# Efficient Finite Field Basis Conversion Techniques (Submission to IEEE P1363a)

Burt Kaliski, Moses Liskov[1] and Yiqun Lisa Yin[2]
RSA Laboratories

**Abstract.** This summary of finite field basis conversion techniques is proposed for inclusion in IEEE P1363 Annex A. Included are some conventional basis conversion techniques, as well as some new storage-efficient basis conversion techniques.

## 1. Introduction

Elements of a finite field can be represented in a variety of ways, depending on the choice of basis for the representation. 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 prime power. The characteristic of the field is the prime $p$ for which $q = p^r$ for some $r \geq 1$. The degree of the field is $m$; its order is $q^m$. A basis for the finite field is a set of $m$ elements in GF($q^m$) $B = \{\omega_0, \ldots ,\omega_{m-1}\}$, such that every element of the finite field can be represented uniquely as a linear combination of basis elements:
where $v[0], \ldots, v[m-1]$ are the coefficients.
The choice of basis is highly relevant to the efficiency of certain functions. However, which basis is chosen is also highly dependent on the individual application. Therefore, converting an element from one

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

representation to another is useful. The bases we will deal with in this document are the normal basis, the polynomial basis, and the dual basis of each of these. However, the techniques described in this document do extend to other bases in some cases.
Annex A.7 of IEEE P1363 [1] concerns this problem. However, the only method shown is the change-of-basis matrix method. The algorithms we present have certain advantages over the matrix method. See [2] for related results.

## 2. Definitions

First of all, we'll need to define the particular bases this document concerns. For more information, see sections 1-8 of Annex A.

[1] RSA Laboratories, 20 Crosby Drive, Bedford, MA 01730 USA. E-mail: burt@rsa.com, mliskov@securitydynamics.com.

[2] RSA Laboratories, 2955 Campus Drive, Suite 400, San Mateo, CA 94403-2507 USA. E-mail: yiqun@rsa.com.

## 2.1 Basis Definitions

Polynomial and normal basis definitions are given in IEEE P1363, Annex A, A.3.4 and A.3.5, respectively. We include them here for ease of reference.
In a polynomial basis, the basis elements are successive powers of an element $t$, called the generator:

$$\omega_i = t^i.$$

A polynomial $f$ of degree $m$ relates the powers of $t$ as follows:

$$f(t) = t^m + f_{m-1}t^{m-1} + ... + f_1t + f_0 = 0$$

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

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

Another common type of basis is a dual basis. Let $\omega_0, ... ,\omega_{m-1}$ be a basis and let $h$ be a nonzero linear function from GF($q^m$) to GF($q$). The dual basis of the basis $\omega_0, ... ,\omega_{m-1}$ with respect to $h$ is the basis $\xi_0, ... ,\xi_{m-1}$, such that for all $i$, $j$,

$$h(\omega_i\xi_j) = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

The dual basis is uniquely defined, and duality is symmetric: the dual basis with respect to $h$ of the basis $\xi_0, ... ,\xi_{m-1}$ is the basis $\omega_0, ... ,\omega_{m-1}$. There is a natural simple choice for $h$, $h_i(v)$, defined to be equal to the $i$th coefficient of $v$, or $v[i]$. The function $h_0$, since it is easy to compute, is used in the functions we present involving the dual basis. If the basis desired is the dual basis with respect to $h$, there is a scaling factor involved; ζ is defined so that $h_0(x\zeta) = h(x)$.

## 2.2 Problem Definitions

The basis conversion problem has two distinct instances: the import problem and the export problem. If two communicating users do not agree on the basis, there must be a conversion performed by one or both of them. Each user has a local representation (the "internal basis," or $B_0$) and each user has information about the other user's representation (the "external basis," or $B_1$).
**Import Problem**
Given an internal basis and an external basis for a finite field GF($q^m$) and the representation $v$ of a field element in the external basis (the "external representation"), determine the corresponding representation $u$ 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 $u$ of a field element, determine the corresponding external representation $v$ of the same field element.

## 3. General Solutions

Let $B_0 = \{\alpha_0, ... ,\alpha_{m-1}\}$ be the internal basis, and $B_1 = \{\omega_0, ... ,\omega_{m-1}\}$ be the external basis. One simple way to convert is to use the matrix Γ, an $m \times m$ matrix whose columns are the internal representations of the external basis. With this matrix, conversion is simply a process of matrix multiplication. Suppose $u$ is the internal representation of an element, and $v$ is its external representation.
**Import**: $u = (\Gamma \times v)$
**Export**: $v = (\Gamma^{-1} \times u)$
The matrix Γ can be easily computed, as described in A.7.3. The running time of these basis conversion algorithms is $O(m^2)$ coefficient operations and the storage requirement is $O(m^2)$ coefficients.

# 4. Storage-Efficient Methods

## 4.1 Import Methods

There are two general import methods presented here:
**Generate-Accumulate**: Generate internal basis representations of external basis elements, and accumulate the internal basis representation. This requires an efficient external basis generator, *Gen*, which takes an index *i* as input, and outputs the internal representation of the external basis element $\omega_i$. (*Gen* can also be represented as an iterator, which takes no input; it merely ouputs the first basis element the first time it's called, the second element the next time, and so on. We define the *Gen* operation as an iterator in this document.)

```
1. $u \leftarrow 0$
2. for $i \leftarrow 0$ to $m$-1
3.         $u \leftarrow u + v[i] \times Gen(i)$
4. endfor
```

**Shift-Insert**: Insert external representation terms and perform an *external* shift on the internal representation. This requires an efficient external shift function, *Shift*, which takes the internal representation of $\Sigma\, a_i\omega_i$ and outputs the internal representation $\Sigma\, a_{i-1}\omega_i$ (for $i = 1$ to $m$-1). It also requires $W_0$, the internal representation of $\omega_0$. An external rotate function, *Rotate*, can also be used.

```
1. $u \leftarrow v[m-1] \times W_0$
2. for $i \leftarrow m-2$ downto 0
3.         $u \leftarrow Shift(u)$
4.         $u \leftarrow u + v[i] \times W_0$
5. endfor
```

A shift function (*ShiftDown*) which shifts the opposite direction can also be used, but this requires $W_{m-1}$, the internal representation of $\omega_{m-1}$.

```
1. $u \leftarrow v[0] \times W_{m-1}$
2. for $i \leftarrow 1$ to $m$-1
3.         $u \leftarrow ShiftDown(u)$
4.         $u \leftarrow u + v[i] \times W_{m-1}$
5. endfor
```

## 4.2 Export Methods

There are two general export methods presented here.
**Generate*-Accumulate**: Select a linear function *h* that is easy to compute, and generate the dual of the external basis with respect to *h*. Then, multiply the internal representation by each element in turn, and use *h* to evaluate external coefficients. This requires an efficient dual-of-external basis generator, *GenDual*, which takes an index *i* as input, and outputs $X_i$, the internal representation of $\xi_i$. As with *Gen*, *GenDual* can be an iterator rather than a function.

```
1. for $i \leftarrow 0$ to $m$-1
2.         $A \leftarrow u \times GenDual(i)$
3.         $v[i] \leftarrow h(A)$
4. endfor
```

**Shift-Extract**: Extract a coefficient from *u*, then externally shift *u* and repeat. This requires both an efficient shift function, *Shift*, and an efficient extraction function, *Extract*, which takes an internal representation of an element as input, and outputs the external coefficient of $\omega_0$. Extraction can be done easily; there is an element $H_0$ for every external basis, such that $Extract(u) = (u \times H_0)[0]$.

```
1. for $i \leftarrow 0$ to $m$-2
```

```
2.      $v[i] \leftarrow Extract(u)$
3.      $u \leftarrow Shift(u)$
4.  endfor
5.  $v[m-1] \leftarrow Extract(u)$
```

## 5. Required Functions

### 5.1 Polynomial External Basis

If the external basis is a polynomial basis, it is assumed that we have the internal representation $T$ of $t$, and the internal representation $I$ of the identity.

**iter** *GenPoly*:

```
1.  $A \leftarrow I$
2.  yield $A$
3.  for $i \leftarrow 1$ to $m-1$
4.      $A \leftarrow A \times T$
5.      yield $A$
6.  endfor
```

The simplest external shift available in this case is a downward shift. In order to perform this shift, we have to be able to extract the term $v[0]$ from $u$. This can be done with a simple multiplication. We also need $T^{-1}$ instead of $T$.

**func** *ShiftDownPoly*($u$)

```
1.  $A \leftarrow v[0] \times I$
2.  $u \leftarrow u - A$
3.  $u \leftarrow u \times T^{-1}$
4.  return $A$
```

In order to calculate the dual of the external polynomial basis, we need $T^{-1}$, $I$, and $Z^{-1}$, where $Z$ is the internal representation of $\zeta$.

**iter** *GenDualPoly*:

```
1.  $A \leftarrow I$
2.  yield $Z^{-1}$
3.  for $i \leftarrow 1$ to $m-1$
4.      $A \leftarrow A \times T^{-1}$
5.      $A \leftarrow A - h_0(A)$
6.      yield $A \times Z$
7.  endfor
```

### 5.2 Normal External Basis

If the external basis is a normal basis, it is assumed that we have the internal representation $T$ of $\theta$, and the internal representation $I$ of the identity.

**iter** *GenNormal*:

```
1.  $A \leftarrow T$
2.  yield $A$
3.  for $i \leftarrow 1$ to $m-1$
4.      $A \leftarrow A^q$
5.      yield $A$
6.  endfor
```

The simplest external shift available in this case is a rotation. Other forms are possible.

**func** *RotateNormal*($u$)

```
1.  $u \leftarrow u^q$
```

```
2. **return** $u$
```

In order to calculate the dual of the external normal basis, we need $Z^{-1}$, where $Z$ is the internal representation of $\zeta$, and $S$, the internal representation of $\sigma$, where $\sigma$ is defined by $h_0(\sigma x) = h_1(x)$.

**iter** *GenDualNormal*:

```
1. $A \leftarrow Z^{-1}$
2. $B \leftarrow S$
3. yield $A$
4. for $i \leftarrow 1$ to $m$-1
5.     $A \leftarrow A \times B$
6.     $B \leftarrow B^q$
7.     yield $A$
8. endfor
```

### 5.3 Dual of Polynomial External Basis

If the external basis is the dual of a polynomial basis, we need only define shifting. Generation of the basis is done by *GenDualPoly*, and generation of the dual is done by *GenPoly*, since the dual of the dual of polynomial basis is the polynomial basis. In order to shift in the dual of polynomial basis, we need $Z$ and $Z^{-1}$, the internal representations of $\zeta$ and $\zeta^{-1}$, respectively, and $T^{-1}$, the internal representation of $t^{-1}$, where $t$ is the polynomial basis generator. Actually, we only need $ZT^{-1}$, $Z^{-1}$, and $I$.

**func** *ShiftPolyDual*($u$)

```
1. $u \leftarrow u \times Z\,T^{-1}$
2. $u \leftarrow u - h_0(u) \times I$
3. $u \leftarrow u \times Z^{-1}$
4. return $u$
```

### 5.4 Dual of Normal External Basis

Similar to above, we only need to define shifting, since generation is done by *GenDualNormal*, and generation of the dual is done by *GenNormal*. In order to shift, we need the constant $SZ^{q-1}$, where $S$ is the internal representation of $\sigma$, and $Z$ is the internal representation of $\zeta$.

**func** *ShiftNormalDual*($u$)

```
1. $u \leftarrow u^q$
2. $u \leftarrow u \times SZ^{q-1}$
3. return $u$
```

## 6. Claimed attributes and advantages

These algorithms for import and export run in time $O(m)$, or $O(m \log q)$ field operations, and require storage for $O(m)$ coefficients.
By contrast, $O(m^2)$ coefficient operations may take more time than $O(m)$ field operations, if the field operations are accelerated, and the storage requirement of $O(m^2)$ coefficients for the matrix method is significantly harder. Normally, $m$ is small enough that either could be handled easily on a normal computer but in constrained environments the advantage to these algorithms is significant.

## 7. Security Considerations

There are no general security considerations for these algorithms; they are merely alternate implementation methods. See Annex D.7 for possible implementation considerations, however.

## 8. Limitations and Disadvantages

These algorithms may take more time than the matrix multiplication method if internal basis operations are not accelerated. Furthermore, if the external basis is none of the ones we have specified, it may be difficult to perform the *Gen*, *Shift*, and *GenDual* functions; the ones we have given do not generalize to all external bases.

## 9. Intellectual property issues

RSA Laboratories has one issued patent [3] and one filed application [4] in the area of basis conversion. RSA Laboratories makes no representations regarding intellectual property claims by other parties.

## 10. References

[1] IEEE. *IEEE P1363: Standard Specifications for Public Key Cryptography*. Draft Version 8, October 5, 1998.

[2] *SAC '98*: "Storage Efficient Finite Field Basis Conversion," Burton S. Kaliski Jr. and Yiqun Lisa Yin, 1998.

[3] "Methods and Apparatus for Efficient Finite Field Basis Conversion," Patent, U.S. Serial No. 5,854,759, Burton S. Kaliski Jr., and Yiqun Lisa Yin, December 1998.

[4] "Efficient Finite Field Basis Conversion Involving A Dual Basis," Patent (pending), U.S. Application Serial No. 09/195,346, Burton S. Kaliski Jr., and Moses Liskov, November 1998.