Efficient Arithmetic Modulo Minimal Redundancy Cyclotomic Primes

Robert Granger, Andrew Moss, and Nigel P. Smart

Abstract—We introduce a family of prime numbers that we refer to as Minimal Redundancy Cyclotomic Primes (MRCPs). The form of MRCPs is such that when using the field representation and multiplication algorithm we present, multiplication modulo these primes can be up to twice as efficient as multiplication of integer residues. This article provides a comprehensive theoretical framework for the use of MRCPs, detailing field and residue representation, conversion and arithmetic algorithms, proofs of correctness, operation counts and parameter generation. As a potential application, we also provide specialised parameters for elliptic curve cryptography, as well as software implementation results.

Index Terms—High speed arithmetic.

1 Introduction

The problem of how to efficiently perform arithmetic in $\mathbb{Z}/N\mathbb{Z}$ is a very natural one, with numerous applications in computational mathematics and number theory, such as primality proving [], factoring [], and coding theory [], for example. It is also of central importance to nearly all public-key cryptographic systems, including the Digital Signature Algorithm [12], RSA [31], and Elliptic Curve Cryptography (ECC) [5]. As such, from both a theoretical and a practical perspective it is interesting and essential to have efficient algorithms for working in this ring, for either arbitrary or special moduli, with the application determining whether generality (essential for RSA for instance), or efficiency (desirable for ECC) takes precedence.

Two closely related factors need consideration when approaching this problem: firstly, how to represent residues; and secondly, how to perform arithmetic on these representatives. An obvious answer to the first question is to use the canonical representation $\mathbb{Z}/N\mathbb{Z} = \{0, \ldots, N-1\}$, with modular multiplication consisting of integer multiplication of residues followed by reduction of the result modulo N, in order to obtain a canonical representative once again. Using this approach, the two components needed for efficient modular arithmetic are fast integer arithmetic, and fast reduction.

In order to reduce the total cost of the process, research has tended to focus on making the reduction stage as efficient as possible. For arbitrary moduli, Montgomery's celebrated algorithm [28] enables reduction to be performed for approximately the cost of a residue by residue multiplication. On the other hand, for very special moduli, in particular the Mersenne numbers $m = 2^k - 1$, modular reduction of a $2k$-bit product of residues is performed via a single modular addition, as is well known.

In 1999 Solinas proposed an extension of this technique to a larger class of integers: the Generalized Mersenne Numbers (GMNs) [33]. As they are a superset, GMNs are more numerous than the Mersenne numbers and contain more primes, while incurring little additional overhead in terms of performance [6]. In 2000, NIST recommended ten finite fields for use in the ECDSA: five binary fields and five prime fields, and due to their performance characteristics the latter of these are all GMNs [12]. Of particular note is an approach due to Bernstein, that utilises a non-standard representation of residues and exploits the floating-point unit of specific instruction-set architectures to great effect, setting the benchmark for high-speed modular arithmetic for ECC [3].

The above approaches make the implicit (though not obviously correct) assumption that the form of the modulus can not affect how one should multiply residue representatives, or at the very least they make no attempt to exploit it. Indeed one could argue that multiplication modulo $2^k - 1$ must be 'near-optimal', since then reduction is nearly free. Such an argument however would be naive, and is a product of considering residues only as integers. To show that this intuition is not necessarily correct, we propose the use of a set of moduli whose form allows one to nearly half the cost of the multiplication step (relative to the multiplication of canonical representatives of the same residues), and yet still permits a very efficient reduction algorithm, which together are competitive with the current fastest modular multiplication algorithms at contemporary ECC security levels [12], and nearly twice as fast for asymptotic

- *R. Granger is with the Claude Shannon Institute for Discrete Mathematics, Coding, Cryptography and Information Security, School of Computing, Dublin City University, Dublin 9, Ireland, and is supported by Science Foundation Ireland Grant No. 06/MI/006.*
  E-mail: rgranger@computing.dcu.ie
- *A. Moss and N.P. Smart are with the University of Bristol, Department of Computer Science, Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB, UK.*
  E-mail: {moss,nigel}@cs.bris.ac.uk

bitlengths. Furthermore the set of primes we propose are far more numerous than the set of GMNs.

The source of the efficiency we present is the combination of a remarkable algebraic identity used by Nogami, Saito, and Morikawa in the context of extension fields [29], together with a residue representation and reduction method proposed by Chung and Hasan [9], which models suitable prime fields as the quotient of an integer lattice by a particular equivalence relation.

The sequel is organised as follows. In §2 we recall some definitions and review related previous work. In §3 we describe the basis of our arithmetic, then in §4-6 we present details of our multiplication, reduction and residue representation respectively. In §7 we show how to ensure input/output stability for modular multiplication and in §8 give our full modular multiplication algorithm. In §9 we give a brief treatment of other arithmetic operations and in §10 present our implementation results. Finally, in §11 we draw some conclusions.

2 Definitions and Previous Work

In this section we introduce the cyclotomic primes and their minimal redundancy subset, the MRCPs, and provide a summary of related work.

2.1 Definitions

We begin with the following.

Definition 1: For $n \geq 1$ let ζ_n be a primitive n-th root of unity. The n-th cyclotomic polynomial is defined by

$$\Phi_n(x) = \prod_{(k,n)=1} (x - \zeta_n^k) = \prod_{d|n} (1 - x^{n/d})^{\mu(d)},$$

where μ is the Möbius function.

Two basic properties of the cyclotomic polynomials are that they have integer coefficients, and are irreducible over $\mathbb{Q}$. These two properties ensure that the evaluation of a cyclotomic polynomial at an integer argument will also be an integer, and that this integer will not inherit a factorisation from one in $\mathbb{Q}[x]$. One can therefore ask whether or not these polynomials ever assume prime values at integer arguments, which leads to the following definition.

Definition 2: For $n \geq 1$ and $t \in \mathbb{Z}$, if $p = \Phi_n(t)$ is prime, we call p an n-th cyclotomic prime, or simply a cyclotomic prime.

Note that for all primes p, we have $p = \Phi_1(p+1) = \Phi_2(p-1)$, and so trivially all primes are cyclotomic primes. These instances are also trivial in the context of the algorithms we present here for performing arithmetic modulo these primes, since in both cases the cyclotomic polynomials are linear, and in this case our algorithms reduce to ordinary Montgomery arithmetic. Hence for the remainder of the article we assume $n \geq 3$.

In addition to being prime-evaluations of cyclotomic polynomials, note that for a cyclotomic prime $p = \Phi_n(t)$, the field $\mathbb{F}_p$ can be modelled as the quotient of the ring of integers of the n-th cyclotomic field $\mathbb{Q}(\zeta_n)$, by the prime ideal $\pi = \langle p, \zeta_n - t \rangle$. This is precisely how one would represent $\mathbb{F}_p$ when applying the Special Number Field Sieve to solve discrete logarithms in $\mathbb{F}_p$, for example [25]. Hence our nomenclature for these primes seems apt.

This interpretation of $\mathbb{F}_p$ for p a cyclotomic prime is implicit within the arithmetic we develop here, albeit only insofar as it provides a theoretical context for it; this perspective offers no obvious insight into how to perform arithmetic efficiently and the algorithms we use make no use of it at all. Similarly the method of Chung and Hasan [9] upon which our residue representation is based can be seen as arising in exactly the same way for the much larger set of primes they consider, with the field modelled as a quotient of the ring of integers of a suitable number field by a degree one prime ideal, just as for the cyclotomic primes.

The goal of the present work is to provide efficient algorithms for performing $\mathbb{F}_p$ arithmetic, for p a cyclotomic prime. As will become clear from our exposition, the following subset of the cyclotomic primes is particularly attractive from an efficiency perspective:

Definition 3: For $m+1$ be an odd prime and let

$$p = \Phi_{m+1}(t) = t^m + t^{m-1} + \cdots + t + 1,$$

Then if p is prime we say it is a *Minimal-Redundancy Cyclotomic Prime (MRCP)*.

MRCPs are so-called because in order to exploit the cyclic structure available, we use not the ring $\mathbb{Z}/\Phi_{m+1}(t)\mathbb{Z}$, but the slightly larger ring $\mathbb{Z}/(t^{m+1}-1)\mathbb{Z}$, into which the first naturally embeds, before applying our residue transformation. Using the larger ring introduces some redundancy into the residue representation which is measured by the ratio $(m+1)/\phi(m+1)$, where $\phi(\cdot)$ is Euler's ϕ-function, which gives the degree of $\Phi_{m+1}(\cdot)$. This ratio takes successive local minima when $m+1$ is prime, hence our chosen name. The reason for using this embedding is two-fold: firstly, it simplifies the reduction algorithm we propose, see §5; secondly, the larger ring also gives rise to the formulae of Nogami *et al.* - hereafter referred to as Nogami's formula - for multiplying residue representatives (cf. §4), and reduces the cost of evaluating them.

2.2 Previous Work

In the context of extension fields, for t prime and a primitive root modulo $m+1$, the ring $\mathbb{F}_t[X]/(\Phi_{m+1}(X)\mathbb{F}_t[X])$ is isomorphic to the field $\mathbb{F}_{t^m}$. In the binary case, i.e., $t = 2$, several authors proposed the use of this polynomial to obtain efficient multiplication algorithms, all of which rely on the observation that the ring $\mathbb{F}_2[X]/(f(X)\mathbb{F}_2[X])$ embeds into the ring $\mathbb{F}_2[X]/((X^{m+1}-1)\mathbb{F}_2[X])$, which possesses a particularly nice cyclic structure [4], [20], [32], [35], but introduces some redundancy. Similarly, this idea applies to any cyclotomic polynomial, and several authors have investigated this strategy, embedding suitably defined extension fields into one of the two

rings $\mathbb{F}_2[X]/((X^k \pm 1)\mathbb{F}_2[X])$ [10], [17], [36]. For odd characteristic extension fields, Gao *et al.* used the latter idea to obtain fast exponentiations algorithms [14], [15], while Nogami *et al.* used the same polynomial modulus to obtain a very fast multiplication algorithm [29], cf. §**??**.

The use of cyclotomic polynomials in extension field arithmetic is therefore evidently well studied. In the context of prime fields however, this work appears to be the first to transfer ideas from the domain of extension field arithmetic to prime fields, and it is in this context that the present article contributes.

With regard to using an embedding of a prime field into an integer ring, Walter introduced the idea of operand scaling [34] in order to obtain a desired representation in the higher-order bits, which aids in the estimation of the quotient when using Barrett reduction [1]. Ozturk *et al.* proposed using fields with characteristics dividing integers of the form $2^k \pm c$, with c small enough to fit into a word, with particular application to ECC [30]. While these moduli permit fast reduction, for a given bitlength they are not at all numerous. Both of these exhibit improved reduction efficiency, however neither improve the multiplication stage. The difference with our approach is that we do both, and in addition, MRCPs appear to have a natural density amongst the primes and are thus plentiful and suitable for numerous applications.

3 MRCP Field Representation

In this section we present a sequence of representations for $\mathbb{F}_p$, with p an MRCP, the final one being the target representation which we use for our arithmetic. We recall the mathematical framework of Chung-Hasan arithmetic, in both the general setting and as specialised to MRCPs, focusing on the underlying theory while deferring explicit algorithms for residue multiplication, reduction and representation until §4-6.

3.1 Chung-Hasan Arithmetic

We now describe the ideas behind Chung-Hasan arithmetic [7]–[9]. The arithmetic was developed for a class of integers they term low-weight polynomial form integers (LWPFIs), whose definition we now recall.

Definition 4: An integer p is a low-weight polynomial form integer (LWPFI), if it can be represented by a monic polynomial $f(t) = t^n + f_{n-1}t^{n-1} + \cdots + f_1 t + f_0$, where t is a positive integer and $|f_i| \leq \xi$ for some small positive integer $\xi < t$.

The key idea of Chung and Hasan is to perform arithmetic modulo p using representatives from the polynomial ring $\mathbb{Z}[T]/(f(T)\mathbb{Z}[T])$. To do so, one uses the natural embedding $\psi : \mathbb{F}_p \hookrightarrow \mathbb{Z}[T]/(f(T)\mathbb{Z}[T])$ obtained by taking the base t expansion of an element of $\mathbb{F}_p$ in the usual representation $\mathbb{F}_p = \{0, \ldots, p-1\}$, and substituting T for t. To compute ψ^{-1} one simply makes the inverse substitution and evaluates the expression modulo p.

The reason for using this ring is straightforward: since ψ^{-1} is a homomorphism, when one computes $z(T) = x(T) \cdot y(T)$ in $\mathbb{Z}[T]$, reducing the result modulo $f(T)$ to give $w(T)$ does not change the element of $\mathbb{F}_p$ represented by $z(T)$, i.e., if $z(T) \equiv w(T) \pmod{f(T)}$, then $z(t) \equiv w(t) \pmod{p}$, since $p = f(t)$. Furthermore, since $f(T)$ has very small coefficients, $w(T)$ can be computed from $z(T)$ using only additions and subtractions. Hence given the degree $2(n-1)$ product of two degree $n-1$ polynomials in $\mathbb{Z}[T]$, its degree $n-1$ representation in $\mathbb{Z}[T]/(f(T)\mathbb{Z}[T])$ can be computed very efficiently. Note that for non-low-weight polynomials this would no longer be the case.

The only problem with this approach is that when computing $z(T)$ as above, the coefficients of $z(T)$, and hence $w(T)$, will be approximately twice the size of the inputs' coefficients, and if further operations are performed the representations will continue to expand. Since for I/O stability one requires that the coefficients be the size of t after each multiplication or squaring, one must somehow reduce the coefficients of $w(T)$ to obtain a standard, or reduced representative, while ensuring that $\psi^{-1}(w(T))$ remains unchanged.

Chung and Hasan refer to this issue as the *coefficient reduction problem* (CRP), and developed three solutions in their series of papers on LWPFI arithmetic [7]–[9]. Each of these solutions is based on an underlying lattice, although this was only made explicit in [9]. Since the lattice interpretation is the most elegant and simplifies the exposition, in the sequel we opt to develop the necessary theory for MRCP arithmetic in this setting.

3.2 Overview of Methodology

Our goal is to develop arithmetic for $\mathbb{F}_p$, where $p = \Phi_{m+1}(t)$, and we begin with the canonical representation $\mathbb{F}_p \cong \mathbb{Z}/\Phi_{m+1}(t)\mathbb{Z}$.

As stated in §2.1, the first map in our chain of representations takes the canonical ring and embeds it into $\mathbb{Z}/(t^{m+1}-1)\mathbb{Z}$, for which the identity map suffices. To map back, one reduces a representative modulo p.

We then apply the Chung-Hasan transformation, which embeds the second ring into $\mathbb{Z}[T]/(T^{m+1}-1)\mathbb{Z}[T]$, by taking the base t expansion of an element of $\{0, \ldots, t^{m+1}-1\}$, and substituting T for t, for instance. We call this map ψ. To compute ψ^{-1} one simply makes the inverse substitution and evaluates the expression modulo $t^{m+1}-1$.

Note that the codomain of ψ can be regarded as an $m+1$-dimensional vector space over $\mathbb{Z}$, with the natural basis $\{T^m, \ldots, T, 1\}$. This interpretation is useful since we can now regard $\mathbb{Z}[T]/(T^{m+1}-1)\mathbb{Z}[T]$ as a quotient of $\mathbb{Z}_{m+1}$ by a certain equivalence relation, for which coefficient/component reduction can be effected using the prescence of a corresponding modular integer lattice.

Since $\mathbb{Z}^{m+1}$ has elements whose components are naturally unbounded, for each $x \in \mathbb{Z}/(t^{m+1}-1)\mathbb{Z}$ there are infinitely many elements of $\mathbb{Z}^{m+1}$ that map via ψ^{-1} to x. In order to obtain a useful isomorphism directly between

$\mathbb{Z}/(t^{m+1}-1)\mathbb{Z}$ and $\mathbb{Z}^{m+1}$, we identify two elements of $\mathbb{Z}^{m+1}$ whenever they map to the same element of $\mathbb{Z}/(t^{m+1}-1)\mathbb{Z}$ via ψ^{-1}, and take the image of ψ to be the quotient of $\mathbb{Z}^{m+1}$ by this equivalence relation. Pictorially we have the following chain, with the final representation being our target:

$$\mathbb{F}_p \subset \mathbb{Z}/(t^{m+1}-1)\mathbb{Z} \hookrightarrow \mathbb{Z}[T]/(T^{m+1}-1)\mathbb{Z}[T] \cong \mathbb{Z}^{m+1}/\sim$$

As mentioned in §3.1, for each coset in $\mathbb{Z}^{m+1}/\sim$, we should like to use a minimal, or in some sense 'small' representative, in order to facilitate efficient arithmetic after a multiplication or a squaring, for example. Since we know the base-t expansion of every $x \in \mathbb{Z}/(t^{m+1}-1)\mathbb{Z}$ gives one such representative for each coset in $\mathbb{Z}^{m+1}/\sim$, for a reduction algorithm we just need to be able to find it, or at least one whose components are of a similar size.

Chung and Hasan related finding such 'nice' or reduced coset representatives to solving a computational problem in an underlying lattice, which we now recall.

3.3 Lattice interpretation

Let $x(T) \in \mathbb{Z}[T]/(T^{m+1}-1)\mathbb{Z}[T]$, where

$$x(T) = x_m T^m + \ldots + x_1 T + x_0.$$

As in §3.2 we consider $x(T)$ to be a vector $\overline{\mathbf{x}} = [x_m, \ldots, x_0] \in \mathbb{Z}^{m+1}$, and write $\overline{\mathbf{x}}^T$ for its transpose. With this interpretation one can see that $\mathbb{Z}/(t^{m+1}-1)\mathbb{Z}$ corresponds to the set of equivalence classes of $\mathbb{Z}^{m+1}/\sim$, where

$$\overline{\mathbf{x}} \sim \overline{\mathbf{y}} \iff \psi^{-1}(\overline{\mathbf{x}}) \equiv \psi^{-1}(\overline{\mathbf{y}}) \pmod{t^{m+1}-1} \quad (1)$$

Given an input vector $\overline{\mathbf{z}}$, which is the output of a multiplication or a squaring, a coefficient reduction algorithm should output a vector $\overline{\mathbf{w}}$ such that $\overline{\mathbf{w}} \sim \overline{\mathbf{z}}$, in the sense of (1), whose components are approximately the same size as t. As observed in [9], the equivalence relation (1) is captured by an underlying lattice, and finding $\overline{\mathbf{w}}$ is tantamount to solving an instance of the *closest vector problem* (CVP) in this lattice. To see why this is, we first fix some notation as in [9].

Let $\overline{\mathbf{u}}$ and $\overline{\mathbf{v}}$ be vectors in $\mathbb{Z}^{m+1}$ such that the following condition is satisfied:

$$[t^m, \ldots, t, 1] \cdot \overline{\mathbf{u}}^T \equiv [t^m, \ldots, t, 1] \cdot \overline{\mathbf{v}}^T \pmod{t^{m+1}-1}$$

Then we say that $\overline{\mathbf{u}}$ *is congruent to* $\overline{\mathbf{v}}$ *modulo* $t^{m+1}-1$ and write this as $\overline{\mathbf{u}} \cong_{t^{m+1}-1} \overline{\mathbf{v}}$. Note that this is exactly the same as saying $\psi^{-1}(\overline{\mathbf{u}}) \equiv \psi^{-1}(\overline{\mathbf{v}}) \pmod{t^{m+1}-1}$, and so $\overline{\mathbf{u}} \sim \overline{\mathbf{v}} \iff \overline{\mathbf{u}} \cong_{t^{m+1}-1} \overline{\mathbf{v}}$.

Similarly, but abusing notation slightly, for any integer $b \neq t^{m+1}-1$, where b is the word base of the underlying architecture, we write $\overline{\mathbf{u}} \cong_b v$ for some integer v satisfying $[t^m, \ldots, t, 1] \cdot \overline{\mathbf{u}}^T \equiv v \pmod{b}$, and say $\overline{\mathbf{u}}$ *is congruent to* v *modulo* b, in this case. We reserve the use of '$\equiv$' to express a component-wise congruence relation, i.e., $\overline{\mathbf{u}} \equiv \overline{\mathbf{v}} \pmod{b}$. Finally, we denote by $\overline{\mathbf{u}} \bmod b$ the component-wise modular reduction of $\overline{\mathbf{u}}$ by b.

The lattice underlying the equivalence relation (1) can now enter the frame. Let $\mathbf{V} = \{\overline{\mathbf{v}}_0, \ldots, \overline{\mathbf{v}}_m\}$ be a set of $m+1$ linearly independent vectors in $\mathbb{Z}^{m+1}$ such that $\overline{\mathbf{v}}_i \cong_{t^{m+1}-1} \overline{\mathbf{0}}$, the all zero vector, for $i = 0, \ldots, m$. Then the set of all integer combinations of elements of $\mathbf{V}$ forms an integral lattice, $\mathcal{L}(\mathbf{V})$, with the property that for all $\overline{\mathbf{z}} \in \mathbb{Z}^{m+1}$, and all $\overline{\mathbf{u}} \in \mathcal{L}$, we have

$$\overline{\mathbf{z}} + \overline{\mathbf{u}} \cong_{t^{m+1}-1} \overline{\mathbf{z}} \quad (2)$$

In particular, the equivalence relation (1) is captured by the lattice $\mathcal{L}$, in the sense that

$$\overline{\mathbf{x}} \cong_{t^{m+1}-1} \overline{\mathbf{y}} \iff \overline{\mathbf{x}} - \overline{\mathbf{y}} \in \mathcal{L}$$

Therefore if one selects basis vectors for $\mathcal{L}$ that have L_∞-norm approximately t, then for a given $\overline{\mathbf{z}} \in \mathbb{Z}^{m+1}$, finding the closest vector $\overline{\mathbf{u}} \in \mathcal{L}$ to $\overline{\mathbf{z}}$ (with respect to the L_∞-norm), means the vector $\overline{\mathbf{w}} = \overline{\mathbf{z}} - \overline{\mathbf{u}}$ is in the fundamental domain of $\mathcal{L}$, and so has components of the desired size. Furthermore, since $\overline{\mathbf{w}} = \overline{\mathbf{z}} - \overline{\mathbf{u}}$, by (2) we have

$$\overline{\mathbf{w}} \cong_{t^{m+1}-1} \overline{\mathbf{z}},$$

and hence solving the CVP in this lattice solves the CRP. In general solving the CVP is NP-hard, but since we can exhibit a good lattice basis for LWPFIs, and a near-optimal lattice basis for MRCPs, it is straightforward.

3.4 Lattice basis and simple reduction

For MRCPs, we use the following basis for $\mathcal{L}$:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & -t \\ -t & 1 & \cdots & 0 & 0 & 0 \\ 0 & -t & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -t & 1 & 0 \\ 0 & 0 & \cdots & 0 & -t & 1 \end{bmatrix} \quad (3)$$

Observe that the L_∞-norm of each basis vector is t, so elements in the fundamental domain will have components of the desired size.

In order to perform a simple reduction that reduces the size of components by approximately $\log_2 t$ bits, write each component of $\overline{\mathbf{z}}$ in base t: $z_i = z_{i,1}t + z_{i,0}$. Then define $\overline{\mathbf{w}}^T$ to be:

$$\begin{bmatrix} z_m \\ z_m \\ \vdots \\ \vdots \\ z_1 \\ z_0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & -t \\ -t & 1 & \cdots & 0 & 0 & 0 \\ 0 & -t & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -t & 1 & 0 \\ 0 & 0 & \cdots & 0 & -t & 1 \end{bmatrix} \begin{bmatrix} z_{m-1,1} \\ z_{m-2,1} \\ \vdots \\ \vdots \\ z_{0,1} \\ z_{m,1} \end{bmatrix}$$

Then $\overline{\mathbf{w}} \sim \overline{\mathbf{z}}$ and each $|w_i| \approx |z_i|/t$, assuming $|z_i| > t^2$. This was the method of reduction described in [7], which requires integer division. The idea described in [8] was based on an analogue of Barrett reduction [1]. The method we shall use, from [9], is based on Montgomery reduction [28] and is superior to both.

3.5 Montgomery lattice-basis reduction

Let b be the word base of the computer architecture. In ordinary Montgomery reduction [28], one has an integer $0 \leq Z < pR$ which is to be reduced modulo p, an odd prime, where here R is the smallest power of the word base larger than p. The central idea is to add a multiple of p to Z such that the result is divisible by R. Upon dividing by R, which is a simple right shift of words, the result is congruent to $ZR^{-1} \pmod p$, and importantly is less than $2p$.

In the context of MRCPs, let $R = b^q$ be the smallest power of b greater than t. The input to the reduction algorithm is a vector $\overline{\mathbf{z}} \in \mathbb{Z}^{m+1}$ for which each component is approximately R^2. The natural analogue of Montgomery reduction is to add to $\overline{\mathbf{z}}$ a vector $\overline{\mathbf{u}} \in \mathcal{L}$ whose components are bounded by R, such that $\overline{\mathbf{z}} + \overline{\mathbf{u}} \equiv [0, \ldots, 0] \pmod R$. Then upon the division of each component by R, the result will be a vector $\overline{\mathbf{w}}$ which satisfies

$$\overline{\mathbf{w}} \cong_{t^{m+1}-1} (\overline{\mathbf{z}} + \overline{\mathbf{u}}) \cdot R^{-1} \cong_{t^{m+1}-1} \overline{\mathbf{z}} \cdot R^{-1},$$

and which has components of the desired size. While this introduces an R^{-1} term into the congruence, as with Montgomery arithmetic, one circumvents this simply by altering the original coset representation of $\mathbb{Z}/(t^{m+1} - 1)\mathbb{Z}$, via the map $x \mapsto xR \pmod{t^{m+1} - 1}$, which is bijective since $\gcd(t^{m+1} - 1, R) = 1$, assuming t is even, see §**??**.

How then does one find a suitable lattice point $\overline{\mathbf{u}}$? For this one use the lattice basis (3), which from here on in we call F. Proposition 3 of [9] proves that $\det F = 1 - t^{m+1}$, and so $\gcd(\det F, R) = 1$. One can therefore compute

$$\overline{\mathbf{u}}^T \stackrel{\text{def}}{=} -F^{-1} \cdot \overline{\mathbf{z}}^T \pmod R, \tag{4}$$

$$\overline{\mathbf{w}}^T \stackrel{\text{def}}{=} (\overline{\mathbf{z}}^T + F \cdot \overline{\mathbf{u}}^T)/R, \tag{5}$$

giving $\overline{\mathbf{w}}$ with the required properties. Observe that the form of these two operations is identical to Montgomery reduction, the only difference being that integer multiplication is replaced by matrix by vector multiplication.

It is easy to see that this is what one requires, since for any $\overline{\mathbf{u}} \in \mathbb{Z}^{m+1}$, we have $F \cdot \overline{\mathbf{u}}^T \in \mathcal{L}$, and so

$$\overline{\mathbf{z}}^T + F \cdot \overline{\mathbf{u}}^T \cong_{t^{m+1}-1} \overline{\mathbf{z}}^T.$$

Furthermore, modulo R we have

$$\overline{\mathbf{z}}^T + F \cdot \overline{\mathbf{u}}^T = \overline{\mathbf{z}}^T + F \cdot (-F^{-1} \cdot \overline{\mathbf{z}}^T \bmod R) \equiv [0, \ldots, 0]^T,$$

ensuring the division of each component by R is exact. Hence $\overline{\mathbf{w}} \cong_{t^{m+1}-1} \overline{\mathbf{z}} \cdot R^{-1}$, as claimed.

In [9], an algorithm was given for computing $\overline{\mathbf{u}}$ and $\overline{\mathbf{w}}$ in (4) and (5) respectively, for an arbitrary LWPFI $f(t)$. The number of word-by-word multiply instructions in the algorithm is $\approx nq^2$, where n is the degree of $f(t)$, and $R = b^q$. For comparison, for ordinary Montgomery reduction modulo an integer of equivalent size this number is n^2q^2, making the former approach potentially very attractive. For our choice of primes, the MRCPs, our specialisation of this algorithm is extremely efficient, as we shall see in §5.

4 MRCP Multiplication

In this section we detail algorithms for performing multiplication. While for the reduction and actual residue representation we consider elements to be in $\mathbb{Z}^{m+1}$, the residue multiplication algorithm arises from the arithmetic of the polynomial ring $\mathbb{Z}[T]/(T^{m+1} - 1)\mathbb{Z}[T]$, and so we use this ring to derive the multiplication formulae.

We defer until §6 the issue of how to truncate from this infinite ring to representatives that can be handled efficiently we have specified the multiplication and reduction algorithms, since these determine what precision will be required.

4.1 Ordinary multiplication formulae

Let $R = \mathbb{Z}[T]/(T^{m+1} - 1)\mathbb{Z}[T]$, and let $\overline{\mathbf{x}} = [x_m, \ldots, x_0]$ and $\overline{\mathbf{y}} = [y_m, \ldots, y_0]$ be elements in R. Then in R the product $\overline{\mathbf{x}} \cdot \overline{\mathbf{y}}$ is equal to $[z_m, \ldots, z_0]$, where

$$z_i = \sum_{j=0}^{m} x_{\langle j \rangle} y_{\langle i-j \rangle}, \tag{6}$$

where the subscript $\langle i \rangle$ denotes $i \pmod{m+1}$. This follows from the trivial property $T^{m+1} \equiv 1 \pmod{T^{m+1}-1}$, and the following calculation. Observe that for $\overline{\mathbf{x}} = \sum_{i=0}^{m} x_i T^i$, and $\overline{\mathbf{y}} = \sum_{j=0}^{m} y_j T^j$, we have:

$$\begin{aligned} \overline{\mathbf{x}} \cdot \overline{\mathbf{y}} &= \sum_{i=0}^{m} x_i \cdot (T^i \cdot \overline{\mathbf{y}}) = \sum_{i=0}^{m} x_i \cdot (\sum_{j=0}^{m} y_j T^{i+j}) \\ &= \sum_{i=0}^{m} x_i \cdot (\sum_{j=0}^{m} y_{\langle j-i \rangle} T^j) = \sum_{j=0}^{m} (\sum_{i=0}^{m} x_i \cdot y_{\langle j-i \rangle}) T^j. \end{aligned}$$

This is of course just the cyclic convolution of $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$.

4.2 Nogami multiplication formulae

Nogami *et al.* proposed the use of so-called all-one polynomials to define extensions of prime fields [29]. In this section we will describe their algorithm in this context, and show how it fits into the framework developed in §3.

Let $\mathbb{F}_p$ be a prime field and assume $f(t) = t^m + t^{m-1} + \cdots + t + 1$ is irreducible over $\mathbb{F}_p$. Then $\mathbb{F}_{p^m} = \mathbb{F}_p[t]/(f(t)\mathbb{F}_p[t])$, which the authors refer to as the all-one polynomial field (AOPF). Using the polynomial basis $\{\omega^m, \omega^{m-1}, \ldots, \omega\}$ - rather than the more conventional $\{\omega^{m-1}, \ldots, \omega, 1\}$ - where ω is a root of $f(t) \bmod p$, elements of $\mathbb{F}_{p^m}$ are represented as vectors of length m over $\mathbb{F}_p$:

$$\overline{\mathbf{x}} = [x_m, \ldots, x_1] = x_m\omega^m + x_{m-1}\omega^{m-1} + \cdots + x_1\omega.$$

Let $\overline{\mathbf{x}} = [x_m, \ldots, x_1]$ and $\overline{\mathbf{y}} = [y_m, \ldots, y_1]$ be two elements to be multiplied. For $0 \leq i \leq m$, let

$$q_i = \sum_{j=1}^{m/2} (x_{\langle \frac{i}{2}+j \rangle} - x_{\langle \frac{i}{2}-j \rangle})(y_{\langle \frac{i}{2}+j \rangle} - y_{\langle \frac{i}{2}-j \rangle}), \tag{7}$$

where the subscript $\langle i \rangle$ here, as in §4.1, denotes $i \pmod{m+1}$. Then we have:

$$\overline{\mathbf{z}} = \overline{\mathbf{x}} \cdot \overline{\mathbf{y}} = \sum_{i=1}^{m} z_i \omega^i,$$

where $z_i = q_0 - q_i$. Nogami *et al.* refer to these coefficient formulae as the *cyclic vector multiplication algorithm*.

This is a remarkable formula, since the number of $\mathbb{F}_p$ multiplications is reduced relative to the schoolbook method from m^2 to $m(m+1)/2$, but at the cost of increasing the number of $\mathbb{F}_p$ additions from $m^2 - 1$ to $3m(m-1)/2 - 1$.

The simple insight of the present work is the observation that one may apply the expressions in (7) to MRCP multiplication, providing that one utilises the representation and reduction methodology of Chung and Hasan, to give a full modular multiplication algorithm.

Note that Karatsuba-Ofman multiplication [23] offers a similar trade-off for extension field arithmetic. Crucially however, as shown in §4.6, when we apply these formulae to MRCPs the number of additions required when using this method is reduced in comparison to those required when using (6). One thus expects the Nogami multiplication-based algorithm to be significantly more efficient.

The original proof given in [29] excluded a couple of intermediate steps, and so for the sake of clarity we give a full proof in §4.4, beginning with the following motivation.

4.3 Alternative bases

Observe that in the set of equations (7), each of the $2(m+1)$ coefficients x_j, y_j is featured $m+1$ times, and so there is a nice symmetry and balance to the formulae. However due to the choice of basis, both x_0 and y_0 are implicitly assumed to be zero. The output $\overline{\mathbf{z}}$ naturally has this property also, and indeed if one extends the multiplication algorithm to compute z_0 we see that it equals $q_0 - q_0 = 0$.

At first sight, the expression $z_i = q_0 - q_i$ may seem a little unnatural. It is easy to change the basis from $\{\omega^m, \ldots, \omega\}$ to $\{\omega^{m-1}, \ldots, \omega, 1\}$: for $\overline{\mathbf{x}} = [x_{m-1}, \ldots, x_0]$ and $\overline{\mathbf{y}} = [y_{m-1}, \ldots, y_0]$, we have:

$$\overline{\mathbf{z}} = \overline{\mathbf{x}} \cdot \overline{\mathbf{y}} = \sum_{i=0}^{m-1} z_i \omega^i,$$

resulting in the expressions $z_i = q_m - q_i$, with q_i as given before. This change of basis relies on the relation

$$\omega^m \equiv -1 - \omega - \cdots - \omega^{m-1} \bmod f(t). \tag{8}$$

Note that in using this basis we have implicitly assumed that instead of x_0 and y_0 being zero, the x_m and y_m occurring in (7) are now zero, and again the above formula is consistent since $z_m = q_m - q_m = 0$. More generally if one excludes ω^k from the basis, then $z_i = q_k - q_i$.

One may infer from these observations that the natural choice of basis is $\{\omega^m, \ldots, \omega, 1\}$, and that the expressions for q_i arise from the arithmetic in the quotient ring $R = \mathbb{F}_p[t]/((t^{m+1} - 1)\mathbb{F}_p[t])$, and not from $\mathbb{F}_p[t]/(f(t)\mathbb{F}_p[t])$.

The most natural basis would therefore seem to be $[\omega^m, \ldots, \omega, 1]$. In this case multiplication becomes

$$\overline{\mathbf{z}} = \overline{\mathbf{x}} \cdot \overline{\mathbf{y}} = \sum_{i=0}^{m-1} z_i \omega^i = \sum_{i=0}^{m-1} (q_m - q_i)\omega^i = \sum_{i=0}^{m} -q_i \omega^i,$$

where for the last equality we have again used equation (8).

4.4 Derivation of coefficient formulae

We now derive the expressions in (7), which we hereafter refer to as the Nogami formulae. Let $\overline{\mathbf{x}} = [x_m, \ldots, x_0] = \sum_{i=0}^{m} x_i \omega^i$, and $\overline{\mathbf{y}} = [y_m, \ldots, y_0] = \sum_{i=0}^{m} y_i \omega^i$. Then in the ring R, as in (6) the product $\overline{\mathbf{x}} \cdot \overline{\mathbf{y}}$ is equal to $\sum_{i=0}^{m} z_i \omega^i$, where

$$z_i = \sum_{j=0}^{m} x_{\langle j \rangle} y_{\langle i-j \rangle}.$$

Of crucial importance is the following identity: for $0 \leq i \leq m$ we have:

$$\begin{aligned} &2\sum_{j=0}^{m} x_{\langle j \rangle} y_{\langle i-j \rangle} - 2\sum_{j=0}^{m} x_{\langle j \rangle} y_{\langle j \rangle} \\ &= -\sum_{j=0}^{m} (x_{\langle j \rangle} - x_{\langle i-j \rangle})(y_{\langle j \rangle} - y_{\langle i-j \rangle}). \end{aligned} \tag{9}$$

To verify this identity observe that when one expands the terms in the right-hand side, the two negative sums cancel with the second term on the left-hand side, since both are over a complete set of residues modulo $m+1$. Similarly the two positive sums are equal and therefore cancel with the convolutions in the first term on the left-hand side.

Nogami *et al.* observed that there is some redundancy in the right-hand side of (9), in the following sense. Firstly, observe that

$$\begin{aligned} \sum_{j=0}^{m} x_{\langle \frac{i}{2}+j \rangle} y_{\langle \frac{i}{2}-j \rangle} &= \sum_{j=0}^{m} x_{\langle \frac{i}{2}+(j-\frac{i}{2}) \rangle} y_{\langle \frac{i}{2}-(j-\frac{i}{2}) \rangle} \\ &= \sum_{j=0}^{m} x_{\langle j \rangle} y_{\langle i-j \rangle}. \end{aligned}$$

One can therefore rewrite the right-hand side of (9) as:

$$-\sum_{j=0}^{m} (x_{\langle \frac{i}{2}+j \rangle} - x_{\langle \frac{i}{2}-j \rangle})(y_{\langle \frac{i}{2}+j \rangle} - y_{\langle \frac{i}{2}-j \rangle}) \tag{10}$$

Noting that the $j = 0$ term is zero, it can therefore be rewritten:

$$\begin{aligned} &-\sum_{j=1}^{m/2} (x_{\langle \frac{i}{2}+j \rangle} - x_{\langle \frac{i}{2}-j \rangle})(y_{\langle \frac{i}{2}+j \rangle} - y_{\langle \frac{i}{2}-j \rangle}) \\ &-\sum_{j=m/2+1}^{m} (x_{\langle \frac{i}{2}+j \rangle} - x_{\langle \frac{i}{2}-j \rangle})(y_{\langle \frac{i}{2}+j \rangle} - y_{\langle \frac{i}{2}-j \rangle}), \end{aligned}$$

which in turn equals

$$\begin{aligned} &-\sum_{j=1}^{m/2}(x_{\langle \frac{i}{2}+j\rangle} - x_{\langle \frac{i}{2}-j\rangle})(y_{\langle \frac{i}{2}+j\rangle} - y_{\langle \frac{i}{2}-j\rangle}) \\ &-\sum_{j=1}^{m/2}(x_{\langle \frac{i}{2}-j\rangle} - x_{\langle \frac{i}{2}+j\rangle})(y_{\langle \frac{i}{2}-j\rangle} - y_{\langle \frac{i}{2}+j\rangle}), \end{aligned} \tag{11}$$

which again equals

$$2\sum_{j=1}^{m/2}(x_{\langle \frac{i}{2}+j\rangle} - x_{\langle \frac{i}{2}-j\rangle})(y_{\langle \frac{i}{2}+j\rangle} - y_{\langle \frac{i}{2}-j\rangle}).$$

For the final equalilty we have used the fact that in the second summation in expression (11) the two terms are the negation of the two terms in the first sum. Hence (9) becomes

$$\begin{aligned} \sum_{j=0}^{m} x_{\langle j\rangle}y_{\langle i-j\rangle} &= \sum_{j=0}^{m} x_{\langle j\rangle}y_{\langle j\rangle} \\ &- \sum_{j=1}^{m/2}(x_{\langle \frac{i}{2}+j\rangle} - x_{\langle \frac{i}{2}-j\rangle})(y_{\langle \frac{i}{2}+j\rangle} - y_{\langle \frac{i}{2}-j\rangle}) \end{aligned} \tag{12}$$

Equation (12) gives an expression for the coefficients of the product $\overline{\mathbf{z}}$ of elements $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$, in the ring R. Assuming these are computed using the more efficient right-hand side, in order to restrict to $\mathbb{F}_p[t]/(f(t)\mathbb{F}_p[t])$, one can reduce the resulting polynomial $\overline{\mathbf{z}}$ by $f(t)$. Note that one does not need to use a smaller basis however, à la Nogami in §4.2, but can in fact reduce by $f(t)$ *implicitly*, without performing any computation; in fact it saves some in the process.

Letting $\langle \overline{\mathbf{x}}, \overline{\mathbf{y}}\rangle = \sum_{j=0}^{m} x_{\langle j\rangle}y_{\langle j\rangle}$, we have:

$$\begin{aligned} \overline{\mathbf{z}} &= \sum_{i=0}^{m} z_i\omega^i = \sum_{i=0}^{m}(-q_i + \langle \overline{\mathbf{x}}, \overline{\mathbf{y}}\rangle)\omega^i \\ &= \sum_{i=0}^{m} -q_i\omega^i + \langle \overline{\mathbf{x}}, \overline{\mathbf{y}}\rangle \sum_{i=0}^{m}\omega^i \equiv \sum_{i=0}^{m} -q_i\omega^i \bmod f(t). \end{aligned}$$

Therefore the first term on the right-hand side of (12) vanishes, so that one needs not even compute it. Thus using the arithmetic in R but implicitly working modulo $f(t)$ is more efficient than performing arithmetic in R alone. This is somewhat fortuitous as it means that while the multiply operation in R is not correct, nevertheless, when one maps back to $\mathbb{F}_p[t]/(f(t)\mathbb{F}_p[t])$ the result is correct.

Since reduction in the ring R has a particularly nice form for our application to prime fields, we choose not to reduce *explicitly* modulo $f(t)$, to give the formulae in §4.3. This also has the effect of eliminating the need to perform the addition of q_0 (or q_m, or whichever term one wants to eliminate when one reduces modulo $f(t)$), simplifying the multiplication algorithm further.

4.5 Application to MRCPs

Since equation (9) is an algebraic identity, it is easy to see that exactly the same expressions apply in the context of MRCPs, and we can replace the formulae (6) with Nogami's formulae. Absorbing the minus sign into the q_i, Algorithm 1 details how to multiply residue representatives.

ALGORITHM 1: **MRCP MULTIPLICATION**

INPUT: $\overline{\mathbf{x}} = [x_m, \ldots, x_0], \overline{\mathbf{y}} = [y_m, \ldots, y_0] \in \mathbb{Z}^{m+1}$
OUTPUT: $\overline{\mathbf{z}} = [z_m, \ldots, z_0] \in \mathbb{Z}^{m+1}$
where $\overline{\mathbf{z}} \cong_{\Phi_{m+1}(t)} \overline{\mathbf{x}} \cdot \overline{\mathbf{y}}$

1. For $i = m$ to 0 do:
2. $\quad z_i \leftarrow \sum_{j=1}^{m/2}(x_{\langle \frac{i}{2}-j\rangle} - x_{\langle \frac{i}{2}+j\rangle}) \cdot (y_{\langle \frac{i}{2}+j\rangle} - y_{\langle \frac{i}{2}-j\rangle})$
3. Return $\overline{\mathbf{z}}$

4.6 Cost comparison

We here give a simple cost model, to indicate the potential performance improvement achieved by using Algorithm 1 rather than the convolution (6).

Let the inputs to the multiplication algorithm have coefficients bounded by b^q, i.e., they each consist of q words. Let $M(q,q)$ be the cost of a q-word by q-word schoolbook multiplication, and let $A(q,q)$ be the cost of addition of two q-word values. We assume that $A(2q,2q) = 2 \cdot A(q,q)$. Then the cost of the multiplication using each method is as follows.

4.6.1 Convolution formulae

For each z_i the multiplication cost is $(m+1) \cdot M(q,q)$, while the addition cost is $m \cdot A(2q,2q)$, which is $2m \cdot A(q,q)$. Since there are $m+1$ terms, the total cost is

$$(m+1)^2 \cdot M(q,q) + 2m(m+1) \cdot A(q,q).$$

4.6.2 Nogami formulae

For each z_i computing each term in the sum costs $(2 \cdot A(q,q) + M(q,q))$, and so computing all these terms costs $\frac{m}{2} \cdot (2 \cdot A(q,q) + M(q,q))$. The cost of summing these is $(\frac{m}{2} - 1)A(2q,2q) = (m-2) \cdot A(q,q)$.

For all the $m+1$ terms z_i the total cost is therefore

$$\frac{m(m+1)}{2} \cdot M(q,q) + 2(m^2-1) \cdot A(q,q).$$

By using the Nogami formulae, we therefore reduce not only the number of multiplications, but also the number of additions, contrary to the case of field extensions, as stated in §4.2.

5 MRCP Reduction

In this section we detail two algorithms for performing reduction for MRCPs. The first, Algorithm 2, assumes only that t is even, which is not a great restriction on the number of t's which for a given $m+1$ produce integers $\Phi_{m+1}(t)$ of a given bitlength. Using this algorithm each reduction reduces the input's components by approximately $\log_2 b$ bits, where b is the wordbase of the underlying architecture.

The second, Algorithm 3, assumes that $t \equiv 0 \bmod 2^l$ for some $l > 1$, and each application of the reduction function reduces the input's components by approximately l bits, and is extremely efficient. Ideally one chooses a t for which $l > (\log_2 t)/2$ so that only two applications of the reduction function are sufficient. In general for other values of l a larger number of reductions may need to be computed, which we consider in §7.

For completeness we present both algorithms, as well as a slight modification of Algorithm 3, which is slightly more efficient. It should be clear from our treatment that for suitably chosen t, Algorithm 3 and its modification are considerably more efficient than Algorithm 2.

In the extreme case that $t = 2^l$, we see that such primes are very similar to the Generalised Mersenne Numbers [33]. In this case, one can use the reduction method detailed in §3.4 without resorting to using its Montgomery version at all. As with GMNs, post-multiplication reduction will necessitate some modular additions/subtractions since the output of a multiply will have length a few bits more than $2\log_2 t$. However the multiplication would clearly be faster thanks to Nogami's formulae.

As we detail in §7, by choosing t, l and $m+1$ carefully, one can avoid the need to perform any final additions or subtractions, as demonstrated in [9].

5.1 MRCP reduction: t even

Following §3.5, in equation (4) we need the matrix $-F^{-1}$:

$$-F^{-1} = \frac{1}{t^{m+1}-1}\begin{bmatrix} 1 & t^m & \cdots & t^3 & t^2 & t \\ t & 1 & \cdots & t^4 & t^3 & t^2 \\ t^2 & t & \cdots & t^5 & t^4 & t^3 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ t^{m-1} & t^{m-2} & \cdots & t & 1 & t^m \\ t^m & t^{m-1} & \cdots & t^2 & t & 1 \end{bmatrix}.$$

The form of F and $-F^{-1}$ allows one to compute $\overline{\mathbf{u}} = -F^{-1} \cdot \overline{\mathbf{z}} \bmod b$, and $F \cdot \overline{\mathbf{u}}$, computed in equation (5), very easily. Since t is even, the following vector may be computed. Let $t[0]$ be the least significant digit of t, written in base b, and let

$$\overline{\mathbf{L}} \stackrel{\text{def}}{=} \frac{1}{t[0]^{m+1}-1}[t[0]^m, t[0]^{m-1}, \ldots, t[0], 1] \bmod b.$$

Algorithm 2 details how to reduce a given an input vector $\overline{\mathbf{z}}$ by b, modulo $t^{m+1}-1$, given the precomputed vector $\overline{\mathbf{L}}$.

ALGORITHM 2: **red1**$_b(\overline{\mathbf{z}})$

```
INPUT:  z̄ = [z_m, ..., z_0] ∈ Z^{m+1}
OUTPUT: red_b(z̄) where red_b(z̄) ≅_{t^{m+1}-1} z̄ · b^{-1}
1.  Set u_0 ← (Σ_{i=0}^{m} L_i · z_i[0]) mod b
2.  For i = m to 0 do:
3.      v_i ← t · u_{⟨i+1⟩}
4.      u_i ← v_i[0] − z_i[0] mod b
5.      w_i ← (z_i + u_i − v_i)/b
6.  Return w̄
```

Algorithm 2 greatly simplifies the reduction algorithm originally given in [9]. This is possible since for $f(t) = t^{m+1}-1$ one can avoid using the matrix $F_2^{'}$ altogether. Doing so allows one to interleave the computation of the vectors $\overline{\mathbf{u}}$ and $\overline{\mathbf{w}}$ defined in (4) and (5) respectively. This has two benefits. Firstly, as we can compute each output component sequentially, we need only store a single component of each vector, rather than $m+1$. Secondly, since when one computes $F \cdot \overline{\mathbf{u}}$ one needs to compute $t \cdot u_{\langle i+1 \rangle}$ for $i = m, \ldots, 0$ (line 3), one obtains $t[0] \cdot u_i$ (the first term on right-hand side of line 4) for free by computing the full product $t \cdot u_{\langle i+1 \rangle}$ first. We therefore avoid recomputing the least significant digit of $t \cdot u_{\langle i+1 \rangle}$ in each loop iteration.

In fact one can do this for any polynomial $f(t) = t^n - c$, with exactly the same algorithm: the only difference being in the definition of the precomputed vector $\overline{\mathbf{L}}$, where $f(t)$ is changed in the denominator. For polynomials with other non-zero coefficients, this does not seem possible, and so Algorithm 2 seems to be as efficient as one could hope for Chung-Hasan reduction with such a weak restriction on the form of t.

Note that in the final loop iteration, u_0 from line 1 is recomputed, which is therefore unnecessary. However, we chose to write the algorithm in this form to emphasise its cyclic structure. Indeed, there is no need to compute u_0 first; if one cyclically rotates $\overline{\mathbf{L}}$ by j places to the left, then the vector $\overline{\mathbf{w}}$ to be added to $\overline{\mathbf{z}}$ (cf. equation (5)) is rotated j places to the left also. One can therefore compute each coefficient of red$1_b(\overline{\mathbf{z}})$ independently of the others using a rotated definition for $\overline{\mathbf{L}}$ (or equivalently by rotating the input $\overline{\mathbf{z}}$). This suggests that a parallelised version of the reduction algorithm may be very attractive. For other polynomials of the form $f(t) = t^n - c$, one needs to alter $\overline{\mathbf{L}}$ more significantly to make this approach work.

It is a rather straightforward to verify that Algorithm 2 correctly produces an output vector in the correct congruency class, via a sequence of simple transformations of [9, Algorithm 3]. However we do not do so here, since we are mainly interested in the far more efficient Algorithm 3.

5.2 MRCP reduction: $t \equiv 0 \bmod 2^l$

Hoping not to cause confusion, for the sake of a uniform nomenclature in this section and throughout the rest of

the paper, we now let $b = 2^l$ where l is not necessarily and usually not the wordsize of an underlying architecture. We denote the cofactor of b in t by c, so that $t = b \cdot c$.

Algorithm 3 details how to reduce a given an input vector $\overline{\mathbf{z}}$ by b, modulo $t^{m+1} - 1$.

ALGORITHM 3: **red2**$_b(\overline{\mathbf{z}})$

INPUT: $\overline{\mathbf{z}} = [z_m, \ldots, z_0] \in \mathbb{Z}^{m+1}$
OUTPUT: $\mathrm{red}_b(\overline{\mathbf{z}})$ where $\mathrm{red}_b(\overline{\mathbf{z}}) \cong_{t^{m+1}-1} \overline{\mathbf{z}} \cdot b^{-1}$

1. For $i = m$ to 0 do:
2. $\quad w_i \leftarrow (z_i + (-z_i \bmod b))/b - c \cdot (-z_{\langle i+1 \rangle} \bmod b)$
3. Return $\overline{\mathbf{w}}$

A simple proof of correctness of Algorithm 3 comes from a specialisation of Algorithm 2. Since $t \equiv 0 \bmod b$, writing t in base b, the vector $\overline{\mathbf{L}}$ becomes

$$\overline{\mathbf{L}} \stackrel{\text{def}}{=} [0, \ldots, 0, -1] \bmod b.$$

Hence for line 1 of Algorithm 2 we have

$$u_0 \leftarrow -z_0[0] \bmod b.$$

Since in line 4 of Algorithm 2, we have $v_i \equiv 0 \bmod b$, we deduce that $u_i = -z_i \bmod b$, and hence we can eliminate the u_i altogether. Each loop iteration then simplifies to

$$\begin{aligned} v_i &\leftarrow t \cdot (-z_{\langle i+1 \rangle} \bmod b) \\ w_i &\leftarrow (z_i + (-z_i \bmod b) - v_i)/b \end{aligned} \tag{13}$$

Upon expanding (13), we obtain

$$\begin{aligned} w_i &\leftarrow (z_i + (-z_i \bmod b))/b - t \cdot (-z_{\langle i+1 \rangle} \bmod b)/b \\ &= (z_i + (-z_i \bmod b))/b - c \cdot (-z_{\langle i+1 \rangle} \bmod b), \end{aligned}$$

as required.

However since we did not provide a proof of correctness of Algorithm 2, we also give a direct proof as follows. Observe that modulo $t^{m+1} - 1$, we have

$$\psi^{-1}(\overline{\mathbf{w}}) \equiv \sum_{i=0}^{m} w_i t^i$$

which is equivalent to:

$$\begin{aligned} &\sum_{i=0}^{m} [(z_i + (-z_i \bmod b))/b - c \cdot (-z_{\langle i+1 \rangle} \bmod b)] t^i \\ \equiv &\sum_{i=0}^{m} (z_i/b) t^i + \sum_{i=0}^{m} (-z_i \bmod b))/b) t^i \\ - &\sum_{i=0}^{m} ((-z_{\langle i+1 \rangle} \bmod b)/b) t^{i+1} \\ \equiv &\sum_{i=0}^{m} z_i t^i/b \pmod{t^{m+1} - 1} \end{aligned}$$

as required.

In terms of operations that may be performed efficiently, we alter Algorithm 3 slightly to give Algorithm 4, which has virtually the same proof of correctness as the one just given.

ALGORITHM 4: **red3**$_b(\overline{\mathbf{z}})$

INPUT: $\overline{\mathbf{z}} = [z_m, \ldots, z_0] \in \mathbb{Z}^{m+1}$
OUTPUT: $\mathrm{red}_b(\overline{\mathbf{z}})$ where $\mathrm{red}_b(\overline{\mathbf{z}}) \cong_{t^{m+1}-1} \overline{\mathbf{z}} \cdot b^{-1}$

1. For $i = m$ to 0 do:
2. $\quad w_i \leftarrow z_i/b + c \cdot (z_{\langle i+1 \rangle} \bmod b)$
3. Return $\overline{\mathbf{w}}$

Note that the first term in line 2 of Algorithm 3 has been replaced by a division by b, which can be performed as a simple shift, while the second term needs only the positive residue modulo b, which can be read efficiently. Hence Algorithm 4 is the one we shall use henceforward.

6 MRCP Residue Representation

So far in our treatment of both multiplication and reduction, for the sake of generality we have assumed arbitrary precision when representing MRCP residues. In this section we specialise to some extent and develop a residue representation that ensures that our chosen algorithms are efficient. As stated in §4 our decisions are informed purely by our chosen multiplication and reduction algorithms (Algorithms 1 and 4), which we believe offer the best performance for MRCPs, at least for relatively small bitlengths which are relevant to ECC for example. In other scenarios or if considering asymptotic performance, one may need to redesign the residue representation and multiplication algorithm accordingly, which we omit here for reasons of space.

We also specialise to single precision t, since this obviates the need for multiprecision arithmetic; utilising the native double precision multipliers that most CPUs possess is more efficient.

For $x \in \{0, \ldots, t^{m+1} - 1\}$ we write $\overline{\mathbf{x}} = [x_m, \ldots, x_0]$ for its base-t expansion, i.e., $\overline{\mathbf{x}} = \sum_{i=0}^{m} x_i t^i$ The base-t representation has positive coefficients, however Algorithm 1 makes use of negative coefficients, so we prefer to incorporate these. We therefore replace the mod function in the conversion with mods, the least absolute residue function, to obtain a residue in the interval $[-t/2, t/2 - 1]$:

$$\mathrm{mods}(x) = \begin{cases} x \bmod t & \text{if } (x \bmod t) < t/2, \\ x \bmod t - t & \text{otherwise.} \end{cases}$$

Using this function, Algorithm 5 converts residues modulo $t^{m+1} - 1$ into the required form [9][Algorithm 1].

ALGORITHM 5: **Conversion to Polynomial Form**

INPUT: An integer $0 \le x < t^{m+1} - 1$
OUTPUT: $\overline{\mathbf{x}} = [x_m \ldots, x_0]$ such that $|x_i| \le t/2$ and $\sum_{i=0}^{m} x_i t^i \equiv x \pmod{t^{m+1} - 1}$

1. For i from 0 to m do:

2. $x_i \leftarrow x \bmod t$
3. $x \leftarrow (x - x_i)/t$
4. $x_0 \leftarrow x_0 + x$
5. Return $\overline{\mathbf{x}} = [x_m, \ldots, x_0]$

The reason for line 4 in Algorithm 5 is to reduce modulo $t^{m+1} - 1$ the coefficient of t^{m+1} possibly arising in the expansion. Note that in this addition, $x \in \{0, 1\}$, and hence $|x_i| \leq t/2$ for each $0 \leq i \leq m$. By construction, we in fact have $-t/2 \leq x_i < t/2$ for $1 < i < m$ while only x_0 can attain the upper bound of $t/2$. There are therefore $t^m(t+1)$ representatives in this format, thus introducing a very small redundancy. Letting $k = \lceil \log_2 t \rceil$, if we assume $t \leq 2^k - 2$, so that $[-t/2, t/2] \subset [-2^k/2, 2^k/2 - 1]$, then the coefficients as computed above can be represented in two's complement in k bits.

Following this conversion, it might seem desirable to define vectors whose components are in $[-2^k/2, 2^k/2 - 1]$ to be reduced, or canonical residue representatives. However, the form of our chosen reduction algorithm indicates that we should add two extra bits of redundancy, as we now explain.

Observe that the second term in line 2 of Algorithm 4, $c \cdot (z_{\langle i+1 \rangle} \bmod b)$, is positive, and in the worst case is k bits long. The first term, z_i/b, is clearly $l = \log_2 b$ bits shorter than z_i. Since one adds these the resulting value may be $k + 1$ bits, or larger, depending on the initial length of the input's components. Furthermore, since we wish to allow negative components, in twos complement the output requires a further bit, giving a minimal requirement of $k + 2$ bits for the output.

Therefore if one wants a reduction algorithm which, when peformed sufficiently many times, outputs an element for which one does not have to perform any modular additions or subtractions to make reduced - which is preferable for efficiency - then the definition of reduced should be at least $k+2$ bits. It is therefore more efficient to use not minimally reduced elements as coset representatives in $\mathbb{Z}^{m+1}/\sim$, as output by Algorithm 5, but slightly larger elements, which we now define.

Definition 5: We define the following set of elements of $\mathbb{Z}^{m+1}$ to be *reduced*:

$$\{[x_m, \ldots, x_0] \in \mathbb{Z}^{m+1} \mid -2^{k+1} \leq x_i < 2^{k+1}\} \tag{14}$$

Note that the amount of redundancy inherent in this representation depends on how close t is to 2^{k+2}.

For any modular multiplication, we assume that the inputs are reduced. We must therefore ensure that the output is reduced also. This leads us to consider I/O-stability.

7 Modular Multiplication Stability

In this section we follow Algorithms 1 and 4. For this analysis we assume the following: $b = 2^l$, $t = c \cdot b$ where $c < 2^{k-l}$ (and hence $t < 2^k - 2$), and that reduced elements have the form (14).

Input elements therefore have components in $\mathbb{I} = [-2^{k+1}, 2^{k+1} - 1]$, and these are representable in $k+2$ bits in two's complement. During the multiplication, terms of the form $x_i - x_j$ are computed.

$$-2^{k+2} + 1 \leq x_i - x_j \leq 2^{k+2} - 1,$$

which therefore fit into $k + 3$ bits in two's complement. The product of two such elements is performed, giving a result

$$-2^{2k+4} + 2^{k+3} - 1 \leq (x_i - x_j) \cdot (y_j - y_i) \leq 2^{2k+4} - 2^{k+3} + 1,$$

which fits into $2k+5$ bits in two's complement. One then adds $m/2$ of these terms, giving a possible expansion of up to $\lceil \log_2 m/2 \rceil$ bits, which as we stipulated in §6 must fit in double precision. To distinguish the wordsize from b, we here call this w. We therefore have a constraint on the size of t (in addition to the constraint $t < 2^k - 2$) in terms of m:

$$\lceil \log_2 (m/2) \rceil + 2k + 5 \leq 2w \tag{15}$$

This inequality determines a constraint on the size of t, given m and w. If one requires larger MRCPs, then one can increase m. For relatively large MRCPs, and a fixed architecture, our policy of t being single precision will limit the number of MRCPs available, an issue which can be overcome by using larger t and smaller m, cf. §**??**.

Assuming (15) is satisfied, one then needs to find the minimum value of $b = 2^l$ such that the result of the multiplication step, when reduced by b a specified number of times, say r, outputs a reduced element. This needs to be done for each (m, k) found in the procedure above. The reason for choosing the minimum such b is to maximise the set of prime-producing cofactors c, which may be useful. On the other hand searching for c of a special form may enable better performance. There is therefore a natural trade-off between the number of such c and the cost of reduction, i.e., whether one reduces twice, or thrice etc. In §10 we exhibit some very specialised such c.

In §6, we showed that one application of Algorithm 4 shortened an input's components by $l-1$ bits, unless the components were already shorter than $(k + 2) + (l - 1)$ bits. Therefore if we stipulate that r reductions should suffice to produce a reduced output, we obtain a bound on l in the following manner. Let

$$h = \lceil \log_2 (m/2) \rceil + 2k + 5$$

Then after one reduction, the maximum length of a component is $h - l + 1$. Similarly after r reductions, the maximum length is $\max\{h - r(l-1), k+2\}$, and we need this to be at most $k + 2$. Hence our desired condition is

$$h - r(l - 1) \leq k + 2$$

Solving for l, we have

$$l \geq 1 + \frac{\lceil \log_2 (m/2) \rceil + k + 3}{r} \tag{16}$$

TABLE 1
Stable parameters for $w = 64$, $r = 2$

$m+1$	k	l	$c <$	$\lceil \log_2 p \rceil$
3	61	33	2^{28}	122
5	61	34	2^{27}	244
7	60	34	2^{26}	360
11	60	34	2^{26}	600
13	60	34	2^{26}	720
17	60	34	2^{26}	960

TABLE 2
Stable parameters for $w = 64$, $r = 3$

$m+1$	k	l	$c <$	$\lceil \log_2 p \rceil$
3	61	23	2^{38}	122
5	61	23	2^{38}	244
7	60	23	2^{37}	360
11	60	23	2^{37}	600
13	60	23	2^{37}	720
17	60	23	2^{37}	960

Using these inequalities it is an easy matter to generate triples $(m+1, k, l)$ which ensure multiplication stability for any r and w. For example, for $r = 2$ and $w = 64$, Tables 1 and 2 give sets of stable parameters for $r = 2$ and $r = 3$ respectively.

The final column gives the maximum bitlength of an MRCP that can be represented with those parameters, though of course using smaller c one can opt for smaller primes. To generate suitable MRCPs, a simple linear search over the values of c of the desired size is sufficient, checking whether or not $\Phi_{m+1}(2^l \cdot c)$ is prime.

8 MRCP Full Modular Multiplication

In this short section, for completeness we can now glue together the parts treated thus far into a full modular multiplication algorithm.

ALGORITHM 6: **MRCP MODMUL**

INPUT: $\overline{\mathbf{x}} = [x_m, \ldots, x_0], \overline{\mathbf{y}} = [y_m, \ldots, y_0] \in \mathbb{I}^{m+1}$
OUTPUT: $\overline{\mathbf{z}} = [z_m, \ldots, z_0] \in \mathbb{I}^{m+1}$ where $\overline{\mathbf{z}} \cong_{\Phi_{m+1}(t)} \overline{\mathbf{x}} \cdot \overline{\mathbf{y}} \cdot b^{-r}$

1. For $i = m$ to 0 do:
2. $\quad z_i \leftarrow \sum_{j=1}^{m/2} (x_{\langle \frac{i}{2} - j \rangle} - x_{\langle \frac{i}{2} + j \rangle}) \cdot (y_{\langle \frac{i}{2} + j \rangle} - y_{\langle \frac{i}{2} - j \rangle})$
3. For i from 0 to $r-1$ do:
4. $\quad \overline{\mathbf{z}} \leftarrow \mathrm{red}_b(\overline{\mathbf{z}})$
5. Return $\overline{\mathbf{z}}$

9 Other Arithmetic Operations

Clearly if modular multiplication was the only operation required then one could do this very efficiently using the discrete logarithm of each non-zero element in $\mathbb{F}_p$ relative to a fixed generator, in which case multiplication becomes addition modulo $p-1$. For nearly all applications however the full of set of field operations is required. While our central focus is on modular multiplication, in this section we very briefly describe how each should be performed for MRCPs with our representation.

9.1 Addition/Subtraction

To perform an addition or subtraction of two reduced elements $\overline{\mathbf{x}}, \overline{\mathbf{y}}$, we first compute the following:

$$\overline{\mathbf{x}} \pm \overline{\mathbf{y}} = [x_m \pm y_m, \ldots, x_0 \pm y_0].$$

Note that the bounds on each of these components is $[-2^{k+2}, 2^{k+2} - 2]$. We therefore require an equivalent reduced element for stability.

Algorithm 5 from [9] shows how to do this in general, and it is easy to specialise to MRCPs. Chung and Hasan refer to this process as *Short Coefficient Reduction* (SCR). For reasons of space we do not explicitly state this algorithm here, but note that in some applications one can not only delay reductions for efficiency purposes [**?**], but can avoid using the SCR algorithm altogether, by incorporating these into the full reductions performed after a multiplication.

For example, during a point multiplication in ECC, one performs a sequence of point additions and point doublings. By analysing the sequence of field operations in each, one can tailor the definition of a reduced element so that only full reductions need be computed, thus averting the use of the SCR algorithm.

9.2 Squaring

When t is single precision, the Nogami formulae do not have any common subexpression as arises for ordinary integer residue squaring. In this case the MRCP squaring algorithm is identical to Algorithm 6.

If t is multiprecision, then the components of a product $\overline{\mathbf{x}} \cdot \overline{\mathbf{y}}$ are computed as a some of integer squares. In this case, one can freely use the same common subexpression elimination to improve squaring efficiency [**?**].

9.3 Inversion

Inversion appears to be difficult to perform in the MRCP representation. We therefore opt to map back to $\mathbb{F}_p$ and perform inversion using the Extended Euclid Algorithm [**?**]. In applications, one often attempts to avoid inversions if at all possible. Indeed in ECC, the use of projective coordinates obviates this requirement for precisely this reason [**?**].

9.4 Equality check

Since our representation has some redundancy, equality checking is also cumbersome, and so we opt to map back to $\mathbb{F}_p$ as for inversions. Again in ECC this is usually only required very infrequently, and so does not greatly impinge upon application efficiency.

10 Implementation and Results

To measure the performance of field arithmetic we must consider the target applications, and the best candidates for each application. For use in ECC we can specialise an algorithm for one particular prime in order to maximise the performance. An extreme example of this approach is given by curve255, using the Crandall prime $2^{255} - 19$ a reduction algorithm is devised that exploits the high throughput of floating-point arithmetic on modern x86 architectures. The algorithm is tied to both the specific structure of the prime, and the characteristics of the target processor. This low-level approach requires the programmer to reschedule the assembly code for each target micro-architecture, but produces the highest performance benchmark to date.

It is difficult to draw a comparison between the non-quantitive properties of algorithms, such as ease of implementation. In comparison with the manually scheduled and optimised code of curve255 we have written two inline macros to perform the core operations in our algorithm. One accumulates the innermost sum of Algorithm 1, $(a+ = (x_i - x_j) * (y_j - y_i))$, while the other performs a single operation of Algorithm 4 reducing a single component. These assembly operations use a mere four of the fifteen available in the x86-64 instruction set. This allows us to rely on normal C code to arrange these macros, and to handle data-storage. As a result the gcc compiler can perform the register scheduling without manual intervention by the programmer. This means that the same code can be reused for any field supported by the algorithm —- the only changes required are the parameter definitions.

To compare our performance quantitatively against the highest performance candidates we have used the mpFq benchmarking system. This has been ported to OS-X 10.5.8 with minor changes and executed on a platform using an Intel Core 2 Duo at 2.2Ghz. This micro-architecture was chosen as it gives the highest performance results for the mpFq benchmarks, which to the authors knowledge are the fastest implementations of prime field arithmetic available. To ensure a fair comparison we have merged our code into mpFq so that all algorithms are being tests with the same timing code. This timer executes 10^6 operations in the field, measuring the elapsed time. The reported figures are the mean execution time for the operation.

Our implementation consists of two inline assembly operations targetted at the Core 2 processor. One accumulates the innermost sum of Algorithm 1, while the other performs a single instance of the reduction operation. Both use the 64-bit operations available on the Core 2 and the extended register set available in x86_64. To generate a particular instance of the family of algorithms we use a simple wrapper written in Python that arranges the sequence of these operations required for the particular parameter choice of m. The storage used is simply C arrays of 64-bit words. The multiplier macro uses just four registers, and the reduction step uses four. We allow the gcc compiler to generate all of the intermediate memory access instructions and schedule the usage of the other 11 registers available.

TABLE 3
Cycle counts for Curve255 and Montgomery arithmetic

Algorithm	Size (bits)	Mult (cycles)
M.	64	30
M.	128	105
M.	192	195
curve255	255	140
M.	256	280
M.	320	407
M.	384	563
M.	448	757
M.	512	981

TABLE 4
Cycle counts for MRCP arithmetic

Parameters	Size (bits)	Mult (cycles)
m=4, l=59, c=3	244	96
m=4, general c	244	112
m=6, $l \geq 34$, $HW(c) = 2$	360	165
m=6, general c	360	182
m=10, general c	600	340

To emphasise the relative simplicity of our implementation, we use only 64-bit scalar operations on the processor, and allow the compiler to schedule most of the output instructions. As a result we reach a throughput of slightly less than one operation per cycle. The mpFq implementation of curve255 uses SSE2 to reach a throughput of almost two operations per cycle (the theoretical maximum on the architecture). Although our implementation is less efficient (because we have spent less programmer time on the machine-dependent optimisation) the performance achieved is still higher. Scheduling a lower-level implementation on the processor would be an interesting challenge.

Within the reduction algorithm we have two main choices that impact performance. If we opt for a generic value of c then more prime fields can be used, but the reduction involves a full `imulq` instruction with relatively high latency. If we specialise our choice of c then we can replace this instruction with a combination of shift and add instructions to improve performance. We have measured the performance of both implementations (using $c = 3$ for the specialised version). The closest size of field that we can implement to curve255 is only 244-bits. This small reduction in field size is compensated by a significant increase in performance. Using the specialised reduction a multiplication operation only take 69% of the time taken by curve255, for a 46% increase in throughput.

11 Conclusion

We propose a modular multiplication algorithm for a large family of prime fields. The multiplication operation is simpler to implement than the highest performing previous work, yet yields higher performance across a wide range of field sizes. The main contribution of this paper is the novel algorithms to perform multiplication and reduction in these fields, and the development of the necessary theory. We present the results from an empirical comparison with the best results from the literature, ensuring a fair comparison by using the highest performing platform for the competing implementations, and reusing the same benchmarking procedure. Against the fixed prime algorithm curve255 for high performance ECC we show a 50% increase in speed. Against the more general Montgomery algorithm we show a 4-fold increase in performance.

By performing better than any competing algorithm across such a wide range of field sizes we present a compelling argument for a single approach to optimised field arithmetic.

Acknowledgements

The authors would like to thank Dan Page for making several very useful observations and suggestions, and Mike Scott for suggesting using special cofactors to improve the reduction algorithm.

References

[1] P. Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor, In *Advances in CryptologyCRYPTO 86* Springer-Verlag, LNCS 263, 311323, 1987.

[2] D.J. Bernstein. A software implementation of NIST P-224. Presentation at the 5th Workshop on Elliptic Curve Cryptography (ECC 2001), University of Waterloo, October 29-31, 2001. Slides available from http://cr.yp.to/talks.html.

[3] D.J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In *Public Key Cryptography - PKC 2006*, LNCS 3958, 207–228. Springer-Verlag, 2006.

[4] I.F. Blake, R.M. Roth and G. Seroussi. Efficient Arithmetic in $GF(2^m)$ through Palindromic Representation. Technical Report HPL-98-134, 1998. Available from http://www.hpl.hp.com/techreports/98/HPL-98-134.html.

[5] Ian Blake, Gadiel Seroussi, and Nigel Smart. Elliptic Curves in Cryptography. *London Mathemtical Society Lecture Note Series*, 265, Cambridge University Press, 1999.

[6] M. Brown, D. Hankerson, J. López, and A. Menezes Software Implementation of the NIST Elliptic Curves Over Prime Fields In *Topics in Cryptology CT-RSA 2001*, LNCS 2020, 250–265, Springer.

[7] J. Chung A. Hasan. More Generalized Mersenne Numbers. In *Selected Areas in Cryptography*, volume 3006 of LNCS, 335 – 347. Springer, 2004.

[8] J. Chung and A. Hasan. Low-Weight Polynomial Form Integers for Efficient Modular Multiplication. In *IEEE Trans. Comput.*, 56-1, 44–57, 2007.

[9] J. Chung and A. Hasan. Montgomery Reduction Algorithm for Modular Multiplication Using Low-Weight Polynomial Form Integers. In *ARITH 18*, 230–239, 2007

[10] G. Drolet. A new representation of elements of finite fields $GF(2^m)$ yielding small complexity arithmetic circuits. *IEEE Trans. Comput.*, 47(9): 938–946, 1998.

[11] ECC Brainpool Standard Curves and Curve Generation. Available from http://www.bsi.bund.de/english/index.htm.

[12] FIPS 186-2. Digital Signature Standard. Federal Information Processing Standards Publication 186-2, US Department of Commerce/N.I.S.T. 2000.

[13] FIPS 186-3. Digital Signature Standard. Federal Information Processing Standards Publication 186-3, US Department of Commerce/N.I.S.T. 2006.

[14] S. Gao, J. von zur Gathen, and D. Panario. Guass periods and fast exponentiation in finite fields. *LNCS*, volume 911, 311-322, Springer-Verlag, 1995.

[15] S. Gao and S. Vanstone. On Orders of Optimal Normal Basis Generators. *Math Comp*, 64(2):1227-1233, 1995.

[16] P. Gaudry and E. Thomé. The mpFq library and implementing curve-based key exchanges. In *SPEED: Software Performance Enhancement for Encryption and Decryption*, ECRYPT Workshop, 49–64, 2007.

[17] W. Geiselmann and D. Grollmann. VLSI design for exponentiation in $GF(2^m)$. *AUSCRYPT'90*, 398–405. Springer-Verlag, 2001.

[18] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. Guide to Elliptic Curve Cryptography. Springer-Verlag 2004.

[19] D. Hankerson, A. Menezes, and S. Vanstone. Software Implementation of Pairings Technical report available from http://www.cacr.math.uwaterloo.cahttp://citeseer.ist.psu.edu

[20] T. Itoh and S. Tsuji. Structure of Parallel Multipliers for a Class of Fields $GF(2^m)$. In *Information and Computers*, vol. 8, 21–40, 1989.

[21] P. Grabher, J. Groszschaedl and D. Page. On Software Parallel Implementation of Cryptographic Pairings. Accepted for SAC 2008. Available from http://eprint.iacr.org/2008/205.

[22] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specification Version 2.1 http://citeseer.ist.psu.edu/jonsson03publickey.html

[23] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. In *Soviet Physics*, Doklady 7, 595–596, 1963.

[24] A.K. Lenstra. Using cyclotomic polynomials to construct efficient discrete logarithm cryptosystems over finite fields. In *Proc ACISP'97*, Springer-Verlag LNCS 1270 (1997), 127-138.

[25] A.K. Lenstra and H.W. Lenstra. The Development of the Number Field Sieve. *LNM 1554*, Springer-Verlag, 1993.

[26] Alfred Menezes, Paul van Oorschot, Scott Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

[27] Alfred Menezes, Edlyn Teske, and Annegret Weng. Weak Fields for ECC. In *Topics in Cryptology - CT-RSA 2004*, 366–386, Springer-Verlag, 2004.

[28] P.L. Montgomery. Modular Multiplication without trial division. *Math. Comp.*, **44**, 519–521, 1985.

[29] Y. Nogami, A. Saito, and Y. Morikawa. Finite Extension Field with Modulus of All-One Polynomial and Representation of Its Elements for Fast Arithmetic Operations. In *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences* Vol.E86-A No.9, 2376–2387, 2003.

[30] E. Ozturk, B. Sunar, and E. Savas. Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In CHES 2004, Speinger Verlag, LNCS 3156, pp 92–106, 2004.

[31] R.L. Rivest, Shamir A., and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, **21**, 120 – 126, 1978.

[32] J.H. Silverman. Fast Multiplication in Finite Fields $GF(2^N)$. In *Proc. Workshop Cryptographic Hardware and Embedded Systems (CHES '99)*, LNCS 1717, 122–134. Springer 1999.

[33] J.A. Solinas. Generalized Mersenne Numbers. Technical report CORR-39, Dept. of C&O, University of Waterloo, 1999. Available from http://www.cacr.math.uwaterloo.cahttp://citeseer.ist.psu.edu/solinas99generalized.html

[34] C.D. Walter. Faster Modular Multiplication by Operand Scaling. *Advances in Cryptology* LNCS 576, 313–323, Springer Verlag, 1992.

[35] J.K. Wolf. Low Complexity Finite Field Multiplication. In *Discrete Math.*, no.s 106/107, 497–502, 1992.

[36] H. Wu, A. Hasan, I. Blake and S. Gao. Finite Field Multiplier Using Redundant Representation. *IEEE Trans. Comput.*, Vol 51, Num 11, Nov 2002.

Robert Granger Biography text here.

Andrew Moss Biography text here.

Nigel P. Smart Biography text here.