Efficient Finite Field Basis Conversion Involving Dual Bases

Burton S. Kaliski Jr. and Moses Liskov

RSA Laboratories
20 Crosby Drive, Bedford, MA 01730
{ burt, moses }@rsa.com

Abstract. Conversion of finite field elements from one basis representation to another representation in a storage-efficient manner is crucial if these techniques are to be carried out in hardware for cryptographic applications. We present algorithms for conversion to and from dual of polynomial and dual of normal bases, as well as algorithms to convert to a polynomial or normal basis which involve the dual of the basis. This builds on work by Kaliski and Yin presented at SAC '98.

1 Introduction

Conversion between different choices of basis for a finite field is an important problem in today's computer systems, particularly for cryptographic operations [1]. While it is possible to convert between two choices of basis by matrix multiplication, the matrix may be too large for some applications, hence the motivation for more storage-efficient techniques. The most likely such application would be in special-purpose hardware devices, but there are others as well.

The paper of Kaliski and Yin [2] introduced the shift-extract and technique of basis conversion, and also gave several storage-efficient algorithms based on those techniques for converting to a polynomial or normal basis. In this paper, we introduce techniques involving the dual of a polynomial or normal basis, including storage-efficient generation of a dual basis and storage-efficient shifting in such a basis. The new techniques result in several new storage-efficient algorithms for converting to and from the dual of a polynomial or normal basis, as well as additional algorithms for converting to a polynomial or normal basis.

2 Background

Elements of a finite field can be represented in a variety of ways, depending on the choice of basis for the representation [3]. Let $GF(q^m)$ be the finite field, and let $GF(q)$ be the ground field over which it is defined, where q is a prime or a prime power. We say that the *characteristic* of the field is p where $q = p^r$ for some $r \geq 1$. For even-characteristic fields, we have $p = 2$. The *degree* of the field is m; its *order* is q^m.

A *basis* for the finite field 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. We write

$$\epsilon = B[0]\omega_0 + B[1]\omega_1 + \cdots + B[m-1]\omega_{m-1}$$

where $B[0], \ldots, B[m-1] \in GF(q)$ are the *coefficients*.

Two common types 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*:

$$\omega_i = \gamma^i.$$

In a normal basis, the basis elements are successive exponentiations of an element γ, again called the generator:

$$\omega_i = \gamma^{q^i}.$$

Another common type of basis is a *dual basis*. Let $\omega_0, \ldots, \omega_{m-1}$ be a basis and let h be a nonzero linear function from $GF(q^m)$ to $GF(q)$, i.e., a function such that for all $\epsilon, \phi \in GF(q^m)$ and $c \in GF(q)$, $h(\epsilon + \phi) = h(\epsilon) + h(\phi)$ and $h(c\epsilon) = ch(\epsilon)$. The dual basis of the basis $\omega_0, \ldots, \omega_{m-1}$ with respect to the function h is the basis $\eta_0, \ldots, \eta_{m-1}$ such that for $0 \le i, j \le m-1$,

$$h(\omega_i \eta_j) = 1 \text{ if } i = j, \text{ 0 otherwise.}$$

Duality is symmetric: the dual basis with respect to h of the basis $\eta_0, \ldots, \eta_{m-1}$ is the basis $\omega_0, \ldots, \omega_{m-1}$. A dual basis can be defined for a polynomial basis, a normal basis, or any other choice of basis, and with respect to a variety of functions.

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 problem has two forms, where we distinguish between the internal basis in which finite field operations are performed, and the external basis to and from which we are converting:

- *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*).
- *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.

Normally, the import and export problem could be solved by using a change of basis matrix, which requires storage for $O(m)$ field elements. Since each field element consists of m base field coefficients, this is $O(m^2)$ coefficients. In constrained environments, this may be too large. What we want are algorithms which require storage for $O(1)$ field elements or $O(m)$ coefficients. The algorithms given in this paper for dual bases satisfy this requirement.

3 Overview of techniques

In the following, the dual of a polynomial basis is called a polynomial* basis, and the dual of a normal basis is called a normal* basis.

3.1 Import algorithms

Given an internal basis and an external basis for a finite field and the representation B of a field element in the external basis, an import algorithm determines the corresponding representation A of the same field element in the internal basis.

Two general methods for determining the internal representation A are described: the generate-accumulate method and the shift-insert method.

Generate-Accumulate method The generate-accumulate method computes the internal representation A by accumulating the products of coefficients $B[i]$ with successive elements of the external basis. The basic form of the algorithm for this method is as follows:

```
proc ImportByGenAccum
    $A \leftarrow 0$
    for $i$ from $0$ to $m-1$ do
        $A \leftarrow A + B[i] \times W_i$
    endfor
endproc
```

As written, this algorithm requires storage for the m values $W_0, \ldots, W_{m-1}$, which are the internal representations of the elements of the external basis. To reduce the storage requirement, it is necessary to generate the values as part of the algorithm. This is straightforward when the external basis is a polynomial basis or a normal basis. For polynomial* and normal* bases, algorithms are given in this paper.

Shift-Insert method The shift-insert method computes the internal representation A by "shifting" an intermediate variable in the external basis and inserting successive coefficients between the shifts. This follows the same concept as the shift-extract method below. Let Shift be a function that shifts an element in the external basis, i.e., a function such as one which given the internal representation of an element with external representation

$$(B[0], B[1], \ldots, B[m-2], B[m-1])$$

computes the internal representation of the element with external representation

$$(B[m-1], B[0], \ldots, B[m-3], B[m-2]).$$

(Other forms of shifting are possible, including shifting in the reverse direction, or shifting where the value 0 rather than $B[m-1]$ is shifted in.)

The basic form of algorithm for this method is as follows ([2], Sec. 3.2, 3.3):

```
proc ImportByShiftInsert
    $A \leftarrow 0$
    for $i$ from $m-1$ downto $0$ do
        Shift($A$)
        $A \leftarrow A + B[i] \times W_0$
    endfor
endproc
```

The direction of the **for** loop may vary depending on the direction of the shift. One advantage of the shift-insert method over the generate-accumulate method is that with a minor increase in storage, this algorithm can be parallelized. That is, if W_0 and $W_{m/2}$ are available, two elements can be inserted per shift. Since the shift is the most work-intensive part of the algorithm, this aids efficiency. This improvement is further discussed in [2].

3.2 Export algorithms

Given an internal basis and an external basis for a finite field and the representation A of a field element in the internal basis, an export algorithm determines the corresponding representation B of the same field element in the internal basis.

Two general methods for determining the external representation B are described: the generate*-evaluate method and the shift-extract method.

Generate*-Evaluate method The generate*-evaluate method computes the external representation B by evaluating products of A with successive elements of a dual of the external basis. For example, the following equation gives the ith coefficient of the external representation:

$$B[i] = h(AX_i)$$

where h is a linear function and $X_0, \ldots, X_{m-1}$ are the internal-basis representations of the elements of the dual of the external basis with respect to the function h. The basic form of algorithm for this method is as follows:

```
proc ExportByGen*Eval
    for $i$ from $0$ to $m-1$ do
        $T \leftarrow A \times X_i$
        $B[i] \leftarrow h(T)$
    endfor
endproc
```

This algorithm requires storage for the m values $X_0, \ldots, X_{m-1}$, which are the internal represenations of the dual of the external basis. As was the case for ImportByGenAccum, to reduce the storage requirement, it is necessary to generate the values as part of the algorithm.

Shift-Extract method The Shift-Extract method computes the external representation A by shifting an intermediate variable in the external basis and *extracting* successive coefficients between the shifts. This follows the same concept as the shift-insert method above, with a similar Shift function and an Extract function that obtains a selected coefficient of the external representation. (The Extract function is similar to the h function in the previous method.)

The basic form of algorithm for this method is as follows ([2], Sec. 3.4, 3.5):

```
proc ExportByShiftExtract
    for $i$ from $m-1$ downto 0 do
        $B[i] \leftarrow$ Extract($A$)
        Shift($A$)
    endfor
endproc
```

Again, the direction of the **for** loop may vary depending on the direction of the shift. As with the shift-insert method above, the shift-extract method can be parallelized to extract multiple coefficients per iteration.

3.3 Summary

For these methods to accomplish our goal of being storage efficient, we depend on the efficiency of some additional functions. For the generate*-evaluate and generate-accumulate methods, we need an efficient dual basis generator. For the shift-insert and shift-extract methods, we need an efficient Shift function that works when the external basis is a normal* or polynomial* basis. An efficient *Extract* function (and hence an efficient method of evaluating a linear function h) is given in [2] (cf. Lemma 3).

4 Polynomial* basis techniques

This section discusses the structure of a polynomial* basis, and presents an efficient basis generation function and an efficient external shift function.

Theorem 1. *Let $1, \gamma, \ldots, \gamma^{m-1}$ be a polynomial basis for $GF(q^m)$, and let $h(\epsilon)$ be a linear function from $GF(q^m)$ to $GF(q)$. Let $h_0(\epsilon)$ be the 1-coefficient of the representation of the element ϵ in the polynomial basis. Let ζ be the element of $GF(q^m)$ such that $h_0(\zeta\epsilon) = h(\epsilon)$. A formula for the dual basis $\eta_0, \ldots, \eta_{m-1}$ of this polynomial basis with respect to h is*

$$\eta_i = \zeta^{-1}\xi_i,$$

where $\xi_0 = 1$ and $\xi_i = \gamma^{-1}\xi_{i-1} - h_0(\gamma^{-1}\xi_{i-1})$.

Proof. We first observe that the value ζ exists since there is a one-to-one correspondence between linear functions and field elements (cf. Lemma 3 of [2]). To prove the correctness of the formula, we use the definition of the dual basis and induction. First, we consider η_0 and observe that $h(\gamma^i \eta_0) = h_0(\gamma^i)$, which is 1 if $i = 0$ and 0 if $1 \leq i \leq m-1$, meeting the definition. Now suppose we know that for $j > 0$ the first $j-1$ elements are correct elements of the dual basis. Then we get the following for the jth element:

$$\begin{aligned} h(\gamma^i \eta_j) &= h_0(\gamma^i \xi_j) \\ &= h_0(\gamma^{i-1}\xi_{j-1}) - h_0(\gamma^i h_0(\gamma^{-1}\xi_{j-1})). \end{aligned}$$

For $i = 0$, this reduces to $h_0(\gamma^{-1}\xi_{j-1}) - h_0(\gamma^{-1}\xi_{j-1}) = 0$. For $1 \leq i \leq m-1$, the equation becomes $h_0(\gamma^{i-1}\xi_{j-1})$, which by induction is 1 if $i = j$ and 0 if $i \neq j$. In both cases the definition is met.

In the following algorithms, Z will be the internal representation of ζ, G will be the internal representation of γ, and I will be the internal representation of the identity element. The value V_0 corresponds to the element such that $(A \times V_0)[0] = h_0(A)$. The value Z corresponds to the function $h(\epsilon) = h_0(\zeta\epsilon)$, and contains the information specific to the choice of dual basis in the following algorithms. Note that if Z is 0, $h(\epsilon) = h_0(0) = 0$, and therefore, h would not be a nonzero linear function. Thus, we can assume Z is nonzero.

4.1 GenPoly*

The algorithm GenPoly* generates the internal representation of the dual basis elements. GenPoly* is an *iterator*; it is meant to be called many times in succession. The first time an iterator is called, it starts from the **iter** line. When a **yield** statement is reached, the iterator returns the value specified by the yield. The next time the iterator is called, it starts immediately after the last **yield** executed; all temporary variables are assumed to retain their values from one call to the next. An iterator ends when the **enditer** line is reached.

```
iter GenPoly*
    $W \leftarrow I$
    yield $Z^{-1}$
    for $i$ from $1$ to $m-1$ do
        $W \leftarrow W \times G^{-1}$
        $T \leftarrow W \times V_0$
        $W \leftarrow W - T[0] \times I$
        yield $W \times Z^{-1}$
    endfor
enditer
```

4.2 ShiftPoly*

With our knowledge of the formula for a polynomial* basis, we can also devise a method for shifting an element's representation in the polynomial* basis. The algorithm simply uses the recursive formula for generating ξ_i from ξ_{i-1}, namely

$$s(\epsilon) = \gamma^{-1}\epsilon - h_0(\gamma^{-1}\epsilon).$$

Theorem 2. *s performs an external shift in the polynomial* basis with respect to h_0, such that $s(\xi_i) = \xi_{i+1}$ for $0 \leq i \leq m-2$ and $s(\xi_{m-1}) = 0$.*

Proof. First we observe that s is linear, i.e., that for all $\epsilon, \phi \in GF(q^m)$, $c \in GF(q)$, $s(\epsilon+\phi) = s(\epsilon)+s(\phi)$ and $s(c\epsilon) = cs(\epsilon)$. Since s is linear, we only have to show that applying it to basis elements is correct. Since s is merely the recursive formula for generating ξ_i from ξ_{i-1}, we know it is correct for all basis elements except ξ_{m-1}. Thus, it remains to show that $s(\xi_{m-1}) = 0$. To see this, define ξ_m as $s(\xi_{m-1})$ and apply the equation from the proof above:

$$h_0(\gamma^i \xi_m) = h_0(\gamma^{i-1}\xi_{m-1}) - h_0(\gamma^i h_0(\gamma^{-1}\xi_{m-1})).$$

For $i = 0$, this cancels to 0. For $1 \leq i \leq m-1$, the equation becomes $h_0(\gamma^{i-1}\xi_{m-1})$, which is 0. Since the values $h_0(\gamma^i \xi_m)$ as i varies correspond to coefficients of the representation of ξ_m in the basis $\xi_0, \ldots, \xi_{m-1}$ and they are all zero, it follows that $\xi_m = 0$.

Since the dual basis $\eta_0, \ldots, \eta_{m-1}$ is just the basis $\xi_0, \ldots, \xi_{m-1}$ scaled by ζ^{-1}, shifting in the dual basis $\eta_0, \ldots, \eta_{m-1}$ is accomplished by computing the function $\zeta^{-1}s(\zeta\epsilon)$. The following is an algorithm for shifting in the polynomial* basis based on this technique.

```
proc ShiftPoly* ($A$)
    $A \leftarrow A \times ZG^{-1}$
    $T \leftarrow A \times V_0$
    $A \leftarrow A - T[0] \times I$
    $A \leftarrow A \times Z^{-1}$
endproc
```

Also note that we can use ShiftPoly* to make a new version of GenPoly* that generates by repeated shifting.

5 Techniques involving the dual of a normal basis

This section discusses the structure of a normal* basis, and presents an efficient basis generation function and an efficient external shift function.

Theorem 3. *Let $\gamma, \ldots, \gamma^{q^{m-1}}$ be a normal basis for $GF(q^m)$, and let $h(\epsilon)$ be a linear function from $GF(q^m)$ to $GF(q)$. Let $h_0(\epsilon)$ be the γ-coefficient of the representation of the element ϵ in the normal basis. Let ζ be the element of $GF(q^m)$ such that $h_0(\zeta\epsilon) = h(\epsilon)$. A formula for the dual basis $\eta_0, \ldots, \eta_{m-1}$ of this normal basis with respect to h is*

$$\eta_i = \zeta^{-1}\xi_i,$$

where $\xi_0 = 1$ and $\xi_i = \sigma\xi_{i-1}^q$, and where σ is the element such that $h_0(\gamma^{q^i}\sigma)$ is 1 for $i = 1$ and 0 for $i = 0$ and $2 \leq i \leq m-1$.

Proof. First, we observe that ζ and σ exist, the latter being an element of the dual basis with respect to h_0. We also observe that $h_0(\sigma\epsilon^q) = h_0(\epsilon)$ for all $\epsilon \in GF(q)$. To prove that the formula is correct, we use the definition of the dual basis and induction. First, we consider η_0 and observe that $h(\gamma^{q^i}\eta_0) = h_0(\gamma^{q^i})$, which is 1 if $i = 0$ and 0 if $1 \le i \le m-1$, meeting the definition. For the induction step, we get the following for the jth element, where $j > 0$:

$$\begin{aligned} h(\gamma^{q^i}\eta_j) &= h_0(\gamma^{q^i}\xi_j) \\ &= h_0(\gamma^{q^i}\sigma\xi_{j-1}^q). \end{aligned}$$

For $i = 0$, the equation becomes $h_0(\gamma^{q^{m-1}}\xi_{j-1})$, which by induction is 0. (Note that $\gamma = \gamma^{q^m}$.) For $1 \le i \le m-1$, it becomes $h_0(\gamma^{q^{i-1}}\xi_{j-1})$, which by induction is 1 if $i = j$ and 0 if $i \ne j$. In both cases the definition is met.

In the following algorithms, G will be the internal representation of γ, S will be the internal representation of σ, and Z will be the internal representation of ζ. As before, we can assume Z is nonzero.

5.1 GenNormal*

Now that we know the general formula for the dual of a normal basis, we can demonstrate a technique for efficiently generating the dual of a normal basis. Like GenPoly*, GenNormal* is written as an iterator.

```
iter GenNormal*
    $T \leftarrow Z^{-1}$
    $W \leftarrow S$
    yield $T$
    for $i$ from 1 to $m-1$ do
        $T \leftarrow T \times W$
        $W \leftarrow W^q$
        yield $T$
    endfor
enditer
```

Theorem 4. *The iterator* GenNormal* *generates the elements of the normal* basis.*

Proof. After the first iteration, GenNormal* outputs the internal representation of ζ^{-1}. At each successive step, the basis is multiplied by successively higher powers of q of σ, so we get $\zeta^{-1}, \zeta^{-1}\sigma, \zeta^{-1}\sigma^{q+1}, \zeta^{-1}\sigma^{q^2+q+1}$, and so on. By our formula, this is the correct list of normal* basis elements.

5.2 ShiftNormal*

There is also an efficient method for doing a rotation of an element in the normal* basis. The algorithm simply uses the recursive formula for generating ξ_i from ξ_{i-1}, namely

$$s(\epsilon) = \sigma\epsilon^q.$$

Theorem 5. *s performs an external shift (actually, a rotation) in the normal* basis with respect to h_0, such that $s(\xi_i) = \xi_{i+1}$ for $0 \le i \le m-2$ and $s(\xi_{m-1}) = \xi_0$.*

Proof. As before, we first observe that s is linear. We again only have to show that that the formula is correct for ξ_{m-1}. To see this, define ξ_m as $s(\xi_{m-1})$ and apply the equation from the proof above:

$$h_0(\gamma^{q^i}\xi_m) = h_0(\gamma^{q^i}\sigma\xi_{m-1}^q).$$

For $i = 0$, the equation becomes $h_0(\gamma^{q^{m-1}}\xi_{m-1})$, which is 1. For $1 \le i \le m-1$, it becomes $h_0(\gamma^{q^{i-1}}\xi_{m-1})$, which is 0. Since the values $h_0(\gamma^{q^i}\xi_m)$ as i varies correspond to coefficients of the representation of ξ_m in the basis $\xi_0, \ldots, \xi_{m-1}$ and they are all zero except for the ξ_0-coefficient, which is 1, it follows that $\xi_m = \xi_0$.

Shifting in the dual basis $\eta_0, \ldots, \eta_{m-1}$ is accomplished by computing the function $\zeta^{-1}s(\zeta\epsilon)$, as before. Based on this, we have the algorithm ShiftNormal*.

```
proc ShiftNormal* ($A$)
        $A \leftarrow A^q$
        $A \leftarrow A \times SZ^{q-1}$
endproc
```

Note that this only requires storage for one value, as SZ^{q-1} can be precomputed. Also note that we can use ShiftNormal* to make a new version of GenNormal*.

6 Conclusion

We have demonstrated efficient algorithms for external shifting and efficient basis generation in the polynomial* and normal* bases. Using these algorithms in the storage-efficient basis conversion methods described above, we can implement the following basis conversion methods: ImportByShiftInsert, ImportByGenAccum, and ExportByShiftExtract for a polynomial* or normal* external basis, and ExportByGen*Eval for a polynomial or normal basis.

References

1. *IEEE P1363: Standard Specifications for Public-Key Cryptography*, draft 11, July 1999. `http://grouper.ieee.org/groups/1363/draft.html`.
2. B.S. Kaliski Jr. and Y.L. Yin. Storage-efficient finite field basis conversion. In S. Tavares and H. Meijer, editors, *Selected Areas in Cryptography '98 Proceedings,* volume 1556 of *Lecture Notes in Computer Science,* pages 81–93. Springer, 1999.
3. R. Lidl and H. Niederreiter. *Finite Fields,* volume 20 of *Encyclopedia of Mathematics and Its Applications.* Addison-Wesley, 1983.

This article was processed using the LaTeX macro package with LLNCS style