On XTR and Side-Channel Analysis

Daniel Page and Martijn Stam

Dept. Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, United Kingdom
{page, stam}@cs.bris.ac.uk

Abstract. Over the past few years, there has been a large volume of work on both attacking elliptic curve cryptosystems (ECC) using side-channel analysis and the development of related defence methods. Lenstra and Verheul recently introduced XTR, a cryptosystem that can compete with ECC in terms of processing and bandwidth requirements. These properties make XTR ideal for use on smart-cards, the devices that suffer most from vulnerability to side-channel attack. However, there are relatively few papers investigating the side-channel security of XTR and although some ECC techniques can be re-used, there are also notable differences. We aim to fill this gap in the literature. We present the first known SPA attack against XTR double exponentiation and two defence methods against such an attack. We also investigate methods of defending XTR against DPA attack.

Keywords: XTR, LUC, finite field, power analysis, side channel attack.

1 Introduction

In 2000 Lenstra and Verheul introduced XTR [16], a cryptosystem using (a sub-group of) the multiplicative group of $\mathbf{F}_{p^6}$ but with a compact representation based on the trace over $\mathbf{F}_{p^2}$ that allows highly efficient arithmetic. Given the current state of affairs in breaking the discrete logarithm problems over either finite fields or elliptic curves, XTR can compete with elliptic curve cryptography (ECC) in terms of both speed and bandwidth. This makes XTR suitable for deployment on similar sorts of constrained devices as ECC, where computational power and storage capacity are both very limited.

Side-channel analysis [13, 12] moves the art of cryptanalysis from the mathematical domain into the practical domain of implementation. Such attacks are based on the assumption that one can observe an algorithm being executed on a processing device and infer details about the internal state of computation from the features that occur. Although timing attacks are the classic example of this technique, in the context of ECC, power analysis is a popular method of monitoring the activity of the processor since smart-cards which commonly implement ECC cryptography draw power from an external and hence accessible source. Using simple equipment, an attacker can collect power profiles from a target smart-card and break the security of a system if the implementation does not include defences mechanisms. Techniques such as electromagnetic (EM) radiation based side-channels are growing in popularity but in this paper we limit our scope to attacks using simple (SPA) and differential (DPA) power analysis.

H. Handschuh and A. Hasan (Eds.): SAC 2004, LNCS 3357, pp. 54–68, 2005.

Due to wide spread use on devices such as smart-cards, there has been a large volume of work on side-channel attacks against ECC and also on defending against these attacks. Conversely, there are relatively few, if any papers addressing the side-channel security of XTR. Although one can borrow some defence methods from ECC and successfully apply them to XTR, there are also notable differences. For instance, ECC techniques based on curve isomorphisms or curve isogenies do not seem to apply, while others, such as exponent splitting, do apply but may not be the best engineering solution. We aim to fill this gap in the literature by investigating possible weaknesses of XTR against side-channel attacks and evaluating different defence techniques.

The paper is organised as follows. We use Section 2 to briefly recap on features of side-channel attacks that relate to subsequent discussion before presenting the XTR cryptosystem in Section 3. In Section 4 we describe an SPA attack against XTR, including some experimental results. We then describe methods for defending XTR against SPA and DPA attack in Sections 5 and 6 respectively.

2 Side-Channel Analysis

At the heart of power analysis is the concept of power traces. When a smart-card performs an operation, it consumes power in proportion to a number of factors such as which computational units are active and the Hamming weights of data involved. The power trace is a profile of how much power is being consumed at a given point in time during execution. Since the power supply on smart-cards is part of the reader terminal, it is both easily controllable and inspected by the attacker unless masking devices are used [23].

Attackers can harness such traces in two main ways. In an SPA attack the attacker executes the algorithm once and gleans information from the types of operation that are performed, focusing mainly on control flow. The attacker uses an operation trace that is constructed by spotting known operation profiles in the power trace. For example, suppose it is possible to differentiate between when a squaring and when a more general group multiplication is performed during an exponentiation using some secret exponent. Using a shorthand of A and D to represent squaring and multiplication, or addition and doubling in the additive case of elliptic curves, this leads to a sequence we call an operation trace. Let us consider the example of a left-to-right binary exponentiation algorithm used to perform ECC point multiplication. From a single trace, the secret exponent can be read immediately if the addition and doubling operations are distinguishable. For example, the operation trace DADD corresponds to an exponent of $(((1 \cdot 2)+1) \cdot 2) \cdot 2 = 12_{10} = 1100_2$. By simply noting where an addition follows a doubling in the operation trace, the attacker can directly read the secret exponent and hence break the presumed security without needing to resort to solving a discrete logarithm problem.

DPA is a more powerful technique that may break systems that are secure against SPA attack. It also filters out noise from a power trace. By running the algorithm many times and focusing mainly on the value of data items used in each execution, the attacker applies statistical techniques to correlate the secret information with features in the collected profiles. To perform these statistical methods on an exponentiation algorithm for example, large numbers of power traces using the same exponent are required. If the

exponentiation is suitably randomised, the samples will be too uncorrelated to provide useful information. However, if no randomised defence measures are in place DPA can be used to break the exponentiation, potentially using a small number of traces. Recent advances in DPA type analysis have seen address calculation as well as actual data values being examined in order to break table or window based exponentiation algorithms [6].

3 XTR

3.1 Description of XTR

XTR [16] is based on the assumed hardness of the discrete logarithm problem in the multiplicative group of $\mathbf{F}_{p^6}$. Using Pohlig-Hellman-like techniques, Lenstra [15] argues that the hardness of the DLP in $\mathbf{F}_{p^6}^*$ must in fact reside in the part that does not lie in any proper subfield. This is called the cyclotomic subgroup and denoted G_{p^2-p+1} since it has order p^2-p+1. Moreover a subgroup G_q of large prime order q is chosen in this cyclotomic subgroup to prevent further Pohlig-Hellman attacks on the smooth part of p^2-p+1.

In XTR elements of G_{p^2-p+1} are represented by their trace over $\mathbf{F}_{p^2}$. For $g \in \mathbf{F}_{p^6}^*$ the trace $\mathrm{Tr}(g)$ over $\mathbf{F}_{p^2}$ is defined as the sum of the conjugates over $\mathbf{F}_{p^2}$ of g

$$\mathrm{Tr}(g) = g + g^{p^2} + g^{p^4} \in \mathbf{F}_{p^2} .$$

Because the order of g divides p^6-1, the trace over $\mathbf{F}_{p^2}$ of g equals the trace of the conjugates over $\mathbf{F}_{p^2}$ of g, hence XTR makes no distinction between an element g and its conjugates over $\mathbf{F}_{p^2}$. If $g \in \mathsf{G}_{p^2-p+1}$ then the element g, or one of its conjugates, can be retrieved from $c = \mathrm{Tr}(g)$ by determining a root of the cubic polynomial $X^3 - cX^2 + c^pX - 1$. Lenstra and Verheul also show that any given $c \in \mathbf{F}_{p^2}$ is the trace of some element in G_{p^2-p+1} if the aforementioned cubic polynomial is irreducible over $\mathbf{F}_{p^2}$. This tightly and provably links the discrete logarithm problem for XTR to that of the corresponding subgroup $\mathsf{G}_q \in \mathbf{F}_{p^6}^*$, and heuristically to that in the entire field $\mathbf{F}_{p^6}^*$.

Henceforth, let p and q be primes with q dividing p^2-p+1. Suggested lengths to provide adequate levels of security are $k = \lg q \approx 160$ and $l = \lg p \approx 170$. Also, let g be a generator of G_q and let $c = \mathrm{Tr}(g)$. Lenstra and Verheul [17] show how p, q and c can be found quickly. In particular, there is no need to find an explicit representation of $g \in \mathbf{F}_{p^6}$.

Throughout this article, c_n denotes $\mathrm{Tr}(g^n) \in \mathbf{F}_{p^2}$, for some p and g of order q dividing p^2-p+1 as above. Note that $c_0 = 3$ and $c_1 = c$. Efficient computation of c_n given p, q and c depends on the recurrence relation

$$c_{n+m} = c_n c_m - c_n^p c_{n-m} + c_{n-2m} , \tag{1}$$

which simplifies for $n = m$ to

$$c_{2n} = c_n^2 - 2c_n^p .$$

Lenstra and Verheul note that the simplification of c_{2n} allows for a considerable speedup of its computation. This speedup will be responsible for providing different traces for A corresponding to c_{n+m} and D corresponding to c_{2n}. We will occasionally abuse notation and use A and D to denote the cost of the operations as well, where $\mathsf{A} \approx 2\mathsf{D}$.

3.2 Binary Exponentiation

Algorithm 1 was introduced by Lenstra and Verheul alongside XTR. They already noted that regardless of the bit being read in Step 3, two doublings and one addition are being performed. Hence the operation trace contains no Shannon information on the exponent apart from its length, provided that for both cases in Step 3 the same order is used [5] (cf. [21, 10] for similar subtleties for the binary Lucas left-to-right algorithm [14]). The algorithm can be slightly adapted to output the triple (c_{n-1}, c_n, c_{n+1}), but care has to be taken (e.g., by adding a dummy operation) that the least significant bit of n is not leaked. The runtime of the algorithm is basically $\mathsf{A} + 2\mathsf{D}$ per exponent bit.

Algorithm 1: Left-to-Right Binary Exponentiation.

On input n and c this algorithm returns c_n or the triple $(c_{2\lfloor \frac{n}{2} \rfloor}, c_{2\lfloor \frac{n}{2} \rfloor+1}, c_{2\lfloor \frac{n}{2} \rfloor+2})$. It maintains as invariant

$$0 \le j < k, \quad a = 1 + \sum_{i=j+1}^{k-1} n_i 2^{i-j}, \quad S_a = (c_{a-1}, c_a, c_{a+1}) \; .$$

(a is carried along for expository purposes only).

1. [Initialization] Set $j \leftarrow k-1, a \leftarrow 1$ as well as $S_a \leftarrow (3, c, c_2)$.
2. [Finished?] If $j = 0$ terminate with output c_{a-1} if $n_0 = 0$ and with output c_a otherwise.
3. [Decrease j] If $n_j = 0$, set $S_a \leftarrow (c_{2(a-1)}, c_{(a-1)+a)}, c_{2a})$ and set $a \leftarrow 2a - 1$; else ($n_j = 1$) set $S_a \leftarrow (c_{2a}, c_{(a+1)+a}, c_{2(a+1)})$ and set $a \leftarrow 2a + 1$. Decrease j by one and go back to the previous step.

3.3 Euclidean Exponentiation

A faster exponentiation routine for XTR was described by Stam and Lenstra [24]. It is based on an adaptation of a Euclidean algorithm by Montgomery [20] using Lucas chains. For ease of notation, we will momentarily use ordinary exponentiation in our description instead of the third order XTR recurrence. The algorithm is based on the invariant relation $A^d B^e = g^n h^m$, where the left-hand side are variables in the algorithm and the right-hand side is the exponentiation to be performed. It is easy to initialise the variables by $(d, e) = (n, m)$ and $(A, B) = (g, h)$. The algorithm then reduces d and e in a way depending on the current values of d and e, updating A and B applying multiplication and squaring operations as execution progresses. Finally, when $e = 0$, the algorithm outputs A and the unprocessed part of the exponent d.

Algorithm 2: Euclidean Double Exponentiation.

Given bases $c_\kappa, c_\lambda, c_{\kappa-\lambda}$, and $c_{\kappa-2\lambda}$ and positive exponents n, m, this algorithm outputs $u = \gcd(n, m)$ and $c_{(n\kappa+m\lambda)/u}$. It uses invariant

$$\begin{aligned} d > 0, \quad e \ge 0, \quad ad + be = n\kappa + m\lambda, \quad \gcd(d, e) = \gcd(n, m), \\ A = c_a, \quad B = c_b, \quad C = c_{a-b}, \quad D = c_{a-2b} \; . \end{aligned}$$

(a and b are carried along for expository purposes only).

Table 1. Two tables that describe the operation of Euclidean exponentiation

Type	Condition	Substitution	Trace
Substitutions if $d \geq e$			
X1	$d \leq 4e$	$(e, d-e)$	A
X2	d even	$(d/2, e)$	ADD
X3	d, e both odd	$((d-e)/2, e)$	ADD
X4	e even	$(e/2, d)$	DD
Substitutions if $e \geq d$			
X5	$e \leq 4d$	$(d, e-d)$	A
X6	e even	$(e/2, d)$	DD
X7	d, e both odd	$((e-d)/2, d)$	ADD
X8	d even	$(d/2, e)$	ADD

(a) A table describing the operations performed by the Euclidean exponentiation algorithm and the SPA trace produced as a result. Note the ordered constraints on values of d and e that dictate the algorithm execution

Type	Condition	Substitution	Trace
Substitutions if $d \geq e$			
X1	$0 \leq e \leq 3d$	$(d+e, d)$	A
X2	$e < \frac{1}{2}d$ and e odd	$(2d, e)$	ADD
X3	$e < \frac{2}{3}d$ and e odd	$(2d+e, e)$	ADD
X4	$e > 8d$ and e odd	$(e, 2d)$	DD
Substitutions if $e \geq d$			
X5	$0 \leq e \leq 3d$	$(d, d+e)$	A
X6	$e < \frac{1}{2}d$ and e odd	$(e, 2d)$	DD
X7	$e < \frac{2}{3}d$ and e odd	$(e, 2d+e)$	ADD
X8	$e > 8d$ and e odd	$(2d, e)$	ADD

(b) A reverse engineering of Euclidean exponentiation which shows the operations used if the algorithm is run in reverse, starting with the pair $(1, 1)$ and moving towards the initial exponent values

1. [Initialization] Let $a = \kappa$, $b = \lambda$, and set $d \leftarrow n$, $e \leftarrow m$, $A \leftarrow c_\kappa$, $B \leftarrow c_\lambda$, $C \leftarrow c_{\kappa-\lambda}$, and $D \leftarrow c_{\kappa-2\lambda}$.
2. [Both even?] Set $f_2 \leftarrow 0$. As long as d and e are both even, replace (d, e) by $(d/2, e/2)$ and f_2 by $f_2 + 1$.
3. [Finished?] If $e = 0$ terminate with output $d2^{f_2}$ and A.
4. [Decrease (d, e)] Substitute (d, e) using the first applicable rule in Table 1a. Update a, b, A, B, C and D accordingly, in order to maintain the invariant. Go back to the previous step.

Table 1a contains the rules that are used to decrease the pair (d, e), including the operations-trace that is left by any particular rule. We have left out the optional ternary rules described by Stam and Lenstra, because the speedup achieved is based upon a clearly recognisable tripling operation and we believe the resulting increase of side-channel leakage is likely to outweigh the gain in speed. On average, the algorithm takes $1.39\mathsf{A} + 1.1\mathsf{D}$ per exponent bit.

A single exponentiation routine can be based on the double exponentiation routine by writing $g^n = g^{n-r}g^r$ where r can be chosen arbitrarily. A sensible way to do this is described in Algorithm 3, where we have thrown out some speedups proposed by Montgomery based on the factorisation of the exponent, since we deem them to vulnerable to attack. If r is chosen as in Step 3 below coprime to the exponent n, the call to Algorithm 2 will result in approximately $0.72 \lg n$ applications of X1 after which a random $(\lg n)/2$-bit random double exponentiation follows. Overall, a single exponentiation will take $1.41\mathsf{A} + 0.55\mathsf{D}$.

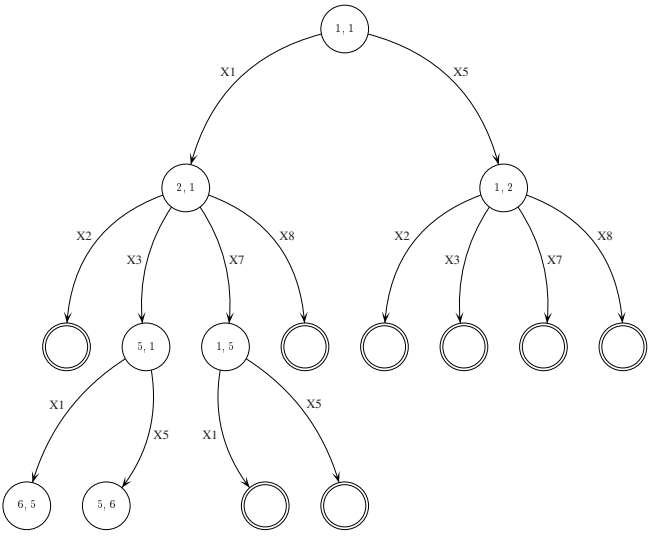

Fig. 1. A parse tree generated by the SPA trace AADDA corresponding to the initial pair $(5, 6)$. Note that the edges are marked with operations that are applied by the exponentiation algorithm to move between the nodes that contain intermediate (d, e) pairs

Algorithm 3: Single Exponentiation.

Given a base c and an exponent n, this algorithm computes c_n.

1. $[d = 1?]$ If $d = 1$, the algorithm terminates with output A.
2. [Initialise new GCD calculation] Set $r \leftarrow \lceil \frac{d}{\phi} \rceil$ and set $(d, e) \leftarrow (r, d - r)$.
3. [Compute "A^p"] Run Algorithm 2 on input $(c, c, 3, c^p)$ and exponents (d, e). Let the output be u and $\tilde{A}$. Set $d \leftarrow u$ and $A \leftarrow \tilde{A}$. Go back to step 1.

4 SPA Attack

Since the single exponentiation routine starts off with a fairly predictable part after which a random double exponentiation takes part, we direct our efforts into analysing the double exponentiation (with coprime exponents). Curiously, attacking double exponentiation algorithms has only received limited attention in the literature so far, although they are essential if precomputation is used and of potential benefit if exponent splitting is used to defend against SPA and DPA.

From the view of an adversary, the Euclidean algorithm starts with unknown values for (d, e) but ends with the known pair $(1, 0)$. In their analysis, the adversary can attempt to run the algorithm backwards by trying to predict which step led to a certain pair (d, e). For instance, $(1, 0)$ is certain to be preceeded by $(1, 1)$ which will be our starting point henceforth. We can view the second-guessing of which operations were performed as movement within a tree of choices where nodes represent (d, e) pairs and edges represent

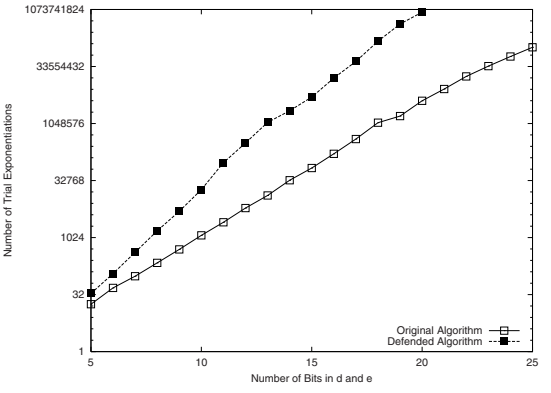

Fig. 2. A graph showing the results of implementing an SPA guided search attack against the original and defended versions of Euclidean exponentiation

the operations performed. As such, there will be a single path through the tree from the initial (d, e) pair to the root of the tree which will be the pair $(1, 1)$. Each leaf in the tree corresponds to a potential exponent. The adversary needs to check the exponents until he has found the correct one, so in order to improve the efficiency of the attack we want to minimise the number of leaves.

By using the collected SPA trace, we can prune the search tree considerably. For example, if we notice the sequence DD we know that neither operation X1 nor X5 will have generated it. We can therefore refine the full tree into what one might call a parse tree derived from the operations observed in the SPA trace. That is, we include only those paths that match what has been observed and prune the rest. Figure 1 shows an example tree for the parse A, ADD, A of a trace AADDA corresponding to the initial pair $(5, 6)$.

As well as using the SPA trace to eliminate impossible paths, we can work from the known root of the tree towards the leaves and use constraints on (d, e) pairs to eliminate operations that could not have occurred. Table 1b illustrates such transformations and constraints, which are essentially the postconditions that follow from the preconditions and the transformations in Table 1a. One can see that rule X6 is almost superfluous: none of the eight steps can actually lead to its precondition, so it can be called at most once from the start in a double exponentiation routine. Also note that X4 is always followed by either X7 or X8 and that the observation of ADDDD implies the last two D's originate from X4. As a more concrete example of pruning based on Table 1b, consider that we arrive at pair $(1, 2)$ and see the sequence ADD. We would ordinarily need to search the paths for operations X2, X3, X7 and X8 but since we know that e must be odd for any of these operations to have occurred, we can eliminate the impossible paths from our search. In Figure 1, empty nodes with double rings denote nodes that are pruned using this method. Using a brute force search of the key space, one might perform around 2^6 trial exponentiations in the worst case since the pair $(5, 6)$ represents six unknown bits. Using the SPA trace to prune this search, we perform only two trials due to the number

of false or impossible paths eliminated. However, with such small values of d and e, it is not clear how the method might perform in the face of recommended key sizes.

In order to empirically investigate this technique with larger parameters, we implemented the search mechanism and ran a large number of experiments. Ambiguity of parses, such as the choice of $\mathsf{A}, \mathsf{ADD}, \mathsf{A}$ over $\mathsf{A}, \mathsf{A}, \mathsf{DD}, \mathsf{A}$ from the trace AADDA is managed inline with our search so that accumulated work is not wasted. We also added two search heuristics to guide the selection of valid paths that rely on us first generating a large number of random keys and constructing some tables from the probability of generating a given trace. These tables allow us to order our selection of paths through the tree based on the probability of their occurrence in our analysis phase. Specifically, we maintain a path history from the root and use it to guide subsequent choices. For example, at node $(2, 1)$ we might descend down edge $\mathrm{X3}$ before edge $\mathrm{X7}$ since that has shown to be a more probable path to target nodes.

From the graph of results in Figure 2 we can see that the average number of trial exponentiations is roughly given by $2^{1.09 \cdot k}$ where k is the length of the exponents. Put more simply, this implies that if an attacker can capture an SPA trace the system is only as secure as where the exponents are about half as long. For example, given 80-bit values of d and e, the attack can recover their value in the same time a naive exhaustive search would take if they were around each around 43-bits. Although not totally devastating for large keys, care must be taken to guard against smaller values of d and e being vulnerable to such an attack. This is especially true since it might be attractive to use such smaller values in constrained environments so as to balance performance against available resources.

For a single exponentiation of a k-bit exponent, this implies the attack succeeds after roughly $2^{0.55 \cdot k}$ trials. Although this is still more than using the Pollard rho method, it is worryingly close. Moreover, it is not unthinkable that in significantly less time part of the exponent can be recovered.

5 SPA Countermeasures

5.1 Adapting the Euclidean Algorithm

Precomputing r. For a fixed exponent, it is possible to precompute r in Algorithm 3 and store this value alongside the exponent. A clever choice of r might increase the number of A's in the operation trace without significantly increasing the total costs. For instance, Montgomery conjectured the existence of an r with only 1,2 and 3 in the continued fraction of n/r. Precomputation of such an r would yield an SPA-resistant algorithm since it would only call rules $\mathrm{X1}$ and $\mathrm{X5}$. We did not attempt to find such a conjectured r, but instead opted for generating a large number of r's coprime to the fixed exponent and picked the one that used the least number of steps in the Euclidean algorithm without binary steps. We formed our r's by dividing n by a deviation of the golden ratio ϕ. The golden ratio has as continued fraction an infinite number of ones. Our deviations also allow 2's and some 3's in the initial part of the continued fraction. For 160-bit exponents we examined about hundred thousand possible values of r per exponent. On average, the cost of the exponentiation, after this precomputation, is about $1.7\mathsf{A}$ per exponent bit.

As a result, for fixed exponents we achieve a SPA-resistant algorithm for hardly any extra work, not taking into account the precomputation. Working harder during precomputation might result in a faster routine, potentially even faster than the unprotected algorithm (this was observed for some smaller bitlengths). Essentially, a small addition chain of a special type is being sought. In principal, similar techniques could apply to ordinary exponentiation with a fixed exponent as well, although we are not aware of any particular family of small addition chains that is resilient against SPA attacks yet having a short certificate (although storing the description of the entire chain might be an option).

This SPA countermeasure excludes several DPA countermeasures from Section 6. This is unfortunate since the SPA countermeasure only makes sense if the same exponent is reused over and over. For random exponents DPA is not an issue, but this particular SPA countermeasure does not work. For random, fresh exponents one could consider constructing the exponent by means of a randomly generated continued fraction with sufficiently small entries. We did not explore this avenue, but it undoubtedly will skew the probability distribution of the exponents in some way.

Choice Randomisation. Making the operation choices non-deterministic is a fairly simple task that to some extent follows work in ECC where addition chains are traversed in random patterns [22]. In order to implement this measure, we propose to relax the bounding conditions that dictate when an operation is selected and then introduce some random factor into the decision making process. For example, one might set the conditions for X1 and X4 to be $d < 2e$ and $2e < d < 4e$ but only execute the later with probability of a half. Although this acts to randomise calculation and help to thwart DPA attack, clearly some investigating of how this might effect security in the context of SPA is required.

We implemented the defense strategy and mounted our search attack from Section 4 against it with results again shown in Figure 2. The introduction of non-determinism clearly hampers the search with the number of trial exponentiations now given by about $2^{1.52 \cdot k}$. This improvement in security is fairly inexpensive; a double exponentiation now takes on average about $1.29\mathrm{A} + 1.53\mathrm{D}$ per exponent bit corresponding to a 3.2% overhead for a single exponentiation. Given the low cost of the method, one might implement further similar transformations that introduce more non-determinism should higher degrees of resistance to SPA attack be required.

For ordinary binary algorithms, using randomization to prevent SPA can actually weaken the algorithm if a small number of traces using the same exponent are known due to attacks based on hidden Markov models [11]. The Euclidean algorithm does not seem to be susceptible to this kind of attack, since a different random choice at one point makes it extremely unlikely that further on in the algorithm the same state is going to occur.

5.2 SPA-Resistant Trace Operations

In ECC the doubling and addition operations are mathematically quite different, so distinguishable traces for them are hard to avoid. Even so, there are some uniform solutions, although they have additional problems [18, 8, 1, 7]. In contrast, for XTR obtaining indis-

tinguishable operations is really only a matter of choice beween accelerated and normal arithmetic. An obvious way to hinder SPA is therefore to not bother speeding things up and hope that the power traces of a real c_{n+m} and a less general c_{n+n} will be indistinguishable. However, there is no need to compromise since we can have SPA-resistant trace operations that largely preserve the speedup and also make the trace of two applications of c_{2n} indistinguishable to one application of c_{n+m}. Similar techniques have been described to do this in the context of ECC [26, 4].

For efficiency purposes, Lenstra and Verheul propose to pick $p \equiv 2 \bmod 3$ and use $\{\zeta_3, \zeta_3^2\}$ as a basis for $\mathbf{F}_{p^2}$, where ζ_3 is a root of the third cyclotomic polynomial $x^2 + x + 1$. Later it was noted by Stam and Lenstra [25] that if $p \equiv 3 \bmod 4$, then one can use $\{1, \zeta_4\}$ without significant loss of efficiency, where ζ_4 is a root of the fourth cyclotomic polynomial x^2+1. Using these bases, the costs of applying c_{2n} are dominated by two modular multiplications, whereas performing c_{n+m} is twice as expensive, costing four modular multiplications and several additions (cf. [16–Lemma 2.1.1], [24–Lemma 2.2], [2–Case $p = 3$], and [2–Case $p^k = 4$]).

The 4:2 ratio in modular multiplications leads us to attempt to try and make two applications of c_{2n} indistinguishable from one application of c_{n+m}. In the Appendix we describe how to do this based on the field representation given by Lenstra and Verheul if p is congruent to 2 modulo 3; and then for the alternative representation if p is congruent to 3 modulo 4. We assume that modular addition and modular subtraction are indistinguishable. We also assume that in the base field computing $a + (a + 2)$, or more generally $a + (b + c)$ where c is very small, is indistinguishable from computing $a + b$. These are all modular additions and if some carry fiddling is allowed, the assumption should hold. However, using Montgomery arithmetic might hamper things, in that 2 has to be represented by the full fledged $2R \bmod p$ where R is the Montgomery radix. In this case an Add' will be equivalent to two ordinary modular additions and some extra dummy additions have to be added to c_{n+m} routine to make up for this.

6 DPA Countermeasures

The algorithms presented in Section 3 are deterministic in their use of data and are therefore likely to be vulnerable against DPA type attacks. One way to prevent DPA attacks is to inject randomness into both the behaviour of the algorithm and the data items it operates on, so that correlation between executions is more difficult. Such techniques have been well studied within the context of ECC and here we describe their applicability for use with exponentiation routines in XTR.

6.1 Binary Exponentiation

Exponent Randomisation. One of the easiest ways to randomise an exponentiation is to add a random multiple of the group order to the original exponent and run the algorithm using the result [12, 3]. That is, one picks a random r and computes c_{n+rq} before recovering the required result at the end, a technique sometimes called exponent blinding. This method is clearly applicable to XTR but suffers from the same significant performance problems as when used in ECC and is therefore not ideal. For example, for

a group of 160 bits in size if r is chosen to be 20 bits long, the exponentiation is slowed down by around 12.5%.

Exponent Splitting. A related technique to randomising the exponent is splitting it into two parts by picking a random $r \in \mathbf{Z}_q$ and rewriting the exponent as $(n-r)+r$. The values $(n-r)$ and r are then used to compute two single exponentiations that are multiplied together to reconstruct the required result. This final multiplication is troublesome in XTR because although c_{n-r} and c_r are known, the required differences c_{n-2r} and c_{n-3r} are typically not and explicitly computing c_{n-2r} and c_{n-3r} alongside other calculations effectively quadruples the cost of a conventional exponentiation. One can bypass this problem by computing g^{n-r} and g^r in $\mathbf{F}_{p^6}$, multiply these two values and trace back the result, but this would be outside the realm of XTR and we do not expect it to be particularly efficient. The final possibility is to use a double exponentiation routine that directly leads to c_n. Stam and Lenstra [24] present a double exponentiation version of the binary algorithm that is suitable for this purpose. In essence, the exponent is rewritten as

$$n \equiv \frac{r}{2^k}(2^k + \frac{(n-r)2^k}{r}) \bmod q,$$

after which the binary algorithm is called twice on exponents $\frac{(n-r)2^k}{r} \bmod q$ and $\frac{r}{2^k} \bmod q$. Hence this countermeasure doubles the execution time required for an exponentiation and can hardly be recommended.

Note that the problems just described also complicates the use of techniques such as meet-in-the-middle [19], where one picks a random point in the binary expansion of an exponent before using a left-to-right algorithm to compute one half and a right-to-left algorithm for the other half.

Base Randomisation. Another method is to randomise the base value by operating modulo some random multiple of the real modulus p and converting back only at the end of an exponentiation. This method applies to XTR although the costs are harder to pin down, since it requires comparing modular multiplications of different length exponents.

Field Randomisation. Han et al. [5] suggest using a field randomisation as a countermeasure, mimicking an ECC countermeasure by Joye and Tymen [9]. Although theoretically XTR works for any field representation, it is essential for its efficiency that the Frobenius endomorphism can be computed almost for free without aversely affecting the costs of an ordinary field multiplication. This combined requirement severely limits the number of field representations that can be used, so using randomised field representations as a countermeasure against DPA will probably be expensive.

Order Randomisation. One could consider randomising the order in which operations are called within Step 3 of the algorithm, for instance using the sequence DDA with probability a half and ADD with probability a half. Although this does randomise the traces in some sense, it does not alter any of the intermediate triples S_a and typically it is leakage such as the Hamming weight of these data items that allows successful DPA attacks. As a consequence, we doubt this countermeasure is suitable to defend against

DPA and note that it has the additional drawback of complicating parallel implementation of the computation required in Step 3.

6.2 Euclidean Exponentiation

Most countermeasures discussed for binary exponentiation apply in equal measure to Euclidean exponentiation (where exponent randomisation takes place before calling Algorithm 3). The main difference is that exponent splitting is actually a very attractive choice now, since it is a necessity part of the Euclidean exponentiation algorithm anyway. If we replace ϕ in Step 2 of Algorithm 3 by a deviation of the golden ratio where the first 20 values in the continued fraction are independently changed to two with probability a half, the runtime of the single exponentiation increases by less than one percent. We believe this to be an efficient and adequate DPA countermeasure.

7 Conclusions

Previous work has shown that the public key cryptosystem XTR can be a high performance alternative to ECC and is especially suited for implementation on constrained, mobile devices such as smart-cards. In this paper, we have presented an analysis of several security issues that are important if XTR is to be used on such devices. By examining issues of side-channel security relating to the double exponentiation used in XTR, we fill a gap in the literature left open by other work in this area.

We presented the first known SPA attack against XTR double exponentiation and two defence methods against such an attack. The first method used a novel randomisation of the exponentiation algorithm while the second borrowed a technique from ECC to construct indistinguishable arithmetic. Finally, we investigated methods of defending XTR against DPA attack, noting that adapting ECC specific techniques require several subtle alterations to cope with the XTR group structure.

As a result, for security against currently known side-channel methods we propose the use of the Euclidean method in Algorithm 2 coupled with indistinguishable arithmetic to guard against SPA and exponent splitting to cope with DPA. This offers a very low performance overhead defence method while achieving a high level of security against side-channel attack. In further work we intend to investigate this security level in physical SPA and DPA experiments and also to explore defence methods for the explicit, i.e. non-trace based, version of XTR.

The authors would like to thank Bart Preneel for his opposition, which led to this research, and Arjen K. Lenstra for his encouragement and proofreading.

References

1. É. Brier and M. Joye. Weierstraß elliptic curves and side-channel attacks. *PKC'02*, LNCS **2274**, pages 335–345.
2. H. Cohen and A. K. Lenstra. Supplement to implementation of a new primality test. *Mathematics of Computation*, 48(177): S1–S4, 1987.

3. J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. *CHES'99*, LNCS **1717**, pages 292–302.
4. C. H. Gebotys and R. J. Gebotys. Secure elliptic curve implementations: An analysis of resistance to power-attacks in a dsp processor. *CHES'02*, LNCS **2523**, pages 114–128.
5. D.-G. Han, J. Lim, and K. Sakurai. On insecurity of the side channel attack on xtr. In *The 2004 Symposium on Cryptography and Information Security (SCIS'04)*, page To appear. The Institute of Electronics, Information and Communication Engineers, 2004.
6. K. Itoh, T. Izu, and M. Takenaka. Address-bit differential power analysis of cryptographic schemes OK-ECDH and OK-ECDSA. *CHES'02*, LNCS **2523**, pages 129–143.
7. T. Izu and T. Takagi. Exceptional procedure attack on elliptic curve cryptosystems. *PKC'03*, LNCS **2567**, pages 224–239.
8. M. Joye and J.-J. Quisquater. Hessian elliptic curves and side-channel attacks. *CHES'01*, LNCS **2162**, pages 93–100.
9. M. Joye and C. Tymen. Protection against differential power analysis for elliptic curve cryptography – an algebraic approach. *CHES'01*, LNCS **2162**, pages 377–390.
10. M. Joye and S.-M. Yen. The Montgomery powering ladder. *CHES'02*, LNCS **2523**, pages 291–302.
11. C. Karlof and D. Wagner. Hidden markov model cryptanalysis. *CHES'03*, LNCS **2779**, pages 17–34.
12. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Crypto'96*, LNCS **1109**, pages 104–113.
13. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. *Crypto'99*, LNCS **1666**, pages 388–397.
14. D. H. Lehmer. Computer technology applied to the theory of numbers. *Studies in Number Theory*, volume 6 of *MAA Studies in Mathematics*, pages 117–151. Math. Assoc. Amer. (distributed by Prentice-Hall, Englewood Cliffs, N.J.), 1969.
15. A. K. Lenstra. Using cyclotomic polynomials to construct efficient discrete logarithm cryptosystems over finite fields. *ACISP'97*, LNCS **1270**, pages 127–138.
16. A. K. Lenstra and E. R. Verheul. The XTR public key system. *Advances in Cryptography—Crypto'00*, LNCS **1880**, pages 1–19.
17. A. K. Lenstra and E. R. Verheul. An overview of the XTR public key system. *The proceedings of the Public-Key Cryptography and Computational Number Theory Conference*, pages 151–180. Verlages Walter de Gruyter, 2001.
18. P.-Y. Liardet and N. P. Smart. Preventing SPA/DPA in ECC systems using the Jacobi form. *CHES'01*, LNCS **2162**, pages 391–401.
19. T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Power analysis attacks on modular exponentiation in smartcards. *CHES'99*, LNCS **1717**, pages 144–157.
20. P. L. Montgomery. Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains. Revised (1992) version from ftp.cwi.nl: /pub/pmontgom/Lucas.ps.gz, 1983.
21. K. Okeya and K. Sakurai. Power analysis breaks elliptic curve cryptosystems secure against timing attack. *Indocrypt'00*, LNCS **1977**, pages 178–190.
22. E. Oswald and M. Aigner. Randomized addition-subtraction chains as a countermeasure against power attacks. *CHES'01*, LNCS **2162**, pages 39–50.
23. A. Shamir. Protecting smart cards from passive power analysis with detached power supplies. *CHES'00*, LNCS **1965**, pages 71–77.
24. M. Stam and A. K. Lenstra. Speeding up XTR. *Asiacrypt'01*, LNCS **2248**, pages 125–143.
25. M. Stam and A. K. Lenstra. Efficient subgroup exponentiation in quadratic and sixth degree extensions. *CHES'02*, LNCS **2523**, pages 318–332.
26. E. Trichina and A. Bellezza. Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks. *CHES'02*, LNCS **2523**, pages 98–113.

A SPA-Resistant Trace Operations

A.1 Field Representation for $p \equiv 2 \bmod 3$

This is the original XTR field representation. Let p be a prime congruent to $2 \bmod 3$, then p generates $\mathbf{Z}_3^*$ and $\Phi_3(x) = x^2 + x + 1 | (x^3 - 1)$ is irreducible in $\mathbf{F}_p$. Let ζ_3 denote a root of $\Phi_3(x)$, then $\zeta^n = \zeta_3^{(n \bmod 3)}$ and in particular $\zeta_3^p = \zeta_3^2$. Hence $\{\zeta_3, \zeta_3^2\}$ is an optimal normal basis of $\mathbf{F}_{p^2}$ over $\mathbf{F}_p$.

Let $c_n \in \mathbf{F}_{p^2}$ be represented by $(c_{n,1}, c_{n,2}) \in (\mathbf{F}_p)^2$, i.e., $c_n = c_{n,1}\zeta_3 + c_{n,2}\zeta_3^2$ (similarly for c_m, c_{n-m} and c_{n-2m}). Using Fermat's little theorem we have that $c_{n,1}^p = c_{n,1}$ etc. and hence $c_n^p = c_{n,2}\zeta_3 + c_{n,1}\zeta_3^2$. We need to consider the computation of c_{2n} and c_{n+m}. The first one is easiest, since

$$c_{2n} = (c_{n,2} - 2(c_{n,1} + 1))c_{n,2}\zeta_3 + (c_{n,1} - 2(c_{n,2} + 1))c_{n,1}\zeta_3^2 \, .$$

Computation of c_{n+m} boils down to computing

$$\begin{aligned} c_{n+m} = {} & ((c_{n-m,1} - (c_{n-m,2} + c_{m,2}))c_{n,1} + (c_{n-m,2} + c_{m,2} - c_{m,1})c_{n,2} + c_{n-2m,1})\zeta_3 + \\ & + ((c_{n-m,2} - (c_{n-m,1} + c_{m,1}))c_{n,2} + (c_{n-m,1} + c_{m,1} - c_{m,2})c_{n,1} - c_{n-2m,2})\zeta_3^2 \, . \end{aligned}$$

If c_{2m-n} is known instead of c_{n-2m} the required Frobenius operation can be easily incorporated into the formula above by swapping the roles of $c_{n-2m,1}$ and $c_{n-2m,2}$. This is especially handy for the binary exponentiation algorithm.

	One c_{n+m} call	*Operation*		Two c_{2n} calls
1.	$t_1 = c_{n-m,2} + c_{m,2}$	Add	Add'	$t_1 = c_{n,1} + (c_{n,1} + 2)$
2.	$t_2 = c_{n-m,1} - t_1$	Sub	Sub	$t_2 = c_{n,2} - t_1$
3.	$t_3 = c_{n,1} \cdot t_2$	Mul	Mul	$c_{2n,1} = c_{n,2} \cdot t_2$
4.	$t_4 = c_{n-2m,1} + t_3$	Add	Add'	$t_4 = c_{n,2} + (c_{n,2} + 2)$
5.	$t_2 = t_1 - c_{m,1}$	Sub	Sub	$t_2 = c_{n,1} - t_4$
6.	$t_3 = c_{n,2} \cdot t_2$	Mul	Mul	$c_{2n,2} = c_{n,1} \cdot t_2$
7.	$c_{n+m,1} = t_3 + t_4$	Add	-	
8.	$t_1 = c_{n-m,1} + c_{m,1}$	Add	Add'	$t_1 = c_{m,1} + (c_{m,1} + 2)$
9.	$t_2 = c_{n-m,2} - t_1$	Sub	Sub	$t_2 = c_{m,2} - t_1$
10.	$t_3 = c_{n,2} \cdot t_2$	Mul	Mul	$c_{2m,1} = c_{m,2} \cdot t_2$
11.	$t_4 = t_3 - c_{n-2m,2}$	Sub	Add'	$t_4 = c_{m,2} + (c_{m,2} + 2)$
12.	$t_2 = t_1 - c_{m,2}$	Sub	Sub	$t_2 = c_{m,1} - t_4$
13.	$t_3 = c_{n,1} \cdot t_2$	Mul	Mul	$c_{2m,2} = c_{m,1} \cdot t_2$
14.	$c_{n+m,2} = t_3 + t_4$	Add	-	

Fig. 3. Indistinguishable arithmetic for $p \equiv 2 \bmod 3$

A.2 Field Representation for $p \equiv 3 \bmod 4$

Lenstra and Stam remark that the representation below can be used for XTR as well. Let p be a prime congruent to $3 \bmod 4$. Then p generates $\mathbf{Z}_4^*$ and $\Phi_4(x) = x^2 + 1$ is irreducible in $\mathbf{F}_p$. Let ζ_4 denote a root of $\Phi_4(x)$, then $\{1, \zeta_4\}$ is a basis of $\mathbf{F}_{p^2}$ over $\mathbf{F}_p$.

Let $c_n \in \mathbf{F}_{p^2}$ be represented by $(c_{n,0}, c_{n,1}) \in (\mathbf{F}_p)^2$, i.e., $c_n = c_{n,0} + c_{n,1}\zeta_4$ (similarly for c_m, c_{n-m} and c_{n-2m}). Using Fermat's little theorem we have that $c_{n,0}^p = c_{n,0}$ etc. and hence $c_n^p = c_{n,0} - c_{n,1}\zeta_4$. Below we list the formulae for c_{2n} and c_{n+m} based on the current field representation

$$c_{2n} = ((c_{n,0} + 1 + c_{n,1})(c_{n,0} + 1 - c_{n,1}) - 1) + 2(c_{n,0} - 1)c_{n,1}\zeta_4 \, .$$

$$\begin{aligned} c_{n+m} = {} & ((c_{n-m,0} + c_{m,0})c_{n,0} + (c_{n-m,1} - c_{m,1})c_{n,1} + c_{n-2m,0}) + \\ & + ((c_{n-m,0} + c_{m,0})c_{n,1} - (c_{n-m,1} - c_{m,1})c_{n,0} + c_{n-2m,1})\zeta_4 \, . \end{aligned}$$

In Figure 4 we show how to make two applications of c_{2n} indistinguishable from one application of c_{n+m}. Note that we are doing some double work for the latter (1 and 7, 3 and 9) and are once more depending on additions with minor carry fiddling (and more so than previously).

	One c_{n+m} call	*Operation*		Two c_{2n} calls
1.	$t_1 = c_{n-m,1} - c_{m,1}$	Sub	Sub'	$t_1 = c_{n,0} + c_{n,0} - 2$
2.	$t_2 = c_{n,1} \cdot t_1$	Mul	Mul	$c_{2n,1} = t_1 c_{n,1}$
3.	$t_1 = c_{n-m,0} + c_{m,0}$	Add	Add'	$t_1 = c_{n,0} + c_{n,1} + 1$
4.	$t_3 = c_{n-2m,0} + t_2$	Add	Sub'	$t_2 = c_{n,0} - c_{n,1} + 1$
5.	$t_4 = c_{n,0} \cdot t_1$	Mul	Mul'	$c_{2n,0} = t_1 t_2 - 1$
6.	$c_{n+m,0} = t_3 + t_4$	Add	-	
7.	$t_1 = c_{m,1} - c_{n-m,1}$	Sub	Sub'	$t_1 = c_{m,0} + c_{m,0} - 2$
8.	$t_2 = c_{n,0} \cdot t_1$	Mul	Mul	$c_{2m,1} = t_1 c_{m,1}$
9.	$t_1 = c_{n-m,0} + c_{m,0}$	Add	Add'	$t_1 = c_{m,0} + c_{m,1} + 1$
10.	$t_3 = c_{n-2m,1} + t_2$	Add	Sub'	$t_2 = c_{m,0} - c_{m,1} + 1$
11.	$t_4 = c_{n,1} \cdot t_1$	Mul	Mul'	$c_{2m,0} = t_1 t_2 - 1$
12.	$c_{n+m,1} = t_3 + t_4$	Add	-	

Fig. 4. Indistinguishable arithmetic for $p \equiv 3 \bmod 4$