Implementation of Elliptic Curve Cryptography with Built-In Counter Measures against Side Channel Attacks

Elena Trichina[1] and Antonio Bellezza[2]

[1] Cryptographic Design Center, Gemplus Technology R & D
Via Pio Emanuelli 1, 00143 Rome, Italy
elena.trichina@gemplus.com
[2] University of Rome "La Sapienza", Rome, Italy
abellezza@tiscalinet.it

Abstract. Many software implementations of public key cryptosystems have been concerned with efficiency. The advent of side channel attacks, such as timing and power analysis attacks, force us to reconsider the strategy of implementation of group arithmetic. This paper presents a study of software counter measures against side channel attacks for elliptic curve cryptosystems.
We introduce two new counter measures. The first is a new implementation technique, namely, homogeneous group operations, which has the property that addition and doubling on elliptic curves cannot be distinguished from side channel analysis. Being inexpensive time-wise, this technique is an alternative to a well-known Montgomery ladder. The second is a non-deterministic method of point exponentiation with precomputations. Although requiring rather large ROM, it provides an effective resistance against high-order power analysis attacks for the price of index re-computations and ROM accesses.
An experimental implementation of NIST-recommended elliptic curves over binary fields with a balanced suite of counter measures built-in in group arithmetic is presented, and the penalty paid is analyzed. The results of the implementation in C on an AMD Duron 600 MHz running Linux are included in the paper.

1 Introduction

With the advance of *side channel attacks* both hard- and software implementations of cryptosystems have to take into account various protection techniques. It has been claimed that all "naive" implementations can succumb to attacks by power analysis. In this paper we do not detail the principles and techniques of different side channel attacks; it had been done elsewhere [17,18,22,10,23]. Instead, we survey and systematize different software counter measures against such attacks, including a couple of new techniques, and suggest a balanced approach to their implementation in the context of elliptic curve cryptosystems (ECC) in binary fields. Giving a due to other excellent papers on practical realizations of ECC [29,31,19], we claim that the contribution of our paper is an attempt to extend the work in [11] to an implementation in which a balanced suite of counter measures against side channel attacks have been built-in on a group operation level.

An implementation of an elliptic curve cryptosystem involves many choices; not only domain parameters (e.g., underlying finite fields, field representations, particular elliptic curves) and group arithmetic, but also field arithmetic, which, in its turn, includes many different methods and algorithms. It is not surprising therefore that although numerous papers on various aspects of ECC implementations had been written, there are only a handful of those that constitute an extensive and careful study that includes all factors involved in the efficient software implementation, notably [11] and [4]. These papers present C implementations on the Pentium 400 MHz workstation of the NIST-recommended elliptic curves over binary and prime fields, respectively.

All published so far complete implementations of ECC have been concerned with efficiency. The advent of side channel attacks force us to reconsider the development cycle. As had been demonstrated in a break-through work by Kocher et al. [17,18], for real life implementations an attacker can be able to perform measurements on the device. If the different operations present different characteristics detectable from the outside (like power consumption profile or duration), the sequence of operations can be discovered by the attacker.

In the most widely used cryptographic systems, an *exponentiation* with a secret exponent is required at some stage. "Exponentiation" in this context is the repeated application of a group operation to the same element. Efficient exponentiation algorithms involve a sequence of squarings, multiplications and possibly divisions (respectively doublings, additions and possibly subtractions in additive notation). Finding out the sequence of operations gives information on the exponent and in some cases uniquely determines it. A detailed systematic description of power analysis attacks on modular exponentiation algorithms can be found in the work of Messerges et al. [22].

Coron [7] generalized DPA attacks to elliptic curve cryptosystems and had shown how to extend DPA to any scalar multiplication algorithm.

Since then, a plethora of papers suggesting various counter measures both, in a general setting and for particular classes of elliptic curves, has been published. We set upon identifying a set of sufficiently general protection techniques that can be employed in a real life scenario. In what follows we survey algorithmic counter measures suggested in the literature, and introduce some original methods. After which an outline of the ECC software implementation in binary fields with a balanced choice of counter measures against side channel attacks is discussed. A description of the experiment with a C implementation on a PC with AMD Duron 600 MHZ running Linux, concludes the paper.

2 Preliminaries and Notation

Since we are interested in elliptic curves, the notation is additive. Thus the basic operations are doubling, addition and subtraction. We focus on elliptic curves over finite fields of characteristic 2, which have an affine equation of the form

$$y^2 + xy = x^3 + ax^2 + b$$

for two elements a and b in the base field. A point on the curve is either identified by a couple of affine coordinates (P_x, P_y) or is the special point P_∞. If P and Q are points on the curve, then an addition $\oplus$ is defined so that $P \oplus Q$ is also a point on the curve and P_∞ is the neutral element. Thus one can also define a subtraction $P \ominus Q$. In what follows, notation from [2] is used.

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on the curve. Assume $P_1, P_2 \neq P_\infty$ and $P_1 \neq -P_2$. The sum $P_3 = (x_3, y_3) = P_1 \oplus P_2$ is computed as follows.

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \qquad y_3 = (x_1 + x_3)\lambda + x_3 + y_1,$$

where $\lambda = (y_1 + y_2)/(x_1 + x_2)$ if $P_1 \neq P_2$, and $\lambda = (y_1)/(x_1) + x_1$ if $P_1 = P_2$.

Each point has also a projective expression of the form (P_X, P_Y, P_Z). If the point is P_∞ then its projective coordinates are $(1, 1, 0)$. Else they are related to the affine coordinates via the transformation rules

$$(P_x, P_y) \mapsto (P_x, P_y, 1)$$
$$(P_X, P_Y, P_Z) \mapsto (P_X/P_Z^2, P_Y/P_Z^3).$$

A curve then is specified by the equation $Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6$. Doubling formulas $(X_3, Y_3, Z_3) = 2(X_1, Y_1, Z_1)$ for the projective equation is:

$$Z_3 = X_1^2 Z_1^2, \qquad X_3 = X_1^4 + bZ_1^4, \qquad Y_3 = bZ_1^4 Z_3 + X_3(aZ_3 + Y_1^2 + bZ_1^4).$$

Addition $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, 1)$ in mixed coordinates is:

$$A = Y_2 Z_1^2, B = X_2 Z_1 + X_1, C = Z_1 B, D = B^2(C + aZ_1^2), Z_3 = C^2, E = AC,$$

$$X_3 = A^2 + D + E, \quad F = X_3 + X_2 Z_3, \quad G = X_3 + Y_2 Z_3, \quad Y_3 = EF + Z_3 G.$$

ECC is based on the general problem of exponentiation in Abelian groups. Various algorithms for exponentiation in the context of cryptography are presented in [21,2]. The survey [8] describes fast methods, including those specialized for elliptic curves. Due to a short bit-length of the exponent, and that it is often generated on-the-fly, sophisticated techniques do not show their advantage. However, certain idiosyncrasies of elliptic curves (e.g., subtraction and addition are almost identical), prompt us to consider together with a classical binary method signed and non-adjacent form (NAF) representation of the exponent [21].

Some algorithms described in this paper work with different exponentiation procedures. In order to be able to interchange them, we rely on a routine that transforms an integer into a sequence indicating the order of operations to perform. Possible contents of the resulting sequence elements are D (double), A (add), S (subtract), suitably encoded.

3 Taxonomy of Software Counter Measures

Three main techniques developed by Kocher et al., are timing attacks [17], and simple (SPA) and differential power analysis (DPA) attacks [18]. Among them, the DPA is the most sophisticated; nevertheless, there are references in the literature when a DPA-protected implementation could be SPA-vulnerable [9]. Moreover, for ECC it seems more difficult to prevent SPA than DPA, as there are good solutions for the latter[7,15,22].

Side channel attacks work because there is a correlation between the physical measurements taken during computations (e.g., power consumption, computing time, EMF radiation, etc.) and the internal state of the processing device, which is itself related to the secret key. While some authors insist on so-called *provable resistance* [6,9] to specific classes of attacks, we share an attitude of [5] and state that, unlike in classical cryptology, in a case of side-channel attacks the definition of an adversarial model is not absolute. Thus, rather than having a *proof* of security, one can have only a strong *evidence* that the countermeasure resists an attack. Note that the security is assessed at the implementation level.

Three general algorithmic techniques to defy side channel attacks are:

1. *Indistinguishability*, which basically means that the only strategy for an adversary is to guess at random. It can be partially achieved by eliminating disparity between group operations involved in exponentiation.
2. *Random splitting of every bit* of the original computation into k shares where each share is equiprobably distributed. Computation then will be carried securely by performing computation only on shares, without reconstructing the original bit [6,9]. In practice, an implementation of this technique for ECC amounts to blinding an exponent or/and a base point with some random mask generated for every run.
3. *Randomization of the execution sequence* which, in order to be successful, must be carried out extensively [6] and, ideally, supported by hardware [27].

In what follows we systematically describe a wide range of counter measures proposed so far for ECC in a general setting.

3.1 Indistinguishability

The first requirement to an effective protection against *timing and SPA* attacks is to ensure that an instruction performed during an execution of a cryptographic algorithm should not be directly dependent on the data being processed.

Double-and-Add Always. Introduced in [7], this counter measure eliminates branch instructions, conditioned by secret data.

```
Algorithm 1 (Double-and-add) ==> Algorithm 2 (Double-and-add always)
Input: P, d=(d_m-1,...,d_0)      Input: P, d=(d_m-1,...,d_0)
  Q <- P                           Q[0] <- P
  for i from m-1 down to 0 do      for i from m-2 down to 0 do
    Q <- 2Q                          Q[0] <- 2Q[0]
    if d_i=1 then Q <-- Q + P        Q[1] <- Q[0] + P
  output Q                           Q[0] <- Q[d_i]
                                   output Q[0]
```

This method increases by 33% the amount of field operations by necessitating "dummy" computations. As has been shown in [28], it makes it susceptible to a new type of attacks, where an attacker induces any temporary random fault into the ALU during the execution of $Q[1] \leftarrow Q[0] + P$ at iteration i. Then, according to whether the final result is incorrect or not, the attacker may deduce if the d_i bit of the exponent is 0 or 1!

The Universal Addition on Elliptic Curves. This method was suggested in [20] for *projective coordinates* for a subclass of elliptic curves in finite fields of characteristic greater than 3. The approach involves a representation of an elliptic curve as the intersection of two quadric surfaces in P^3. A non-standard definition of point addition has the advantage that the set of equations holds when two points are equal, thus, naturally eliminating the difference between point addition and doubling. The price to pay is that every point addition requires 16 field multiplications and 3 squarings. Even after various optimizations (that involve the Jacobi representation and a windowing method, and require additional memory), a total increase in computation costs is 70% in comparison with a standard projective coordinate method.

Independently, a similar approach was developed for Hessian curves in [14], where due to the high symmetry of the Hessian parameterization, the same algorithm can be used for point addition and point doubling. An implementation of point addition requires 12 field multiplications, providing 33% improvement over Jacobian curves.

Unfortunately, both Jacobian and Hessian parameterizations are not fully general. Brier and Joye later extended the technique in [3] to general Weierstrass elliptic curves and come up with the unified formulae, which requires 7 multiplications, 1 inversion and 3 squarings for affine coordinates and 18 multiplications for projective. This is very expensive in comparison with the method proposed in this paper while providing the same protection.

A Montgomery Ladder. The advantages of Montgomery scalar multiplication [24] in the context of elliptic curves were re-discovered several times [19,25, 13,16]. The Montgomery ladder is based on the binary method and the observation that the x-coordinate of the sum of two points whose difference is known, can be computed in terms of only x-coordinates of the involved points. Incidentally, it is also the most promising candidate for a"side channel indistinguishable" exponentiation. The Montgomery ladder protected with a "no-branches" technique is given below.

```
Algorithm 3 (Montgomery method) => Algorithm 4 (Modified Montgomery)
Input: P, d=(d_m-1,...,d_0)        Input: P, d=(d_m-1,...,d_0)
Output: dP                         Output: dP
  P[1] <- P                          P[1] <- P
  P[0] <- 2P                         P[0] <- 2P
  for i from m-2 down to 0 do        for i from m-2 down to 0 do
    if d_i=1 then                      b <- (1-d_i)
      P[1] <- P[1]+P[0]
      P[0] <- 2P[0]
    else
      P[0] <- P[1]+P[0]
      P[1] <- 2P[1]
  return P[1]
```

```
    P[b] <- P[0]+P[1]
    P[d_i] <- 2P[d_i]

  return P[1]
```

The advantage of this method is perfectly symmetric, memory-efficient computations. The disadvantage is increase in computation costs since both, point addition and point doubling, are computed in each iteration.

An optimization of the Montgomery method based on a new formulae for computing x-coordinate for addition of two points, was suggested in [19]. In affine case, a new formulae requires 2 field inversions, 2 multiplications, 2 squarings and 4 additions per iteration; for projective coordinates, it involves only 6 field multiplications, 6 squarings, and 3 additions per iteration. This is comparable cost-wise with our homogeneous group operation method described later, but has an advantage of using less memory, since y-coordinate is not used.

Universal Exponentiation Algorithm. Introduced in [5], this is an elegant attempt to design a *provably* SPA-resistant exponentiation method. Using a representation of the exponent with addition chains, the authors "transfer" the security of the exponentiation method actually implemented in the exponent itself (i.e., in a secret data). The requirement here is that an exponent must be represented as an addition chain in a secure environment. The algorithm works with virtually all exponentiation methods. It reads triplets of values $(\gamma(i) : \alpha(i) : \beta(i))$ meaning that the content of register $R[\alpha(i)]$ must be multiplied by (added to) the content of register $R[\beta(i)]$ and the result must be written into register $R[\gamma(i)]$. The exponent d then is represented by the register sequence $\Gamma(d) = \{(\gamma(i) : \alpha(i), \beta(i))\}_{0 \leq i \leq m-1}$. In order to break the algorithm, an adversary must be able to differentiate among the triplets and to recover all their values. The advantage of this method is that it introduces only a small amount of extra computations; the disadvantage is large memory usage.

3.2 Splitting Variables

In practice, the general encoding proposed in [6,9] amounts to blinding an exponent and a base point with some random mask to prevent *DPA* attacks.

Blinding an Exponent. Randomization of the exponent is inexpensive and useful technique. It comes in few flavors.

An additive mask is usually used in the ECC context [7]. It consists in adding a multiple of the element order to the exponent. If N is the group order, then $(kN)P = 0$ for any integer k and for any element P in the group. To compute dP, where d is the secret exponent and P is a base point, one first blinds d

with kN, where k is a (small) random number generated for every run, and N is the group order. Hence, $Q = (d + kN)P = dP + (kN)P = dP$. As d usually is the size of the group order, choosing a t-bit integer k increases the size of the exponent by about t bits.

An exponent splitting [5] is a variant of this technique, where d is represented as a sum of a random k and $d-k$, and exponentiation is carried out in two steps: $R \to kP$; $\quad Q \to (d-k)R$. The values of both, k and $d-k$ are required to recover the value of d, which makes it less attractive than the method above.

A multiplicative mask is a multiplicative analogue of this idea. Let N be the group order. If k is a unit in the multiplicative group $(\mathbf{Z}/\mathbf{Z}N, \times)$ and k^{-1} its inverse, then $d = k(k^{-1}d)(\mod N)$ for any exponent d. This means that calling $d' := k^{-1}d(\mod N)$, for any group element P one has the following equivalent exponentiations:

$$\begin{array}{llll} (1) & P \overset{d}{\mapsto} dP & (3) & P \overset{d'}{\mapsto} d'P \overset{k}{\mapsto} k(d'P) \\ (2) & P \overset{k}{\mapsto} kP \overset{d'}{\mapsto} d'(kP) & (4) & P \overset{kd'}{\mapsto} (kd')P \end{array}$$

Sequence (1) is a straightforward exponentiation; sequence (4) is a sub-case of the additive blinding technique. Sequences (2) and (3) are performed in two phases. We focus on (2). The overhead is given by the exponentiation with exponent k, since both d and d' are on average the size of N. To keep overhead low, one can choose k to yield a fast exponentiation, for instance by choosing it randomly among the elements of $(\mathbf{Z}/\mathbf{Z}N)^*$ of at most t bits, with a small t. As in the additive blinding, one can trade-off the degree of randomization (and thus the security) for speed.

If sequence (2) is used, the element fed to the second phase is not controlled by an attacker. The first phase, on the other hand, doesn't leak any information whatsoever on the secret exponent, depending uniquely on the input element and the random integer k. Thus one gets at the same time additional bonuses of a change in the sequence of point operations and of randomization of the input point. Cost-wise, the method is comparable with the additive mask, although precomputations are somewhat slower.

Blinding a Base Point. Blinding a base point is necessary if we assume that an attacker knows how points are represented in memory during computations [7]. An accepted technique to mask the input by applying some bijective function for which it is easy to compute the final correction.

Adding to the input a "perturbation point" [7] involves storing a couple of "random" points $(R, -dR)$ and updating it at each run by doubling both elements. Exponentiation of an input P with an exponent d is performed in three steps: $(1) P' \leftarrow P + R$, $\quad (2) Q' \leftarrow dP' = dP + dR$, $\quad (3) Q \leftarrow Q' + (-dR) = dP$.

As the argument $P + R$ of the exponentiation phase does not depend exclusively on the input, it is not possible to choose points with particular properties. However, if d changes at each run, computing $-dR$ for final correction doubles group operations, and must be protected as well.

Randomizing projective coordinates of a point [7] is achieved almost for free. Projective coordinates of a point is not unique because $(X, Y, Z) = (kX, kY, kZ)$ for every $k \neq 0$ in the finite field. Hence, before each new execution of dP, the projective coordinates of P can be randomized with a random k. It makes it impossible for an attacker to predict any specific bit of the binary representation of P, thus rending a DPA attack infeasible.

3.3 Randomization of the Execution Sequence

Non-deterministic execution of the sequence of instructions, although does not provide perfect protection against statistical techniques, increases the number of measurements.

Randomized Addition/Subtraction chain. [26] exploits the fact that dP can be computed with different execution sequences; for instance, for $9P$: $P \rightarrow 2P \rightarrow 4P \rightarrow 8P \rightarrow 9P$ or $P \rightarrow 2P \rightarrow 4P \rightarrow 5P \rightarrow 10P \rightarrow 9P$. Randomization can be achieved by inserting a random decision that can rearrange a sequence of additions, subtractions and doubling in a signed binary algorithm. The idea is to randomly substitute a long chain of 1's (which correspond to "double-and-add") in the signed binary representation of the exponent by a block of zeros ("double") followed by -1 (subtract). Similar substitution is applied to isolated 0's inside a block of 1's. The algorithm needs some 9% more additions. The authors claim that it makes DPA attacks really difficult.

4 Homogeneous Group Operations

In the choice of doubling and addition algorithms for elliptic curves, it is often advisable to work in projective coordinates, which allows us to avoid costly inversions in the field. For elliptic curves over fields of characteristic 2, complexities for doubling performed on projective coordinates and addition and subtraction on mixed affine-projective coordinates with projective result, are: Here *Mult* stands for multiplication, *Squar* for squaring, and *Add* for addition.

The problem we address here is the relevant computational difference between addition/subtraction and doubling, which in implementation reflects in differences detectable by an attacker. Observing that the complexity for point doubling is about half of the complexity for addition/subtraction, one can think of splitting the complex subroutines into two parts, each one approximately equivalent to a point doubling. This is actually possible, as we show below.

When choosing an elliptic curve, setting $a = 1$ makes the total number of products in addition and subtraction exactly twice the number for doubling,

Table 1. Complexities of group operations in projective coordinates

	General case			**Case $a = 1$**		
Curve operation	Mult	Squar	Add	Mult	Squar	Add
Doubling	5	5	4	5	5	4
Addition	11	3	7	10	3	7
Subtraction	11	3	8	10	3	8

thus avoiding dummy operations. We present atomic units of computations in the case $a = 1$. A general case slightly differs in the addition/subtraction part, while the doubling subroutine is the same.

Before proceeding, we observe that computing the subtraction $P \ominus Q$ of two points corresponds to computing $P \oplus (-Q)$. If Q has affine coordinates (Q_x, Q_y), then $-Q$ has affine coordinates $(Q_x, Q_y + Q_x)$. Thus, the algorithms for addition and subtraction only differ by a field addition, and we will consider them at the same time. Furthermore, if the affine point to be subtracted is the same throughout a loop, one can compute $Q_x + Q_y$ once. Here are the subroutines.

Doubling. The input is given by the coordinates (A_X, A_Y, A_Z) of the point.

$$
\begin{array}{lll}
\lambda_1 \longleftarrow A_Z^2 & \lambda_6 \longleftarrow \widetilde{b}\lambda_1 & \lambda_{11} \longleftarrow \lambda_7^4 \\
\lambda_2 \longleftarrow A_X\lambda_1 & \lambda_7 \longleftarrow A_X + \lambda_6 & \lambda_{12} \longleftarrow \lambda_{10}\lambda_{11} \\
\lambda_3 \longleftarrow A_Y A_Z & \lambda_8 \longleftarrow \lambda_4^2 & \lambda_{13} \longleftarrow \lambda_9 + \lambda_{12} \\
\lambda_4 \longleftarrow A_X^2 & \lambda_9 \longleftarrow \lambda_8\lambda_2 & \\
\lambda_5 \longleftarrow \lambda_2 + \lambda_4 & \lambda_{10} \longleftarrow \lambda_5 + \lambda_3 &
\end{array}
$$

Here $\widetilde{b}$ is such that $\widetilde{b}^4 = b$. It can be computed once and for all when the curve is set. The output point has projective coordinates $(\lambda_{11}, \lambda_{13}, \lambda_2)$. The sequence of operations is SMMSAMASMASSMA.

Addition/Subtraction – Phase I (Case $a = 1$). The input is given by projective coordinates (A_X, A_Y, A_Z) and affine coordinates (B_x, B_y).

$$
\begin{array}{lll}
\mu_1 \longleftarrow \lambda_7^2 & \lambda_4 \longleftarrow B_y \quad (addition) & \lambda_7 \longleftarrow A_Z\lambda_6 \\
\lambda_1 \longleftarrow A_Z^2 & \quad or & \lambda_8 \longleftarrow A_Y + \lambda_5 \\
\lambda_2 \longleftarrow \lambda_1 B_x & \lambda_4 \longleftarrow B_y + B_x \quad (subtraction) & \lambda_9 \longleftarrow \lambda_8 B_x \\
\lambda_3 \longleftarrow \lambda_1 A_Z & \lambda_5 \longleftarrow \lambda_4\lambda_3 & \lambda_{11} \longleftarrow \lambda_7 + \lambda_8 \\
 & \lambda_6 \longleftarrow A_X + \lambda_2 &
\end{array}
$$

The sequence of operations is SMMAMAMAMA.

Addition/Subtraction – Phase II (Case $a = 1$). This subroutine is executed right after the previous one, of which it uses some of the partial results.

$$\begin{array}{lll} \mu_1 \longleftarrow \lambda_7^2 & \mu_5 \longleftarrow \mu_1 + \mu_3 & \mu_9 \longleftarrow \lambda_9 + \mu_2 \\ \mu_2 \longleftarrow \lambda_7\lambda_4 & \mu_6 \longleftarrow \lambda_6\mu_4 & \mu_{10} \longleftarrow \mu_9\mu_1 \\ \mu_3 \longleftarrow \lambda_8\lambda_{11} & \mu_7 \longleftarrow \mu_5 + \mu_6 & \mu_{11} \longleftarrow \mu_8 + \mu_{10} \\ \mu_4 \longleftarrow \lambda_6^2 & \mu_8 \longleftarrow \mu_7\lambda_{11} & \end{array}$$

The output point has projective coordinates $(\mu_7, \mu_{11}, \lambda_7)$.The sequence of operations is SMMSAMAMAMA.

The shortest operation sequence (including a couple of dummy operations) for all three subroutines is $SMMSAMASMASSMA$.

Table 2. Complexities of homogeneous vs normal group operations

	General case			**Case $a = 1$**		
Curve operation	Mult	Squar	Add	Mult	Squar	Add
Doubling (old)	5	5	4	5	5	4
Doubling (new)	6	5	4	5	5	4
Addition (old)	11	3	7	10	3	7
Addition (new)	12	10	8	10	10	8
Subtraction (old)	11	3	8	10	3	8
Subtraction (new)	12	10	8	10	10	8

Table 2 gives the complexities for various options. Observe that squarings and additions in the field of characteristic 2 are very cheap. Thus we are wasting little more than one field multiplication per curve operation in the general case, and just a few additions and squarings in the case $a = 1$.

An Exponentiation Algorithm Using Homogeneous Point Operations. The classical algorithms can be used to implement an exponentiation that shows to the outside world a sequence of homogeneous rounds. The subroutines for doubling, addition-phase I, subtraction-phase I and addition/subtraction-phase II have been implemented in C. In order to achieve homogeneity, they are invoked with the same kind of parameters, i.e. with 5 field elements representing the projective and affine coordinates of two curve points. The result is a point in projective coordinates, i.e. a triplet of field elements.

Phase I of addition and subtraction returns the first input point (the one in projective coordinates). The subroutines are called F_i with i being 0 for doubling, 1 for addition/I, 2 for subtraction/I and 3 for addition-subtraction/II. F_1 and F_2 also have side-effects, namely they store in memory partial results which are used by F_3. We suppose that at the precomputation step, the exponent is transformed in a secure environment into a sequence b[1], ..., b[d] corresponding to procedure calls.

```
Algorithm 5 (Exponentiation with homogeneous group operations)
Input: point P=(P_x,P_y) in affine coordinates,
       sequence b[1],...,b[d] of atomic unit calls for exponent d
Output: point dP in affine coordinates
  (S_X,S_Y,S_Z) <- (1,1,0) (initialise the result)
   for i from 1 to d-1 do
      (S_X,S_Y,S_Z) <- F_{b[i]}(S_X,S_Y,S_Z,I_x,I_y)
Final conversion
    S_x <- S_X/S_Z^2
    S_y <- S_Y/S_Z^3
Return the point with affine coordinates (S_x,S_y)
```

5 Non-deterministic Right-to-Left Method with Precomputations

Non-deterministic right-to-left method with precomputations had been actually implemented in our experiment. It is very efficient but requires a lot of ROM.

An algorithm is based on the observation that $dP = d_{m-1}2^{m-1}P + ... + d_1 2P + d_0 P$, where $(d_{m-1}, ..., d_1, d_0)$ is a binary representation of the exponent d, d_0 being the LSB, and d_{m-1} the MSB of d. Thus, if the multiples of point P are precomputed in advance, it does not matter in which order the bits of the exponent d are scanned as long as one can associate the bit d_i with the precomputed multiple $2^i P$. Here is our solution.

```
Algorithm 6 (Non-deterministic right-to-left with precomputations)
Input: array A=[2^{m-1}P,...,2P, P] of precomputed multiples of P
       d = (d_{m-1},...,d_1, d_0) bits of an exponent
Output: dP
  M <- 0                     -- initialise M to zero
  W <- permute([m-1, ...,0]) -- W contains randomly permuted indices
  for all elements j from W do
    M <- M + d_{W[j]}*A[W[j]]
  return M
```

Instead of permutation, one can just rotate A and d simultaneously j positions to the left or to the right; $0 \leq j \leq m-1$ is a random updated at each run. To enhance the randomization, the method of random splitting and rotating can be applied recursively to the sub-arrays of indices. This method is beneficial for protocols where the base point P does not change often. However, it requires additional ROM to store precomputed multiples of P. Considering that the size of field elements for the ECC is around 200 bits, the total amount of memory in question is $25 \times \log N$ bytes, where N is a group order.

6 A Point of a Small Order as a Perturbation Point

A point of a small order on the curve as a perturbation point is a variation on the idea of blinding a base point. Elliptic curves of use for cryptographic purposes have a group of points with order a big prime N times a cofactor h, where h is usually small. We suggest choosing as the perturbation point R a point of order h. We suppose the cofactor not to be too small (a cofactor of 2 would be no good) but such that its bit-length l_h is not bigger than, say, 1/5 of bit-length l_N of N.

This presents some disadvantages but allows on-the-fly computation of the point and the correction at each run with much fewer group operations. To be useful, a cofactor h has to be at least 4. Although [1] contains an example with the bit-length $l_h = 16$ and $l_N = 161$, the trend to have a very small cofactor weakens our method.

We compute a random point of order h as the multiple of Q by a random factor r between 0 and $h - 1$. The final correction to apply is $T := -drQ$. We can actually compute it as $(-dr \mod h)Q$. The factor is again between 0 and $h - 1$. As the bit-length of h is considerably less than the bit-length of N, the exponentiations to get R and T are faster than the main exponentiation and require some l_h additions. The computation of rQ and $(-dr \mod h)Q$ must be masked. As r changes after each run, one has to take care mostly of single-run analysis.

One can speed up and simultaneously mask computations of rQ and $(-dr \mod h)Q$ by storing $2Q, 4Q, \ldots, 2^{l_h-1}Q$ and using fast right-to-left binary algorithm in a non-deterministic manner, as described in the previous chapter. This technique had actually been implemented.

7 Computer Experiments

We implemented some of the ideas explained in the previous chapters. For finite fields, we implemented in C basic operations from scratch following [11].

Field Arithmetics. As polynomial basis, we followed [11] and [1], choosing the following irreducible polynomials for the fields in our experiments.

$$\begin{aligned}
\mathbf{F}_{2^{163}} &= \mathbf{F}_2[x]/(x^{163} + x^7 + x^6 + x^3 + 1) \\
\mathbf{F}'_{2^{163}} &= \mathbf{F}_2[x]/(x^{163} + x^8 + x^2 + x + 1) \\
\mathbf{F}_{2^{233}} &= \mathbf{F}_2[x]/(x^{233} + x^{74} + 1) \\
\mathbf{F}_{2^{283}} &= \mathbf{F}_2[x]/(x^{283} + x^{12} + x^7 + x^5 + 1)
\end{aligned}$$

For multiplication right to left and left to right comb methods, plus a base 16 left to right comb method were implemented. As the inversion algorithm we chose the extended Euclidean algorithm. Sample timings taken on an AMD Duron 600 MHz running Linux for selected fields are given in Table 3.

Exponentiation on Elliptic Curves. We implemented both binary and signed NAF exponentiation for elliptic curves over $F_{2^{163}}$, choosing a curve E_1 with $a = 1$ and a general curve E_2 with the parameters specified below.

Table 3. Timing for field operations in μs for various fields

Operation	$\mathbf{F}_{2^{163}}$	$\mathbf{F}'_{2^{163}}$	$\mathbf{F}_{2^{233}}$	$\mathbf{F}_{2^{283}}$
Addition	0.16	0.16	0.17	0.16
Squaring	0.34	0.37	0.38	0.53
Product: RL comb	7.00	7.13	9.86	18.76
Product: LR comb	10.04	10.10	14.93	18.72
Product: Base 16 LR comb	2.78	2.80	3.61	4.59
Inversion	36.02	36.05	60.50	77.97

Point operations were performed using the standard sequence of operations and the homogeneous versions. Table 4 summarizes the penalty for a side-channel protection. To get the following timings, the generating points were multiplied by a random exponent both with standard and homogeneous operations. For E_1 the fact that $a = 1$ was taken into account in both versions.

As for a real-life scenario, a suite of built-in counter measures included:

- indistinguishability via homogeneous operations (3-17% overhead);
- a multiplicative exponent masking, where the first step (multiplication with a small random k) had been implemented via a non-deterministic right-to-left binary method with precomputations, providing an additional benefit of blinding a base point in a process (15% overhead).

The total costs amounted to 2133 field multiplications, 1773 squarings, 1416 additions and 3 inversions, while a non-protected version would have had 1683 multiplications, 1121 squarings, 1200 additions and 1 inversion. In other words, some 30% increase in computation time constitutes the total penalty.

Elliptic curve parameters	
E_1	
F	$\mathbf{F}_2[x]/(x^{163} + x^7 + x^6 + x^3 + 1)$
a	01
b	02 0A601907 B8C953CA 1481EB10 512F7874 4A3205FD
N	04 00000000 00000000 000292FE 77E70C12 A4234C33
h	02
Q_x	03 FB02D922 0A5E7980 D9C7C192 AFC7EDC4 19B261E4
Q_y	05 F8692B70 5F82AAF2 7E41D4D3 82D9E359 98979F99
E_2	
F	$\mathbf{F}_2[x]/(x^{163} + x^8 + x^2 + x + 1)$
a	07 2546B543 5234A422 E0789675 F432C894 35DE5242
b	00 C9517D06 D5240D3C FF38C74B 20B6CD4D 6F9DD4D9
N	04 00000000 00000000 000292FE 77E70C12 A4234C33
h	02
Q_x	07 AF699895 46103D79 329FCC3D 74880F33 BBE803CB
Q_y	06 434AB98E 1F769093 2FA04BCA 9ED0479D 4B5FC954

8 Conclusion

From our experiment it follows that the cost of counter measures against side channel attacks need not to be prohibitively high. With a choice of well-balanced suite of complementary counter measures the total overhead can be as low as 30%. We want to emphasize that the objective of the experiment was an

Table 4. Exponentiation timings in μs

	binary	NAF		binary	NAF
E_1			E_2		
standard	5355	4699	standard	5642	4932
homogeneous	5512	4847	homogeneous	6581	5770
overhead	2.9%	3.1%	overhead	16.6%	17.0%

"average-case" analysis, rather than finding the fastest or the most secure solution. This explains, for example, the choice of the field $GF(2^n)$ and the field representation. It is well-known that operations in $GF(2^n)$ can be efficiently implemented in both, hardware and software, and that, although the normal basis representation offers a very fast squaring, the polynomial representation is more widely used.

The choice of projective coordinates for ECC allows us to use homogeneous group operations, thus achieving SPD resistance with only 3–17% overhead. One can argue that using projective coordinates is a good idea only in prime fields, but not in $GF(2^n)$ [30]. However, the choice of coordinates must take into account not only performance, but also security of the implementation. Performance-wise, it makes sense to use projective coordinates when the ratio inversion/ multiplication is greater than 3. Since this ratio depends very much on the chosen multiplication algorithm (see the table in the previous section), the choice of coordinates cannot be made solely on the criteria of the field.

Projective coordinates offer actually two benefits when considering side channel attack resistance. They can be used to defy DPA attacks by randomizing coordinates as in [7] as well as SPA and timing attacks using our homogeneous group operations method.

The experiment made us also wiser. If we were to give an advise on an allround suite of counter measures against side channel attacks, including timing, SPA, DPA, and fault attacks, we would recommend the following.

– Blinding the base point by randomizing projective coordinates.
– Randomization of the exponent can be done using both, additive or multiplicative masks; the latter has an additional benefit of randomization of the execution sequence, and partially, of a base point.
– Use the securized Montgomery ladder with optimized group operations (on x-coordinates only) as suggested in [19]. The overhead is comparable with the one achieved in our experiment, but the algorithm is naturally more regular and thus, more secure.

References

1. ANSI X9.62: *The elliptic curve digital signature algorithm (ACDSA)*, draft, July 1997.
2. I. Blake, G. Seroussi, N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press (1999).
3. Brier, E., Joye,: Weierstrass elliptic curves and side-channel attacks. Proc. *Public Key Cryptography (PKC 2002)*, LNCS **2274** (2002) 335–345
4. Brown, M., Hankerson, D., Lopez, J., Menezes, A.: Software implementation of the NIST elliptic curves over prime fields. Proc. *CT-RSA 2001*, LNCS **2020** (2001) 250–265
5. Clavier, C., Joye, M.: Universal exponentiation algorithm: A first step towards provable SPA-resistance. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 300–308
6. Chari, S., Jutla, C., Rao, J., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. Proc. *Advances in Cryptology – Crypto'99*, LNCS **1666** (1999) 398–412
7. Coron, J.-S.: Resistance against differential power analysis attacks for elliptic curve cryptosystems. Proc. *Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS **1717** (1999) 292–302
8. Gordon, D. M.: A survey of fast exponentiation methods. J. *Algorithms* **27** (1998) 129–146
9. Goubin, L., Patarin, J.: DES and differential power analysis. Proc. *Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS **1717** (1999) 158–172
10. Hasan, M. A.: Power analysis attacks and algorithmic approaches to their countermeasures for Koblitz cryptosystems. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2000)*, LNCS **1965** (2000) 93–108
11. Hankerson, D., Hernandez, J. L., Menezes, A.: Software implementation of elliptic curve cryptography over binary fields. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2000)*, LNCS **1965** (2000)
12. IEEE Std 1363-2000. *IEEE Standard Specification for Public-Key Cryptography.* IEEE Computer Society, August 29, 2000
13. Izu, T., Takagi, T.: A fast parallel elliptic curve multiplication resistant against side channel attacks. Proc. *Public Key Cryptography (PKC 2002)*, LNCS **2274** (2002) 280–296
14. Joye, M., Quisquater, J-J.,: Hessian elliptic curves and side-channel attacks. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 402–410
15. Joye, M., Tymen, Ch.: Protection against Differential Power Analysis for elliptic curve cryptography - An algebraic approach. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 377–390
16. Joye, M., Yen, S-M.: The Montgomery powering ladder. In these proceedings.
17. Kocher, P.: Timing attacks on implementations of Diffie-Hellmann, RSA, DSS, and other systems. Proc. *Advances in Cryptology – Crypto'96*, LNCS **1109** (1996) 104–113
18. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. Proc. *Advances in Cryptology – Crypto'99*, LNCS **1666** (1999) 388–397
19. Lopez, J., Dahab, R.: Fast multiplication on elliptic curves over $GF(2^n)$ without precomputations. Proc. *Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS **1717** (1999) 316–327

20. Liardet, P.-Y., Smart, N. P.: Preventing SPA/DPA in ECC systems using the Jacobi form. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 391–401
21. Menezes, A. J., van Oorschot, P. C., Vanstone, S. A.: *Handbook in Applied Cryptography.* CRC Press, 1997
22. Messerges, T. S., Dabbish, E. A., Sloan, R. H.: Power analysis attacks on modular exponentiation in smartcards. Proc. *Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS **1717** (1999)
23. Messerges, T. S., Dabbish, E. A., Sloan, R. H.: Examining smart-card security under the threat of power analysis attacks. *IEEE Transactions on Computers*, **51** (5) (2002) 541–552
24. Montgomery, P.: Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computations*, **48** (1987) 243–264
25. Okeya, K., Sakaurai, K.: Power analysis breaks Elliptic Curve cryptosystems even secure against the timing attacks. Proc. *INDOCRYPT 2000*, LNCS **1977** (2000) 178–190
26. Oswald, E., Aigner, M.: Randomized addition-subtraction chains as a countermeasure against power attacks. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 39–50
27. May, D., Muller, H.L., Smart, N.P.: Non-deterministic processors. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 28–38
28. Yen, S-M., Kim, S-J., Lim, S-G., Moon, S-J: A counter-measure against one physical cryptanalysis may benefit another attack. Proc. *Information Security and Cryptology (ICISC 2001)*, LNCS **2288** (2002) 414–427
29. De Win,. E., Bosselaers, A., Vandenberghe, S., De Gersem, P., Vandewalle, J.: A fast software implementation for arithmetic operations in $GF(2^n)$. Proc. *Asiacrypt '96*, LNCS **1163** (1996) 65–76
30. De Win,. E., Mister, S., Preneel, B., Wiener M.: On the performance of signature schemes based on elliptic curves. Proc. *An Algorithmic Number Theory Symposium*, LNCS **1423** (1998)
31. Woodbury, A., Bailey, D., Paar, Ch.: Elliptic curve cryptography on smart cards without coprocessors. Proc. *The Forth Smart Card Research and Advanced Applications Conf.*, September 20–22, 2000 Bristol, UK