Side Channel Analysis of Practical Pairing Implementations: Which Path is More Secure?

Claire Whelan and Mike Scott

School of Computing, Dublin City University
Ballymun, Dublin 9. Ireland
{cwhelan, mike}@computing.dcu.ie

Abstract. We present an investigation into the security of three practical pairing algorithms; the Tate, Eta and Ate pairing, in terms of side channel vulnerability. These three algorithms have recently shown to be efficiently computable on the resource constrained smart card, yet no in depth side channel analysis has yet appeared in the literature. Since the secret parameter input to the pairing can potentially be entered in either of the two possible positions, there exist two avenues of attack, i.e. $e(P, Q)$ or $e(Q, P)$ where P is public and Q is private. We analyse the core operations fundamental to pairings and not only highlight how each implementation may potentially succumb to a side channel attack, but also show how one path is more susceptible than the other in Tate and Ate. For those who wish to deploy pairing based systems we make a simple suggestion to improve resistance to SCA.

Keywords: Side Channel Analysis (SCA), Pairing Based Cryptography, Correlation Power Analysis (CPA), Tate Pairing.

1 Introduction

Pairings are a relatively new primitive in the world of cryptography. Pairings are bilinear maps, which make them attractive for cryptographic constructions. Since their introduction in the constructive sense[1], a multitude of pairing based protocols have been suggested and a handful of efficient pairing implementations have been developed. We refer the reader to [1] for a comprehensive listings of such papers. One aspect that has hindered the widespread adoption of pairings, from a commercial perspective, is whether they were efficiently computable on resource constrained devices such as the smart card and could compete with existing cryptographic constructions. Recently Scott et al. [14] have answered this question and present impressive timings for the computation of the Tate [3], Ate [9] and Eta [2] pairings on a 32 bit smart card.

An equally popular but more mature field is that of Side Channel Analysis (SCA). SCA has advanced immeasurably since its breakthrough into the security community almost a decade ago [10]. Almost every cryptographic construction,

[1] Initially pairings were suggested for cryptanalytic purposes [11].

especially those intended for use in the smart card, have been subject to some form of SCA or another. These powerful attacks, which do not play by the rules of traditional cryptanalysis, have proven successful against most algorithms.

While some SCA has been performed on elliptic curve cryptosystems, the most researched attack which targets point scalar multiplication, is not applicable to pairings. In 2005 Page and Vercauteren [12] presented the first side channel analysis of a pairing algorithm, when a fault attack was applied to the Duursma-Lee algorithm [7] for characteristic three. Preliminary simple and differential power analysis (SPA and DPA) in a generic sense was also briefly addressed.

We present the first security analysis of the BKLS algorithm for the Tate pairing [3], the Ate pairing [9], and the BGOhES algorithm for a truncated version of the Eta pairing [2] in terms of SCA. We evaluate these algorithms assuming the presence of an adversary capable of passive differential side channel attacks. In particular we assess the candidate pairing algorithms based on the attack, correlation power analysis (CPA) [6]. Our reasoning for using CPA is that it focuses on words of data at a time instead of selection functions, and it overcomes some of the shortcomings of differential power analysis (DPA) such as ghost peaks [6]. We theoretically analyse the pairings using empirical knowledge about SCA to aid in determining potential weaknesses. However we approach our analysis from a different perspective. Instead of focusing on specific algorithms for specific operations, we focus on how operations are computed from a structural perspective. We also assess each candidate pairing algorithm based on the prospect of the secret parameter being entered in either parameter position and subsequently present recommendations for protocol designers in employing pairings based on our analysis.

The paper is organised as follows. A brief overview of the candidate pairing algorithms and CPA is presented in section 2. In section 3 we analyse the core pairing operations in terms of how they may be attacked using side channels from a structural sense. We define a strategy of attack for each pairing algorithm and consequently compare Tate, Eta and Ate in section 4. We present possible countermeasures and address their effectiveness in deterring SCA in section 5. Finally we conclude and summarise our findings in section 6. Note that the specific pairing algorithms themselves can be found in appendix A.

2 Background

We briefly review relevant details on pairings and CPA.

2.1 Overview of Practical Pairings

Pairings are functions which map a pair of elliptic curve points to an element of a multiplicative group of an underlying finite field. Algorithms A.1, A.2 and A.3 describe implementations of Tate, Ate and Eta respectively. Each of these algorithms are efficiently computable on a 32 bit smart card, executing in under half a millisecond [14].

The BKLS [3] algorithm is a particularly fast method for computing the Tate Pairing $e(P, Q)$, where $P \in E(\mathbb{F}_p)$ is a point on the base curve and $Q \in E'(\mathbb{F}_{p^{k/d}})$ is a point on the d-th order twist with embedding degree k, where d is at least 2 when k is even [4]. BKLS can be calculated over supersingular or non-supersingular curves over finite fields of arbitrary characteristic.

The Ate pairing [9] $a(P, Q)$ is the most recently discovered pairing algorithm, and is potentially faster than BKLS for non-supersingular curves. Ate cleverly observes that it is more efficient to make the first parameter $P \in E'(\mathbb{F}_{p^{k/d}})$ and the second parameter $Q \in E(\mathbb{F}_p)$.

The BGOhES algorithm for the Eta pairing, is a generalisation of Duursma-Lee pairing algorithm for the Tate pairing with a truncated loop. The pairing $\eta_T(P, Q)$ is calculated on a supersingular curve over small characteristic, where both parameters P and Q are elements in $E(\mathbb{F}_{2^m})$.

Each pairing algorithm ultimately consists of an application of Millers algorithm followed by a final exponentiation. The notable difference between the three pairing is that Ate and Eta both have half length loops compared to Tate.

2.2 Correlation Power Analysis

The basis for correlation power analysis (CPA) [6] and other forms of passive differential SCA is that there exists a relationship between the data being processed during a computation and detectable physical manifestations such as power consumption and electromagnetic emanations. This dependance is magnified by capturing numerous acquisitions of the target in operation and then applying statistical analysis techniques to differentiate the signal of interest from noise.

Specifically, CPA builds an hypothetical model based on assumptions made about what constitutes energy dissipation. Then for key guesses, the correctness of a guess is established by estimating what the consumption of such data would be (based on the model) and then comparing it to actual data. This is generally performed using a correlation test such as Pearson's correlation coefficient. We make use of these findings in our analysis.

3 Side Channel Analysis of Naive Pairings

In a number of pairing based protocols, either the P or Q parameter is secret. For example, in Boneh and Franklin's identity based encryption [5] the critical operation involving the secret key in a pairing is the decryption operation. Although we are analysing pairings in isolation, the associated side channel security of pairings have implications in the bigger picture.

In order to perform critical analysis of the candidate pairing algorithms, it is necessary to analyse the core pairing operations in terms of how much information they can potentially leak. In this section we will analyse the finite field calculations central to pairings. Before we address these operations individually, we make some observations about pairings.

3.1 Pairing Observations

We note the following possible opportunistic observations about pairings:

1. The secret parameter can potentially be entered as the first or second parameter in the pairing. If the curve is supersingular and a distortion map $\psi(.)$ is used, as is the case with the Eta pairing, the parameters to the pairing $e(P, Q)$ can be switched, i.e. $e(Q, P)$ will yield the same result. In the case of the Tate and Ate pairing, while the parameter can take either path, it must hold for the entire protocol. Therefore this presents us with two avenues of attack; the P path and the Q path. We note that depending on which path is most vulnerable to SCA, such implications may lead to a simple method of defence.
2. Due to point compression we only need to extract the x coordinate of the secret point. Once this is found there are only two possibilities for y. Therefore we restrict our attention to the secret x coordinate.
3. We will try to focus on operations which involve elements from the base field $\mathbb{F}_q$, where $q = p$ or 2^m since extension field elements $\mathbb{F}_{q^k}$ are up to k bit times larger than that of base field elements.

3.2 Structural Analysis of Core Pairing Operations

One of the key requirements in performing a differential side channel attack is to identify an exploitable operation in the algorithm which involves some known (or computable) data and the secret key. Since elliptic curve arithmetic ultimately relies on the underlying finite field, we will restrict our analysis to multiplication, squaring, square root and reduction over the binary field and multiplication and reduction over the prime field. We refer the reader to appendix A to see when and where such operations are used in the candidate pairings.

We briefly recap on binary field and prime field arithmetic, since the candidate pairing implementations are over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$.

Characteristic two finite fields $\mathbb{F}_{2^m}$ are constructed using polynomial basis representation: $a(z) = \{a_i z^{m-1} + a_{i-1} z^{m-2} + \ldots + a_2 z^2 + a_1 z + a_0 \mid a_i \in \{0, 1\}\}$ where $a(z) \in \mathbb{F}_{2^m}$ has degree at most $m - 1$. Arithmetic over $\mathbb{F}_{2^m}$ is modulo the irreducible polynomial $f(z)$. We will represent $a(z) \in \mathbb{F}_{2^m}$ as the concatenation of w bit blocks: $a(z) = a_{m/w-1}|a_{m/w-2}|\ldots|a_0$, where w is the underlying processor's word length.

Characteristic p finite fields, $\mathbb{F}_p$, where p is a large prime, consist of the integers $0, 1, 2, \ldots, p - 1$ with arithmetic modulo p. Let $n = \lceil \log_2 p \rceil$ be the bit length of p. We will represent the elements $a \in \mathbb{F}_p$ as the concatenation of w bit blocks: $a = a_{n/w-1}|a_{n/w-2}|\ldots|a_0$.

Since we only need to deterministically calculate partial output of target operations, we revert back to the most basic methods for insight.

Multiplication. The most straightforward method for multiplication is the shift and xor method for $\mathbb{F}_{2^m}$ and the operand scanning method for $\mathbb{F}_p$. These methods are very similar, and so will only describe the former.

The multiplication[2] of two $\mathbb{F}_{2^m}$ elements $a(z) = \sum_{i=0}^{m-1} a_i z^i$ and $b(z) = \sum_{i=0}^{m-1} b_i z^i$ will produce the binary polynomial $c(z) = \sum_{i=0}^{2m-1} c_i z^i$, with degree $2m-1$. The shift and xor method involves multiplying words of $b(z)$ by words of $a(z)$ at a time. This process is depicted in figure 1. In the smart card system used by Scott et al. [14] they used a special binary polynomial multiplication instruction. The main distinction between this method and the multiplication of two $\mathbb{F}_p$ elements is that instead of xor-ing, addition (with carry bits) is performed.

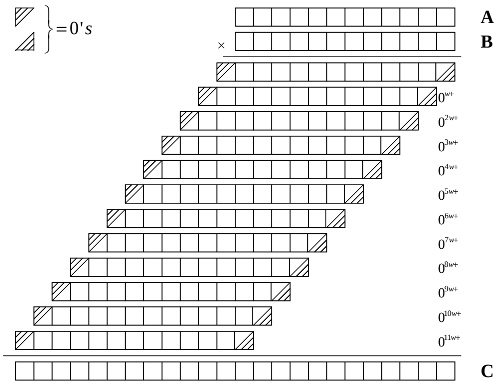

Fig. 1. Multiplication of $\mathbb{F}(2^m)$ elements: the shift and xor method.

A simple power analysis (SPA) attack on the bitwise shift and xor method was suggested by Page et al. in [12], which could easily be extended to apply to the operand scanning method. However, since it is unlikely that this basic algorithm will be favoured in a constrained embedded device, it is doubtful that SPA will work. Other attacks of modular multiplication have also been suggested. Walter [15] demonstrates how Montgomery multiplication can be attacked with SPA if an extra reduction is included.

In reality a number of multiplication algorithms can be implemented. We suggest that instead of focusing on the multiplication algorithm itself, we focus on the result of the multiplication. Due to the structural evolution of multiplication, which the basic algorithms allow us to easily see (as in figure 1), we can easily identify which data portions effect the resulting product value (or partial product). A possible side channel attack of the multiplication operation is as follows:

Let the target operation for a CPA attack be the multiplication of two finite field elements. Note that we will deal with the act of multiplication and reduction separately for the moment. Let x be an n-bit known (or computable) value

[2] In the context of binary fields by multiplication we mean carry-free binary polynomial multiplication.

by the adversary and k be an n-bit unknown secret value. Let $y = x \cdot k$ be the resulting $2n$-bit product. We represent x, k and y as the concatenation of w bit blocks: $x = x_{n/w-1}|x_{n/w-2}|\ldots|x_0$, $k = k_{n/w-1}|k_{n/w-2}|\ldots|k_0$ and $y = y_{2n/w-1}|y_{2n/w-2}|\ldots|y_0$ accordingly.

Since multiplication is not a suitable selection or partition function, and CPA is the attack of choice, w-bit portions of k will be extracted at a time[3]. To identify the target input block we denote x_l and k_l to be the l^{th} w-bit block of x and k respectively, where $0 \leq l \leq n/w - 1$. To identify the hypothetical output block we denote y_r to be the r^{th} w-bit block of y, where $0 \leq r \leq 2n/w - 1$.

If we are dealing with implementations over the binary field, there are two possible positions from which the attack can commence, either the most or least significant word of k, since all middle words of the product $x{\cdot}k$ are polluted by the outermost words. If the implementation is over the prime field, we are restricted to commencing from the least significant word only since carry propagation will significantly effect all other words. We will describe the case where we begin searching the least significant word of k, k_0. First all the data for the correlation test is produced.

Algorithm 1 Generate hypothetical output of the multiplication $x_0 \cdot k_0$ for all possible k_0, where x_0 is known. N is the number of times the algorithm is executed and consequently relates to the number of acquisitions captured.

```
INPUT: $x_0$
OUTPUT: $H_0$
for $0 \leq j < 2^w$ do
    for $1 \leq i \leq N$ do
        $k_0 = j$
        $y_0 = k_0 \cdot x_0$
        $H_0(i,j) = y_0$
    end for
end for
return $H_0$
```

This will produce a $N \times 2^w$ matrix H_0 detailing the hypothetical product of all 2^w possible k_0's and the N known x_0's. Note that the actual multiplication of $k_0 \cdot x_0$ will produce a $2w$ bit product y_0, however only the least significant word of this is required as entry in H_0. The most significant word of y_0 contributes to the subsequent product word y_1.

To identify which is the correct least significant word k_0, the correlation is calculated between each row in H_0 and a discrete time interval in the acquired physical traces where the target operation is being executed. The hypothesis with the highest correlation, is identified as the correct least significant word k_0.

[3] In the case of Scott's implementation, where $w = 32$ it will be a computationally intensive task to extract one word. However it is possible to calculate the partial correlation by just focusing on practical portions of w at a time.

To extract the remaining interior words of k, the attack proceeds similar to algorithm 1. It can be seen from figure 1 that in the $2n$-bit product y, all middle words are influenced by more than one word in k. Therefore k_1 cannot be found unless k_0 is known, and k_2 cannot be found unless k_1 and k_0 is known, etc. Therefore, line 5 in algorithm 1 is replaced by $y_r = (k_l \cdot x_0) + ($ auxiliary words $)$ for $1 \leq l \leq n/w - 2$. For instance $y_1 = (k_1 \cdot x_0) + (k_0 \cdot x_1) + (k_0 \cdot x_0)$ and $y_2 = (k_2 \cdot x_0) + (k_1 \cdot x_1) + (k_0 \cdot x_2) + (k_1 \cdot x_0)$.

The computational cost of such an attack is $l \times 2^w$. Note that we can improve on this slightly when analysing binary field implementations. By observing the fact that the most and least significant words of k can be independently calculated (i.e. no middle words of k influence the multiplied output), we can simultaneously calculate hypotheses for k_0 and $k_{n/w-1}$. Once both of these terms have been extracted, then the search can step inwards, i.e. calculate hypotheses for k_1 and $k_{n/w-2}$ simultaneously, etc. This reduces the cost of extracting k to $\frac{l}{2} \times 2^w$.

Squaring. A variety of fast multiplication algorithms exist for squaring finite field elements. Here we will view squaring in its simplest form which is the multiplication of $x \cdot k$, where $x = k$, and so the attack just described can be applied in the same way to the squaring operation.

Square Root. The square root method is only called on in the Eta pairing implementation, and so only square root calculation over the binary field will be discussed. An efficient method for calculating the square root can be obtained from the observation that $\sqrt{a}$ can be expressed in terms of the square root of the element z [8]. Basically the value a is split into it's odd and even coefficients, as depicted in figure 2, and then the odd portion is multiplied by $\sqrt{z}$ and subsequently added to the even portion. If the irreducible polynomial $f(z)$ is a trinomial or pentanomial, then an efficient formula for calculating $\sqrt{z}$ can be used. For example, $\sqrt{z} = x^{\frac{m+1}{2}} + z^{\frac{n+1}{2}} (\bmod f(z))$ when $f(z)$ is a trinomial and $\sqrt{z} = z^{\frac{m+1}{2}} + z^{\frac{n+1}{2}} + z^{\frac{q+1}{2}} + z^{\frac{r+1}{2}} (\bmod f(z))$ when $f(z)$ is a pentanomial.

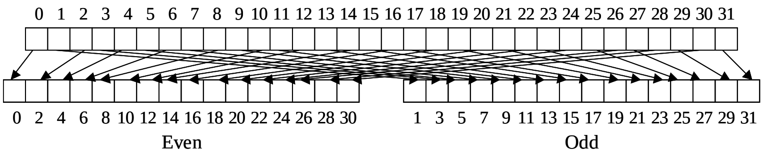

Fig. 2. Square Root of $\mathbb{F}(2^m)$ elements where $w = 32$.

Since the act of square root is a single operand operation, the only way the act of calculating the square root of a secret value can be used in a side channel attack is if, after a prediction about a (and the resulting $\sqrt{a}$) has been made, this value is later used in an operation involving known data. The hypothetical output of this following operation is then used to verify the hypotheses. This is a form of second order attack and will be described later in more detail. For now we will describe how a value entered into the square root function changes in a structural sense, so that by knowing a portion of a we can deterministically calculate what a portion of $\sqrt{a}$ will be.

Let k equal the unknown secret value as before. The $\sqrt{k}$ is calculated as follows: First k is split into left and right chunks. Given that we will be predicting w bits of k at a time, this means that we will know $\frac{w}{2}$ bits of odd, and $\frac{w}{2}$ bits of even. We will denote these half words by k_o and k_e respectively.

In the multiplication step, two $\frac{m}{2}$ bit quantities, $k_o \cdot \sqrt{z}$, will be multiplied to produce an m bit product. Since we know one of the multipliers $\sqrt{z}$ we can predict what a portion of the product will be as before. This portion will be $\frac{w}{2}$ bits. Since there will not be multiple reduction steps (there might not even be one), we can calculate what the final output of the reduction will be.

The final step to the square root operation is the addition (xor) of the even values from before k_e and the product $k_o \cdot \sqrt{z} \pmod{f(z)}$. Since we know the $\frac{w}{2}$ bits of even, and $\frac{w}{2}$ bits of the product, we can calculate $\frac{w}{2}$ of the value $\sqrt{k}$. This can be carried on to the next step in the algorithm, where we can determine how these $\frac{w}{2}$ bits effect the result of the next operation. As before, we have two possible positions from which this attack can commence; the most significant word and the least significant word. Once either of these has been established the middle neighbour words can be searched for.

Even though $\frac{w}{2}$ bit portions are used to verify w bit hypotheses of k, the computational costs of extracting k is still $l \times 2^w$ or $\frac{l}{2} \times 2^w$ if k is attacked simultaneously from both ends.

Reduction. Almost all operations over finite fields are coupled with reduction. The protocol for modulo operations depends on the implementation, i.e. reduction can be performed either concurrently or consecutively. If reduction is performed consecutively, the attacks of the preceding operations can be applied as described. If reduction is performed concurrently, we will have to revise our attack strategy.

In practical implementations generally the chosen moduli is of special form such that it permits fast reduction. For example, over the binary field, irreducible trinomials or pentanomials are preferred.

Straightforward reduction can be performed using the shift and subtract method, where subtract over $\mathbb{F}_{2^m}$ is xor and subtract over $\mathbb{F}_p$ involves borrow bits. $a(z) \equiv b(z) \pmod{f(z)}$ or $a \equiv b \pmod{p}$ basically involves lining the modulus up with the most significant bit of a (or $a(z)$) and subtracting to produce an intermediate value t. The modulus is then repeatedly lined up with intermediate t's until the the bit length (or degree) of t is less than the bit length of p (or degree of $f(z)$).

If repeated reduce is implemented, then it is more difficult to definitively calculate the hypothetical output of interest. For example in the case of multiplication, if we are to predict partial output of $c \equiv a \cdot b \pmod{p}$, we must be able to calculate all of the product $a \cdot b$. Knowing only portions of $a \cdot b$ is not sufficient since the waterfall effect of reduction will lose these portions in a manner unpredictable by the adversary.

The implication of repeated reduce is that intermediate output of the calculation of $c \equiv a \cdot b \pmod{p}$ must now be used for the hypothesis testing. A possible attack might proceed as follows: We will describe this attack for the case where modular multiplication is the operation of interest, this technique may be applied similarly to other methods. Let x equal the known data, and k equal the unknown secret data as before. To extract k_0, we can hypothetically calculate the intermediate output $t \equiv x \cdot k_0$, where t is the intermediate $(m + w)$-bit result. t will then be reduced by p to once again produce a m-bit value. Since the modulus is public, the resultant m-bit value (or even portions of it) will be used to verify the correct k_0. To extract k_1, partial hypothetical output of $(t \equiv x \cdot k_1 \pmod{p}) + (t \equiv x \cdot k_0 \pmod{p})$, where we assume k_0 has been found, is calculated. This process is repeated until no words of k remain unknown.

4 Possible Attacks

So far we have described how individual operations may be attacked using side channels. Now we will put these attacks into context as we describe when and where in the three candidate pairing algorithms these operations are performed and how they can be exploited to extract secret data. For each algorithm we will assess both paths where the first scenario details the situation where Q is secret (*Case 1*) and the second where P is secret (*Case 2*). Note that each case will be addressed relating to the specific implementation details given in [14].

4.1 BKLS

BKLS [3] (algorithm 2, A.1) implements the Tate pairing $e(P, Q)$, where the first input parameter P is a point of order r on the base curve $E(\mathbb{F}_p)$, and the second parameter Q, is a point on the twist $E'(\mathbb{F}_{p^{k/2}})$. $e(P, Q)$ evaluates to an element in the finite field $\mathbb{F}_{p^k}$, where p is a large prime and k is the embedding degree. In the specific implementation described in [14] $k = 2$ and so the points P and Q have coordinates in $\mathbb{F}_p$. This means the coordinates (x, y) will be approximately the same length as the bit length of p.

Case 1. When P is public, since the order r will be a published parameter we can generate all intermediate jP values where $1 \leq j \leq r$ (for the calculation of rP). Q on the other hand will remain static throughout the pairing computation. The target operation in the algorithm involving the secret Q is: $m_j = y_j - \lambda_j(x_Q + x_j) - y_Q i$ where x_j, y_j and λ_j are known, $i = \sqrt{-1}$ and x_Q and y_Q are secret.

Since we only need extract x_Q, we can focus on the operation $\lambda_j(x_Q + x_j)$. As we will know x_j and λ_j, we can employ the attack of the multiplication operation described in 3.2 and extract words of x_Q at a time.

High order SCA can be applied here since we will know x_j, y_j, λ_j for all j, and so can calculate the hypothetical output of $\lambda_j(x_Q + x_j)$ at multiple points.

Case 2. Conversely in the scenario where P is secret and Q is public, our known value remains static through the attack, and the secret parameter is constantly changing. Intuitively this makes this path more difficult to attack, as even if intermediate values of P are recovered, the original P must be extracted requiring point subtraction and knowledge of the number of previous additions that have already been performed. On further inspection, based on our analysis of finite field operations, this avenue of attack is actually not possible for the following reason; In order to make an hypothesis, there must exist an operation where the adversary can deterministically calculate how the input effects the output (even partially). However given a section of x_j it is impossible to deterministically calculate any of the subsequent x_{j+1}, where x_{j+1} is the result from either the doubling or addition of the previous jP, since calculation of x_{j+1} requires knowledge of y_j. So even if we make predictions for the value of x_{j+1} in $\lambda_{j+1}(x_Q + x_{j+1})$, we have no way of determining what λ_{j+1} is and more importantly what the original x_j is.

This natural property of the BKLS algorithm can actually act as a deterrent by enforcing the secret parameter to be entered as the first parameter to the pairing.

4.2 Ate

The Ate pairing $a(P, Q)$ [9] (algorithm 3, A.2) is also computed over the prime field, where P is chosen as a point of order r over the twisted curve $E'(\mathbb{F}_{p^{k/d}})$ with embedding degree k, and Q is chosen over the base field $E(\mathbb{F}_p)$. Here point scalar multiplication (the accumulation of P to rP) is calculated over the extension field $E'(\mathbb{F}_{p^{k/d}})$ (see [9] for details). This means that the underlying finite field arithmetic fundamental to point addition and doubling is performed over $\mathbb{F}_{p^{k/d}}$. However, the coordinates of Q will be over $\mathbb{F}_p$. In the specific implementation described in [14] $k = 4$ and $d = 2$.

Case 1. As with BKLS when P is public and Q is private the target operation is $m_j = i^2 y_Q - i(i^2 y_j/2 + \lambda_j(i^2 x_j/2 + x_Q))$ where elements from the twist jP are untwisted (hence the division by 2) and combined with Q to construct the miller variable $m_j \in \mathbb{F}_{p^k}$. Note in this case $i = \sqrt{-2}$.

Isolating the operation $\lambda_j(i^2 x_j/2 + x_Q)$ involving the secret coordinate x_Q involves addition of an element in $\mathbb{F}_p$ to an element in $\mathbb{F}_{p^2}$ elements and multiplication over $\mathbb{F}_{p^2}$. Note $a + b$ where $a = x_1 + y_1 i \in \mathbb{F}_p$ where $y_1 = 0$, and $b = x_2 + y_2 i \in \mathbb{F}_{p^2}$, simply involves adding the real coefficient of a to the real coefficient of b.

To attack this operation, we once again utilise the observation about multiplication. Even though multiplication is now being performed over $\mathbb{F}_{p^2}$, we can still think in terms of multiplication over $\mathbb{F}_p$. So say $a = x_1 + y_1 i$ and $b = x_2 + y_2 i \in \mathbb{F}_{p^2}$, $a \cdot b$ is simply $(x_1 \cdot x_2 - 2y_1 \cdot y_2) + (x_1 \cdot y_2 + y_1 \cdot x_2)i$ where the internal multiplications are over $\mathbb{F}_p$. Relating to the our attack of $\lambda(i^2 x_j/2+x_Q)$, this means we will be able to calculate the partial output of $x_1 \cdot x_2$ and $y_1 \cdot x_2$ where x_2 relates to the real coefficient of x_j which is added to x_Q. Note that we will be able to calculate all other portions of $a \cdot b$. Once again the attack of the multiplication operation as described in 3.2 can be employed to extract x_Q.

Case 2. The case where P is secret is almost analogous to the BKLS case, and hence appears to be impossible to attack. In Ate an attack would be even more complex since the calculation of jP involving point addition and doubling is over the extension field $\mathbb{F}_{p^2}$ and thus involves more complex arithmetic.

4.3 Eta

The Eta algorithm (algorithm 4, A.3) is quite different from Tate and Ate. The implementation [2] for consideration is applicable to supersingular elliptic curves over the binary field $E(\mathbb{F}_{2^m})$. The Eta pairing $\eta_T(P,Q)$ evaluates to an element in $\mathbb{F}_{2^{km}}$. Both parameters are points on the curve $E(\mathbb{F}_{2^m})$. No distortion map is explicitly used, as the map to the extension field is integrated into the algorithm.

Unlike BKLS and Ate, some preliminary computation takes place outside the loop. Operations on points are also completely avoided. The paths for P and Q are almost symmetric and so attack strategies for both paths are almost equivalent. The only difference is that where the square root of x_P and y_P is calculated, the squaring of x_Q and y_Q is performed.

Case 1. Here there are two main points of attack. The first point of attack is outside the loop; $f \leftarrow u \cdot (x_P + x_Q + 1) + y_P + y_Q + b + 1 + (u + x_Q)s + t$ where the first value in $\mathbb{F}_{2^{km}}$ is constructed. This is enabled by the incorporation of the public elements s and $t \in \mathbb{F}_{2^{km}}$.

Assuming that x_Q is secret and x_P is known, the operation $u \cdot (x_P + x_Q + 1)$ where $u = x_P + 1$, can be focused on. This operation basically involves addition (xor) and multiplication modulo the known irreducible polynomial $f(z)$. Hypothetical partial output of $u \cdot (x_P + x_Q + 1)$ can be calculated by guessing w-bit portions of x_Q.

The second point of attack is the squaring operation, i.e. $x_Q \leftarrow x_Q^2$. This squared value is subsequently used in the next round of the loop in $g \leftarrow u \cdot (x_P + x_Q) + y_P + y_Q + x_P + (u + x_Q)s + t$ where u in this calculation is x_P. By purely calculating what the hypothetical output of a portion of $x_Q \leftarrow x_Q^2$ is, we can analyse how this portion affects subsequent operations.

For example, assuming we have guessed what the least significant word of x_Q is, and calculated the least significant word of the resulting x_Q^2. This means that we can can hypothetically calculate $u \cdot (x_P + x_Q + 1)$ after the first and second round of the for loop and still be able to easily trace back to the original Q.

Case 2. Eta is unique to Tate and Ate in that the two paths in the pairing are almost symmetric and so are equally as vulnerable. Similar to the attack of Q

there are two main points of attack. The first point of attack is again outside the loop, where the operation $u \cdot (x_P + x_Q + 1)$ can be attacked.

The second point of attack is the square root function inside the loop. Given that we can deterministically calculate $\frac{w}{2}$ bits of hypothetical w bits of $\sqrt{x_P}$, this means that we can test how this predicted output effects the output of a number of subsequent operations to perform high order SCA. For example we can calculate the hypothetical output $u \cdot (x_P + x_Q)$ and $x_P + (u + x_Q)s + t$.

The only real obstacle that Eta provides is that regardless of whether the secret takes the either path, it is dynamic. Therefore if the adversary extracts an intermediate secret value they must work back to get the original point. This is in contrast to Ate and Tate which can be attacked at any point in the algorithm.

5 Possible Countermeasures and their Implications

A number of countermeasures have already been anticipated to protect pairings against SCA [13], [12]. Taking advantage of bilinearity, the secret point can simply be blinded. A pairing can be calculated as $e(P, Q) = e(aP, bQ)^{1/ab}$ where a and b are random values or $e(P, Q) = e(P, Q+R)/e(P, R)$ where R is a random point. While these may be effective in deterring SCA since a new random value will be used every time the pairing is called, they are expensive, ultimately requiring point scalar multiplication and calculation of two pairings respectively.

Another more subtle countermeasure proposed by [13] observes that repeated multiplication of the miller variable m in BKLS and Ate (or f in Eta) by a random element in $\mathbb{F}_p$ (or $\mathbb{F}_{2^m}$) will have no effect on the final pairing value since they will be eliminated in the final exponentiation. This is a less expensive deterrent only requiring a field multiplication per iteration of the miller loop.

In order for this countermeasure to be effective, we recommend that the random value must not only be multiplied by the miller variable, but must be multiplied by all intermediate values that make up the miller variable. For example in the case of Tate; $m_j = r \cdot y_j - \lambda_j (r \cdot x_Q + r \cdot x_j) - r \cdot y_Q i$ where $r \in \mathbb{F}_p$. If a new random value is multiplied at every iteration of the loop, the attacks we have presented would no longer be possible.

6 Conclusion & Recommendations

We have presented the first passive differential side channel analysis of the Tate, Ate and Eta pairing. We performed this investigation in an analytical sense, where empirical knowledge of side channel attacks was used to determine where and how operations in the candidate algorithms could be exploited. We presented an attack of the multiplication, square root and reduction operations over finite fields, from a slightly different perspective. Instead of focusing on how these operations could be performed, we simply focus on trying to deterministically calculate partial output based on the structural expansion of basic algorithms.

We assessed the three candidate pairing algorithms based on an attack of the two paths a secret can take. From this we found that although none of the algorithms assessed proved to be resistant to SCA, Tate and Ate if implemented with the secret being stationed in the first parameter could withstand such attacks. Eta however, which is the most efficient algorithm computationally, is open to attack from either path proving that speed may not be the main consideration when choosing the best implementation.

From our findings we recommend two straightforward deterrents to protocol designers implementing pairing based protocols to protect against SCA: 1. If implementing the Tate or Ate pairing, ensure that the secret parameter is positioned in the first parameter (i.e. the secret takes the P path). 2. If implementing any of the pairings (but more specifically Eta), we recommend the adoption of the simple countermeasure proposed by [13] where intermediate finite field multiplication by a random value will successfully mask sensitive data.

References

1. P. Barreto. Pairing based crypto lounge. URL: `http://paginas.terra.com.br/informatica/paulobarreto/pblounge.html`.
2. P. Barreto, S. Galbraith, C. O'hEigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. Cryptology ePrint Archive: Report 2004/375. URL: `http://eprint.iacr.org/2004/375`.
3. P. Barreto, H. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing based cryptosystems. In *Advances in Cryptology - CRYPTO 02*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer Verlag, 2002.
4. P. S. L. M. Barreto, B. Lynn, and M. Scott. On the selection of pairing friendly groups. In *Selected Areas in Cryptography - SAC 2003*, Lecture Notes in Computer Science, Ottawa, Canada, 2003.
5. Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology - CRYPTO 01*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Verlag, 2001.
6. E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In M. Joye and J.J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 04*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29, 2004.
7. I. M. Duursma and H. S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In *Advances in Cryptology - Asiacrypt 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 111–123. Springer Verlag, 2003.
8. K. Fong, D. Hankerson, J. Lopez, and A. Menezes. Field inversion and point halving revisited. CACR Technical Report, CORR 2003-18. URL: `http://www.cacr.math.waterloo.ca/`, 2003.
9. F. Hess, N. Smart, and F. Vercauteren. The eta pairing revisited. Cryptology ePrint Archive: Report 2006/110. URL: `http://eprint.iacr.org/2006/110`.
10. P. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss and other systems. In *Advances in Cryptology - CRYPTO 96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer Verlag, 1996.
11. A.J. Menezes, T. Okamoto, , and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. In *IEEE Transactions on Information Theory*, volume 39, pages 1639–1646, 1993.

12. D. Page and F. Vercauteren. Fault and side-channel attacks on pairing based cryptography. In Luca Breveglieri and Israel Koren, editors, *Fault Detection and Tolerance in Cryptography - FDTC 05*, pages 56–66, 2005.
13. M. Scott. Computing the tate pairing. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304, 2005.
14. M. Scott, N. Costigan, and W. Abdulwahab. Implementing cryptographic pairings on smart cards. Cryptology ePrint Archive: Report 2006/144, URL: `http://eprint.iacr.org/2006/144`.
15. C. Walter. Simple power analysis of unified code for ecc double and add. In M. Joye and J. J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 04*, volume 3156 of *Lecture Notes in Computer Science*, pages 191–204, 2004.

A Practical Pairing Implementations

A.1 The Tate Pairing

Algorithm 2 Computation of $e(P, Q)$ on $E(\mathbb{F}_p) : y^2 = x^3 + Ax + B$, where P is a point of prime order r on $E(\mathbb{F}_p)$ and Q is a point on the twisted curve $E'(\mathbb{F}_p)$

```
INPUT: $P = (x_P, y_P), Q = (x_Q, y_Q)$
OUTPUT: $m \in \mathbb{F}_p$
$m = 1$
$x_A, y_A \leftarrow x_P, y_P$
$n = r - 1$
for $i \leftarrow \lfloor \lg(r) \rfloor - 2$ to 0 do
    if $n_i = 1$ then
        $(\lambda, x_T, y_T) \leftarrow$ double$(A, A)$
        $g = y_A - \lambda(x_Q + x_A) - i.y_Q$
        $x_A, y_A \leftarrow x_T, y_T$
        $m = m^2 \cdot g$
        $(\lambda, x_T, y_T) \leftarrow$ add$(T, P)$
        $g = y_A - \lambda(x_Q + x_A) - i.y_Q$
        $x_A, y_A \leftarrow x_T, y_T$
        $m = m \cdot g$
    else
        $(\lambda, x_T, y_T) \leftarrow$ double$(A, A)$
        $g = y_A - \lambda(x_Q + x_A) - i.y_Q$
        $x_A, y_A \leftarrow x_T, y_T$
        $m = m^2 \cdot g$
    end if
end for
$m = \frac{\bar{m}}{m}$
return $V_{p+1/r}(m_r)$
```

A.2 The Ate Pairing

Algorithm 3 Computation of $a(P, Q)$ on $E(\mathbb{F}_p) : y^2 = x^3 + Ax + B$, where P is a point of prime order r on the twisted curve $E'(\mathbb{F}_{p^2})$ and Q is a point on the base curve $E(\mathbb{F}_p)$

```
INPUT: $P = (x_P, y_P), Q = (x_Q, y_Q)$
OUTPUT: $m \in \mathbb{F}_{p^2}$
$m = 1$
$x_A, y_A \leftarrow x_P, y_P$
$n = t - 1$
for $i \leftarrow \lfloor \lg(n) \rfloor - 2$ to $0$ do
    if $n_i = 1$ then
        $(\lambda, x_T, y_T) \leftarrow \text{double}(A, A)$
        $g = i^2 y_Q - i(i^2 y_A/2 + \lambda(i^2 x_A/2 + x_Q))$
        $x_A, y_A \leftarrow x_T, y_T$
        $m = m^2 \cdot g$
        $(\lambda, x_T, y_T) \leftarrow \text{add}(A, P)$
        $g = i^2 y_Q - i(i^2 y_A/2 + \lambda(i^2 x_A/2 + x_Q))$
        $x_A, y_A \leftarrow x_T, y_T$
        $m = m \cdot g$
    else
        $(\lambda, x_T, y_T) \leftarrow \text{double}(A, A)$
        $g = i^2 y_Q - i(i^2 y_A/2 + \lambda(i^2 x_A/2 + x_Q))$
        $x_A, y_A \leftarrow x_T, y_T$
        $m = m^2 \cdot g$
    end if
end for
$m = \frac{\bar{m}}{m}$
return $V_{p^2+1/r}(m_r)$
```

A.3 The Eta Pairing

Algorithm 4 Computation of $\eta_T(P,Q)$ on $E(\mathbb{F}_{2^m}) : y^2 + y = x^3 + x + b$

```
INPUT: $P = (x_P, y_P), Q = (x_Q, y_Q)$
OUTPUT: $f \in \mathbb{F}_{2^{km}}$
$u \leftarrow x_P + 1$
$f \leftarrow u \cdot (x_P + x_Q + 1) + y_P + y_Q + b + 1 + (u + x_Q)s + t$
for $i \leftarrow 1$ to $(m+1)/2$ do
    $u \leftarrow x_P, x_P \leftarrow \sqrt{x_P}, y_P \leftarrow \sqrt{y_P}$
    $g \leftarrow u \cdot (x_P + x_Q) + y_P + y_Q + x_P + (u + x_Q)s + t$
    $f \leftarrow f \cdot g$
    $x_Q \leftarrow x_Q^2, y_Q \leftarrow y_Q^2$
end for
return $f^{(2^{2m}-1)(2^m-2^{(m+1)/2}+1)(2^{(m+1/2)}+1)}$
```