Miller's Algorithm in Pairing-Based Cryptography

Po-Jen Chen[R08943023], Da-Wei Lin[R08943020]

Energy-Efficient Circuit and System Lab,
Graduate Institute of Electronics Engineering, National Taiwan University.

Abstract. This term project describes the behaviors of Miller's algorithm in pairing-based cryptography and how [1] implemented the pairing computation efficiently with some mathematical skills. Lastly, we proposed a modified Miller's algorithm against side channel attack, the biggest threat to cryptographic systems.

Keywords: Miller's algorithm · Pairing-Based Cryptography · Tower extension · Divisor · Side Channel Attack.

1 Introduction

The protocol solutions provided by pairing-based cryptography can only be made practical if one can efficiently compute bilinear pairings at high levels of security. Back in 1986, Victor Miller proposed in [4], [5] an iterative algorithm that can evaluate rational functions from scalar multiplications of divisors, thus allowing to compute bilinear pairings at a linear complexity cost with respect to the size of the input. Since then, several authors have found further algorithmic improvements to decrease the complexity of Miller's algorithm by reducing its loop length, and by constructing pairing-friendly elliptic curves and pairing-friendly tower extensions of finite fields.

From 2001 to 2006, types of pairing cryptography start to develop, including the Weil pairing, the Tate pairing, and the optimal Ate pairing in the following equations 1 2 3. Those pairings are very complicated in math and non-trivial to compute even by using Miller's algorithm. Because pairings are computationally expensive, making these computation faster is current research.

$$\widehat{e}(P,Q) = f_P(A_Q)/f_Q(A_P) \tag{1}$$

$$\widehat{e}(P,Q) = f_P(A_Q) \tag{2}$$

$$\widehat{e}(P,Q) = [f_P(Q) \cdot l_P(\phi Q) \cdot l_P(-\phi^2 Q)]^{(q^k-1)/r} \tag{3}$$

Besides, Miller's algorithm is used to compute $f_P(Q)$ and it consisted of two major parts, Miller Loop and Final Exponentiation. The bottleneck in the pairing computation is in the latter because of the extremely high exponent. By implementing in the tower extensions of finite fields, we can do the power in the lower finite fields and put them together therefore. In addition, side channel attack (SCA), the biggest threats to cryptographic applications, needs to be considered carefully. Thus, some dummy operations would be added to make power information independent of the data being processed.

In this term project, we will review the cryptography in Section 2 and explain Miller's algorithm in Section 3. Then, we describe some practical skills in both software and hardware in Section 4. After a brief introduction of how pairings are implemented, we will give an approach against SCA in section 5. Finally, we concluded the project in Section 6.

2 Cryptography Review

2.1 Bilinear Map

Roughly speaking, an asymmetric bilinear pairing can be defined as the non-degenerate bilinear mapping,

$$\widehat{e} : G_1 \times G_2 \to G_3 \tag{4}$$

where both G_1, G_2 are finite cyclic additive groups with prime order r, whereas G_3 is a multiplicative cyclic group whose order is also r. Additionally, as it was mentioned above, for cryptographic applications it is desirable that pairings can be computed efficiently. When $G_1 = G_2$, we say that the pairing is symmetric, otherwise, if $G_1 \neq G_2$, the pairing is asymmetric. And a bilinear map is a function such that

$$\widehat{e}(aP, bP) = \widehat{e}(P, Q)^{ab}, \forall P, Q \in G, \forall a, b \in Z. \tag{5}$$

With this map, we can establish relationship between cryptographic groups and make Decisional Diffie-Hellman (DDH) easy in one of them in the process.

2.2 Divisor

The math behind why pairing functions work is quite tricky and involves quite a bit of advanced algebra going even beyond what we have seen so far, but we provide an outline as following. First of all, we need to define the concept of a divisor, basically an alternative way of representing functions on elliptic curve points. A divisor of a function basically counts the zeroes and the infinities of the function. Take examples, let us fix some point $P = (P_x, P_y)$, and consider the following line function: $f(x, y) = x - P_x$. The divisor is $[P] + [-P] - 2[O]$ (the square brackets are used to represent the fact that we are referring to the presence of the point P in the set of zeroes and infinities of the function, not the point P itself; $[P] + [Q]$ is not the same thing as $[P + Q]$) The reason is as follows:

1. The function is equal to zero at P, since x is P_x so $x - P_x = 0$
2. The function is equal to zero at $-P$, since $-P$ and P share the same x coordinate

3. The function goes to infinity as x goes to infinity, so we say the function is equal to infinity at O. Theres a technical reason why this infinity needs to be counted twice, so O gets added with a multiplicity of -2 (negative because its an infinity and not a zero, two because of double counting).

Now, let us consider a line function: $ax+by+c=0$ which passes through points P and Q. By the definition of elliptic curve group operation, the line also passes through $-P-Q$ and it goes to infinity dependent on both x an y. So the divisor becomes $[P]+[Q]+[-P-Q]-3[O]$.

For any two functions F and G, the divisor of $F \cdot G$ is equal to the divisor of F plus the divisor of G, which means $(F \cdot G) = (F) + (G)$, so for example if $f(x,y) = P_x - x$, then $(f^3) = 3[P] + 3[-P] - 6[O]$; P and $-P$ are triple-counted to account for the fact that f^3 approaches 0 at those points three times as quickly in a certain mathematical sense.

3 Miller's Algorithm

Take the Tate pairing for example, we want to compute $\widehat{e}(P,Q) = f_p(Q)^{(q^{12}-1)/r}$[1] where f is the function with divisor $(f_c) = c[P_0] - [cP_0] - (c-1)[O]$ if we know the base point P_0 and the constant c such that $P = c[P_0]$ in the finite field F_q. In Miller's algorithm, Miller Loop computes $f_P(Q)$ and Final Exponentiation raise the result to the power of $(q^{12}-1)/r$ whose details are showed bellow.

3.1 Miller Loop

The tricky problem in Miller Loop is that the base point and the constant would not know from the input of the algorithm. Instead, we treat the input point P as the alternative base point and use the group order r to replace the constant. In other words, we have to find the rational function with divisor $(f_r) = r[P] - [rP] - (r-1)[O]$. To explain why the alternative approach works, we need to know how to find f_{a+b} from f_a and f_b, which is also the major equation in Miller Loop and shown in algorithm 1. Then, if we take f_{r+1} into algorithm 1, we can notice that

$$f_{r+1} = f_r \cdot f_1 \cdot \frac{g_{r,1}}{g_{r+1,-r-1}} = f_r \frac{g_{O,-P}}{g_{P,-P}} = f_r \tag{6}$$

And also

$$f_{r+1} = f_{[O]+[P]} = f_{[P]} = f_1 \tag{7}$$

Therefore, we can adopt those alternative variables to realize the rational function with specific divisors.

The computations in Miller Loop consisted of two parts, elliptic curve point multiplication (ECPM) and operations of rational function where Algorithm 2 shows the steps of Miller Loop. First, we focus on the point T who does

[1] r is the curve order.

ECPM once to compute $rP = O$. That is, if the scanned bit r_i is 1, do a pair of elliptic curve point addition (ECPA) and elliptic curve point double (ECPD); if the scanned bit r_i is 0, do an ECPD afterwards. Second, the rational function f simply follows the point T by using Algorithm 1. Besides, squaring and multiplication in finite field are needed in this step.

Algorithm 1 How to compute f_{a+b}

Input: f_a, f_b
Output: f_{a+b}

$(f_a) = a[P] - [aP] - (a-1)[O]$
$(f_b) = b[P] - [bP] - (b-1)[O]$
$(g_{a,b}) = [aP] + [bP] + [-aP - bP] - 3[O]$
$(g_{(a+b),-(a+b)}) = [aP + bP] + [-aP - bP] - 2[O]$
$(f_a) \cdot (f_b) \cdot \frac{(g_{a,b})}{(g_{(a+b),-(a+b)})}$
$= (a+b)[P] - [aP + bP] - (a+b-1)[O]$
$= (f_{a+b})$

Algorithm 2 Miller Loop

Input: $P, Q, r = \sum_{i=0}^{l-1} r_i 2^i$, where $r_i \in \{0, 1\}$
Output: $f_P(Q)$

$T \leftarrow P; f \leftarrow 1$
for i from $l-2$ to 0 **do**
  $f \leftarrow f^2 \cdot g_{T,T}(Q); T \leftarrow 2T$
  if $r_i = 1$ **then**
    $f \leftarrow f \cdot g_{T,P}(Q), T \leftarrow T + P$
return f

3.2 Final Exponentiation

Normally, the size of the prime q is at least 254-bits so it is obviously difficult to raise the result of Miller Loop to the power of $(q^{12}-1)/r$ in the large extension finite field, showned in Algorithm 3. Besides, the exponent $(q^{12}-1)/r$ comes from the elliptic curve we choose, also knowns as a Barreto-Naehrig elliptic curve whose embedding degree is equal to 12. Since $k = 12 = 2^2 \cdot 3$, the tower extensions can be created using irreducible binomials only. This is because $x^k - \beta$ is irreducible over F_q provided that $\beta \in F_q$ is neither a square nor a cube in F_q. Hence, the tower extension can be constructed by simply adjoining a cube or square root of such element β and then the cube or square root of the previous root. This process should be repeated until the desired extension of the tower has been reached.

Algorithm 3 Final Exponentiation

Input: $f_P(Q)$

Output: $f_P(Q)^{(q^{12}-1)/r}$

1: $f \leftarrow f^{(q^{12}-1)/r}$

2: **return** f

Accordingly, we decided to represent $F_{q^{12}}$ using the tower extension, namely, we first construct a quadratic extension, which is followed by a cubic extension and then by a quadratic one, using the following irreducible binomials:

$$\begin{aligned} F_{q^2} &= F_q[u]/(u^2-\beta), \beta = -5, \\ F_{q^6} &= F_{q^2}[v]/(v^3-\xi), \xi = u, \\ F_{q^{12}} &= F_{q^6}[w]/(w^2-v). \end{aligned} \tag{8}$$

We first remark that the field extension $F_{q^{12}}$ can be also represented as a sextic extension of the quadratic field, i.e., $F_{q^{12}} = F_{q^2}[W]/(W^6-u)$, with $W = w$. Hence, we can write $f = g + hw \in F_{q^{12}}$, with $g, h \in F_{q^6}$ such that $g = g_0 + g_1 v + g_2 v^2, h = h_0 + h_1 v + h_2 v^2$ where $g_i, h_i \in F_{q^2}$ for $i = 0, 1, 2$. This means that f can be equivalently written as, $f = g + hw = g_0 + h_0 W + g_1 W^2 + h_1 W^3 + g_2 W^4 + h_2 W^5$. We note that the q-power of an arbitrary element in the quadratic extension field F_{q^2} can be computed essentially free of cost as follows. Let $b \in F_{q^2}$ be an arbitrary element that can be represented as $b = b_0 + b_1 u$. Then, $(b)^{q^{2i}} = b$ and $(b)^{q^{2i-1}} = \overline{b}$ with $\overline{b} = b_0 - b_1 u$ for $i \in N$. Moreover, performing squaring[3] is extremely efficiently in the cyclotomic subgroup of $F_{q^6}^{\times}$ for $q \equiv 1 \pmod 6$. Thus, with the identity $W^q = u^{(q-1)/6} W$, we can write $(W^i)^q = \gamma_{1,i} W^i$ with $\gamma_{1,i} = u^{i(p-1)/6}$ for $i = 1, \cdots, 5$. From the definitions given above, we can compute f^q as

$$\begin{aligned} f^q &= (g_0 + h_0 W + g_1 W^2 + h_1 W^3 + g_2 W^4 + h_2 W^5)^q \\ &= \overline{g_0} + \overline{h_0} W + \overline{g_1} W^{2q} + \overline{h_1} W^{3q} + \overline{g_2} W^{4q} + \overline{h_2} W^{5q} \\ &= \overline{g_0} + \overline{h_0}\gamma_{1,1} W + \overline{g_1}\gamma_{1,2} W^2 + \overline{h_1}\gamma_{1,3} W^3 + \overline{g_2}\gamma_{1,4} W^4 + \overline{h_2}\gamma_{1,5} W^5 \end{aligned} \tag{9}$$

In this way, the equation above has a computational cost of 5 multiplications in F_q and 5 conjugations in F_{q^2}. We can follow a similar procedure for computing f^{q^2} and f^{q^3}, which are arithmetic operations required in the hard part of the final exponentiation. For that, we must pre-compute and store the per-field constants $\gamma_{1,i} = u^{i(p-1)/6}, \gamma_{2,i} = \gamma_{1,i} \cdot \overline{\gamma_{1,i}}$ and $\gamma_{3,i} = \gamma_{1,i} \cdot \gamma_{2,i}$ for $i = 1, \cdots, 5$.

4 Practical Skills in Software and Hardware

4.1 Modular Reduction

This subsection describes several optimizations for some operations over F_{q^2}. In Algorithm 4, consider $A, B, C \in F_{q^2}$ and $C = c_0 + c_1 u = A \cdot B$, then

$c_0 = a_0b_0 - 5a_1b_1$ and $c_1 = (a_0+a_1)(b_0+b_1) - a_0b_0 - a_1b_1$. Hence, only three multiplications over F_q need to be computed, reducing one multiplication compared to common operation over F_{q^2}. Thus, it may seem that three **mod512** operations are necessary. However, we can keep the results of products **mul256**(s,t), **mul256**(a_0,b_0), and **mul256**(a_1,b_1) in the 512-bit integers. After all additions and subtractions are done, we can do a **mod512** in order to get c_0 and c_1.

Algorithm 4 Optimized Multiplication over F_{q^2}

Input: $A, B \in F_{q^2}$ where $A = a_0 + a_1u, B = b_0 + b_1u$
Output: $C \in F_{q^2}$ where $C = c_0 + c_1u$

$s \leftarrow$ **addNC**(a_0, a_1)
$t \leftarrow$ **addNC**(b_0, b_1)
$d_0 \leftarrow$ **mul256**(s, t)
$d_1 \leftarrow$ **mul256**(a_0, b_0)
$d_2 \leftarrow$ **mul256**(a_1, b_1)
$d_0 \leftarrow$ **subNC**(d_0, d_1)
$d_0 \leftarrow$ **subNC**(d_0, d_2)
$c_1 \leftarrow$ **mod512**(d_0)
$d_2 \leftarrow 5d_2$
$d_1 \leftarrow d_1 - d_2$
$c_0 \leftarrow$ **mod512**(d_1)
return $C \leftarrow c_0 + c_1u$

In addition to optimization for multiplication over F_{q^2}, we can save lots of costly checks after modulo addition or modulo subtraction when storing values in 256-bits integers. With selected 254-bits prime q satisfying $7q < N$, we can safely add/subtract the operands without carry check in Algorithm 5.

Algorithm 5 Optimized Squaring over F_{q^2}

Input: $A \in F_{q^2}$ where $A = a_0 + a_1u$
Output: $C = A^2 \in F_{q^2}$

$t \leftarrow$ **addNC**(a_1, a_1)
$d_1 \leftarrow$ **mul256**(t, a_0)
$t \leftarrow$ **addNC**(a_0, q)
$t \leftarrow$ **subNC**(t, a_1)
$c_1 \leftarrow 5a_1$
$c_1 \leftarrow$ **addNC**(c_1, a_0)
$d_0 \leftarrow$ **mul256**(t, c_1)
$c_1 \leftarrow$ **mod512**(d_1)
$d_1 \leftarrow$ **addNC**(d_1, d_1)
$d_0 \leftarrow$ **subNC**(d_0, d_1)
$c_0 \leftarrow$ **mod512**(d_0)
return $C \leftarrow c_0 + c_1u$

4.2 Pipelined Scheme

In terms of the hardware implementation, some have implemented fully pipelined 12-stage F_{q^2} multiplier, with Karatsuba method and Lazy Reduction method, as shown in Fig. 1. By using iterative accumulator mechanism, it can relax the data dependency resistance. Moreover, some would employ radix-4 unified division [2] for the Montgomery inversion operation, which usually takes $3l$ cycles, results in at most l cycles.

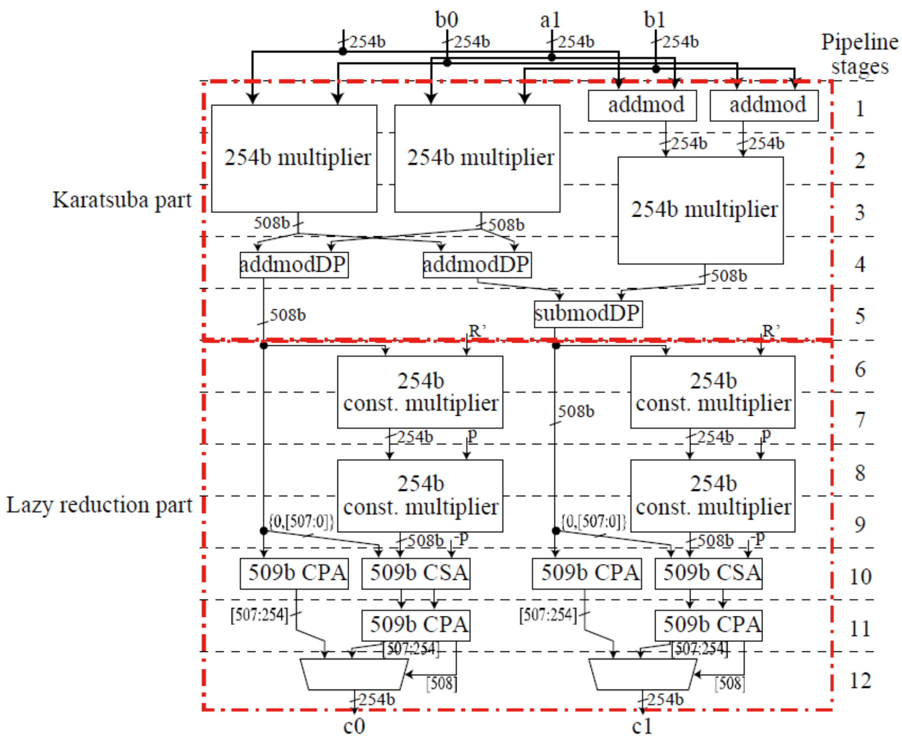

Fig. 1. 12-stage fully pipelined F_{q^2} multiplier.

5 SCA Countermeasure

SCA, the biggest threat to PKC, focuses on attacking hardware physical state like power or time which is the key dependent information. So chips need additional hardware or special algorithm to avoid leakage information during operation. Power consumption attacks are based on the observation that the power consumed at a given time during cryptographic process is related to the instruction being executed and the data being manipulated. And power consumption analysis may also enable to distinguish between instruction being executed. For example, it might be possible to distinguish between point doubling and point addition in Algorithm 6, thereby revealing the bits of the integer d. In order to be resistant against SPA, the instructions performed during a cryptographic algorithm should not depend on the data being processed, e.g. there should not be any branch instructions conditioned by the data. It is easy to modify Algorithm 6 to achieve this goal, which shown in Algorithm 7.

Algorithm 6 Double-and-add Algorithm

Input: an integer d, and a base point P
Output: dP

1: $Q \leftarrow P$
2: **for** i from $l-2$ to 0 **do**
3: $\quad Q \leftarrow 2Q$
4: $\quad$ **if** $d_i = 1$ **then** $Q \leftarrow Q + P$
5: **return** Q

Algorithm 7 Double-and-add Always Algorithm

Input: an integer d, and a base point P
Output: dP

1: $Q[0] \leftarrow P$
2: **for** i from $l-2$ to 0 **do**
3: $\quad Q[0] \leftarrow 2Q[0]$
4: $\quad Q[1] \leftarrow Q[0] + P$
5: $\quad Q[0] \leftarrow Q[d_i]$
6: **return** $Q[0]$

Based on the concept of Algorithm 7, we proposed a modified Miller's algorithm in 8, adding some dummy operations in Miller Loop to eliminate the power message. However, high computational overhead leading to significant performance loss is inevitable due to extra ECPA calculations with the enlarged curve order. Furthermore, there is no need to modify Final Exponentiation because of the equivalent exponent at specific elliptic curve.

Algorithm 8 Modified Miller's Algorithm

Input: points P, Q
Output: $\widehat{e}(P, Q)$
Miller Loop:
$T[0] \leftarrow P; f_0 \leftarrow 1$
for i from $l-2$ to 0 **do**
  $f_0 \leftarrow f_0^2 \cdot g_{T[0],T[0]}(Q); T[0] \leftarrow 2T[0]$
  $f_1 \leftarrow f_0 \cdot g_{T[0],P}(Q); T[1] \leftarrow T[0] + P$
  $f_0 \leftarrow f_{r_i}; T[0] \leftarrow T[r_i]$
return f
Final Exponentiation:
$f \leftarrow f^{(q^{12}-1)/r}$
return f

6 Conclusion

This term project describes the details of Miller's algorithm from the view point of implementation. For software designers who want to use pairing-based cryptography in their works, try to use open sources as much as possible because they are computationally expensive and non-trivial to compute. On the other hand, for hardware implementation, ones should start from algorithmic level instead of hardware architecture because many practical skills are derived from math. Next, by implementing Miller Loop with our modified algorithm, pairings will be effectively resistant to simple power analysis. Besides, as computing power is improving, the security level should expend correspondingly, which also means that we should reduce pairing operations or use embedded devices to make them faster.

References

1. Beuchat, J.L., González-Díaz, J.E., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal ate pairing over barreto–naehrig curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing-Based Cryptography - Pairing 2010. pp. 21–39. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
2. Chen, Y., Lee, J., Liu, P., Chang, H., Lee, C.: A dual-field elliptic curve cryptographic processor with a radix-4 unified division unit. In: 2011 IEEE International Symposium of Circuits and Systems (ISCAS). pp. 713–716 (May 2011). https://doi.org/10.1109/ISCAS.2011.5937665
3. Granger, R., Scott, M.: Faster squaring in the cyclotomic subgroup of sixth degree extensions. In: Nguyen, P.Q., Pointcheval, D. (eds.) Public Key Cryptography – PKC 2010. pp. 209–223. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
4. Miller, V.S.: Short programs for functions on curves. In: IBM THOMAS J. WATSON RESEARCH CENTER (1986)
5. Miller, V.S.: The weil pairing, and its efficient calculation. Journal of Cryptology **17**(4), 235–261 (Sep 2004). https://doi.org/10.1007/s00145-004-0315-8, https://doi.org/10.1007/s00145-004-0315-8