

Storage-Efficient Basis Conversion Techniques

Contribution to IEEE P1363a

Leo Reyzin and Burt Kaliski, RSA Laboratories
reyzin@mit.edu, bkaliski@rsasecurity.com

February 18, 2000

Abstract. This contribution proposes text for possible inclusion in IEEE P1363a specifying storage-efficient finite field basis conversion techniques. Like IEEE P1363a, it is written as updates to the IEEE P1363 document. It is intended for discussion and review at the March 16-17, 2000, IEEE P1363 working group meeting. The contribution has not yet been approved by the working group.

For more information, please see the previous IEEE P1363a submission, B. Kaliski, M. Liskov and Y.L. Yin, "Efficient finite field basis conversion techniques," April 1999, which is available from <http://grouper.ieee.org/groups/1363/addendum.html>.

Copyright © 2000 by the Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved.

This is a contribution to an unapproved draft of a proposed IEEE Standard, subject to change. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities. If this document is to be submitted to ISO or IEC, notification shall be given to IEEE Copyright Administrator. Permission is also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position. Other entities seeking permission to reproduce portions of this document for these or other uses must contact the IEEE Standards Department for the appropriate license. Use of information contained in the unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P. O. Box 1331
Piscataway, NJ 08855-1331, USA

Annex A (Informative) Number-Theoretic Background (updated)

Add new Sections A.7.5, A.7.6, A.7.7 and A.7.8 to IEEE P1363 as indicated below:

A.7.5 Storage-Efficient Conversion Techniques

The matrix multiplication technique described in A.7.1 requires one to store m^2 binary coefficients for the matrix Γ or Γ^{-1} and perform linear algebra for each conversion. The techniques below, which are based on [KY98] require much smaller storage, after certain steps that depend only on the choice of basis, at the expense of a few extra operations per conversion. They generally utilize field multiplication in the internal basis rather than linear algebra; if field multiplication in the internal basis is optimized, the performance of these algorithms is comparable with the matrix multiplication method.

As in A.7.1, let B_1 be the basis in which the user performs field operations, and B_0 be the other basis. Section A.7.6 considers the task of importing (converting a representation in B_0 to a representation in B_1) and Section A.7.8 that of exporting (converting a representation in B_1 to a representation in B_0). The two tasks are inherently different, because it is assumed that the user can perform field operations in B_1 but not in B_0 .

A.7.6 Storage-Efficient Import

Input: a field degree m ; a (polynomial or normal) basis B_0 with field polynomial $p_0(t)$; a bit string \underline{a} of length m . (Note: in the case of a Gaussian normal basis, the field polynomial can be computed via A.7.2.)

Output: the element $v \in GF(2^m)$ (represented in the internal basis B_1) such that the representation of v in B_0 is \underline{a} .

If B_0 is a Polynomial Basis:

Setup (once per basis)

1. Let u be a root of $p_0(t)$ represented with respect to B_1 . (u can be computed via A.5.6; see also Note 2 below.)

Conversion (per each element)

2. Let $\underline{a} = (a_{m-1} \dots a_2 a_1 a_0)$.
3. Set $e \leftarrow 1$ (the element one of $GF(2^m)$, represented with respect to B_1).
4. If $a_{m-1} = 0$, then set $v \leftarrow 0$; else set $v \leftarrow e$.
5. For i from $m-2$ downto 0 do
 - 5.1 Set $v \leftarrow vu$.
 - 5.2 If $a_i = 1$, set $v \leftarrow v + e$.
6. Output v .

If B_0 is a Normal Basis:

Setup (once per basis)

1. Let u be a root of $p_0(t)$ represented with respect to B_1 . (u can be computed via A.5.6.)

Conversion (per each element)

2. Let $\underline{a} = (a_0 \ a_1 \ a_2 \ \dots \ a_{m-1})$.
3. If $a_0 = 0$, then set $v \leftarrow 0$; else set $v \leftarrow u$.
4. For i from 1 to $m - 1$ do
 - 4.1 Set $v \leftarrow v^2$.
 - 4.2 If $a_i = 1$, set $v \leftarrow v + u$.
5. Output v .

NOTES

1—Step 1 of the above algorithms needs be performed only once for each pair B_0 and B_1 ; the value u can then be stored and used every time an element needs to be imported. This is analogous to (but more space-efficient than) storing the matrix Γ to perform the import operation.

2—This algorithm may be invoked to facilitate division using the Extended Euclidean algorithm of A.4.4. Specifically, when B_1 is a normal basis, one needs to export elements α and β to a polynomial basis B_0 before computing $\gamma = \alpha / \beta$ via the Extended Euclidean algorithm, and then import γ back to B_1 . In that case, the best choice for B_0 is the polynomial basis with polynomial $p(t)$, where $p(t)$ is the normal polynomial for B_1 . Then, in Step 1, u should not be computed via A.5.6; instead, u will simply be the element whose representation with respect to B_1 is $(100\dots 0)$. (Computing u via A.5.6 would require one to use division and could thus cause infinitely deep recursion if division is again implemented via basis conversion combined with the Extended Euclidean algorithm.)

A.7.7 Multiplication Matrix for Polynomial Basis

The export method presented in A.7.8 requires a computation of the (leftmost) multiplication matrix M for the basis B_1 . For a polynomial basis, the matrix can be computed by the following algorithm. An algorithm for a normal basis is given in A.6.3.

Input: a field degree m ; a polynomial basis B_1 .

Output: the matrix M such that m_{ij} is the leftmost bit of the product of the i -th and j -th basis vectors of B_1 .

1. Let t be the root of the polynomial defining B_1 (t is represented in B_1 by the m -bit string $(00\dots 010)$).
2. Let $r := t^{m-1}$ (represented in B_1 by the m -bit string $(100\dots 0)$).
3. Let $b_0 := b_1 := \dots := b_{m-2} := 0$ and let $b_{m-1} := 1$.
4. For i from m to $2m - 2$ do
 - 4.1 Let $r := rt$.
 - 4.2 Let b_i be the leftmost bit of the representation of r in B_1 .
5. Set

$$M \leftarrow \begin{pmatrix} b_{2m-2} & b_{2m-3} & \Lambda & b_{m-1} \\ b_{2m-3} & b_{2m-4} & \Lambda & b_{m-2} \\ \vdots & \vdots & \ddots & \vdots \\ M & M & O & M \end{pmatrix}$$

6. Output M .

A.7.8 Storage-Efficient Export

Input: a field degree m ; a (polynomial or normal) basis B_0 with field polynomial $p_0(t)$; an element $v \in GF(2^m)$ (represented in the internal basis B_1). (Note: in the case of a Gaussian normal basis, the field polynomial can be computed via A.7.2.)

Output: the bit string \underline{a} of length m such that the representation of v in B_0 is \underline{a} .

If B_0 is a Polynomial Basis:

Setup (once per basis)

1. Compute the matrix Γ via A.7.3. Let u be a root of $p_0(t)$ represented with respect to B_1 . (It is computed in A.7.3.) Compute $w := u^{-1}$ via A.4.4. (See Note 3 below.)
2. Set $\Delta \leftarrow \Gamma^{-1}$ and let $\underline{\delta} = (\delta_{0,0} \ \delta_{1,0} \ \dots \ \delta_{m-1,0})$ be a row-vector whose elements are the 0-th column of Δ .
3. Compute the multiplication matrix M for B_1 via A.6.3 or A.7.7 (note that A.6.3 includes a separate efficient method for computing the multiplication matrix in the case of a Gaussian normal basis using A.3.7).
4. Let $\underline{z} = \underline{\delta} M^{-1}$. Note that \underline{z} is an m -element bit string; let z be the element of $GF(2^m)$ whose representation in B_1 is \underline{z} .

Conversion (per each element)

5. Set $v \leftarrow vz$.
6. For $i = 0$ to $m - 2$
 - 6.1 Let a_i be the leftmost bit of the representation of v in B_1 .
 - 6.2 If $a_i = 1$, then set $v \leftarrow v - z$.
 - 6.3 Set $v := vw$.
7. Let a_{m-1} be the leftmost bit of the representation of v in B_1 .
8. Output $\underline{a} := (a_{m-1} \ \dots \ a_2 \ a_1 \ a_0)$.

If B_0 is a Normal Basis:

Setup (once per basis)

1. Compute the matrix Γ via A.7.3.
2. Set $\Delta \leftarrow \Gamma^{-1}$ and let $\underline{\delta} = (\delta_{0,m-1} \ \delta_{1,m-1} \ \dots \ \delta_{m-1,m-1})$ be a row-vector whose elements are the $(m - 1)$ -st column of Δ .
3. Compute the multiplication matrix M for B_1 via A.7.7.
4. Let $\underline{z} = \underline{\delta} M^{-1}$. Note that \underline{z} is an m -element bit string; let z be the element of $GF(2^m)$ whose representation in B_1 is \underline{z} .

Conversion (per each element)

5. For $i = m - 1$ downto 1
 - 5.1 Compute $w := vz$ and let a_i be the leftmost bit of the representation of w in B_1 .
 - 5.2 Set $v := v^2$.
6. Compute $w := vz$ and let a_0 be the leftmost bit of the representation of w in B_1 .
7. Output $\underline{a} := (a_0 \ a_1 \ a_2 \ \dots \ a_{m-1})$.

NOTES

1—Steps 1-4 of the above algorithms need be performed only once for each pair B_0 and B_1 ; the value z (and w , in the case of polynomial basis) can then be stored and used every time an element needs to be imported. This is analogous to (but more space-efficient than) storing the matrix Γ^{-1} to perform the import operation.

2—The per-element conversion operations in both algorithms (steps 5-7 of the first algorithm and steps 5-6 of the second) involve only finite field operations. An alternative approach that is also storage-efficient combines finite field operations and dot product operations (a dot product $\underline{a} \cdot \underline{b}$ of two m -bit vectors $\underline{a} = (a_0 \ a_1 \ a_2 \ \dots \ a_{m-1})$ and $\underline{b} = (b_0 \ b_1 \ b_2 \ \dots \ b_{m-1})$ is a single bit defined as $a_0 b_0 \oplus a_1 b_1 \oplus \dots \oplus a_{m-1} b_{m-1}$). For a polynomial basis, steps 5-7 would be replaced with

5. Set $e \leftarrow 1$ (the element one of $GF(2^m)$, represented with respect to B_1).
6. For $i = 0$ to $m - 2$
 - 6.1 Let \underline{v} be the m -element bit string representing v in B_1 , and compute $a_i := \underline{v} \cdot \underline{s}$.
 - 6.2 If $a_i = 1$, then set $v \leftarrow v - e$.
 - 6.3 Set $v := v w$.
7. Let \underline{v} be the m -element bit string representing v in B_1 , and compute $a_{m-1} := \underline{v} \cdot \underline{s}$.

For a normal basis, steps 5-6 would be replaced with

5. For $i = m - 1$ downto 1
 - 5.1 Let \underline{v} be the m -element bit string representing v in B_1 , and compute $a_i := \underline{v} \cdot \underline{s}$.
 - 5.2 Set $v := v^2$.
6. Let \underline{v} be the m -element bit string representing v in B_1 , and compute $a_0 := \underline{v} \cdot \underline{s}$.

In both alternatives, the element z is not needed, so steps 3 and 4 of the algorithm can be omitted and A.7.7 is not needed.

3—As also noted above, this subroutine may be invoked to facilitate division using the Extended Euclidean algorithm of A.4.4. In such a case, in Step 1, u and Γ should be computed via A.7.4 rather than A.7.3 (u will simply be the element whose representation with respect to B_1 is $(100\dots0)$). $w = u^{-1}$ should be computed by raising u to the power $2^m - 2$ via A.4.3 (computing w via the extended Euclidean algorithm would require one to use basis conversion again, and could thus result in infinitely deep recursion).

Annex F (Informative) Bibliography (updated)

Add the following new reference:

[KY98] B.S. Kaliski Jr. and Y.L. Yin, "Storage-efficient finite field basis conversion," S. Tavares and H. Meijer, editors, *Selected Areas in Cryptography – SAC '98 Proceedings, Lecture Notes in Computer Science* **1556** (1999), Springer, 81-93.