INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Project-Team SPACES*

# *Solving Problems through Algebraic Computation and Efficient Software*

*Lorraine*

THEME SYM

# Table of contents

# 1. Team

**Team Leader**
Paul Zimmermann [ research director, INRIA, HdR ]

**Administrative Assistant**
Céline Simon [ until Sep. 30th ]
Emmanuelle Deschamps [ from Oct. 1st ]

**Staff member (CNRS or INRIA)**
Pierrick Gaudry [ research scientist, CNRS ]
Guillaume Hanrot [ research scientist, INRIA, HdR ]
Vincent Lefèvre [ research scientist, INRIA, until Sep. 15th, 2006 ]
Emmanuel Thomé [ research scientist, INRIA ]
Marion Videau [ assistant professor, Université Henri Poincaré, from Sep. 1st, 2006 ]

**PhD. student**
Laurent Fousse [ MESR grant, UHP, defense on Dec. 4th, 2006 ]
Thomas Houtmann [ DGA grant, DGA, defense planned in 2007 ]

**Post-doctoral fellows**
Jean-Paul Cerri [ IUFM Lorraine, until Sep. 1st ]
Nuno Franco [ Auxiliar Professor, University of Evora, Portugal ]
Damien Stehlé [ MESR grant, UHP, until Sep. 1st ]

# 2. Overall Objectives

## 2.1. Overall Objectives

Until the beginning of 2004, the Spaces project-team was a joint project-team of INRIA Lorraine and INRIA Rocquencourt. The main objective of this team was to solve systems of polynomial equations and inequations. The focus was on algebraic methods which are more robust and frequently more efficient than purely numerical tools. The members of the team located in Paris was mostly working on the algebraic aspects, whereas the task of the members located in Nancy was to devise arithmetic tools which could enhance the efficiency of the formal method (mainly real arithmetic, but also more exotic ones, like modular or $p$-adics).

Since the beginning of 2004, the "Paris subteam" has decided to create their own project team. In the same time, due to several arrivals in the project-team, the main interest of the "Nancy subteam" has somewhat shifted towards arithmetics, algorithmic number theory and their application in cryptology. A new project-team named CACAO has been created on October 9, 2006. However, from the beginning of 2004, all work done in the Nancy part of the SPACES project should be thought of as being related to the goals of the CACAO project team. We thus choose to present the objectives of the latter project, and results obtained with respect to them. The present activity report is thus a "joint report" Spaces-Cacao.

The objectives of the project-team have been along the following lines:

- Studying the arithmetic of curves of small genus $> 1$, having in mind applications to cryptology,
- Improving the efficiency and the reliability of arithmetics in a very broad sense (i.e., the arithmetics of a wide variety of objects).

These two objectives strongly interplay. On the one hand, arithmetics are, of course, at the core of optimizing algorithms on curves, starting evidently with the arithmetic of curves themselves. On the other hand, curves can sometimes be a tool for some arithmetical problems like integer factorization.

To reach these objectives, we have isolated three key axes of work:

- **Algebraic Curves**: the main issue here is to investigate curves of small genus $> 1$ over finite fields (base field $\mathbb{F}_{p^n}$, for various $p$ and $n$), i.e., mainly: to compute in the Jacobian of a given curve, to be able to check that this variety is suitable for cryptography (cardinality, smoothness test) and to solve problems in those structures (discrete logarithm). Applications go from number theory (integer factorization) to cryptography (an alternative to RSA).
- **Arithmetics**: we consider here algorithms working on multiple-precision integers, floating-points numbers, $p$-adic numbers and finite fields. For such basic data structures, we do not expect new algorithms with better asymptotic behavior to be discovered; however, since those are first-class objects in all our computations, every speedup is most welcome, even by a factor of 2
- **Linear Algebra and Lattices**: Solving large linear systems is a key point of factoring and discrete logarithm algorithms, which we need to investigate if curves are to be applied in cryptology. Lattices are central points of the new ideas that have emerged over the very last years for several problems in computer arithmetic or discrete logarithms algorithms.

Starting Fall 2006, Spaces will lead an ANR research grant on the topic of the Number Field Sieve integer factorization algorithm. This research will be done in collaboration with the teams of LIX (École polytechnique, Palaiseau) and IECN (Nancy). The three research axes set above in the objectives of Spaces fit well in the perspective of this research project.

Another new direction of research has started since Fall 2006 with the arrival of Marion Videau, who has been hired as an assistant professor at UHP, coming from the CODES project-team (Rocquencourt). This should allow the project-team to start an axis around symmetric primitives for cryptology; this is an interesting complement to the expertise already present regarding asymmetric (and especially curve-based) primitives for cryptology.

# 3. Scientific Foundations

## 3.1. Introduction

### *3.1.1. Algebraic Curves*

Though we are interested in curves by themselves, the applications to cryptology remain a motivation of our research. Therefore, we start by introducing these applications, since they may serve as a guideline to the reader in this somewhat technical section.

*3.1.1.1. Curves and Cryptology*

The RSA cryptosystem — the *de facto* standard in public-key cryptography — requires large keys, at least 1024 bits currently. Algebraic curves offer a better level of security for a smaller key size, say 160 bits currently for elliptic curves. They are not specifically used as curves. In practice, a very general construction due to El Gamal associates to any group a cryptosystem, this cryptosystem being secure as soon as the so-called *Diffie-Hellman* problem (or its decision variant) is difficult:

$$\text{Given } g \in G,\ g^a \text{ and } g^b \text{ for some integers } a \text{ and } b, \text{ compute } g^{ab}.$$

Currently, the only way to attack this problem is to tackle the more difficult *discrete logarithm problem*:

$$\text{Given } g \in G \text{ and } g^a, \text{ find } a,$$

which, in the case of the El Gamal system, is equivalent to the so-called attack on the key (given the public part of the key, recover the secret part). We shall only discuss the discrete logarithm problem in this document, since it is widely believed that the two problems are in fact equivalent.

This problem is easy when the underlying group is $\mathbb{Z}$ or $(\mathbb{Z}/N\mathbb{Z}, +)$. Classically, multiplicative groups of finite fields are used; however, they can be attacked by algorithms very similar to those existing for factoring, and thus require the same key-size to ensure security.

A trend initiated by Koblitz and Miller and followed by many others is to use as "cryptographic groups" the group ("Jacobian") associated by classical arithmetical geometry to a given algebraic curve.

To use such a group for cryptographic applications, the key algorithmic points are the following:

- have an explicit description of the group and the group operation, as efficient as possible (the speed of ciphering and deciphering being directly linked to the efficiency of the group operation);
- undertake an as thorough as possible study of the security offered by those groups.

The second point should again be split into two steps: study of the behavior of the group under "generic attacks" (avoiding small cardinality, avoiding cardinalities with no large prime factor), and trying to devise "ad hoc" attacks. The first step amounts more or less to being able to compute the cardinality of the group; the second one to try as hard as possible to find a way to compute discrete logarithms in this group.

This section now proceeds as follows; we introduce the basic objects (curves and Jacobians) and their properties relevant to the following problems: group structure and arithmetic, cardinality, discrete logarithm.

Finally, and in a somewhat independent way, curves and their Jacobians can be used for integer factorization; we shall also review that point.

*3.1.1.2. Jacobian*

A central role is played by a certain algebraic variety of higher dimension associated to a given curve $C$, its Jacobian $J(C)$, which comes with a natural group structure. We shall not define it, but rather state its most important properties:

1. $C$ embeds as a sub-variety of $J(C)$,
2. $J(C)$ is an abelian variety, i.e., has an (abelian) group structure such that group operations (addition, inversion) can be written as rational functions of the coordinates.
3. if $C$ has genus $g$, $J(C)$ is a variety of dimension $g$ (note that in full generality one only knows how to embed it in a space of dimension $2^{2g}$, i.e. to give many equations in $2^{2g}$ variables rather than, for instance, one equation in $g+1$ variables).

The most important feature of the Jacobian is the fact that it comes with a natural group structure, which is the key point for its uses in applications to primality, factorization, and cryptology.

*3.1.1.3. Curves over Finite Fields*

We intend to focus on the case $\mathbb{K} = \mathbb{F}_q$ and its extensions, with subsidiarily a study of the cases where $\mathbb{K}$ is a number field or a completion of a number field, since those happen to be related to the previous one by reduction/lifting techniques.

In this setting, the situation is rather rigid; the cardinality of the curve over any $g$ extension fields of the base field determines the cardinality over all extensions, and the cardinality of the Jacobian. We also have "sharp" estimates for the cardinality, namely $|\sharp C(\mathbb{F}_q^k) - (q^k+1)| \leq 2g\sqrt{q^k}$ and $(\sqrt{q}-1)^{2g} \leq \sharp J(\mathbb{F}_q) \leq (\sqrt{q}+1)^{2g}$ (the so-called Weil bounds).

The cardinalities have several interpretations, which usually yield different strategies for computing them. We shall review them in an informal way in section 3.1.1.5.

*3.1.1.4. Discrete Logarithm*

In this part, we generalize slightly the setting, since we shall also discuss later some aspects on discrete logarithms over finite fields. We shall hence assume that $G$ is an abelian group, where we want to solve the equation

$$g^x = h$$

where $g, h$ are given elements from $G$, and the unknown $x$ is an integer. This is known as the *discrete logarithm* problem (DL for short). A first remark, due to Nechaev [49], is that if one uses only operations in the group, one needs at least $(\sharp G)^{1/2}$ operations to compute a discrete logarithm. One of the quests of cryptology is finding a so-called "Nechaev group", for which there are provably no algorithms for computing discrete logarithms faster than $(\sharp G)^{1/2}$; it currently appears that elliptic curves are the best candidates to be Nechaev groups, hence the interest in cryptology.

On the other hand, two classical algorithms (Pollard's $\rho$ method and Shanks' baby-step giant-step) allow one to compute a discrete logarithm in any group $G$ in time $O(\sharp G^{1/2})$. The complexity of the "general discrete logarithm" is thus completely known. However, for a family of groups or even a specific group, faster algorithms might exist. We shall discuss some of those algorithms in the sequel.

*3.1.1.5. State of the art*

3.1.1.5.1. Arithmetic in the Jacobian

As a group, the Jacobian is defined as a quotient of the free group generated by points; as any definition based on a quotient, it is not very tractable for explicit computations. It is necessary to devise a specific representation of elements and specific algorithms to deal with computations in the Jacobian. Though general methods exist [39], the most interesting methods usually take advantage of the specific curve one is dealing with, or even of the specific model of the curve to get a more efficient algorithm.

In the case of elliptic curves, the problem is quite easy; the classical chord-and-tangent rule yields by simple calculations easy-to-implement formulas. One can still improve somewhat upon those formulas. The situation however is quite different as soon as higher genus curves are involved.

In the case of hyperelliptic curves, a now classical algorithm due to Cantor [28] explains how to implement efficiently arithmetic in their Jacobian; numerous improvements have been obtained since, including explicit formulas [43], [51] which are more efficient in practice than Cantor's algorithm.

Another family of curves has received interest from the cryptology community in recent years, namely the $C_{ab}$ family. In that case, algorithms have been obtained by Arita [24] using Gröbner bases computations, then for a sub-family, a more efficient method was devised by Galbraith, Paulus and Smart [30] and a common setting was then found by Harasawa and Suzuki [38]. Since then, more efficient algorithms were obtained by using suitable orderings for the Gröbner basis computation in Arita's method, and explicit formulas derived in some cases [26]. However, recent work by Diem and Thomé almost completely dismisses non-hyperelliptic $C_{ab}$ curves, as far as cryptology is concerned.

3.1.1.5.2. Computing the Cardinality

The question of point counting over finite fields is of central importance for applications to cryptography, see Section 4.1. Recall that we are given an algebraic curve $C$ of genus $g$, over a finite field $\mathbb{F}_q$, and we would like to count the number of points of the Jacobian of this curve.

First, for the sake of completeness, we should mention two classical ways to somewhat reverse the problem, i.e., to construct the curve and its number of points at the same time: using *Koblitz curves* and *complex multiplication*.

Those two methods are extremely efficient, especially the first one, but the main drawback is that they introduce some unnecessary structure in the curves they construct; in particular, Koblitz's method yields curves with a large ring of automorphisms. This can be used to speed up discrete logarithm computations, and should thus be considered as a weakness from the cryptographic point of view.

Let us now turn to actual point counting algorithms. Hasse-Weil's theorem states that computing a certain polynomial $P(t)$ is enough to obtain the cardinality (in particular, the cardinality of the jacobian over the base field is exactly $P(1)$). There are several interpretations for the polynomial $P(t)$, which yield different strategies for computing it:

- $\ell$-adic characterization: Schoof's algorithm [52] and its improvements and extensions, is especially suitable in large characteristic. It is well understood for elliptic curves; the hyperelliptic case, though already studied [33], [32], would still benefit from significant improvement.

- $p$-adic characterization: Lift the curve to an extension of $\mathbb{Q}_p$, and compute $P(t)$ over this extension. This is the core of Satoh's method and its AGM variants. This method is the most efficient in small characteristic.
- Monsky-Washnitzer characterization and Kedlaya's algorithm. Again, this is suitable for small or medium primes. This method is interesting by its generality.

3.1.1.5.3. Discrete Logarithm

In the case of Jacobians of curves, at the time being, no other general algorithm is known. This is the key interest of curves for cryptology, and the reason for which rather small keys give the same level of security that much larger keys in the case of RSA.

However, many ad hoc methods, which exploit (or demonstrate) the weakness of certain families of curves, exist. Let us quote Pohlig-Hellman method (if the cardinality of the group is smooth, hence the interest in computing the cardinality!), Index calculus (for discrete logarithms over finite fields, leading to subexponential complexities), and some weaker instances of curves (trace 0, supersingular, Weil descent, small extension fields). For curves, higher genus have been showed to be weaker than generic groups for $g \geq 5$ by Gaudry [31] and then by further work for $g \geq 3$, see Section 6.2.

## 3.2. Linear Algebra

### *3.2.1. Huge Linear Systems*

Huge linear systems are frequently encountered as last steps of "index-calculus" based algorithms. Those systems correspond to a particular presentation of the underlying group by generators and relations; they are thus always defined on a base ring which is $\mathbb{Z}$ modulo the exponent of the group, typically $\mathbb{Z}/2\mathbb{Z}$ in the case of factorization, $\mathbb{Z}/(q^n-1)\mathbb{Z}$ when trying to solve a discrete logarithm problem over $\mathbb{F}_{q^n}^*$.

Those systems are often extremely sparse, meaning that they have a very small number of non-zero coefficients.

The classical, naive elimination algorithm of Gauss yields a complexity of $O(n^3)$, when the matrix considered has size $n \times n$. However, if we assume that we can perform a matrix multiplication in time $O(n^\omega)$, algorithms exist which lower this complexity to $O(n^\omega)$. Furthermore, if we make assumptions on our matrix (mainly that it is sparse, meaning that a matrix-vector product can be computed in time $O(n^\theta)$ for some $\theta < 2$), then specialized algorithms (Lanczós, Wiedemann [59]) relying only on evaluation of matrix-vector products yield a complexity of $O(n^{1+\theta})$, typically $O(n^2)$ for the very sparse matrices ($\theta = 1$) that we often encounter.

### *3.2.2. Lattices*

Many problems described in the other sections, but also numerous problems in computer algebra or algorithmic number theory, involve at some step the solution of a linear problem or the search for a short linear combination of vectors lying in a finite-dimensional Euclidean space. As examples of this, we could cite factoring and discrete logarithms methods for the former, finding worst cases for the Table Maker's Dilemma in computer arithmetic for the latter (see Section 3.3.5).

The important problem in that setting is, given a "bad" basis of a lattice, to find a "good" one. By good, we mean that it consists of short, almost orthogonal vectors. This is a difficult problem in general, since finding the shortest nonzero vector is already NP-hard, under probabilistic reductions.

In 1982, Lenstra, Lenstra, and Lovász [46] defined the notion of a LLL-reduced basis and described an algorithm to compute such a basis in polynomial time, namely $O(n^2 \log M)$ linear algebra steps (of type matrix-vector multiplication), or $O(n^4 \log M)$ operations [53] on coefficients at most $O(n \log M)$, therefore giving a $O(n^6 \log^3 M)$ bit complexity if the underlying arithmetic is naive.

### *3.2.3. State of the art*

*3.2.3.1. Large Linear Systems*

Recall that the systems we are dealing with are usually systems with coefficients in a finite ring, which can be either small ($\mathbb{F}_2$), or quite a large ring.

*3.2.3.2. Lanczós' method*

Given a symmetric matrix $A$ and a vector $x$, Lanczós' method computes, using Gram-Schmidt process, an orthogonal basis $(w_1, \cdots, w_n)$ of the subspace generated by $\{x, Ax, \cdots, A^n x\}$ for the scalar product $[x, y] = (x|Ay)$. As soon as one finds an isotropic vector $w_i$, i.e., $[w_i, w_i] = 0$, one has $w_i^t A w_i = 0$. In our situation, we take $A = B^t B$, where we want to find a vector in the kernel of $B$; we thus have $(w_i B)^t B w_i = 0$. Over a finite field this does not always imply $Bw_i = 0$, but this remains true with probability close to 1 over a finite ring of large characteristic. This approach works over $\mathbb{F}_2$ as well, but with some caution.

*3.2.3.3. Wiedemann's method*

Given a matrix $A$ (not necessarily symmetric) and a vector $x$, Wiedemann's algorithm looks for a trivial linear combination of the vectors $A^i x$, $i \geq 1$. Such a relation can be written as $\sum_{i=1}^{n} a_i A^i x = 0$. Now, if $u = \sum_{i=1}^{n} A^{i-1} x$ is a nonzero vector, we have $Au = 0$, and $u$ is a vector of the kernel of $A$. The linear combination, in turn, is searched by choosing a random vector $y$ and computing the elements $\alpha_i = yA^i x$. If a relation of the type we are looking for exists, then $\alpha_i$ is a linear recurring sequence of order $n$. Given $2n$ elements of the sequence, Berlekamp-Massey's algorithm allows one to compute the coefficients of the recurrence. Thus, with $O(n)$ matrix-vectors and $O(n)$ vector-vector products, one hopes to recover a vector of the kernel. The overall complexity is thus, on average, $O(n^{1+\theta})$, as announced.

*3.2.3.4. Parallel and distributed algorithms*

Algorithms for solving large sparse linear systems have been designed with implementation, and parallelism or distribution in mind, or both. The Lanczós and Wiedemann algorithms have "block" versions [47], [29], which one can use in order to take advantage of an advanced computing facility, like a massively parallel computer, or a much cheaper resource like a computer cluster, which can be turned into an effective task force. A key problem is therefore the identification of the computational tasks which either can, or cannot be effectively spanned across many processors or machines. In the case of a computer cluster, evaluating the cost of communications between nodes taking part to the computation is of course very important. To this regard, the different algorithms (block or non-block versions, Lanczós or Wiedemann) do not compare equally. A variety of running times can be obtained depending on the exact characteristics of the input system (size, density, definition field), the number of computing nodes, and on the choice of certain parameters of the algorithms (for the block versions).

The block Wiedemann algorithm has been used by Thomé [55], [56] in the course of solving a $500,000 \times 500,000$ linear system defined over $\mathbb{F}_p$, where $p$ is a prime of 183 decimal digits. This computation was made feasible using an algorithm based on the Fast Fourier Transform (FFT), which permitted broader distribution of the computation [57].

Today, block versions of the Lanczós and Wiedemann algorithms are a necessity for who wants to solve linear systems encountered in record-size factoring problems, discrete logarithm problems, or in some other cases. Yet, a precise account on the positive and negative sides of both block algorithms, and a formulation of their preferred setting, seems to be missing.

## 3.3. Arithmetics

We consider here the following arithmetics: integers, rational numbers, integers modulo a fixed modulus $n$, finite fields, floating-point numbers and $p$-adic numbers. We can divide those numbers in two classes: *exact numbers* (integers, rationals, modular computations or finite fields), and *inexact numbers* (floating-point and $p$-adic numbers).

Algorithms on integers (respectively floating-point numbers) are very similar to those on polynomials (respectively Taylor or Laurent series). The main objective in that domain is to find new algorithms that make operations on those numbers more efficient. These new algorithms may use an alternate number representation.

### *3.3.1. Integers*

The integral types of the current processors have a width $w$ of either 32 or 64 bits. This means that, using hardware instructions, one is only able to compute modulo $2^{32}$ or $2^{64}$. An arbitrary precision integer is then usually represented under the form $n = \sum_{i=0}^{l} n_i 2^{iw}$, with $n_i$ a machine integer. In algorithmic terms, it means that a multiprecision integer is an array of machine integers. Naive operations can then be defined by using the classical "schoolbook methods" in base $2^w$, with linear complexity in the case of addition and subtraction, and quadratic complexity in the case of division and multiplication.

### *3.3.2. Integers Modulo n and Finite Fields*

Integers modulo $n$ are usually represented by the representative of their class in the interval $[0, n-1]$ or sometimes $]-n/2, n/2]$.

Addition, subtraction and multiplication are obtained from the corresponding operation over the integers, after reduction modulo $n$. This means that after each operation, a reduction modulo $n$ must be performed. This is not very costly in the case of addition and subtraction (where it implies a single test and half the time another addition of subtraction), but implies a division in the case of multiplication.

The modular division is a completely different operation, and amounts to compute a so-called extended gcd of $x$ and $n$, i.e., a pair $a, b$ with $ax + bn = 1$. This is classically performed by the Euclidean algorithm or one of its variants, and is thus, in practice, by far the most costly operation. Many improvements in low-level algorithms are obtained by choosing suitable representations of objects which avoid divisions modulo $n$.

Finite fields can be separated in two types. Prime fields correspond to the integers modulo $n$ for prime $n$. Extension fields are algebraic extensions of those prime fields, i.e., $\mathbb{Z}/p\mathbb{Z}[X]$ modulo an irreducible polynomial $P(X)$. Elements of a non-prime finite field are thus often represented as polynomials of elements of a prime field. This means that ideas from polynomial arithmetic can, and should be used.

A difficult case is the case when $p^{\deg P}$ (the cardinality of the field) is large whereas neither $p$ nor $\deg P$ really are. The case where $p$ is large is indeed a classical case where we have to deal with arithmetic with large integers, and fast algorithms exist in that case. The case where $p$ is small and $\deg P$ large corresponds to the realm of fast polynomial arithmetic. However, in the "middle range", neither $p$ nor $\deg P$ are large enough to justify the use of fast techniques. This is also at the core of some technical theoretical difficulties.

### *3.3.3. p-adic Numbers*

A $p$-adic number is defined as the formal limit of a sequence $(x_n)$ of integers such that $x_i = x_{i+1} \mod p^i$. One could think of it as a formal series $\sum_{n\geq 0} a_n p^n$, with $a_n \in \{0, \cdots, p-1\}$, though alternative representations are sometimes more efficient for some computations. In particular, a $p$-adic number given to the precision $n$ is simply an element of $\mathbb{Z}/p^n\mathbb{Z}$.

The $p$-adic numbers offer the capability of lifting information known in a finite field to a field of characteristic zero, keeping some structure information at the same time. They are extensively used by many algorithms in computer algebra and algorithms related to algebraic curves, together with their extensions.

When we are trying to lift information from a nonprime finite field, say $\mathbb{F}_q$ for $q = p^n$, we are led to introduce algebraic extensions of $\mathbb{Z}_p$; algebraic extensions of the $p$-adics can be of two types, *unramified extensions* and *ramified extensions*; roughly speaking, ramified extensions contain fractional powers of $p$.

In practice, we are mostly interested in the case of small $p$ and unramified extensions. Of lesser importance are $p$-adic integers for large $p$, and extensions of these, because the algorithms we have in mind are generally not practical for large $p$. Yet, this is not necessarily the case for any possible $p$-adic algorithm, hence this point of view may change. At present, our application realms do not call for $p$-adic arithmetic requiring computations in ramified extensions, but this may change in the future as well.

### *3.3.4. Floating-point Numbers and the IEEE-754 Standard*

When discussing inexact types, one stumbles very quickly on two critical difficulties:

- since approximation is inherent to the manipulation of inexact types, how should approximation be performed? This amounts to defining the (necessarily) finite set of numbers that can be exactly represented (format),
- even if the two operands of an operation can be exactly represented, in general the result cannot be. How should one define the result of an operation (rounding)? This is the key for a precise and portable semantics of floating-point computations.

From now on, we shall focus on the floating-point numbers, which are the main inexact data type, at least from the practical point of view.

*3.3.4.1. Formats.*

A floating-point format is a quadruple $(\beta, n, E_{min}, E_{max})$; a floating-point number in that format is of the form

$$\pm(b_0.b_1...b_{n-1})\cdot\beta^e,$$

where $\beta$ is the *base* — usually 2 or 10 —, $n$ is the *significand width*, $e \in [E_{min}, E_{max}]$ is the *exponent*, and the $b_i$ are the *digits*, $0 \leq b_i < \beta$. The IEEE-754 standard defines four binary floating-point formats (single precision, single-extended, double precision, double-extended), the single-extended format being obsolete:

| format | total width | $\beta$ | significand width | $E_{min}$ | $E_{max}$ |
|---|---|---|---|---|---|
| single | 32 | 2 | 24 | $-126$ | $+127$ |
| double | 64 | 2 | 53 | $-1022$ | $+1023$ |
| double-extended | $\geq 79$ | 2 | $\geq 64$ | $\leq -16382$ | $\geq +16383$ |

The on-going revision (754r) forgets about the single-extended and double-extended formats, and defines a new quadruple precision format (binary128). It also defines new decimal formats:

| format | total width | $\beta$ | significand width | $E_{min}$ | $E_{max}$ |
|---|---|---|---|---|---|
| binary128 | 128 | 2 | 113 | $-16382$ | $+16383$ |
| decimal32 | 32 | 10 | 7 | $-95$ | $+96$ |
| decimal64 | 64 | 10 | 16 | $-383$ | $+384$ |
| decimal128 | 128 | 10 | 34 | $-6143$ | $+6144$ |

*3.3.4.2. Rounding.*

The IEEE-754 standard defines four rounding modes: rounding to zero, to $+\infty$, to $-\infty$, and to nearest-even. It requires that any of the four basic arithmetic operations $(+, -, \times, \div)$, and the square root, must be *correctly rounded*, i.e., the rounded value of $a \diamond b$ for $\diamond \in \{+, -, \times, \div\}$ must be the closest one to the exact value (assuming that the inputs are exact) — as if one were using infinite precision — according to the rounding direction. (In case of an exact result lying exactly in the middle of two consecutive machine numbers, the nearest-even mode chooses that with an even mantissa, i.e., ending with $b_{n-1} = 0$ in binary.)

### 3.3.5. The Table Maker's Dilemma

Let $f$ be a mathematical function (for example the exponential, the logarithm, or a trigonometric function), and a given floating-point format $(\beta, n, E_{min}, E_{max})$. Assume $\beta = 2$, i.e., a binary format for simplicity. Given a floating-point number $x$ in that format, we want to determine the floating-point number $y$ in that format — or in another output format — that is closest to $f(x)$ for a given rounding mode. In that case, we say that $y \longleftarrow f(x)$ is *correctly rounded*. The problem here is that we cannot compute an infinite number of bits of $f(x)$. All we can do is to compute an approximation $z$ to $f(x)$ on $m > n$ bits, with an error bounded by one *ulp* (unit in last place). Consider for example the arc-tangent function, with the double-precision number $x = 4621447055448553 \cdot 2^{-11}$, and rounding to nearest. We have in binary:

$$\begin{aligned}\arctan x \;=\;& 1.10010010000111111011010101000100010000100101010011 00\\ & 1000000000000000000000000000000000000000000000000111011...,\end{aligned}$$

where the first line contains 53 significant bits, and the second one has 45 consecutive zeros. If $m \leq 99$, we'll get as approximation $z = \underbrace{1.100...100}_{53}\ 1000...000$, which is exactly the middle of two double-precision numbers, and therefore we will not be able to determine the correct rounding of $\arctan x$. We say that $x$ is a *worst case* for the arctan function and rounding to nearest. Since a given format contains a finite number of numbers — at most $2^{64}$ for double-precision —, the maximal working precision $m$ required for any $x$ in that format is finite. The Table Maker's Dilemma (TMD for short) consists in determining that maximal working precision $m_{max}$ needed, which depends on $f$, the format and the rounding mode, and possibly the corresponding worst cases $x$. Once we know $m_{max}$, we can design an efficient routine to correctly round $f$ as follows: (i) compute a $m_{max}$-bit approximation $z$ to $f(x)$, with an error of at most one ulp, (ii) round $z$.

### *3.3.6. State of the art*

#### *3.3.6.1. Integers*

Most basic algorithms for integers are believed to be optimal, up to constant factors. The main goal here is thus to save on those constant factors. For the multiplication, one challenge is to find the best algorithm for each input size; since the thresholds between the different algorithms (naive, Karatsuba, Toom-Cook, FFT) are machine-dependent, there is no theoretical answer to that question. The same holds for the problem of finding which kind of FFT (Mersenne, Fermat, complex, Discrete Weighted Transform or DWT) is the fastest one for a given application or input size.

For the division, it is well known that it can be performed — as any algebraic operation — in a constant times that of the corresponding multiplication: for example, a $n \times n$ product corresponds to a $(2n)/n$ division. One main challenge is to decrease that constant factor, say $d$. In the naive (quadratic) range, we have $d = 1$, but already in the Karatsuba range, the best known implementation has $d = 2$ [27]. (Van der Hoeven [60] gives an algorithm with $d = 1$, however its implementation seems tricky, and its memory usage is superlinear.)

#### *3.3.6.2. Floating-point numbers*

Algorithms for floating-point numbers make great use from those for integers. Indeed, a binary floating-point number may be represented as an integer significand multiplied by $2^e$. Multiplication of two floating-point numbers therefore reduces to the product of their significands; this product is in fact a *short product*, since only the high part is needed (assuming all numbers have the same precision). Despite some recent theoretical advances [37] [48], no great practical speedup has been obtained so far for the computation of a short product with respect to the corresponding plain product. The same holds for division, though extension of the ideas of the middle-product [36] to floating-point numbers might allow one to gain somewhat on division.

#### *3.3.6.3. Integers modulo n.*

A special case of integer division is when the divisor $n$ is constant. This happens in particular in modular or finite field computations (discrete logarithm and factorization via ECM for instance). There are basically two kinds of algorithms in that case: (i) Barrett's division [25] precomputes an approximation to $1/n$, which is used to get an approximation to the quotient, which after a second product yields an approximate remainder, (ii) Montgomery's reduction precomputes $-1/n \mod \beta^k$ (where the input $n$ has $k$ words in base $\beta$) which gives in two products the value of $c\beta^{-k} \mod n$, for $c$ having $2k$ words in base $\beta$. Both algorithms perform two products with operands of size equal to the size of $n$. These products are in fact short products, but according to the above remark, the global cost is close to that of two plain products. A speedup can be obtained in the FFT range, where the second product (to obtain the remainder) produces a known high part (resp. low part) in Barrett's division (resp. Montgomery's reduction); using the fact that the FFT computes that product modulo $2^m \pm 1$, one can save a factor of two for that product, with a global gain of 25%. Together with caching the transform of the input $n$ and of its approximate inverse, one approaches $d = 1$. These ideas still need to be implemented in common multiple-precision software.

#### *3.3.6.4. p-adic numbers.*

Recently, a large number of new "$p$-adic" algorithms for solving very concrete problems have been designed, notably for counting points on algebraic varieties defined over finite fields. The application of such algorithms to coding theory or cryptology is immediate, as this is a considerable aid for quickly setting up elliptic curve cryptosystems, or for finding good codes. Some of these algorithms have been listed in Section 3.1.1.5.2.

In such algorithms, computations are carried out in "$p$-adic structures", but this vague wording reflects a relatively wide variety of mathematical structures (not unrelated to the underlying finite field, of course). We are frequently led to computing in the ring of 2-adic integers, which can be regarded as the integers modulo $2^n$ for some variable precision $n$. Also, just as extensions of $\mathbb{F}_2$ are very common in computer algebra in general, the ring of integers of unramified extensions of 2-adic numbers plays an important role.

*3.3.6.5. The Table Maker's dilemma.*

Some instances of the TMD are easy. For example, for an algebraic function of total degree $d$, we get an upper bound of $m_{max} \leq dn + O(1)$ [42], which is attained when $d = 2$. Another easy case is the base conversion, where the TMD reduces to $O(E_{max} - E_{min})$ computations of continued fractions [35].

However, in general, and especially for non-algebraic functions, the TMD is a difficult problem, because we know no rigorous upper bound for $m$, or the corresponding upper bound is much too large. However, a quick-and-dirty statistical analysis shows that for a $n$-bit input format (including the exponent bits if needed), the worst case is about $m \approx 2n$. But to determine a rigorous bound, the only known methods are based on exhaustive search. Basically, they compute a $2n$-bit approximation to $f(x)$ for every $x$ in the given format, and see how many consecutive zeros or ones appear after (or from) the round bit. This naive approach has complexity $\Theta(2^n)$. Fortunately, faster — but still exponential — methods do exist. The first one is Lefèvre's algorithm [45], [44], with a complexity of $2^{2n/3+\varepsilon}$. An improved algorithm of complexity $2^{4n/7+\varepsilon}$ is given in [54].

# 4. Application Domains

## 4.1. Cryptology

The main application domain of our project is cryptology. As it has been mentioned several times in this document, curves have taken an increasing importance in cryptology over the last ten years. Various works have shown the usability and the usefulness of elliptic curves in cryptology, standards [41] and real-world applications.

We collaborate with the TANC project-team from INRIA Futurs and École polytechnique on the study of the suitability of higher genus curves to cryptography (mainly hyperelliptic curves of genus two, three) This implies some work on three concrete objectives, which are of course highly linked with our main theoretical objectives:

1. improvement of the arithmetic of those curves, so as to guarantee fast enough ciphering-deciphering;
2. fast key generation. This rests on fast computations in the curve and in the ability to quickly compute the cardinality. Another approach (complex multiplication) is followed by TANC.
3. study of the security of the algorithmic primitives relying on curves. This implies attempts at solving discrete logarithms problems in Jacobians using the best known techniques, so as to determine the right key-size.

We also have connections to cryptology through the study and development of the integer LLL algorithm, which is one of the favourite tools to cryptanalyse public-key cryptosystems. For example, we can mention the cryptanalysis of knapsack-based cryptosystems, the cryptanalyses of some fast variants of RSA, the cryptanalyses of fast variants of signature schemes such as DSA or Elgamal, or the attacks against lattice based cryptosystems like NTRU. The use of floating-point arithmetic within this algorithm dramatically speeds it up, which renders the afore-mentioned cryptanalyses more feasible.

## 4.2. Computational Number Theory Systems

We have strong ties with several computational number theory systems, and code written by members of the project-team can be found in the Magma software and in the Pari/GP software.

### *4.2.1. Magma*

Magma (http://magma.maths.usyd.edu.au/magma/) is the leading computational number theory software. It also has some features of computer algebra (algebraic geometry, polynomial system solving) but not all of what is expected of a computer algebra system. It is developed by the team of John Cannon in Sydney, and while it describes itself as a non-commercial system, it is sold to cover the development cost, porting and maintaining.

In many areas, programs originating from very specialized research works are ported into MAGMA by their authors, who are invited to Sydney for this purpose. Several members of our project-team have already visited Sydney; there has even been an official collaboration supported by the French embassy in Sydney involving people from 3 groups in France (Toulouse, Palaiseau, Nancy) in 2000-2002. Gaudry, Thomé, and Zimmermann have had the occasion to visit the MAGMA group in Sydney in 2001 in order to implement within MAGMA some code they had written for their personal research (on computing the cardinality of Jacobians of hyperelliptic curves, on computing discrete logarithms in $\mathbb{F}_{2^n}$, and on the ECM factorization algorithm, respectively). Zimmermann visited again the MAGMA group in April 2005 to help integrating MPFR and LIBECM into MAGMA.

The Magma system now uses MPFR (see Section 5.2) for its multiple-precision floating-point arithmetic.[1].

### *4.2.2. Pari/GP*

Pari/GP is a computational number theory system which comes with a library which can be used to access Pari functions within a C program. It has originally been developed at the Bordeaux 1 university, and is currently maintained (and expanded) by Karim Belabas, from Bordeaux University. It is free (GPL) software. We sometimes use it for validation of our algorithms.

Again, some code written by members of the project has been incorporated into Pari.

## 4.3. Arithmetics

Another indirect transfer is the usage of MPFR in GCC (*Gnu Compiler Collection*), originally for the GFORTRAN compiler[2], now in the *middle-end*[3], which is a language-independent phase. MPFR is currently used at compile-time, to convert expressions like $\sin(3.1416)$ into binary double-precision, when the rounding mode can be statically determined. Finally, we should mention another usage of our software by the GCC team: GMP-ECM is used as efficiency test for release candidates of the gcc compiler, up from version 3.3.

The MPFR library is also used by the **CGAL** software, a library for computational geometry developed at INRIA Sophia-Antipolis. The **CGAL**[4] group is currenly only using it for converting rationals to multi-precision floating-point numbers, but plans to write its own interval arithmetic atop of MPFR in the near future, since double-precision interval arithmetic quickly fails for its problems (e.g. circle intersections).

# 5. Software

## 5.1. Introduction

An important part of the research done in the SPACES project is published within software.

## 5.2. MPFR

**Keywords:** *IEEE 754*, *arbitrary precision*, *correct rounding*, *floating-point number*.

[1] https://magma.maths.usyd.edu.au/magma/export/mpfr_gmp.shtml
[2] Cf. thread *Remove GMP in favor of MPFR* at http://gcc.gnu.org/ml/fortran/2004-07/msg00005.html.
[3] http://gcc.gnu.org/bugzilla/show_bug.cgi?id=29335
[4] http://www.cgal.org

**Participants:** Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Paul Zimmermann [contact].

MPFR is one of the main pieces of software developed by the SPACES team. MPFR is a library for computing with arbitrary precision floating-point numbers, together with well-defined semantics, distributed under the LGPL license. In particular, all arithmetic operations are performed according to a rounding mode provided by the user, and all results are guaranteed correct to the last bit, according to the given rounding mode.

From September 2003 to August 2005, P. Pélissier joined the MPFR team, as a Junior technical staff, to help improve the efficiency of MPFR for small precision (up to 200 bits, in particular in double, double extended and quadruple precision). He also greatly improved the portability of the library, and added the use of LIBTOOL to enable dynamic libraries. P. Pélissier is now working for SopraGroup — a small company near Toulouse, subcontractor for Airbus Industry — on the validation of A380 commands.

In October 2005, the MPFR team took part in the "many digits" friendly competition organized by the group of Henk Barendregt at the University of Nijmegen, Netherlands[5]. The competition consisted in 24 real values, that had to be computed with the largest possible precision (up to one million digits) in the least possible time. The MPFR team won that competition, where commercial software like Maple or Mathematica were also represented.

MPFR 2.2.1 was released on November 29, 2006.

Several software systems use MPFR, for example: the KDE calculator Abakus by Michael Pyne; CGAL (Computational Geometry Algorithms Library) developed by the Geometrica team (INRIA Sophia-Antipolis); Gappa, by Guillaume Melquiond (ARENAIRE team); Genius Math Tool and the GEL language, by Jiri Lebl; GCC; Giac/Xcas, a free computer algebra system, by Bernard Parisse; the iRRAM exact arithmetic implementation from Norbert Müller (University of Trier, Germany); the Magma computational algebra system; and the Wcalc calculator by Kyle Wheeler.

Finally, a paper [11] has been written summarizing the objectives, architecture, and features of MPFR. It will appear in 2007.

## 5.3. MPC

**Keywords:** *IEEE 754*, *arbitrary precision*, *complex floating-point number*, *correct rounding*.

**Participants:** Andreas Enge, Paul Zimmermann [contact].

MPC is a complex floating-point library developed on top of the MPFR library, and distributed under the LGPL license. It is co-written with Andreas Enge (TANC team, INRIA Futurs). A complex floating-point number is represented by $x + iy$, where $x$ and $y$ are real floating-point numbers, represented using the MPFR library. The MPC library currently implements all basic arithmetic operations, and the exponential function, all with correct rounding on both the real part $x$ and the imaginary part $y$ of any result.

## 5.4. GMP-ECM

**Participants:** Laurent Fousse, Pierrick Gaudry, Paul Zimmermann [contact].

GMP-ECM is a program to factor integers using the Elliptic Curve Method. Its efficiency comes both from the use of the GNU MP library, and from the implementation of state-of-the-art algorithms. GMP-EMC contains a library (LIBECM) in addition of the binary program (ECM). The binary program is distributed under GPL, while the library is distributed under LGPL, to allow its integration into other non-GPL software. For example, the Magma computational number theory software uses LIBECM, up from version V2.12 of Magma.

[5] http://www.cs.ru.nl/~milad/manydigits/

Since October 2005 where this project moved to gforge.inria.fr, and up to September 2006, there were about 1000 downloads. According to the "table of champions" maintained by Richard Brent[6], the ten largest ECM factors were found using GMP-ECM, including the current ECM record (67 digits).

GMP-ECM is used by many mathematicians and computer scientists to factor integers, either for fun or for for real purpose; for example it can be used to prove the primality of an integer, since several primality tests require to factor a given proportion of a number [34].

## 5.5. Exhaustive Tests of the Mathematical Functions

**Participant:** Vincent Lefèvre.

The programs to search for the worst cases for the correct rounding of mathematical functions (exp, log, sin, cos, etc.) using Lefèvre's algorithm have still been improved. In particular, several steps use Maple to perform multiple-precision interval arithmetic; the Maple interface had to be completely redesigned to work with Maple 9.5, and is now provided by a separate Perl module `Maple.pm`.

The results are used:

- by us, to detect bugs in MPFR and in the GNU C library (glibc);
- by the ARENAIRE team, for their implementation of the mathematical functions with correct rounding.

## 5.6. Worst-cases of Mathematical Functions

**Participants:** Guillaume Hanrot, Damien Stehlé [contact].

Bacsel is a still evolving efficient implementation (10000 lines of C code) of the SLZ algorithm for finding worst cases of elementary functions. A no longer up-to-date version is available on http://www.loria.fr/~stehle. A release should occur before the end of 2006.

## 5.7. Floating-point LLL

**Participant:** Damien Stehlé [contact].

fpLLL is a program (10000 lines of C code) initiated as a proof-of-concept for the papier [50]. It is an efficient implementation of several variants of the program described in this paper, from a "fast" variant where the output basis is not guaranteed to be LLL-reduced (but should be on most inputs), to a completely rigorous variant. The underlying floating-point arithmetic can also be selected by the user (machine double precision, DPE, MPFR). This code is already distributed on http://www.loria.fr/~stehle, and should evolve into a library in the near future. A tailored version of this code is used in BACSEL. This program is by far more stable and efficient than its main competitors, NTL, GP/Pari. It is also more stable and efficient than Magma LLL code, but this latter code is under complete rewriting by D. Stehlé.

## 5.8. Correctly rounded quadratures

**Participant:** Laurent Fousse [contact].

CRQ is a library for arbitrary-precision numerical integration (quadrature) developped by Laurent Fousse. It is based on the MPFR library and distributed under the LGPL. Its aim is to extend the idea of correct rounding (present in the IEEE754 norm and in MPFR) to the more complex operation of numerical integration, or at least to bound its error. Its impact is currently limited as no numerical software is using it. The operation of numerical integration is commonplace in symbolic and numerical systems like Maple or Mathematica, but none of them have the goal to provide a rigorous bound on the error.

[6] http://wwwmaths.anu.edu.au/~brent/ftp/champs.txt

## 5.9. Local fields

**Participant:** Emmanuel Thomé [contact].

Mploc is a `C` library for computing in $p$-adic fields and their unramified extensions. The focus is mainly on $\mathbb{Z}_p$ for prime $p$, and unramified extensions of $\mathbb{Z}_2$. The ability to compute in these structures is important to several applications, for example counting zeta functions of algebraic varieties —this application encompasses the problem of point counting on algebraic curves—. In a similar realm, some algorithms for constructing curves via complex multiplication methods have a $p$-adic analogue: the Mploc library can be used for this purpose.

The Mploc library is already distributed[7] and used, although several performance improvements are sought. The library presently gathers 3,000 lines of `C` source code.

## 5.10. Finite fields

**Participants:** Pierrick Gaudry, Emmanuel Thomé [contact].

Mpfq is (yet another) library for computing in finite fields. The purpose of Mpfq is not to provide a software layer for accessing finite fields determined at runtime within a computer algebra system like Magma, but rahter to give a very efficient, optimized code for computed in finite fields precisely known at *compile time*. Mpfq is not restricted to a finite field in particular, and can adapt to finite fields of any characteristic and any extension degree. Cryptology being one of the contexts of application, however, Mpfq somehow focuses on prime fields and on fields of characteristic two.

Mpfq's ability to generate specialized code for desired finite fields differentiates this library from existing software. The performance achieved is far superior. Mpfq can be readily used for example to assess the throughput of an efficient software implementation of a given cryptosystem. Such an evaluation is the purpose of the "EBats" benchmarking tool[8].

The library's purpose being the *generation* of code rather than its execution, the working core of Mpfq consists of roughly 5,000 lines of Perl code, which generate most of the currently 13,000 lines of `C` code. Mpfq is currently under active development, and a first release is expected in 2007.

# 6. New Results

## 6.1. Floating-Point Arithmetic

**Participants:** Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Damien Stehlé, Paul Zimmermann.

Two problems remain to find all worst cases of the standard C99 functions in the double precision IEEE 754 format:

- periodic functions with large arguments, for example $\sin x$ for $x$ near $2^{1024}$. The distance between two consecutive floating-point numbers being large — here $2^{971}$ — with respect to the function period — here $2\pi$ — the classical methods (Lefèvre's and the SLZ algorithms) cannot be applied.
- two-variable functions like $x^y$, $\sqrt{x^2+y^2}$ or $\operatorname{atan}\frac{y}{x}$. The problem here is that the input set has up to $2^{128}$ elements.

We have obtained a first result for the first problem. Namely, if $\mu$ is the distance between two consecutive floating-point numbers, the idea is to search for an integer multiple $\tau = q\mu \mod \Pi$ that is small after reduction by the period $\Pi$, using for example the continued fraction from $\mu/\Pi$. Then we apply the classical methods (Lefèvre's and the SLZ algorithms) to arithmetic progressions of the form $x_i = x_0 + i\tau$, instead of $x_i = x_0 + i\mu$ in the classical case. The obtained complexity is slightly worse than in the classical case, because $\tau$ is not as small as $\mu = \operatorname{ulp}(x_i)$. We were however able to find some non-trivial bad cases in a few days of computing time, for example:

[7] http://www.loria.fr/~thome/software/mploc
[8] http://www.ecrypt.eu.org/ebats/

$$\sin(5501214608935005\cdot 2^{971}) = 0.0\underbrace{01001100011001110010111010011110001101101001000111111}_{53}0^{45}\,1011...$$

We expect this first result will lead to new developments. In particular a complete description of the $x_0 + i\mu$ that are in a small interval modulo the period $\Pi$ might lead to a still better algorithm.

Another important topic in computer arithmetic is the search for polynomial approximations to functions. Indeed, instead of implementing a sophisticated algorithm to compute a function, it is often more efficient to precompute good polynomial approximations over sub-intervals, and to evaluate only these approximations. However, in order to get a good control of the evaluation error, one should use polynomials with floating-point coefficients. This problem has been studied jointly with N. Brisebarre [17] (Arénaire project-team, Lyon). We have shown that, when studying approximation in the $L^2$ sense, ie. finding a polynomial $P$ making

$$\int_I (P-f)^2 d\mu,$$

minimal for some function $f$ and some measure $\mu$ over $I$, it is possible to find the best polynomial $P$ with floating-point coefficients; this has been shown by reducing the problem to a problem of finding a closest vector in a lattice. We also show that the reduction can be performed the other way, thus showing that $L^2$-approximation is NP-hard.

The common work with Richard Brent and Colin Percival on the fine error analysis of the complex floating-point multiplication was accepted for publication in *Mathematics of Computation*, and is currently in press [10].

## 6.2. Algorithmic number theory and cryptology

**Participants:** Pierrick Gaudry, Emmanuel Thomé.

### *6.2.1. Discrete logarithm in jacobian of curves*

The paper written on the algorithm developed in 2005 (using a double large prime variation for the discrete logarithm problem, DLP for short, in jacobian of curves) with Diem and Thériault has been accepted. In the most important cases, genus 3 and genus 4 curves, it brings the following changes to the complexity of the DLP in the jacobian of a curve over a finite field with $q$ elements:

| $g$ | Index calculus [31] | 1 large prime [58] | our algo [12] |
|---|---|---|---|
| 3 | $q^{3/2}$ | $q^{10/7}$ | $q^{4/3}$ |
| 4 | $q^{8/5}$ | $q^{14/9}$ | $q^{3/2}$ |

Another important contribution in our work was to do computational experiments in order to demonstrate that the asymptotically fast algorithms were also the fastest in practice, already for small sizes.

Our work has since been improved by Diem to curves of small degree. In the particular case of non-hyperelliptic curves of genus 3, this has been more precisely studied by Diem and Thomé [18]. The conclusion is that for those curves, there exists an attack with complexity $O(q)$, and the algorithm is quite practical. This result is a very important one: until recently curves of genus 3 were studied for potential replacement of elliptic curves in cryptosystems; it is now clear that there is not much hope in that direction for non-hyperelliptic curves.

Another contribution in the context of discrete logarithms has been obtained by Enge and Gaudry. For a general curve of large enough genus $g$ over a finite field $q$, the complexity of a discrete log computation is in $L_{q^g}(1/2)$, where $L()$ is the classical subexponential function (this has been recently proven in a rigorous way by Hess [40]). Enge and Gaudry [19] have shown that for plane curves having a particular shape of degrees in $x$ and $y$, this complexity can been reduced heuristically to $L_{q^g}(1/3)$, recovering the kind of complexity we have for integer factorization or discrete logarithms in finite fields.

### 6.2.2. Point counting

On the point counting side, after a series of lectures given at IHP, Gaudry has written a survey article [20] that takes a snapshot of the current situation and highlight the major difficulties that should be overcome to continue, in particular in the non-elliptic, large characteristic case.

The paper [9], whose redaction started when Gaudry was at LIX has been accepted for publication. This contains an efficient algorithm getting some information on the number of points when the characteristic of the base field is medium-sized. The technique that is used also provided the best known deterministic algorithm for factoring integers.

In the case of elliptic curves over large prime field, Gaudry and Morain have cleaned the phase called "Eigenvalue computation" of the SEA algorithm [15]. Using algorithmic tools of computer algebra, they reduced the theoretical complexity of this phase and its practical running time.

The alternate approach to point counting is the CM method that produces a curve together with its number of points. In a collaboration of Gaudry, Houtmann, Weng, Ritzenthaler and Kohel, started at LIX, it is shown that a 2-adic algorithm can be used to speed-up the computations in the case of genus 2 curves. This work [14] has been recently accepted.

### 6.2.3. Miscellaneous

Gaudry and specialists of protocol design have worked together to improve the efficiency of key-exchange when elliptic curves are used as a building block. This ended up in a fast protocol [13] that is provably secure in the standard model (no random oracle needed).

## 6.3. Lattices

**Participant:** Damien Stehlé.

A study of the behaviour of LLL on average [16] has been undertaken by Nguyen and Stehlé. It rests on a fine probabilistic modelling of the situation. The main results obtained are the fact that, contrary to a common belief, the behaviour of LLL in practice is not better than the worst-case bounds given by the theoretical analysis. In particular, one should expect the first vector of an LLL-reduced basis of a $d$-dimensional lattice $L$ to be of length $\approx (1.02)^d \lambda_1(L)$, where $\lambda_1(L)$ is the first minimum of the lattice. Furthermore, similar results are obtained on the complexity of the algorithm. These results are supported by practical experiments.

# 7. Contracts and Grants with Industry

## 7.1. MPQS

**Participant:** Paul Zimmermann.

MPQS is a program that factors integers using the Multiple Polynomial Quadratic Sieve, developed by Scott Contini and Paul Zimmermann. It is distributed under GPL from http://www.loria.fr/~zimmerma/free/. A license agreement is under discussion with Waterloo Maple Inc. (WMI), to enable the use of a fixed version of the MPQS software within the Maple computer algebra software.

# 8. Other Grants and Activities

## 8.1. National Initiatives

### 8.1.1. ANR CADO

**Participants:** Pierrick Gaudry, Guillaume Hanrot, Emmanuel Thomé, Paul Zimmermann.

The team has obtained a financial support from the ANR (“programme blanc”) for a project, common with the TANC project-team and the number theory team of the mathematics lab in Nancy of studying the number field sieve algorithm.

#### *8.1.2. ANR RAPIDE*

**Participants:** Guillaume Hanrot, Marion Videau, Paul Zimmermann.

The team has obtained a financial support from the ANR ("Sécurité et Informatique", SETIN2006 program) for a common research project with CODES project team, XLIM laboratory (Arithmetic, codes and cryptography group) of the University of Limoges, and the CITI laboratory (Middleware - Security group) of INSA in Lyon. This project is coordinated by Marion Videau.

The research aimed concerns stream ciphers especially the ones designed for constrained environments. This is a particularly hot topic as it takes place in a context where one can say that there is no such ciphers that can presently be declared secure. It also participates in the analysis efforts towards the final evaluation of the proposals submitted to the eSTREAM stream cipher project issued by ECRYPT, the European Network of Excellence in Cryptology.

### 8.2. European Initiatives

#### *8.2.1. PAI with Berlin*

**Participant:** Pierrick Gaudry.

We have a grant from the French Ministry of Foreign Affairs in the PAI program (Programme d’Actions Intégrées) with Germany. This is an exchange research program with Florian Heß and the “Algebra und Zahlentheorie” group in the TU Berlin. The topic fits with our overall objectives, since the goal is to investigate new methods in number theory and geometry with a view towards cryptology.

## 9. Dissemination

### 9.1. Scientific Animation

#### *9.1.1. RNC’7 Conference*

The members of the project have organized the 7th Real Numbers and Computers conference (RNC’7) in July 2006 (see http://rnc7.loria.fr). E. Thomé was publicity chair, L. Fousse and V. Lefèvre were organizing a “friendly competition”, C. Simon was in charge of the invited speakers and the conference budget, and G. Hanrot, P. Zimmermann were co-chairs of the program committee, and editors of the conference proceedings [8].

#### *9.1.2. Other conferences*

Emmanuel Thomé co-organizes the *Journées Nationales de calcul Formel*, to be held in Luminy in 2007.

Emmanuel Thomé participates to the program committee of the C2 workshop (*Codage et Cryptographie*), to be held in Eymoutiers in october 2006.

### 9.2. Leadership within Scientific Community

G. Hanrot and P. Zimmermann have been program co-chairs of the RNC’7 conference, that took place in Nancy in July 2006.

## 9.3. Committees memberships

G. Hanrot is vice-head of the Project Committee of INRIA Lorraine. He is also an appointed member of the INRIA Commission d'Évaluation, of the Mathematics "Commissions de Spécialistes" from Universités Montpellier 2, Henri-Poincaré Nancy 1-Nancy 2-INPL, Jean-Monnet Saint-Étienne. He was a member of the hiring committee for CR2 at INRIA Futurs and INRIA Sophia-Antipolis in 2006. He is a member of the steering committee of the RNC conference. In 2006, he was one of the reviewers of the PhD theses of R. Dupont (École polytechnique) and G. Melquiond (E.N.S. Lyon), and a member of the committee for M. Abouzaid PhD thesis (Bordeaux 1).

P. Zimmermann is also an elected member from the INRIA Evaluation Committee, and of the Computer Science "Commission de Spécialistes" from University Henri Poincaré Nancy 1. He is also a member of the steering committee of the RNC conference, of the program committee of ARITH'18 (to be held in 2007), and of the editorial board of a special issue of *Journal of Logic and Algebraic Programming* on the development of exact real number computation.

P. Gaudry is an appointed member of the Computer Science "Commissions de Spécialistes" from Universités Henri-Poincaré Nancy 1 and Paris 8.

## 9.4. Teaching

P. Gaudry and G. Hanrot gave each three 3 hours lectures at MPRI (Master Parisien de Recherche en Informatique) about algorithmic number theory, in the Cryptology course.

P. Gaudry and G. Hanrot are members of the jury of "agrégation externe de mathématiques", a competitive exam to hire high school teachers.

# 10. Bibliography

## Major publications by the team in recent years

[1] Y. BILU, G. HANROT, P. VOUTIER. *Existence of primitive divisors of Lucas and Lehmer sequences*, in "J. Reine Angew. Math.", vol. 539, 2001, p. 75–122.

[2] Y. BUGEAUD, G. HANROT. *Un nouveau critère pour l'équation de Catalan*, in "Mathematika", vol. 47, 2000, p. 63–73.

[3] D. DEFOUR, G. HANROT, V. LEFÈVRE, J.-M. MULLER, N. REVOL, P. ZIMMERMANN. *Proposal for a Standardization of Mathematical Function Implementation in Floating-Point Arithmetic*, in "Numerical Algorithms", vol. 37, n$^{o}$ 1-2, 2004, p. 367–375.

[4] L. FOUSSE, P. ZIMMERMANN. *Accurate Summation : Towards a Simpler and Formal Proof*, in "5th Conference on Real Numbers and Computers 2003 - RNC5, Lyon, France", Sept. 2003, p. 97–108.

[5] V. LEFÈVRE. *Moyens arithmétiques pour un calcul fiable*, Thèse de doctorat, École Normale Supérieure de Lyon, 2000.

[6] F. ROUILLIER, P. ZIMMERMANN. *Efficient Isolation of Polynomial Real Roots*, in "Journal of Computational and Applied Mathematics", vol. 162, n$^{o}$ 1, 2003, p. 33-50.

[7] D. STEHLÉ, V. LEFÈVRE, P. ZIMMERMANN. *Worst Cases and Lattice Reduction*, in "16th IEEE Symposium on Computer Arithmetic 2003 - ARITH-16'03, Santiago de Compostela, Spain", Jun. 2003, p. 142-147.

# Year Publications

## Books and Monographs

[8] G. HANROT, P. ZIMMERMANN. , G. HANROT, P. ZIMMERMANN (editors). *Proceedings of the 7th Conference on Real Numbers and Computers (RNC'7)*, 2006, http://hal.inria.fr/inria-00107213/en/.

## Articles in refereed journals and book chapters

[9] A. BOSTAN, P. GAUDRY, E. SCHOST. *Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator*, in "SIAM Journal on Computing", 2006, http://hal.inria.fr/inria-00103401/en/.

[10] R. BRENT, C. PERCIVAL, P. ZIMMERMANN. *Errors Bounds on Complex Floating-Point Multiplication*, in "Mathematics of Computation", 2006, http://hal.inria.fr/inria-00107268/en/.

[11] L. FOUSSE, G. HANROT, V. LEFÈVRE, P. PÉLISSIER, P. ZIMMERMANN. *MPFR: A Multiple-Precision Binary Floating-Point Library With Correct Rounding*, in "ACM Transactions on Mathematical Software", vol. 33, n$^{o}$ 2, June 2007, http://hal.inria.fr/inria-00103655/en/.

[12] P. GAUDRY, E. THOMÉ, N. THÉRIAULT, C. DIEM. *A double large prime variation for small genus hyperelliptic index calculus*, in "Mathematics of Computation", 2006, http://hal.inria.fr/inria-00000897/en/.

## Publications in Conferences and Workshops

[13] O. CHEVASSUT, P.-A. FOUQUE, P. GAUDRY, D. POINTCHEVAL. *The Twist-AUgmented technique for key exchange*, in "PKC 2006, 04/2006, New York, USA", M. YUNG, Y. DODIS, KIAYIAS, T. MALKIN (editors). , Lecture notes in computer science, vol. 3958, Springer-Verlag, 2006, p. 410-426, http://hal.inria.fr/inria-00103433/en/.

[14] P. GAUDRY, T. HOUTMANN, A. WENG, C. RITZENTHALER, D. KOHEL. *The 2-adic CM method for genus 2 curves with application to cryptography*, in "Asiacrypt 2006, 12/2006, Shangai", X. LAY, K. CHEN (editors). , Lecture notes in computer science, vol. 4284, Springer-Verlag, 2006, p. 114-129, http://hal.inria.fr/inria-00103435/en/.

[15] P. GAUDRY, F. MORAIN. *Fast algorithms for computing the eigenvalue in the Schoof-Elkies-Atkin algorithm*, in "ISSAC '06: Proceedings of the 2006 international symposium on symbolic and algebraic computation, 07/2006, Genoa, Italy", J.-G. DUMAS (editor). , ACM Press, 2006, p. 109 - 115, http://hal.inria.fr/inria-00001009/en/.

[16] P. NGUYĒN, D. STEHLE. *LLL on the average*, in "The 7th Algorithmic Number Theory Symposium, Allemagne", Lecture Notes in Computer Science, vol. 4078, Springer-Verlag, 2006, p. 238–256, http://hal.ccsd.cnrs.fr/ccsd-00107309/en/.

## Internal Reports

[17] N. BRISEBARRE, G. HANROT. *Floating-Point $L^2$-Approximations*, Rapport de recherche INRIA, n$^{o}$ RR-6058, 2006, http://hal.inria.fr/inria-00119254/en/.

### Miscellaneous

[18] C. DIEM, E. THOMÉ. *Index calculus for non-hyperelliptic curves of genus 3*, 2006, http://hal.inria.fr/inria-00107290/en/.

[19] A. ENGE, P. GAUDRY. *An $L(1/3+\varepsilon)$ algorithm for the discrete logarithm problem in low degree curves*, Preprint, 14 pages, 2006.

[20] P. GAUDRY. *Algorithmes de comptage de points d'une courbe définie sur un corps fini*, Preprint, 28 pages, 2006.

## References in notes

[21] L. C. GUILLOU, J.-J. QUISQUATER (editors). , Lecture Notes in Comput. Sci., Proc. International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 1995, vol. 921, 1995.

[22] B. PRENEEL (editor). , Lecture Notes in Comput. Sci., Proc. International Conference on the Theory and Application of Cryptographic Techniques, Brugge, Belgium, May 2000, vol. 1807, Springer–Verlag, 2000.

[23] C. D. WALTER, Ç. K. KOÇ, C. PAAR (editors). , Lecture Notes in Comput. Sci., Proc. 5th International Workshop on Cryptographic Hardward and Embedded Systems, Sep. 8–10, 2003, vol. 2779, Springer–Verlag, 2003.

[24] S. ARITA. *Algorithms for computations in Jacobians of $C_{ab}$ curve and their application to discrete-log-based public key cryptosystems*, in "Proceedings of Conference on The Mathematics of Public Key Cryptography, Toronto, June 12–17", 1999.

[25] P. BARRETT. *Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor*, in "Advances in Cryptology, Proceedings of Crypto'86", A. M. ODLYZKO (editor). , Lecture Notes in Comput. Sci., vol. 263, Springer–Verlag, 1987, p. 311–323.

[26] A. BASIRI, A. ENGE, J.-C. FAUGÈRE, N. GÜREL. *The arithmetic of Jacobian groups of superelliptic cubics*, To appear in Math. Comp..

[27] C. BURNIKEL, J. ZIEGLER. *Fast Recursive Division*, Research Report, n$^{\text{o}}$ MPI-I-98-1-022, MPI Saarbrücken, 1998, http://data.mpi-sb.mpg.de/internet/reports.nsf/NumberView/1998-1-022.

[28] D. G. CANTOR. *Computing in the Jacobian of a hyperelliptic curve*, in "Math. Comp.", vol. 48, n$^{\text{o}}$ 177, 1987, p. 95–101.

[29] D. COPPERSMITH. *Solving linear equations over $\mathrm{GF}(2)$ via block Wiedemann algorithm*, in "Math. Comp.", vol. 62, n$^{\text{o}}$ 205, 1994, p. 333–350.

[30] S. D. GALBRAITH, S. PAULUS, N. P. SMART. *Arithmetic on Superelliptic Curves*, in "Math. Comp.", vol. 71, n$^{\text{o}}$ 237, 2002, p. 393–405.

[31] P. GAUDRY. *An algorithm for solving the discrete log problem on hyperelliptic curves*, in "Advances in Cryptology – EUROCRYPT 2000", B. PRENEEL (editor). , Lecture Notes in Comput. Sci., Proc. International

Conference on the Theory and Application of Cryptographic Techniques, Brugge, Belgium, May 2000, vol. 1807, Springer–Verlag, 2000, p. 19–34.

[32] P. GAUDRY, É. SCHOST. *Cardinality of a genus 2 hyperelliptic curve over $GF(5 \cdot 10^{24}+41)$*, E-mail to the NMBRTHRY list, 2002, http://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind0209&L=nmbrthry&T=0&P=1827.

[33] P. GAUDRY, E. SCHOST. *Construction of Secure Random Curves of Genus 2 over Prime Fields*, in "Advances in Cryptology – EUROCRYPT 2004", C. CACHIN, J. CAMENISCH (editors). , Lecture Notes in Comput. Sci., vol. 3027, Springer-Verlag, 2004, p. 239–256.

[34] B. GRÉGOIRE, L. THÉRY, B. WERNER. *A computational approach to Pocklington certificates in type theory*, in "Proceedings of FLOPS'06", Lecture Notes in Comput. Sci., vol. 3945, Springer–Verlag, 2006.

[35] M. HACK. *On intermediate precision required for correctly-rounding decimal-to-binary floating-point conversion*, in "Proceedings of RNC'6, Schloß Dagstuhl, Germany, November 15-17", 2004.

[36] G. HANROT, M. QUERCIA, P. ZIMMERMAN. *The middle product algorithm, I. Speeding up the division and square root of power series*, in "Appl. Algebra Engrg. Comm. Comput.", n$^{\text{o}}$ 14, 2004, p. 415–438.

[37] G. HANROT, P. ZIMMERMAN. *A long note on Mulders' short product*, in "J. Symbolic Comput.", n$^{\text{o}}$ 37, 2004, p. 391–401.

[38] R. HARASAWA, J. SUZUKI. *Fast Jacobian group arithmetic on $C_{ab}$ curves*, in "ANTS-IV", W. BOSMA (editor). , Lecture Notes in Comput. Sci., vol. 1838, Springer–Verlag, 2000, p. 359–376.

[39] F. HESS. *Computing Riemann-Roch spaces in algebraic function fields and related topics*, in "J. Symbolic Comput.", vol. 33, 2002, p. 425–445.

[40] F. HESS. *Computing relations in divisor class groups of algebraic curves over finite fields*, Preprint, 2004.

[41] IEEE. *P1363: Standard specifications for public key cryptography*, http://www.manta.ieee.org/groups/1363/.

[42] T. LANG, J.-M. MULLER. *Bounds on Runs of Zeros and Ones for Algebraic Functions*, in "Proceedings of the 15th IEEE Symposium on Computer Arithmetic", IEEE Computer Society, 2001, p. 13–20.

[43] T. LANGE. *Formulae for Arithmetic on Genus 2 Hyperelliptic Curves*, Preprint, 2003.

[44] V. LEFÈVRE, J.-M. MULLER. *Worst Cases for Correct Rounding of the Elementary Functions in Double Precision*, in "Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'15)", N. BURGESS, L. CIMINIERA (editors). , IEEE Computer Society, 2001, p. 111-118.

[45] V. LEFÈVRE. *Moyens arithmétiques pour un calcul fiable*, Thèse de doctorat, École Normale Supérieure de Lyon, 2000.

[46] A. K. LENSTRA, H. W. LENSTRA, L. LOVÁSZ. *Factoring Polynomials with Rational Coefficients*, in "Mathematische Annalen", vol. 261, 1982, p. 515–534.

[47] P. L. MONTGOMERY. *A block Lanczos algorithm for finding dependencies over* GF(2), in "Advances in Cryptology – EUROCRYPT '95", L. C. GUILLOU, J.-J. QUISQUATER (editors). , Lecture Notes in Comput. Sci., Proc. International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 1995, vol. 921, 1995, p. 106–120.

[48] T. MULDERS. *On Short Multiplications and Divisions*, in "Appl. Algebra Engrg. Comm. Comput.", vol. 11, n$^{o}$ 1, 2000, p. 69–88.

[49] V. NECHAEV. *Complexity of a determinate algorithm for the discrete logarithm*, in "Math. Notes", vol. 55, n$^{o}$ 2, 1994, p. 165–172.

[50] P. NGUYĚN, D. STEHLÉ. *Floating-Point LLL Revisited*, in "Proceedings of Eurocrypt 2005", Lecture Notes in Comput. Sci., vol. 3494, Springer-Verlag, 2005, p. 215–233.

[51] J. PELZL, T. WOLLINGER, J. GUAJARDO, C. PAAR. *Hyperelliptic Curve Cryptosystems: Closing the Performance Gap to Elliptic Curves*, in "CHES 2003", C. D. WALTER, Ç. K. KOÇ, C. PAAR (editors). , Lecture Notes in Comput. Sci., Proc. 5th International Workshop on Cryptographic Hardward and Embedded Systems, Sep. 8–10, 2003, vol. 2779, Springer–Verlag, 2003, p. 351–365.

[52] R. SCHOOF. *Elliptic curves over finite fields and the computation of square roots mod p*, in "Math. Comp.", vol. 44, 1985, p. 483–494.

[53] A. SCHÖNHAGE. *Factorization of Univariate Integer Polynomials by Diophantine Approximation and an Improved Basis Reduction Algorithm*, in "Proc. of ICALP '84", Lecture Notes in Comput. Sci., Springer–Verlag, 1984, p. 436–447.

[54] D. STEHLÉ, P. ZIMMERMANN, V. LEFÈVRE. *Searching Worst Cases of a One-Variable Function Using Lattice Reduction*, in "IEEE Transactions on Computers", vol. 54, n$^{o}$ 3, Mar 2005, p. 340-346, http://hal.inria.fr/inria-00000379.

[55] E. THOMÉ. *Computation of discrete logarithms in* $\mathbb{F}_{2^{607}}$, in "Advances in Cryptology – ASIACRYPT 2001", C. BOYD, E. DAWSON (editors). , Lecture Notes in Comput. Sci., vol. 2248, Springer–Verlag, 2001, p. 107–124.

[56] E. THOMÉ. *Discrete logarithms in* GF($2^{607}$), E-mail to the NMBRTHRY list, 2002, http://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind0202&L=NMBRTHRY&P=R658&I=-3.

[57] E. THOMÉ. *Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm*, in "J. Symbolic Comput.", vol. 33, n$^{o}$ 5, 2002, p. 757–775.

[58] N. THÉRIAULT. *Index calculus attack for hyperelliptic curves of small genus*, in "Advances in Cryptology – ASIACRYPT 2003", C. LAIH (editor). , Lecture Notes in Comput. Sci., vol. 2894, Springer-Verlag, 2003.

[59] D. H. WIEDEMANN. *Solving sparse linear equations over finite fields*, in "IEEE Trans. Inform. Theory", vol. IT–32, n$^{o}$ 1, 1986, p. 54–62.

[60] J. VAN DER HOEVEN. *Relax, but don't be too lazy*, in "J. Symbolic Comput.", vol. 34, n$^{o}$ 6, 2002, p. 479–542.