ABSTRACT

Cryptographic pairing (bilinear mapping) is a core algorithm for various cryptography protocols. It is computationally expensive and inefficiently computed with general purpose processors. Although there has been previous work looking into efficient hardware designs for pairing, most of these systems use small characteristic curves which are incompatible with practical software designs. In this paper, we propose novel processor architecture for pairing-based cryptography applications using large characteristic curves. It takes advantage of some unique FPGA features such as huge aggregated memory bandwidth and massively parallel computation logic to achieve high performance and high energy efficiency. The proposed architecture is ideal for server-side applications requiring flexibility, performance and energy efficiency.

Keywords: Finite Field Arithmetic Unit, Weil Pairing, Multi Precision Arithmetic, Pairing Processor.

1. Introduction

Pairing-based cryptography [1] is the use of a pairing between elements of two cryptographic groups to a third group to construct cryptographic systems. The central idea is the construction of a mapping between two useful cryptographic groups which allows for new cryptographic schemes based on the reduction of one problem in one group to a different, usually easier problem in the other group.

a) Bilinear Mapping.

For this Pairing-based cryptography protocols utilize bilinear mapping. It allows some important features such as identity-based cryptographic schemes and short signature schemes. The major pairing-based construct is the bilinear map. Consider two groups G1 and G2 of prime order q. For clarity, we denote G1 using additive notation and G2 using multiplicative notation, even though the group operations in G1 and G2 may well be different from the well-known arithmetic addition and multiplication. We consider P and Q two generators of G1, and we write

\[ aP = P + P + \ldots + P \text{(a times)} \]

We now consider the mapping \( e \) as follows:

\[ e : G1 \times G1 \rightarrow G2 \]

(Note that we do not know how to build a self-bilinear map, \( G1 \times G1 \rightarrow G1 \). This would be quite powerful.) Useful bilinear maps have three properties:

i) Bilinearity:

\[ \forall P, Q \in G1, \forall a, b \in \mathbb{Z}^*q, \]
\[ e(aP, bQ) = e(P, Q)^{ab} \]

ii) Non-Degeneracy

If everything maps to the identity, that’s obviously not interesting:
\[ \forall P \in G_1, P \neq 0 \Rightarrow \{e(P, P)\} = G_2 \]
(e(P, P) generates G2)

iii) Computability

e is efficiently computable.

We can find G1 and G2 where these properties hold: the Weil and Tate pairings prove the existence of such constructions. Typically, G1 is an elliptic-curve group and G2 is a finite field.

b) Advantages

In recent years there has been a paradigm shift from traditional computing to “Cloud Computing”. Instead of computing with local infrastructure, computations are provided as a service over the Internet (“The Cloud”). This generates huge demand for pairing-based cryptography [5] computations to maintain system integrity.

This work differs from previous hardware designs in three major ways. First, we target large characteristic pairings that are comparatively well-understood and more compatible with client-side software. Second, instead of implementing a pairing kernel for a specific algorithm and curve, we take a higher-level approach and design a novel architecture that is Parameterizable to different bit-widths and capable of implementing different pairing algorithms. Finally, the proposed architecture takes ad-vantage of modern FPGA features to provide flexibility for system scaling.

2. Background

A cryptography pairing is a bilinear mapping from two groups to another group. The known implementations of these pairings – the Weil and Tate pairings – involve fairly complex mathematics.

a) Weil Pairing

In mathematics, the Weil pairing is a construction of roots of unity by means of functions on an elliptic curve \( E \), in such a way as to constitute a pairing (bilinear form, though with multiplicative notation) on the torsion subgroup of \( E \).

i) Formulation

Suppose \( E \) is defined over a field \( K \). Given an integer \( n > 0 \) (we require \( n \) to be prime to \( \text{char}(K) \)) such that \( K \) contains a primitive \( n \)-th root of unity, then the \( n \)-torsion on \( E(K) \) has known structure, as a Cartesian product of two cyclic groups of order \( n \). The basis of the construction is of an \( n \)-th root of unity \( w_{P, Q} \) for given points \( P, Q \in E(K)[n] \),

\[ w(P, Q) \in \mu_n \]

for given points \( P, Q \in E(K)[n] \), here

\[ E(K)[n] = \{ T \in E(K) | n \cdot T = O \} \]

and \( \mu_n = \{ x \in K | x^n = 1 \} \)

By a direct argument one can define a function \( F \) in the function field of \( E \) over the algebraic closure of \( K \), by its divisor:

\[ (F) = \sum (P + k \cdot Q) - \sum (k \cdot Q) \]

with sums for \( 0 \leq k < n \). In words \( F \) has a simple zero at each point \( P + kQ \), and a simple pole at each point \( kQ \). Then \( F \) is well-defined up to multiplication by a constant. If \( G \) is the translation of \( F \) by \( Q \), then by construction \( G \) has the same divisor. One can show that

\[ \frac{G}{F} \neq 1 \]

unless \( P \) and \( Q \) generate cyclic subgroups one of which is inside the other.

In fact then \( G/F \) would yield a function on the isogenous curve \( E/C \) where \( C \) is the cyclic subgroup generated by \( Q \), having just one simple pole. Such a function cannot exist, as follows by proving the residue at the pole is zero, a contradiction.

So, if we define

\[ w(P, Q) = \frac{G}{F} \]

we shall have an \( n \)-th root of unity (translating \( n \) times must give 1) other than 1. With this definition it can be shown that \( w \) is anti-symmetric and bilinear, giving rise to a non-degenerate pairing on the \( n \)-torsion.
b) Tate Pairing

Let \( E \) be an elliptic curve over a field \( K_0 \). Let \( n \) be a positive integer which is coprime to the characteristic of the field \( K_0 \) [8]. The set of \( n \)th roots of unity is defined to be

\[
\mu_n = \{ u \in K_0 : u^n = 1 \}.
\]

Let the field \( K = K_0(\mu_n) \) to be the extension of \( K_0 \) generated by the \( n \)th roots of unity. Define

\[
E(K)[n] = \{ P \in E(K) : [n]P = \infty \}
\]

\[
nE(K) = \{ [n]P : P \in E(K) \}
\]

Then \( E(K)[n] \) is a group of exponent \( n \) and \( nE(K) \) is a subgroup of \( E(K) \) and the quotient group \( E(K)/nE(K) \) is a group of exponent \( n \). Define

\[
(K_n)^n = \{ u^n : u \in K \}
\]

Then the groups \( K'=(K_n)^n \) and \( \mu_n \) are isomorphic.

Tate Pairing over Finite Fields

Let \( K_0 = \mathbb{F}_q \) be a finite field. Let \( E \) be an elliptic curve defined over \( K_0 \) and let \( n \) be an integer coprime to \( q \) which divides \( \#E(K_0) \). The field \( K = K_0(\mu_n) \) is some finite extension \( F_q^k \) where \( k \) is called the embedded degree or security multiplier and for simplicity it is the smallest integer such that \( n \) divides \( (q^k - 1) \).

The value of Tate pairing is an equivalence class in \( F_q^n \) \( F_q^n \) and for practical reasons this value can be raised to the power \( (q^k - 1)/n \) so that a unique representation of this class can be obtained. Then the bilinear pairing can be defined as:

\[
e(P, Q) = (P, Q)^{(q^n - 1)/n}
\]

which maps into the group \( \mu_n \subset F_q^* \), rather than the group \( F_q^*/(F_q^*)^n \).

c) Software and Hardware Friendly Cryptography Pairing

We divide cryptographic pairings into two groups: “hardware-friendly” and “soft ware-friendly”. Hardware-friendly pairings [3] have Galois fields with small characteristics (i.e. binary or ternary fields) and their security strength comes from the large power in their extension fields. (Galois field so named in honor of Évariste Galois is a field that contains a finite number of elements). They are preferred in dedicated hardware designs because operations in these fields have short carry-chains and the computation of different coefficients in the polynomial can be independently evaluated and thus, it is easy to accelerate them through parallel computation. Conversely, software-friendly pairings use Galois fields with large prime numbers and small powers in their extension field. Operations in these fields have long carry-chains and straightforward parallelization of the arithmetic operations is generally not possible.

Although hardware-friendly curves favor dedicated hardware computation, they may not suitable for traditional desktop processors. This is because traditional CPUs are optimized towards large native word widths (e.g. 32-bit or 64-bit). Thus, computing over hardware-friendly curves may require wasting computational power calculating small pieces of data (e.g. 1-3 bits) with wide arithmetic units. As many more coefficients are required with hardware-friendly curves to achieve the same security level as one over a software-friendly curve, it requires disproportionately more time to compute using a conventional processor.

In addition to this, in the “Cloud Computing” paradigm, client-side computers are usually inexpensive, making dedicated hardware for cryptographic pairing infeasible. Employing hardware-friendly curves in such a paradigm would significantly penalize clients’ performance and should be avoided.

Another reason for using software-friendly curves is the fact that cryptographic pairings are relatively restrictive algorithms and there is simply much more research on curve / field combinations for software-friendly pairings. For example, a family of combinations named Barreto-Naehrig curves has been discovered. Such a family of curves is important because it allows a potential system developer to adaptively select different curves based upon criteria such as requisite security strength or the availability of computational resources.

d) Field Programmable Gate Arrays

Field programmable gate arrays (FPGAs) are high performance hardware devices with the flexibility of software. It is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence “field-programmable”. The FPGA configuration is generally specified using a hardware description language (HDL). The programmability allows them to be specialized to particular computations. This specialization leads to resource efficiency, which can be used for parallel computation to achieve high throughput.

FPGAs are increasingly used in conventional high performance computing applications where computational kernels (Fast Fourier transform, convolution etc) are performed on the FPGA instead of a microprocessor. FPGA implementation of these kernels offers order of magnitude performance improvements over microprocessors. Other benefits are in terms of
power used: a FPGA implementation of FFT or convolution is expected to consume lesser power than a microprocessor. Low-power usage is due to the lower clock rate and literally no wasted cycles for instruction fetch/decode in FPGAs.

Energy efficiency is another advantage of using FPGAs over general purpose processors. As the circuits can be specialized for particular computations, energy over-head is reduced. This is especially important for server-side applications where power consumption and cooling are critical issues.

FPGAs especially find applications in any area or algorithm that can make use of the massive parallelism offered by their architecture. One such relevant area to this paper is code breaking, in particular brute-force attack, of cryptographic algorithms.

3. Finite Field Arithmetic Unit

Our pairing-based processor is divided into two parts. The finite field arithmetic unit (ALU) carrying out computation and a control unit specialized for pairing algorithms

a) Multi precision arithmetic

Although finite field arithmetic of both the base field and the extension field is required in pairing, the extension field arithmetic can be decomposed and only base field arithmetic has to be supported. In software-friendly pairing, the prime number $q$ is usually large in order to provide sufficient security strength (i.e. 256-bit). Straightforward integer arithmetic circuits fail to achieve good performance with such large numbers due to the long carry chains. A common solution to the problem is multiple-precision arithmetic [2]. Instead of computing a finite field arithmetic operation in a single clock cycle, we break it down into multiple limbs and process them sequentially. Throughout the paper, we use $N$ to represent the total number of bits of the finite field, $W$ to represent the width of each limb $K=[N/W]$ and be the number of limbs in each word. We represent each limb of a variable $A$ with coefficients $a_i$.

i) Multi precision addition

Addition and subtraction are performed on two integers having the same number of base $b$ digits. To add or subtract two integers of different lengths, the smaller of the two integers is first padded with $0$’s on the left (i.e., in the high-order positions) [7].
can be adapted if more advanced multiplication techniques are desired. The integer multiplication data path is constructed with partial product generators (PPGs) and an adder tree. Let x and y be integers expressed in radix b representation:
\[ x = (x_n x_{n-1} \ldots x_1 x_0)_b \] and \[ y = (y_{t-1} \ldots y_1 y_0)_b. \]
The product x·y will have at most \((n + t + 2)\) base b digits. A single-precision multiplication means the multiplication of two base b digits. If \(x_i\) and \(y_j\) are two base b digits, then \(x_i \cdot y_j\) can be written as \((uv)_b\), where u and v are base b digits, and \(u\) may be 0.

Algorithm

Multiple-precision multiplication

INPUT: positive integers \(x\) and \(y\) having \(n + 1\) and \(t + 1\) base b digits, respectively.

OUTPUT: the product \(x \cdot y = (w_{n+t+1} \ldots w_0)_b\) in radix b representation.

1. For \(i\) from 0 to \((n + t + 1)\) do: \(w_i \leftarrow 0\).
2. For \(i\) from 0 to \(t\) do the following:
   2.1 \(c \leftarrow 0\).
   2.2 For \(j\) from 0 to \(n\) do the following:
      2.2.1 \(w_{i+j} \leftarrow v\), \(c \leftarrow 0\).
      2.2.2 \(c \leftarrow u\).
   2.3 \(w_{i+t+1} \leftarrow u\).
3. Return((\(w_{n+t+1} \ldots w_0\))).

Another approach for multiplication is also possible without changing the processor architecture.

Algorithm

The computations of \(P(x) \leftarrow x^d P(x) \mod F(x)\)

\[
P(x) \leftarrow B_s(x)A(x) + P(x) \mod F(x)\]

\[
V_1 = x^s \sum_{i=m-g-1}^{m-1} p_i x^i, \]
\[
V_2 = x^s \sum_{i=m-g}^{m-1} p_i x^i \mod F(x), \]
\[
V_3 = B_s(x)A(x) \mod F(x)\]

Note that \(V_1\) is a g-bit shift of the lower \(m - g\) bits of \(P(x)\). \(V_2\) is a g-bit shift of the upper \(g\) bits of \(P(x)\) followed by a modular reduction. \(V_3\) requires a polynomial multiplication and reduction where the operand polynomials have degree \(g - 1\) and \(m-1\).

Efficient Group Level Multiplication

INPUT: \(A(x), B(x), \) and \(F(x)\)

OUTPUT: \(P(x) \leftarrow A(x)B(x) \mod F(x)\)

1. For \(k = s - 2\) downto 0 do
   \[
   V_1 = x^s \sum_{i=0}^{m-g-1} p_i x^i, \]
   \[
   V_2 = x^s \sum_{i=m-g}^{m-1} p_i x^i \mod F(x), \]
   \[
   V_3 = B_k(x)A(x) \mod F(x)\]
   \[
P(x) = V_1 + V_2 + V_3\]

4. Pairing Processor

There are two common approaches in constructing a pairing processor using a finite field ALU.

a) Approach 1

First approach is to use hardwired state machines. Although such an approach is more efficient and provides compact and efficient designs, it is difficult to customize the controller for different curves and algorithms to explore design space. This approach is not compatible for change in algorithms. Any change in algorithm has to change the processor architecture and it may affect the efficiency of the system.

b) Approach 2

Another approach is to combine the ALU with a general purpose processor such as Xilinx MicroBlaze [6]. This approach allows us for modification in pairing algorithms and also curve characteristics through software. However most general purpose processors offer functionality unnecessary for pairing functions which in turn increases the overhead on the system and limits its performance. These problems can be solved by designing a special purpose processor and leveraging FPGA’s reconfigurability. Now taking the advantage from the special features of the Miller’s loops of every pairing algorithm, we can design a processor specialized for pairing. We encode the branching sequences into a circular shift and use a special instruction to carry out these data independent branching. As the number of JSR (Jump and Shift Register) instructions is fixed in every pairing algorithm, the shift register is restored to its initial value after each pairing is completed.
In this paper we presented processor architecture for large characteristic pairing-based cryptography. The processor is Parameterizable to different curves/ pairings and optimized for modern FPGAs. Hence, it is suitable for cryptosystems requiring variable strength security. We proposed a system that provides a superior throughput while offering better energy efficiency compared with traditional software-based pairing systems. To best of our knowledge, this is the first time that energy efficiency has been considered in pairing-based cryptography.

5. Conclusion

In this paper we presented processor architecture for large characteristic pairing-based cryptography. The processor is Parameterizable to different curves/ pairings and optimized for modern FPGAs. Hence, it is suitable for cryptosystems requiring variable strength security. We proposed a system that provides a superior throughput while offering better energy efficiency compared with traditional software-based pairing systems. To best of our knowledge, this is the first time that energy efficiency has been considered in pairing-based cryptography.

6. References


[3]. Hardware architectures of Elliptic Curve based Cryptosystems by Chang Shu.

[4]. Division by Integers using Multiplication by Chuck Pheatt, Emporia State University.

[5]. A Handbook of Applied Cryptography by A.Menezes.


