Storage-Efficient Finite Field Basis Conversion

Burton S. Kaliski Jr.[1] and Yiqun Lisa Yin[2]

[1] RSA Laboratories, 20 Crosby Drive, Bedford, MA 01730, burt@rsa.com
[2] RSA Laboratories, 2955 Campus Drive, San Mateo, CA 94402, yiqun@rsa.com

Abstract. The problem of finite field basis conversion is to convert from the representation of a field element in one basis to the representation of the element in another basis. This paper presents new algorithms for the problem that require much less storage than previous solutions. For the finite field $GF(2^m)$, for example, the storage requirement of the new algorithms is only $O(m)$ bits, compared to $O(m^2)$ for previous solutions. With the new algorithms, it is possible to extend an implementation in one basis to support other bases with little additional cost, thereby providing the desired interoperability in many cryptographic applications.

1 Introduction

Finite field arithmetic is becoming increasingly important in today's computer systems, particularly for cryptographic operations. Among the more common finite fields in cryptography are odd-characteristic finite fields of degree 1, conventionally known as $GF(p)$ arithmetic or arithmetic modulo a prime, and even-characteristic finite fields of degree greater than 1, conventionally known as $GF(2^m)$ arithmetic, where m is the degree. Arithmetic in $GF(2^m)$ (or any finite field of degree greater than 1) can be further classified according to the choice of basis for representing elements of the finite field; two common choices are a polynomial basis and a normal basis.

For a variety of reasons, including cost, performance, and compatibility with other applications, implementations of $GF(2^m)$ arithmetic vary in their choice of basis. The variation in choice affects interoperability, since field elements represented in one basis cannot be operated on directly in another basis. The problem of interoperability limits the applicability of implementations to cryptographic communication. As an example, if two parties wish to communicate with cryptographic operations and each implements finite field arithmetic in a different basis, then at least one party must do some conversions, typically before or after communicating a field element or at certain points in the cryptographic operations. Otherwise, the results of the cryptographic operations will be different.

It is well known that it is possible to convert between two choices of basis for a finite field; the general method involves a matrix multiplication. However, the matrix is often too large. For instance, the change-of-basis matrix for $GF(2^m)$ arithmetic will have m^2 entries, requiring several thousand bytes or more of storage in typical applications (e.g., $m \approx 160$). While such a matrix may be reasonable to store in a software implementation, it is likely to be a significant burden in a low-cost hardware implementation.

We describe in this paper new algorithms for basis conversion for normal and polynomial bases that require much less storage than previous solutions. Our algorithms are also very efficient in that they involve primarily finite-field operations, rather than, for instance, matrix multiplications. This has the advantage of benefiting from the optimizations that are presumably already available for finite-field operations.

With our algorithms, it is possible to extend an implementation in one basis so that it supports other choices of basis, with only a small additional cost in terms of circuitry, program size, or storage, relative to typical cryptographic applications of finite-field arithmetic, such as elliptic curve cryptosystems. Our work applies both to even-characteristic and odd-characteristic finite fields of degree greater than one, though even-characteristic arithmetic is the most likely application of our work, since it is more common in practice. We also suggest how to generalize our algorithms to other choices of basis than polynomial and normal bases.

2 Background

In this section, we introduce some basic notation and definitions. Let $GF(q)$ denote the finite field with q elements where $q = p^r$ for some prime p and integer $r \geq 1$. The *characteristic* of the field is the prime p. For even-characteristic fields, we have $p = 2$. Throughout the paper, we use $GF(q^m)$ to denote the finite field defined over the ground field $GF(q)$; the degree of $GF(q^m)$ over $GF(q)$ is m.

A *basis* for the finite field $GF(q^m)$ is a set of m elements $\omega_0, \ldots, \omega_{m-1} \in GF(q^m)$ such that every element of the finite field can be represented uniquely as a linear combination of basis elements. That is, given an element $\epsilon \in GF(q^m)$, we can write

$$\epsilon = \sum_{i=0}^{m-1} B[i]\omega_i$$

where $B[0], \ldots, B[m-1] \in GF(q)$ are the *coefficients*. The row vector $B = (B[0], \ldots, B[m-1])$ is called the *representation* of the element ϵ in the basis $\omega_0, \ldots, \omega_{m-1}$. Once the basis is chosen, rules for field operations (such as addition, multiplication, inversion) can be derived.

Elements of a finite field can be represented in a variety of ways, depending on the choice of basis for the representation. Two common choices of basis are a polynomial basis and a normal basis. In a polynomial basis, the basis elements are successive powers of an element γ (called the generator), that is, $\omega_i = \gamma^i$. The element γ must satisfy certain properties, namely that the powers $\gamma^0, \ldots, \gamma^{m-1}$ are linearly independent. In a normal basis, the basis elements are successive exponentiations of an element γ (again called the generator), that is, $\omega_i = \gamma^{q^i}$. In this case, the successive exponentiations must be linearly independent. Each basis has its own advantages and disadvantages in terms of implementation, and some discussions can be found in [4].

The *basis conversion* or *change-of-basis* problem is to compute the representation of an element of a finite field in one basis, given its representation in another basis. The general solution to the problem is to apply the change-of-basis matrix relating the two bases. Suppose that we are converting from the representation B of ϵ in the basis $\omega_0, \ldots, \omega_{m-1}$ to another basis. Let W_i be the representation of the element ω_i in the second basis, and let M, the change-of-basis matrix, be the $m \times m$ matrix whose columns are $W_0, \ldots, W_{m-1}$. It follows that the representation A of the element ϵ in the second basis can be computed as the matrix-vector product $A^T = MB^T$ where we view A and B as row vectors of dimension m. A change-of-basis matrix is invertible, and we can convert in the reverse direction by computing $B^T = M^{-1}A^T$.

The change-of-basis-matrix solution is straightforward and effective. But it is limited by two factors. First, the matrix M is potentially quite large, consisting of m^2 coefficients. Moreover, if we wish to convert in both directions, we must store the matrix M^{-1} as well, or else compute it, which could be time-consuming. Second, the operations involved in computing the matrix-vector product, while involving coefficients in the ground field, are not necessarily implementable with operations in either basis. Thus, the conversion process may not be as efficient as we would like.

Another approach to conversion is to multiply by elements of a dual basis (see Page 58 of [6]), but the storage requirement will again be quite large, if the entire dual basis is stored.

Our objective is to overcome the difficulties of the approaches just described. We wish to convert from one basis to another without involving a large amount of storage or requiring a large number of operations. And, we would like to take advantage of the built-in efficiency of finite field operations in one basis, rather than implementing new operations, such as matrix multiplications. We will call the basis in which finite field operations are primarily performed the *internal basis*. The other basis will be called the *external basis*. The conversion operation from the external basis to the internal basis will be called an *import* operation, and the reverse an *export* operation.

The specific problems to be solved are thus as follows.

- *Import problem.* Given an internal basis and an external basis for a finite field $GF(q^m)$ and the representation B of a field element in the external basis (the *external representation*), determine the corresponding representation A of the same field element in the internal basis (the *internal representation*) primarily with internal-basis operations, and with minimal storage.
- *Export problem.* Given an internal basis and an external basis for a finite field $GF(q^m)$ and the internal representation A of a field element, determine the corresponding external representation B of the same field element primarily with internal-basis operations, and with minimal storage.

The more general problem of converting from one basis to another with operations in a third basis is readily solved by importing to and re-exporting from the third basis; thus, our algorithms for converting to and from an internal basis will suffice for the more general problem.

3 Conversion Algorithms

In this section, we present four conversion algorithms for the import and export problem. We will focus our discussion on the case that both bases are defined over the same ground field $GF(q)$, and that the coefficients in the ground field are represented the same way in both bases. Section 4 addresses the case that the bases are defined over different ground fields, or that the coefficients are represented differently.

We require that the external basis is a polynomial basis or a normal basis, so that elements in the external basis have either the form

$$\epsilon = \sum_{i=0}^{m-1} B[i]\gamma^i \tag{1}$$

or the form

$$\epsilon = \sum_{i=0}^{m-1} B[i]\gamma^{q^i} \tag{2}$$

where γ is the generator of the external basis and $B[0], \ldots, B[m-1] \in GF(q)$ are the coefficients of the external representation. However, as discussed in Section 4, similar algorithms may be constructed for other choices of basis. In addition, we assume that the internal representation G of the generator γ is given, which is reasonable in most practical settings.[1]

We make no assumptions on the internal basis, other than that it is defined over the same ground field $GF(q)$ and that the representation of the coefficients in the ground field is the same. Our algorithms involve the same sequence of operations whether the internal basis is a polynomial basis, a normal basis, or some other type of basis. Thus, as examples, our algorithms can convert from a polynomial basis to a normal basis, from a normal basis to a polynomial basis, from a polynomial basis with one generator to a polynomial basis with another generator, or from a normal basis with one generator to a normal basis to another generator, to give a few possibilities.

We assume that addition, subtraction, and multiplication operations are readily available in the internal basis. A special case of multiplication, which can be more efficient, is scalar multiplication, where one of the operands is a coefficient, i.e., an element of the ground field.

In the following, we denote by I the internal representation of the identity element.

[1] If not, it can be computed given information about the internal and external bases, such as the minimal polynomial of the generator. There may be several acceptable internal representations of the generator, and hence several equivalent internal representations of a given element. For interoperability we need only that conversion into and out of the internal basis involve the same choice of G.

3.1 Basic Techniques for Conversion

Before presenting our conversion algorithms, we first describe some useful techniques which will serve as the building blocks of the conversion algorithms.

Algorithms for importing from an external basis can be constructed based on a direct computation of Equations (1) or (2). Since the internal representation G of the generator γ is given, we can easily convert each external basis element into its internal representation using only operations in the internal basis. In Sections 3.2 and 3.3, we give alternatives to these algorithms which are amenable to further performance improvements.

Algorithms for exporting to an external basis, however, cannot be constructed in the same direct manner. The major obstacle comes from the following fact: It is not obvious how to convert each internal basis element into its external representation using *only* operations in the internal basis, even given the external representation of the generator. So instead of converting the basis element, we will use some new techniques, and they are described in the following three lemmas.

Lemma 1. *Suppose the external basis is a polynomial basis with a generator γ. Let B be the external representation of an element ϵ, and let B' be the external representation of the element $\epsilon\gamma^{-1}$. Then for all indexes $0 < i < m-1$,*

$$B'[i] = B[i+1]$$

provided that $B[0] = 0$.

Lemma 3.1 shows that if the external basis is a polynomial basis, then multiplication by the inverse γ^{-1} of the generator γ shifts the coefficients down, provided that the coefficient at index 0 is initially 0. The result leads to an algorithm for exporting to a polynomial basis: compute the coefficient $B[0]$, subtract $B[0]$, multiply by G^{-1}, and repeat, computing successive coefficients of B.

Related to this, multiplication by the generator γ shifts coefficients up, provided that $B[m-1] = 0$. Rotation of the coefficients in either direction is also possible, though we will not need it for our algorithms.

Lemma 2. *Suppose the external basis is a normal basis. Let B be the external representation of an element ϵ, and let B' be the external representation of the element ϵ^q. Then for all indexes $0 \leq i \leq m-1$,*

$$B'[i] = B[(i-1) \bmod m].$$

Lemma 3.2 shows that if the external basis is a normal basis, then raising to the power q rotates the coefficients up. The result leads to an algorithm for exporting to a normal basis: compute the coefficient $B[m-1]$, raise to the power q, and repeat.

We still need a technique for obtaining the coefficient $B[0]$ or $B[m-1]$. From the fact that the coefficients of the internal and external representations are related by a change-of-basis matrix M $B^T = M^{-1}A^T$, we know that a coefficient $B[i]$ can be obtained by a linear combination

$$B[i] = \sum_{j=0}^{m-1} M^{-1}[i,j]A[j]$$

where the values $M^{-1}[i,j] \in GF(q)$ are elements of the matrix M^{-1}. We can thus obtain a coefficient $B[i]$ by operations over the ground field $GF(q)$. We may also compute the coefficient $B[i]$ with only internal-basis operations, as will be shown in the next lemma.

In preparation, we will need to consider the multiplication matrices for the internal basis. Let $\omega_0, \ldots, \omega_{m-1}$ be the internal basis. The multiplication matrix for the coefficient at index k, denoted K_k, is the $m \times m$ matrix whose $[i,j]$th element, $0 \le i,j < m$, is the coefficient at index k of the representation in the internal basis of the product $\omega_i \omega_j$. In other words, the matrices are defined so that for all i, j, $0 \le i,j < m$,

$$\omega_i \omega_j = \sum_{k=0}^{m-1} K_k[i,j]\omega_k.$$

The multiplication matrices is invertible. It follows from this definition that the coefficient at index 0 of the product of two internal representations R and S, which we may write as $(R \times S)[0]$, is equal to RK_0S^T.

Lemma 3. *Let $s_0, \ldots, s_{m-1}$ be elements of $GF(q)$, and let K_0 be the multiplication matrix for the coefficient at index 0 in the internal basis. Then for any internal representation A of an element,*

$$\sum_{i=0}^{m-1} s_j A[j] = (A \times V)[0],$$

where V is defined as $V^T = K_0^{-1}[s_0 \cdots s_{m-1}]^T$.

Proof. Since the multiplication matrix K_0 is invertible, the element V exists. By definition of multiplication, we have $(A \times V)[0] = AK_0V^T$. It follows directly that $(A \times V)[0]$ equals the desired linear function.

Lemma 3.3 shows that any linear combination of coefficients of the internal representation of an element may be computed with internal-basis operations. To generalize the result, we denote by V_i the value such that

$$B[i] = (A \times V_i)[0],$$

i.e., the one where the values $s_0, \ldots, s_{m-1}$ are the matrix row $M^{-1}[i,0], \ldots,$ $M^{-1}[i,m-1]$. Like the internal representation G of the generator of the external basis, a value V_i is particular to an external basis; a different set of values V_i would be needed for each external basis with which one might want to convert.

In what follows, we present four conversion algorithms, for importing and exporting with external polynomial and normal bases. For each algorithm, we measure the number of full multiplications and scalar multiplications involved and the amount of storage required for constants and intermediate results. As noted previously, there are direct import algorithms based on direct computation of Equations (1) and (2); we give different versions that allow further performance improvements.

As we will see, each basic conversion algorithm requires the storage of only one or two constants (vectors of length m). So for $GF(q^m)$ the total storage requirement is $O(m \log q)$ bits, compared to $O(m^2 \log q)$ bits for previous solutions.

3.2 Importing from a Polynomial Basis

Algorithm IMPORTPOLY converts from a polynomial-basis representation for $GF(q^m)$ over $GF(q)$ to an internal representation over the same ground field, primarily with internal-basis operations.

Input: $B[0], \ldots, B[m-1]$, the external representation to be converted
Output: A, the corresponding internal representation
Constants: G, the internal representation of the generator of the external basis

```
proc ImportPoly
$A \leftarrow 0$
for $i \leftarrow m-1$ down to $0$ do
        $A \leftarrow A \times G$
        $A \leftarrow A + B[i] \times I$
endfor
```

The algorithm processes one coefficient per iteration, scanning from highest index to lowest, accumulating powers of G up to G^i for each $B[i]$ term. It involves m full multiplications (and may also involve m scalar multiplications, depending on the form of I), and requires storage for one constant.

We can view the above algorithm as computing the internal representation A according to the following formula.

$$A = B[0] \times I + G \times (B[1] \times I + G \times (B[2] \times I + \cdots + G \times (B[m-1] \times I) \cdots)).$$

So IMPORTPOLY bears some similarity to the method of evaluating a polynomial $f(x)$ at a given value x^* using Horner's rule [2]. More specifically, $f(x^*)$ can be evaluated with $O(m)$ operations by rewriting $f(x)$ as follows.

$$f(x^*) = a_0 + x^*(a_1 + x^*(a_2 + \cdots + x^*(a_{m-1})) \cdots)).$$

There are some distinctions between the two methods. The inputs to basis conversion are the coefficients $B[0], \ldots, B[m-1]$, and the generator G is fixed. In contrast, the input to polynomial evaluation is the value x^*, and the coefficients $a_0, \ldots, a_{m-1}$ are fixed.

It is possible to reduce the number of iterations of the loop and thereby improve performance by processing more than one coefficient per iteration. One of the ways to do this is, in the case that m is even, to change the loop to

```
for $i \leftarrow m/2 - 1$ down to 0 do
    $A \leftarrow A \times G$
    $A \leftarrow A + B[i + m/2] \times G^{m/2} + B[i] \times I$
endfor
```

where $G^{m/2}$ is an additional constant. Although the number of scalar multiplications remains the same, the number of full multiplications can be reduced by up to a factor of k, if k coefficients are processed per iteration. In addition, all k coefficients can be processed in parallel.

Another improvement is to unroll the first iteration and start with $A \leftarrow B[m-1] \times I$.

3.3 Importing from a Normal Basis

Algorithm ImportNormal converts from a normal-basis representation for $GF(q^m)$ over $GF(q)$ to an internal representation over the same ground field, primarily with internal-basis operations.

Input: $B[0], \ldots, B[m-1]$, the external representation to be converted
Output: A, the corresponding internal representation
Constants: G, the internal representation of the generator of the external basis

```
proc ImportNormal
$A \leftarrow 0$
for $i \leftarrow m - 1$ down to 0 do
    $A \leftarrow A^q$
    $A \leftarrow A + B[i] \times G$
endfor
```

The algorithm processes one coefficient per iteration, scanning from highest index to lowest, accumulating powers of G up to G^{q^i} for each $B[i]$ term. (We make use of the fact that $(A + B[i] \times G)^q = A^q + B[i] \times G^q$). It involves m exponentiations to the power q and m scalar multiplications, and requires storage for one constant, in addition to the intermediate results for exponentiation. (The exponentiation will typically involve about $1.5 \log_2 q$ multiplications and require storage for one intermediate result, though better performance is possible if the internal basis is a normal basis. For $q = 2$, the exponentiation will be just a squaring.)

As with ImportPoly, it is possible to improve performance by unrolling the first iteration or by processing more than one coefficient per iteration.

3.4 Exporting to a Polynomial Basis

Algorithm ExportPoly converts from an internal representation for $GF(q^m)$ over $GF(q)$ to an external polynomial-basis representation over the same ground field, primarily with internal-basis operations.

Input: A, the internal representation to be converted
Output: $B[0], \ldots, B[m-1]$, the corresponding external representation
Constants:
G^{-1}, the internal representation of the inverse of the generator of the external basis
V_0, the value such that $(A \times V_0)[0] = B[0]$ (see Lemma 3.3)

```
proc ExportPoly
$A \leftarrow A \times V_0$
for $i \leftarrow 0$ to $m-1$ do
        $B[i] \leftarrow A[0]$
        $A \leftarrow A - B[i] \times V_0$
        $A \leftarrow A \times G^{-1}$
endfor
```

The algorithm computes one coefficient per iteration, applying the observations previously given, with the additional enhancement of premultiplying by the value V_0.[2] The algorithm involves $m+1$ full multiplications and m scalar multiplications, and requires storage for two constants. The input A is modified by the algorithm.

As with the previous algorithms, more than one coefficient can be processed per iteration. An additional constant such as $V_{m/2}/V_0$ is required, where $V_{m/2}$ is the value such that $(A \times V_{m/2})[0] = B[m/2]$; the coefficient $B[i+m/2]$ would be computed as

```
$T \leftarrow A \times V_{m/2}/V_0$
$B[i+m/2] \leftarrow T[0]$
```

The number of multiplications is not reduced in this case, due to the method for computing the coefficient $B[i+m/2]$. The improvement would be more significant if the coefficients were computed as a direct linear function of A, provided that such computation were efficient.

Another improvement is to unroll the last iteration and end with $B[m-1] = A[0]$.

[2] This is the reason that the correction step involves subtracting the value $B[i] \times V_0$ rather than $B[i] \times I$. The alternative to premultiplying A by V_0 is to multiply it by V_0 during each iteration before computing the coefficient $B[i]$; but this involves an additional multiplication per iteration.

3.5 Exporting to a Normal Basis

Algorithm ExportNormal converts from an internal representation for $GF(q^m)$ over $GF(q)$ to a normal-basis representation over the same ground field, primarily with internal-basis operations.

Input: A, the internal representation to be converted
Output: $B[0], \ldots, B[m-1]$, the corresponding external representation
Constants: V_{m-1}, the value such that if $(A \times V_{m-1})[0] = B[m-1]$ (see Lemma 3.3)

```
proc ExportNormal
for $i \leftarrow m-1$ down to $0$ do
        $T \leftarrow A \times V_{m-1}$
        $B[i] \leftarrow T[0]$
        $A \leftarrow A^q$
endfor
```

The algorithm computes one coefficient per iteration, applying the observations previously given. The algorithm involves m exponentiations to the power q and m full multiplications, and requires storage for one constant and one intermediate result, T, in addition to the intermediate results for exponentiation. The input A, though modified by the algorithm, returns to its initial value.

As with ExportPoly, it is possible to improve performance by unrolling the last iteration or by processing more than one coefficient per iteration.

4 Extensions

The algorithms presented so far all assume that the ground field is the same for the internal and the external basis and have the same representation.

If the ground fields are the same but have different representations, the individual coefficients can be converted through techniques similar to those for the entire representation.

If the ground fields have different representations, however, we can convert individual "subcoefficients" of each coefficient, where the subcoefficients are elements of $GF(p)$. In the import algorithms, we would add terms of the form $B[i][j] \times H_j$ (or $B[i][j] \times GH_j$) to A, where $B[i][j]$ is a subcoefficient and H_j is the internal representation of an element of the ground-field basis. In the export algorithms, we would multiply by values like $V_{0,j}$ where $(A \times V_{0,j}) = B[0][j]$. The storage requirements for these methods would depend on the degree of the ground field over $GF(p)$, and would be modest in many cases. More storage-efficient algorithms are possible, however, involving techniques different than those described here.

The algorithms presented here are examples of a more general class of conversion algorithms, where successive coefficients are processed as an internal representation is "shifted" or "rotated" in terms of its corresponding external-basis representation. For the algorithms here, such "external" shifting or rotation is accomplished by such operations as exponentiation, multiplication by G, or multiplication by G^{-1} (combined with subtraction of the $B[0]$ coefficient), depending on the algorithm. The import algorithms interleave shifting with insertion of coefficients into the internal representation, by addition of a the term $B[i] \times I$ or $B[i] \times G$; the export algorithms interleave shifting with extraction of the coefficients, by a computation of the form $(A \times V_0)[0]$.

Algorithms of the same class may be constructed for any choices of basis for which an efficient external shifting or rotation operation can be defined.

5 Applications

Many public-key cryptosystems are based on operations in large finite mathematical groups, and the security of these cryptosystems relies on the computational intractability of computing discrete logarithms in the underlying groups. The group operations usually consist of arithmetic in finite fields, in particular $GF(p)$ and $GF(2^m)$. In this section, we focus on the application of our conversion algorithms to elliptic curve cryptosystems over $GF(2^m)$ [5]. The general principles extend to other applications as well.

At a high level, an elliptic curve over $GF(2^m)$ is a set of points which form a group with respect to certain rules for adding two points. Such an addition consists of a series of field operations in $GF(2^m)$. In a generic implementation using projective coordinates, adding two distinct points needs 10 multiplications and 3 squarings, and doubling a point needs 5 multiplications and 5 squarings. [3]

Elliptic curve cryptosystems over $GF(2^m)$ that are of particular interest today are analogs of conventional discrete logarithm cryptosystems. Such elliptic curve schemes (e.g., EC Diffie-Hellman and EC DSA [1][3]) usually involve one or two *elliptic curve scalar multiplications* of the form $Q = kP$ where P and Q are points and k is an integer of length about m. Again in a generic implementation, such an operation can be done with m doublings of a point and $m/2$ additions of two distinct points, giving a total of $(3 \times m/2 + 5 \times m) = 6.5m$ field squarings and $(10 \times m/2 + 5 \times m) = 10m$ field multiplications.

To illustrate the cost of conversion, we consider a general situation in which two parties implement some elliptic curve scheme over $GF(2^m)$ with different choices of basis. In such a situation, each elliptic curve scalar multiplication in the scheme would require at most two conversions by one of the parties, one before and one after the operation.[4] The following table compares the cost of two conversions (back and forth) with the cost of one elliptic curve scalar multiplication.

[3] In general, the number of operations depends on the particular formulas and constraints on the parameters of the curve. The number given here is based on Annex A "Number-theoretic algorithms" in [3].

[4] For example, suppose Alice has a polynomial basis and Bob a normal basis, and they want to agree on a shared key using EC Diffie-Hellman. After obtaining Bob's public key P, Alice would import P from the normal basis to the polynomial basis, compute $Q = kP$ in the polynomial basis, and export Q from the polynomial basis back to the normal basis. Alternatively, Bob could perform the conversions.

operation	multiplications	squarings
ImportPoly + ExportPoly	$2m+1$	0
ImportNormal + ExportNormal	m	$2m$
EC scalar multiplication	$10m$	$6.5m$

Based on the above table, we can estimate the extra cost of conversion compared with one elliptic curve scalar multiplication. When the external basis is a polynomial basis, the extra cost (ImportPoly + ExportPoly) is around $2/(10 + 6.5 \times 0.5) = 15\%$ for an internal polynomial basis (assuming squaring is twice as fast as multiplication for an internal polynomial basis) and about $2/10 = 20\%$ for an internal normal basis (since squarings are essentially free in an internal normal basis). Similarly, when the external basis is a normal basis, the extra cost (ImportNormal + ExportNormal)is about $(1 + 2 \times 0.5)/(10 + 6.5 \times 0.5) = 15\%$ for an internal polynomial basis and about $1/10 = 10\%$ for an internal normal basis. To summarize, the extra cost of conversion is between 10% and 20%, depending on the implementation.

Overall, we conclude that our conversion algorithms incur only a small extra cost in an elliptic curve cryptosystem, and that the memory requirement is quite reasonable: only a few additional constants or intermediate values need to be stored. The cost can be reduced still further by additional optimizations such as processing more than one coefficient at a time, with the only additional requirement being the storage of a small number of additional elements.

6 Conclusions

We have described in this paper several new algorithms for basis conversion that require much less storage than previous solutions. Our algorithms are also very efficient in that they involve primarily finite-field operations. This has the advantage of benefiting from the optimizations that are presumably already available for finite-field operations.

As examples, our algorithms can convert from a polynomial basis to a normal basis and from a normal basis to a polynomial basis. Related algorithms can convert to and from other choices of basis for which an efficient "external shifting" operation can be defined. Our algorithms are particularly applicable to even-characteristic finite fields, which are typical in cryptography.

The variation in choice of basis for representing finite fields has affected interoperability, especially of cryptosystems. With our algorithms, it is possible to extend an implementation in one basis so that it supports other choices of basis at only a small additional cost, thereby providing the desired interoperability and extending the set of parties that can communicate with cryptographic operations.

Acknowledgment

We would like to thank the IEEE P1363 working group for motivating this problem, Leonid Reyzin for helpful discussions, and the anonymous referees for their comments. We would also like to thank Rob Lambert for bringing Horner's Rule to our attention.

References

1. ANSI X9.62: *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, draft, November 1997.
2. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.
3. IEEE P1363: *Standard Specifications for Public-Key Cryptography*, Draft version 3, May 1998. `http://stdsbbs.ieee.org/groups/1363/draft.html.`
4. A. Menezes, I. Blake, X. Gao, R. Mullin, S. Vanstone, and T. Yaghoobian. *Applications of Finite Fields.* Kluwer Academic Publishers, 1993.
5. A. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.
6. R. Lidl and H. Niederreiter. *Finite Fields,* volume 20 of *Encyclopedia of Mathematics and Its Applications.* Addison-Wesley, 1983.