# On distributed computing on elliptic curves


## Tommi Meskanen

Department of Mathematics, University of Turku
FIN-20014, Turku, Finland
and
Turku Centre for Computer Science TUCS
e-mail: tommes@utu.fi

## Ari Renvall

Department of Mathematics, University of Turku
FIN-20014, Turku, Finland
e-mail: ariren@utu.fi

## Paula Steinby

Department of Mathematics, University of Turku
FIN-20014, Turku, Finland
e-mail: pauste@utu.fi



**Turku Centre for Computer Science**
**TUCS Technical Report No 401**
**March 2001**

**ISBN 952-12-0807-4**
**ISSN 1239-1891**

**Abstract**

Let $C$ be a device of limited computing power, and assume it to have access to another device $A$ with superior capacities. We consider the situation where $C$ is supposed to calculate the basic operation of elliptic curve cryptography: the scalar multiplication of a point $P$ on the curve. We investigate whether $C$ can take advantage of $A$'s computing power, and to what extent $C$ needs to trust $A$.

**Keywords:** cryptography, elliptic curves, distributed computing

**TUCS Research Group**
Theory Group: Mathematical Structures in Computer Science

# 1 Arithmetic on elliptic curves

Denote by $\mathbf{F}_q$ the field with $q$ elements. If the field is binary, then the (non-supersingular) elliptic curve $E(\mathbf{F}_q)$ over $\mathbf{F}_q$ is defined as the set of points

$$E(\mathbf{F}_q) = \{(x, y) \mid y^2 + xy = x^3 + ax^2 + b\} \cup \mathcal{O} \qquad (a, b \in \mathbf{F}_q, b \neq 0),$$

where $\mathcal{O}$ is the point at infinity. (In non-binary fields the term $ax^2$ must be replaced by $ax$ and the condition $4a^3 + 27b^2 \neq 0$ should hold). If the group operation is suitably defined, then $E(\mathbf{F}_q)$ is an Abelian group with $\mathcal{O}$ as the neutral element. It is customary to adopt additive notation, so we denote the group operation by $+$. This addition can be performed using the basic arithmetic operations of the base field. Below we give the equations for a binary field. Let $P = (x, y) \neq \mathcal{O}$ be a point on the curve. Then

$$\begin{aligned} -P &= (x, x + y) \\ 2P &= (u, v) \\ &= (\theta^2 + \theta + a, \theta(x + u) + u + y), \end{aligned}$$

where $\theta = x + \dfrac{y}{x}$. Moreover, if also $Q = (x', y') \neq \mathcal{O}$, $Q \neq P$ and $Q \neq -P$ (i.e. $x \neq x'$), then

$$\begin{aligned} P + Q &= (u', v') \\ &= (\theta'^2 + \theta' + a + x + x', \theta'(x + u') + u' + y), \end{aligned}$$

where $\theta' = \dfrac{y + y'}{x + x'}$.

Elliptic curve cryptosystems are El Gamal based systems, hence their security is based on the difficulty of the discrete logarithm problem in the group $(E(\mathbf{F}_q), +)$. More specifically, EC based systems exploit the fact that given points $P$ and $kP$ (for some unknown $k$) on a curve $E(\mathbf{F}_q)$, it is practically impossible to calculate $k$. The users of an elliptic curve cryptosystem all share the same domain parameters $(m, f(x), a, b, G, r)$. Of these $m$ and $f(x)$ fix the underlying field and the presentation of field elements, $a$ and $b$ define the curve and $G$ is a point on the curve of order $r$. The private key of a user is then simply an integer $s < r$, and the corresponding public key is the point $Q = sG$.

For an extensive study on elliptic curves in cryptography we refer to [BSS]. Here, as an example, we describe the ECDSA signature algorithm, which is included in all ECC standards. Let $s$ be $A$'s private key and $Q$ the public key. A signature for a message $m$ is then computed as follows.

1. First $A$ computes a representative $h(m)$ of the message, where $h$ is a hash function agreed beforehand (SHA-1, for example).

2. Then $A$ selects a random integer $k$, computes the point $P = kG$ and converts the $x$-coordinate of it to an integer $c$.

3. $A$'s signature for the message $m$ will then be a pair $(c, d)$, where $d = k^{-1}(h(m) + sc) \pmod r$.

A verifier $B$ can check the validity of a signature $(c, d)$ for a message $m$ using $A$'s public key $Q = sG$:

1. $B$ computes the hash value $h(m)$.

2. $B$ computes the point $R = d^{-1}(h(m)G + cQ)$ and converts the $x$-coordinate to an integer $c'$

3. If $c = c'$ then $B$ accepts the signature.

It is an easy task to verify that if both $A$ and $B$ follow their algorithms then $P = R$, and thus also $c = c'$.

As we see, in ECDSA (and in fact in any cryptographic primitive employing elliptic curves) the most time consuming operation is that of scalar multiplication of a point. This motivates us to concentrate on how to securely and efficiently distribute the computation of $kP$.

# 2 Scalar multiplication of a point

In the following we introduce our favourite algorithm for scalar multiplication: the *Fixed Base Windowing* algorithm (FBW for short). Slightly modified, we find it the most suitable for efficient and secure calculation of $kP$. FBW seems recommendable even in the cases where the base $P$ is not fixed. The feature that all doublings occur in the precomputation phase and all additions during the actual computation diminishes the chance of a successful *side-channel attack*.

## 2.1 The FBW algorithm

The Fixed Base Windowing algorithm is a general exponentiation algorithm, as it can be applied in any group. As the name suggests, it is specially designed to situations where the base is fixed. A significant part of the computation consists of constructing a precomputation table $T$, the contents of which only depends on the base. If we always have the same base, then it suffices to calculate $T$ just once and save it for further use. But, if necessary, we can also compute $T$ each time separately.

Below we present the basic version of FBW. We adopt the notation from elliptic curves setup.

Let $P = (x, y)$ be a point of order $r$ on some curve $E(\mathbf{F}_q)$, and let $k$ be a positive integer less than $r$. Our task is to compute the point $kP$. The idea of the algorithm is to look at the multiplier $k$ through a “window” of fixed width $w$. For notational convenience we assume that the length of $k$'s binary representation is $n = lw$, and we denote $k = \sum_{i=0}^{l-1} k_i 2^{iw}$, where $0 \leq k_i < 2^w$. Denote also $P_i = 2^{iw}P$ $(i = 0, 1, \ldots, l-1)$; hence $kP = \sum_{i=0}^{l-1} k_i P_i$.

**Algorithm 1**

```
Q := P; R := 0; B := 0;
For i := 0 To l-1 Do (* precomputation *)
  T[i] := Q;
  For j := 1 to w Do Q := 2*Q;
For i := 2^w-1 Downto 1 Do (* computation depending on k *)
  For j := 0 To l-1 Do
    If k[j] = i Then B := B + T[j];
  R := R + B;
Return(R)
```

First part of the algorithm is to generate the precomputation table $T[i] = P_i$. Then the product $kP$ is “gathered” to $R$ such that each table element $T[i]$ is included in $B$ when there are $k_i$ additions R := R + B left. This sums up to $R = \sum k_i P_i = kP$. The first part of the algorithm consists of doublings only, whereas there are just additions in the second part. This is a strength (compared to, for instance, the traditional double-and-add -algorithm) against so-called side-channel attacks.

Let us consider our model of distributed computing: when computing the product $kP$ (with FBW), $C$ takes advantage of $A$'s computing power. We recommend $C$ to use $A$'s assistance with the precomputation only, and the actual computation to be performed solely by $C$. There are at least two good reasons for this. First, if $A$ is used in computations which depend on the coefficient $k$, then $C$ has to ask $A$ for something specific. That would bring on a need for duplex data transfer. Second, it seems impossible to build a scheme where $C$ would profit from $A$'s computation without having to reveal any relevant information on $k$.

## 3 Models of computation

The main issue of this article is to analyze the benefits of distributed computation when using elliptic curve cryptography. As scalar multiplication is the basic operation in EC cryptography, and most of the time in any cryptographic EC operation is spent computing $kP$, it seems natural to concentrate on it.

Let $C$ be a device to calculate $kP$. Assume that $k$ is known to $C$ only, whereas $P$ is public. Suppose that $A$ is another device with superior computing power, and that $A$ is willing to help $C$ in the computations. As an example, $C$ could be a smart card inserted in a mobile phone $A$.

We consider three different models.

I $A$ does not participate in the computation at all, so $C$ does everything independently. Thus, no data needs to be transferred between $A$ and $C$.

II $A$ is used during the computation, but all data that $A$ provides is checked by $C$. No information on $k$ is given to $A$, so $C$ doesn't need to trust $A$.

III $A$ participates in the computations, and $C$ trusts that all data $A$ gives to $C$ is correct. If $A$ really is honest, the information it gains does not reveal anything of $k$.

Model I is included to give the basis for relevant comparisons of the benefits of the other two models. In I and II the result of $C$ is never incorrect, but in model III this is not necessarily the case if $A$ cheats. It should also be noted, that we really can't make any weaker assumptions on $A$ than we do in model III. The same information that an honest $A$ may learn is available also to an eavesdropper listening the traffic between $A$ and $C$. Moreover, if $A$ learns $k$, it could as well do all calculations by itself, making $C$ useless.

In the following we assume that $kP$ is computed using the Fixed Base Windowing algorithm, and that $A$ is used only in the precomputation (if at all). We also assume that $|k| = n = lw$. These choices are motivated by the following reasons:

- Based on our tests, a modification of FBW seems a recommendable choice for calculating $kP$, as it combines efficiency with suitability to avoid side-channel attacks.
- If $A$ helps $C$ during the precomputation, then $A$ can send the required data without $C$'s request. On the other hand, if $A$ helps $C$ in calculations depending on $k$, then first $C$ needs to request some specific data from $A$. As a consequence more data needs to be transmitted between them. In the smart-card environment data transmission is pretty slow, so also from this point of view the more precomputation our algorithm uses the better.
- If $C$'s help request depends on $k$, there is always a risk that some information on $k$ gets revealed. In our experience it seems difficult to use $A$ during computations that depend on $k$.

We represent the complexity of an algorithm as a triple $(s, m, i)$, where $s$, $m$ and $i$ stand for the number of squarings, multiplications and inversions, respectively. For example, the complexity of both doubling and ordinary addition on elliptic curves is $(1, 2, 1)$.

## 3.1 Model I

This is the traditional model, where $C$ needs to do all computations by itself. Looking at algorithm 1 it is easy to count the complexity. In the precomputation we need $n$ doublings, so the complexity is $(n, 2n, n) = (lw, 2lw, lw)$. In the actual computation we need $c_k = l + 2^w - 1$ additions. Thus, the total complexity is $(lw + c_k, 2(lw + c_k), lw + c_k)$.

## 3.2 Model II

Let us now discuss how $C$ could utilize $A$'s computing power during the precomputation. It seems reasonable to reduce the problem to performing a single doubling, since computing the table points consists of consecutive doublings. So, let us look at the situation where $A$ and $C$ need to calculate the point $Q = 2P$. We denote $P = (x, y)$ and $Q = (u, v)$.

As $A$ is not trusted, $C$ must be able to check that every piece of information given by $A$ is correct. Therefore, a precondition for the success of this model is to find operations such that performing them is more costly than verifying the results.

In [Knu] it was shown that, under certain conditions, point halving costs less than point doubling. Thus, one possible idea is that $A$ simply gives $C$ the point $Q = 2P$, and $C$ verifies this by checking that $\frac{1}{2}Q = P$. However, this approach has some problems. An efficient point halving algorithm requires relatively much memory, which is not acceptable in our main application where $C$ is a smart card. More importantly, if $C$ does have the necessary resources for efficient point halving, it would be best exploited by substituting a "halve-and-add" algorithm (see [Knu]) for the "double-and-add" algorithm in Model I.

In the following we present two ideas, both of which are based on a simple observation: the result of a field inversion is easily verified by performing one field multiplication. Probably in any implementation of a binary field, multiplication costs reasonably less than an inversion. Further, let us point out that using the doubling formula involves computing the value $\theta = x + \frac{y}{x}$. These facts together motivate us to suggest a scheme, where $A$ after computing the value of $\theta$ hands it over to $C$, who first checks its validity and then makes use of it in computing the coordinates of $Q$.

**Algorithm 2**

| | | |
|---|---|---|
| $A \to C$ | $\theta = x + \frac{y}{x}$ | |
| $C$ | $x(\theta + x) = y?$ | ($*$ checking $\theta$ $*$) |
| $C$ | $u = \theta^2 + \theta + a$ | |
| $C$ | $v = \theta(x + u) + u + y$ | |

Computing the complexity of Algorithm 2 is straightforward. $C$ needs one multiplication to verify that $A$ has given the correct $\theta$; computing $u$ and

$v$ requires one squaring and a multiplication, resp. This sums up to $(1,2,0)$ for one doubling and $(lw, 2lw, 0)$ for all precomputation. Altogether we save time equal to that needed for $lw$ inversions.

Our second algorithm for model II is based on the idea of representing a point as pair $(x,\theta)$ instead of $(x,y)$ (where $\theta = x + \frac{y}{x}$). If $P = (x,y) \sim (x,\theta)$ and $Q = 2P = (u,v) \sim (u,\theta')$ then, by the doubling formula:

$$\begin{aligned}
u &= \theta^2 + \theta + a \\
\theta' &= u + vu^{-1} = u + (\theta(x+u) + u + y) \cdot u^{-1} \\
&= u + (\theta(x+u) + u + x(\theta + x)) \cdot u^{-1} \\
&= u + (u(\theta+1) + x^2) \cdot u^{-1} \\
&= \theta + u + 1 + x^2 u^{-1}
\end{aligned}$$

The complexity of one doubling using this method is $(1,2,1)$. Thus, there would be no time savings if $C$ doubled points this way. But if $C$ receives the value $\theta'$ from $A$, then $C$ only has to check its validity by making sure that $x^2 = u(\theta' + \theta + u + 1)$. The cost of the validation operation is $(1,1,0)$, and computing $u$ requires one additional squaring.

**Algorithm 3**

$$\begin{array}{cl}
C & u = \theta^2 + \theta + a \\
A \to C & \theta' = u + vu^{-1} \\
C & x^2 = u(\theta' + \theta + u + 1)?
\end{array}$$

In the computation of $kP$ we need the actual $y$-coordinate of every $w$th table element. Since $y = x(\theta + x)$, this costs an additional multiplication per element. Altogether, the complexity of precomputation will then be $(2lw, lw+l, 0)$. We notice that algorithm 3 is more efficient than algorithm 2, if $w$ squarings in the base field can be computed faster $w-1$ multiplications.

### Model III

Analyzing this model is trivial. As $A$ is trusted, $A$ can simply give the whole precomputation table for $C$, who only needs to do computations depending on $k$. Thus $C$'s total complexity in this case is $(c_k, 2c_k, c_k)$.

It should be noted, however, that this model is quite dangerous. In the following section we consider the risks.

## 4 What if $A$ cheats?

In this section we consider what damage can be done if $C$ trusts a cheating $A$ in our model III above. The obvious consequence is that $C$'s calculations are then incorrect, which in itself might be very harmful. But the real danger is that the false result might reveal some secret information.

Assume again that $C$ is trying to compute $kP$, where $k$ is secret and $P$ is public. Suppose that $A$ gives $C$ a precomputation table $T$, and $C$ (trusting $A$) computes the point $X_T = \sum_{i=0}^{l-1} k_i T[i]$ using the FBW algorithm. The point $X_T$ is again public information, so also $A$ learns it. From $A$'s point of view $k$ is a uniformly distributed random variable. Theoretically, the information that $X_T$ reveals about $k$ is $I(X_T, k) = H(X_T) - H(X_T \mid k)$, where $H(Y)$ is the entropy of a random variable $Y$: $H(y) = \sum_{y \in Y} p(y) \log_2(p(y))$. In our case $X_T$ is completely determined by $k$, thus $H(X_T \mid k) = 0$ and $I(X_T \mid k) = H(X_T)$.

For simplicity, suppose that $k$ is an integer selected uniformly from all integers of length $n = lw$: $0 \leq k < 2^n$.

Suppose that $A$ has been honest, so the table $T$ is correct. Then $X_T = kP$ and it is easy to verify that $H(X_T) = n$. In other words, $X_T$ completely determines $k$ (as it should). The problem is that to extract the available information $A$ needs to compute the discrete logarithm.

On the other hand, assume that $A$ has given $C$ a table consisting of only one non-zero point $T[i]$. Then $X_T = k_i T[i]$ and $H(X_T) = w$. This means that $X_T$ reveals $w$ bits of information about $k$. Again, to extract this information $A$ needs to compute the discrete logarithm. But in this case it is trivial, as there are only $2^w$ alternatives.

There is a natural generalization of this attack. $A$ can select $T$ to consist of a "small" number of non-zero points. For example, suppose that $T[i_0 + i] = 2^{iw} \cdot T[i_0]$ for some $i_0$ and $i = 0, \ldots, j-1$ and the rest $T[i] = \mathcal{O}$. Then $H(X_T) = jw$ and, if $j$ is small enough, $A$ can learn the secret bits of $k_{i_0}, \ldots, k_{i_0+j-1}$. The optimal choice for $j$ depends on $A$'s computational resources.

The attacks described above can be avoided if $C$ checks that the table elements $T[i]$ are all non-zero. Unfortunately this is of little help, as we can modify the attack in so many ways that $C$ cannot possibly rule out all alternatives. For example, $A$ can choose all the table points as a small multiple of some point $R$. If $T[i] = iR$, then $X_T = (\sum i k_i) R$. It can be computed that $H(X_T) \approx 11,5$ if $n = 160$ and $w = 4$.

From the above it follows that, unless $C$ is absolutely certain that $A$ is honest, $C$ should avoid our model III.

Let us finally consider a setup somewhere between models II and III. Suppose that, as in algorithm 2, $A$ gives $C$ the values $\theta$, but $C$ does not verify their validity (or alternatively verifies only some of them at random). But not even this idea seems too promising. If $A$ manages to give $C$ two incorrect $\theta$'s in succession, then $A$ has a good chance to force the table elements $T[i]$ to any point $(\alpha, \beta)$ it desires. For example, $A$ could make $T$ to repeat itself by forcing $T[\frac{l}{2}] = T[0]$, whence $T[\frac{l}{2} + 1] = T[1]$ and so on. In this case it is easy to compute that $I(X_T, k) < \frac{n}{2} + w$. In other words, the security provided by $k$ is roughly halved.

Let us describe how $A$ can cheat. Denote by $\varphi(P, \theta)$ the point computed by algorithm 2 without the verification of $\theta$. Hence, if $\theta$ is correct, $\varphi(P, \theta) = 2P$. Given a starting point $P_0 = (x, y)$ and a target point $P_2 = (\alpha, \beta)$, $A$'s task is to find such values $z$ and $t$ that $\varphi(P_1, t) = P_2$, where $P_1 = (u, v) = \varphi(P_0, z)$. Necessarily $t$ should satisfy the equation $t^2 + t + a = \alpha$. This is easily done if $\mathrm{Tr}(a + \alpha) = 0$. It remains to select $z$ such that the resulting $y$-coordinate will be $\beta$. Applying the doubling formulae we obtain

$$\begin{aligned}
\beta &= t(u + \alpha) + \alpha + v \\
&= t(z^2 + z + a + \alpha) + \alpha + \\
&\quad z(x + z^2 + z + a) + z^2 + z + a + y \\
&= z^3 + tz^2 + (t + x + a + 1)z + (t(a + \alpha) + a + y + \alpha).
\end{aligned}$$

Thus, if $A$ can solve $z$ from the equation above, $A$'s chances for a successful cheating are good. $A$'s task is to solve a polynomial equation of third degree in a binary field. The number of solutions is always either 0, 1 or 3. The following lemma from [BRS] tells us that the equation has at least one solution with probability greater than 50%. In the same article an efficient method to calculate the solution is given.

**Lemma.** The equation $z^3 + c_1 z^2 + c_2 z + c_3 = 0$ ($c_i \in \mathbf{F}_{2^m}$) has exactly one solution in $\mathbf{F}_{2^m}$, if and only if $\mathrm{Tr}(\frac{(c_2 + c_1^2)^{3/2}}{c_3 + c_1 c_2}) \neq \mathrm{Tr}(1)$.

# 5 An example

To get a better understanding on the benefits of distributed computation we consider a concrete example. Currently a 1024 bit RSA modulus is considered to provide adequate security for most applications. The same level of security using elliptic curve cryptography is achieved if the order of the group $E(\mathbf{F}_q)$ is roughly $2^{160}$. Then the length of $k$'s binary representation is (approximately) 160 bits, and, from the point of view of FBW's efficiency, the optimal size for the window ($w$) is 3–5.

Thus, assume that $C$ is calculating a point $kP$ on a curve over a field of size $2^{160}$, and that an untrusted $A$ helps $C$ (model II). Suppose that $C$ uses the FBW algorithm with $w = 4$ and $l = 40$. To measure the possible time savings we need to know the relative costs of the basic operations: squaring, multiplication and inversion on the underlying field. Naturally these figures depend heavily on the available resources. We consider two different environments.

If $C$ is a personal computer, then (according to our own implementation) one multiplication takes roughly twice the time required for one squaring, and an inversion costs approximately four multiplications. Then, in model I, the fastest way for $C$ to compute the precomputation table for $P$ is to double

$P$ repeatedly using the formula given in the first section. The complexity of it is $(1, 2, 1)$, thus the relative cost for the whole precomputation is $160 + 2 \cdot 320 + 8 \cdot 160 = 2080$.

In model II we shall use algorithm 3, because squaring is cheaper than multiplication. The complexity of one doubling is then $(2, 1, 0)$, and one additional multiplication is needed for the $l$ table points. The cost for precomputation is then $320 + 2 \cdot (160 + 40) = 720$. This means that the time required for precomputation is cut to roughly one third.

To obtain the total saving we should take into account also the computation depending on $k$. This is performed similarly in both models requiring 55 point additions, and the cost of it is $(55+2\cdot 110+8\cdot 55 = 715)$. Therefore the calculation time for $kP$ is approximately halved in model II.

If $C$ is a smart card with hardwired field multiplication, then squaring and multiplication (which in this case are performed similarly) are really fast compared with inversion. In the following we assume that one inversion takes the same time as 40 multiplications. In this case it pays to use projective instead of the affine coordinates (which we have used so far) to present points. This follows, since if computations are performed using projective coordinates, then we can almost completely avoid inversions (with the cost of additional squarings and multiplications). For details we refer to [BSS], here it suffices to present a table for the complexities of point doubling and point addition using different coordinates. (In the mixed case the other summand is given in affine, the other in projective coordinates.)

| | complexity as $(s, m, i)$ | | |
|---|---|---|---|
| | affine | mixed | projective |
| Addition | $(1, 2, 1)$ | $(3, 11, 0)$ | $(5, 15, 0)$ |
| Doubling | $(1, 2, 1)$ | – | $(5, 5, 0)$ |

As we notice, the complexity of one doubling is now $(5, 5, 0)$. The time required for precomputation is thus $5 \cdot 160 + 5 \cdot 160 = 1600$. The table points are then given in projective coordinates, and the complexity of the remaining 55 point additions is $(15, 5, 0)$ each. Moreover, the result needs to be converted from projective coordinates back to affine coordinates, which requires one inversion, three multiplications and one squaring. This yields to an additional cost of $15 \cdot 55 + 5 \cdot 55 + 1 + 3 + 40 \cdot 1 = 1144$, and thus the total cost for computing $kP$ is 2744.

Using model II the most efficient precomputation is performed via affine coordinates and employing algorithm 2. The complexity of one doubling is $(1, 2, 0)$, which gives $160 + 2 \cdot 160 = 480$ for the cost of precomputation. The remaining 55 point additions should again be done using projective coordinates. Of these additions 40 belong to the mixed category, the rest 15 being purely projective. This gives a cost of $(40 \cdot 3 + 15 \cdot 5) + (40 \cdot 11 + 15 \cdot 15 + 3) + 40 \cdot 1 = 904$, as again the final result needs to be converted back to affine coordinates. The total cost is then 1384, which again is about 50% of the cost of model I.

We summarize the results in the table below. It should be remembered that the times are relative, no comparison between the PC setup and smart card setup can be done. Also, these figures give only the maximum possible savings with our methods. We have not included $A$'s computation times and, more importantly, the times required for data transfer between $A$ and $C$. The main application we have had in mind is a smart card. Unfortunately with the current technology i/o times to and from the card are too slow to exploit these methods.

| | precomputation | | | complete computation | | |
|---|---|---|---|---|---|---|
| | compl. | cost | sav. | compl. | cost | sav. |
| PC | | | | | | |
| model I | $(160, 320, 160)$ | 2080 | | $(215, 430, 215)$ | 2795 | |
| model II | $(320, 200, 0)$ | 720 | 65% | $(375, 310, 55)$ | 1435 | 49% |
| smart card | | | | | | |
| model I | $(800, 800, 0)$ | 1600 | | $(1076, 1628, 1)$ | 2744 | |
| model II | $(160, 320, 0)$ | 480 | 70% | $(356, 988, 1)$ | 1384 | 50% |

## 6 Conclusion

We have shown that in elliptic curve cryptography a device with restricted computing power can exploit external computational resources without jeopardizing the security. In theory about 50% of the computation times can be saved using the described methods. We have also considered the dangers of trusting the external device.

## References


[BRS] E. R. Berlekamp, H. Rumsey and G. Solomon, On the Solution of Algebraic Equations over Finite Fields, Information and Control 10 (1967), pp. 553-564

[BSS] I. Blake, G. Seroussi and N. Smart, Elliptic Curves in Cryptography, Cambridge University Press, Cambridge, 1999.

[Knu] Erik Woodward Knudsen, Elliptic Scalar Multiplication Using Point Halving, Advances in Cryptology - Asiacrypt'99, LNCS 1716, Springer-Verlag (1999), pp. 135-149.

[MOV] A. Menezes, P. van Oorschot and S. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, Florida, 1996.

**Turku Centre for Computer Science**
**Lemminkäisenkatu 14**
**FIN-20520 Turku**
**Finland**

http://www.tucs.abo.fi

**University of Turku**

- **Department of Mathematical Sciences**

**Åbo Akademi University**

- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

**Turku School of Economics and Business Administration**

- **Institute of Information Systems Science**