb{J}$ (more precisely, $\mathbb{J}(\mathbb{K})$) denote the quotient of the group $\mathbb{D}^0$ of divisors of degree zero defined over $\mathbb{K}$ by the subgroup $\mathbb{P}$ of principal divisors coming from $G, H \in \mathbb{K}[u, v]$. $\mathbb{J} = \mathbb{D}^0/\mathbb{P}$ is called the Jacobian of the curve. If $D_1, D_2 \in \mathbb{D}^0$ then we write $D_1 \sim D_2$ if $D_1 - D_2 \in \mathbb{P}$; D_1 and D_2 are said to be equivalent divisors.

The divisors of the Jacobian can be represented as a pair of polynomials $a(u)$ and $b(u)$, where the coefficients are elements of a finite field [3]. The divisor D represented by polynomials will be denoted as $div(a, b)$.

The DLP on $\mathbb{J}(\mathbb{K})$ can be stated as follows: given two divisors $D_1, D_2 \in \mathbb{J}(\mathbb{K})$, determine the smallest integer m such that $D_2 = mD_1$, if such an m exists. The binary algorithm and its variants [5, 6] can be used to efficiently compute mD. The main operations in the algorithm are group addition and group doubling. These group operations will be the subject of the next section. The idea that Jacobians of hyperelliptic curves are suitable for DL cryptosystems was first introduced by Koblitz in [1].

2.2. Group Operations on a Jacobian

This section gives a brief description of the algorithms used for adding and doubling two divisors on a Jacobian. These group operations will be performed in two steps. First we have to find a semi-reduced divisor $D' = \mathrm{div}(a', b')$, such that $D' \sim D_1 + D_2 = \mathrm{div}(a_1, b_1) + \mathrm{div}(a_2, b_2)$ in the group $\mathbb{J}$. In the second step we have to reduce the semi-reduced divisor $D' = \mathrm{div}\ (a', b')$ to an equivalent divisor $D = (a, b)$. Algorithm 1 describes the addition of divisors.

Algorithm 1 ***Addition***
Input: $D_1 = \mathrm{div}(a_1, b_1)$, $D_2 = \mathrm{div}(a_2, b_2)$
Output: $D = \mathrm{div}(a, b) = D_1 + D_2$

1. $d = \gcd(a_1, a_2, b_1 + b_2 + h)$
   $= s_1 a_1 + s_2 a_2 + s_3(b_1 + b_2 + h)$
2. $a'_0 = a_1 a_2/d^2$
   $b'_0 = [s_1 a_1 b_2 + s_2 a_2 b_1 +$
   $s_3(b_1 b_2 + f)]d^{-1} (\mathrm{mod} a)$
3. *While* $\deg a'_k > g$ *do*
   3.1 $a'_k = \frac{f - b'_{k-1} h - (b'_{k-1})^2}{a'_{k-1}}$
   3.2 $b'_k = (-h - b'_{k-1}) \bmod a'_k$
4. *Output* $(a = a'_k, b = b'_k)$

Doubling a divisor is easier than the general addition, because $a = a_1 = a_2$ and $b = b_1 = b_2$. Hence, to perform the doubling of a divisors, steps 1 and 2 can be simplified as follows:

1. $d = \gcd(a, 2b + h)$
   $= s_1 a + s_3(2b + h)$
2. $a'_0 = a^2/d^2$
   $b'_0 = [s_1 ab + s_3(b^2 + f)]d^{-1} (\mathrm{mod} a'_0)$

3. HARDWARE ARCHITECTURE

In order to build an HECC processor, as shown in Figure 1, one has to provide good architectures for the time critical underlying operations. Hence, we focus on the development of optimized field and polynomial arithmetic architectures. Once we obtain efficient architectures for these operations, the implementation of the group operations is straight forward.

3.1. Field Arithmetic

This section summarizes the architectures for field operations used to perform the underlying field arithmetic. We will concentrate on fields $\mathbb{F}_{2^m}$, because fields of characteristic 2 are best suited for hardware architectures. An element $A \in \mathbb{F}_{2^m}$ will be represented as a polynomial $A(x) = \sum_{i=0}^{m-1} a_i x^i$, $a_i \in \mathbb{F}_2$. The irreducible polynomial will be denoted as $F(x)$.

Addition of two field elements $A, B \in \mathbb{F}_{2^m}$ is accomplished by a bitwise XORing of the field elements. This means: $C(x) = \sum_{i=0}^{m-1} c_i x^i = A(x) + B(x) = \sum_{i=0}^{m-1}((a_i + b_i) \bmod 2)x^i$. In terms of hardware, we need m parallel XOR units.

Multiplication of two field elements is calculated with the digit-serial multiplier introduced in [7]. We chose this architecture because it provides a good trade–off between area, power consumption, and performance. Let D be the digit-size, then the multiplication of two field elements can be expressed as:

$A(x)B(x) \mod F(x)$
$\equiv (A(x)\sum_{i=0}^{k_D-1} B_i(x)x^{Di}) \mod F(x)$
$\equiv (\sum_{i=0}^{k_D-1} B_i(x)(A(x)x^{Di} \bmod F(x)))$
$\bmod F(x)$

The product is calculated in $\lceil m/D \rceil$ clock cycles, as D bits are processed in one clock cycle.

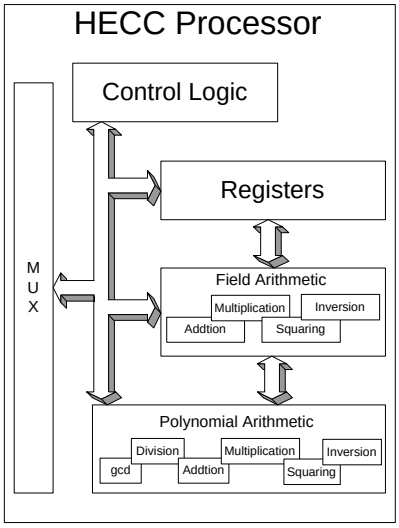

Figure 1: HECC Processor

Similarly, *squaring* can be accomplished as: $A^2(x) \equiv \sum_{i=0}^{m-1} a_i x^{2i} \bmod F(x); a_i \in \mathbb{F}_2$. The bit-parallel squarer is capable of computing a square in one clock cycle if one uses a fixed polynomial $F(x)$. The squaring requires at most $(r-1)(m-1)$ gates, where r represents the number of non-zero coefficients of the field polynomial.

Inversion can be computed using the Extended Euclidean Algorithm (EEA). Algorithm 4 describes how to perform an inversion over $\mathbb{F}_{2^m}$ according to [8].

Algorithm 2 ***Field-Inversion*** *[8]*
Input: $a \in \mathbb{F}_{2^m}$, $a \neq 0$
Output: $a^{-1} \bmod f(x)$

1. *Set* $b \leftarrow 1$, $u \leftarrow a$, $v \leftarrow f$, $c \leftarrow 0$
2. *While* $\deg(u) \neq 0$ *do:*
   - 2.1 $j \leftarrow \deg(u) - \deg(v)$
   - 2.2 *If* $j < 0$ *Then:* $u \leftrightarrow v$, $b \leftrightarrow c$, $j \leftarrow -j$
   - 2.3 $u \leftarrow u + x^j \cdot v$, $b \leftarrow b + x^j \cdot c$
3. *return(b)*

This algorithm is well suited for hardware because it does not use any divisions, rather it uses multiplications of field elements by x^j and additions. The multiplication by x^j is a j-position left shift, which is a very efficient operation and carries virtually no delay in hardware.

3.2. Polynomial Arithmetic

This section describes all the polynomial arithmetic necessary to implement Algorithm 1.

Polynomial addition can be done by bitwise XORing, since the coefficients of the polynomials are elements in $\mathbb{F}_{2^m}$. We have to add at most $(\deg[P(u)] + 1)$ field elements, where $P(u)$ is the higher input polynomial and $\deg[P(u)]$ is the degree of $P(u)$. Therefore we need $[(\deg[P(u)]+1) \cdot m]$ gates to implement a polynomial adder. The whole addition can be performed in one clock cycle.

A block diagram of our polynomial multiplier is shown in Figure 2:

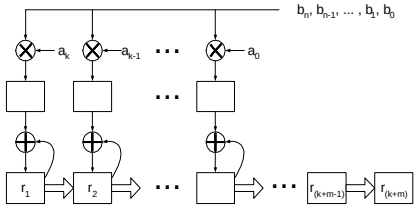

Figure 2: Polynomial Multiplication

The multiplier in Figure 2 parallelizes k coefficient multiplications. We multiply $A(u) = \sum_{i=0}^{k} a_i u^i$ by $B(u) = \sum_{i=0}^{n} b_i u^i$, where $a_i, b_i \in \mathbb{F}_{2^m}$. We start by multiplying the nth coefficient of $B(u)$ by $A(u)$. We continue this process by computing $b_i \cdot A(u)$, $i = n-1, ..., 0$. The result of each scalar multiplication is accumulated to the total result and is then shifted by the length of one coefficient. Note, that one scalar multiplication takes $\lceil m/D \rceil$ clock cycles, that is the same as one field multiplication.

Algorithm 3 performs a *polynomial squaring.*

Algorithm 3 ***Polynomial Squaring***
Input: $A(u) = \sum_{i=0}^{k} a_i u^i$, $a_i \in \mathbb{F}_{2^m}$
Output: $C(u) = \sum_{i=0}^{2k} c_i u^i$, $c_i \in \mathbb{F}_{2^m}$

1. *For* $j = 0$ *to* k
   - 1.1 $c_{2j} \leftarrow (a_j)^2 \bmod P(x)$
2. *Return* $(C(u))$

We need $\deg(A(u)) + 1$ field squarers to perform one polynomial squaring. A polynomial squaring takes one clock cycle, if the field squarer computes a result in one clock cycle.

The *gcd* of two polynomials can be calculated with a modified version of the EEA. The modification avoids numerous field inversions, full polynomial multiplications, and polynomial divisions, thus making the algorithm more efficient. In what follows, LC() denotes the leading coefficient of the polynomial.

Algorithm 4 ***Polynomial gcd***

Input: $r_0(u) = \sum_{i=0}^{k} a_i u^i$,
$r_1(u) = \sum_{i=0}^{n} b_i u^i$,
$a_i, b_i \in \mathbb{F}_{2^m}$, $\deg(r_0) > \deg(r_1)$
Output: $d(u) = \gcd(r_0(u), r_1(u))$
$= s(u)r_0(u) + t(u)r_1(u)$

1. $s_0(u) \leftarrow 1$, $s_1(u) \leftarrow 0$,
   $t_0(u) \leftarrow 0$, $t_1(u) \leftarrow 1$
2. *While* $r_1 \neq 0$ *do:*
   - 2.1 $j \leftarrow \deg(r_1) - \deg(r_0)$
   - 2.2 *if* $j < 0$ *then:* $r_0 \leftrightarrow r_1$,
     $t_0 \leftrightarrow t_1$, $s_0 \leftrightarrow s_1$, $j \leftarrow -j$
   - 2.3 $t_1 \leftarrow \mathrm{LC}(r_0) \cdot t_1 + u^j \cdot \mathrm{LC}(r_1) \cdot t_0$,
     $s_1 \leftarrow \mathrm{LC}(r_0) \cdot s_1 + u^j \cdot \mathrm{LC}(r_1) \cdot s_0$
   - 2.4 $r_1 \leftarrow \mathrm{LC}(r_0) \cdot r_1 + u^j \cdot \mathrm{LC}(r_1) \cdot r_0$
3. *Set* $d(u) \leftarrow \mathrm{LC}(r_0)^{-1} \cdot r_0(u)$, $s(u) \leftarrow \mathrm{LC}(r_0)^{-1} \cdot s_0(x)$, $t(x) \leftarrow \mathrm{LC}(r_0)^{-1} \cdot t_0(x)$
4. *Return* $(d(u), s(u), t(u))$

The multiplications in Steps 2.3 and 2.4 are scalar multiplications and we only have to perform one field inversion (Step 3).

For *polynomial division* we developed an algorithm that avoids field inversions and full polynomial multiplication

Algorithm 5 ***Polynomial Division***

Input: $A(u) = \sum_{i=0}^{k} a_i u^i$,
$B(u) = \sum_{i=0}^{n} b_i u^i$,
$a_i, b_i \in \mathbb{F}_{2^m}$, $\deg(A) > \deg(B)$
Output: $q(u)$, $r(u)$,
where $A(u) = q(u) \cdot B(u) + r(u)$

1. $inverse \leftarrow \mathrm{LC}[B(u)]^{-1}$
2. *For* $j = (\deg[A(u)] - \deg[B(u)])$ *down to* 0 *do:*
   - 2.1 $factor \leftarrow \mathrm{LC}[(A(u)] \cdot inverse$
   - 2.2 $factor \leftarrow factor \cdot u^j$
   - 2.3 $temp\_B(u) \leftarrow B(u) \cdot factor$
   - 2.4 $A(u) \leftarrow A(u) + temp\_B(u)$,
     $q(u) \leftarrow q(u) + factor$
3. *Set* $r(u) \leftarrow A(u)$
4. *Return* $(q(u), r(u))$

Multiplication by u^j is a left shift by j-coefficients. We also notice that we only need to do one field inversion (Step 1).

The *polynomial inverse* can be calculated via the polynomial gcd implementation. In this case the inverse is stored in $s(x)$ of Algorithm 4.
More detailed considerations of the HECC hardware architecture can be found in [9].

4. CONCLUSIONS

In this paper we first summarized the properties of HECC necessary for the development of suitable hardware architectures. We also presented architectures to implement the necessary field operations and polynomial arithmetic in hardware efficiently. In particular the development of an efficient algorithm for the polynomial division and the calculation of the Extended Euclidean Algorithm in the polynomial ring was demonstrated.

5. REFERENCES

[1] N. Koblitz, "A Family of Jacobians Suitable for Discrete Log Cryptosystems", *Crypto '88*, LNCS, Springer-Verlag, Berlin.

[2] N. Koblitz, "Algorithms and Computation in Mathematics", *Algebraic Aspects of Cryptography*, Springer-Verlag, 1998.

[3] N. Koblitz, "Hyperelliptic Cryptosystem", *Journal of Cryptology*, 1989.

[4] I. Blake, G. Seroussi and N. Smart, "Elliptic Curves in Cryptography", London Mathematical Society Lecture Notes Series 265, Cambridge University Press, Reading, MA, 1999.

[5] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York 1996.

[6] D. M. Gordon, "A Survey of Fast Exponentiation Methods", *Journal of Algorithms*, Volume 27, 1998.

[7] L. Song and K. K. Parhi, "Low-Energy Digit-Serial/Parallel Finite Field Multipliers", *Journal of VHDL Signal Processing Systems*, 1 -17, 1997.

[8] D. Hankerson, J. L. Hernandez and A. Menezes, "Software Implementation of Elliptic Curve Cryptography Over Binary Fields", *CHES '99*, Springer Verlag, August 2000.

[9] T. Wollinger, "Computer Architectures for Cryptosystems Based on Hyperelliptic Curves", *M.S. Thesis*, Worcester Polytechnic Institute, April 2001.

[10] N. Smart", "On the Performance of Hyperelliptic Cryptosystems", Advances in Cryptology - EUROCRYPT '99, LNCS, Springer-Verlag, 1999.