SAM Threshold Sharing Schemes Version 1.0

Committee Specification 02 Update

~~19 August~~21 September 2021

This stage:
https://docs.oasis-open.org/sam/sam-tss/v1.0/cs03/sam-tss-v1.0-cs03.docx (Authoritative)
https://docs.oasis-open.org/sam/sam-tss/v1.0/cs03/sam-tss-v1.0-cs03.html
https://docs.oasis-open.org/sam/sam-tss/v1.0/cs03/sam-tss-v1.0-cs03.pdf

Previous stage:
https://docs.oasis-open.org/sam/sam-tss/v1.0/cs02/sam-tss-v1.0-cs02.docx (Authoritative)
https://docs.oasis-open.org/sam/sam-tss/v1.0/cs02/sam-tss-v1.0-cs02.html
https://docs.oasis-open.org/sam/sam-tss/v1.0/cs02/sam-tss-v1.0-cs02.pdf

~~**Previous stage:**~~
~~https://docs.oasis-open.org/sam/sam-tss/v1.0/cs01/sam-tss-v1.0-cs01.docx (Authoritative)~~
~~https://docs.oasis-open.org/sam/sam-tss/v1.0/cs01/sam-tss-v1.0-cs01.html~~
~~https://docs.oasis-open.org/sam/sam-tss/v1.0/cs01/sam-tss-v1.0-cs01.pdf~~

Latest stage:
https://docs.oasis-open.org/sam/sam-tss/v1.0/sam-tss-v1.0.docx (Authoritative)
https://docs.oasis-open.org/sam/sam-tss/v1.0/sam-tss-v1.0.html
https://docs.oasis-open.org/sam/sam-tss/v1.0/sam-tss-v1.0.pdf

Technical Committee:
OASIS Security Algorithms and Methods (SAM) TC

Chairs:
Tim Chevalier (Tim.Chevalier@netapp.com), NetApp
Tony Cox (tony.cox@cryptsoft.com), Cryptsoft Pty Ltd.

Editors:
Tim Chevalier (Tim.Chevalier@netapp.com), NetApp
Tim Hudson (tjh@cryptsoft.com), Cryptsoft Pty Ltd.

Abstract:

This document is intended for developers and architects who wish to design systems and applications that utilize threshold sharing schemes in an interoperable manner.

Status:

This document was last revised or approved by the OASIS Security Algorithms and Methods (SAM) TC on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=sam#technical.

TC members should send comments on this document to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/sam/.

This document is provided under the Non-Assertion Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this document, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/sam/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Key words:

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] and [RFC8174] when, and only when, they appear in all capitals, as shown here.

Citation format:

When referencing this document, the following citation format should be used:

[SAM-TSS-v1.0]

SAM Threshold Sharing Schemes Version 1.0. Edited by Tim Chevalier and Tim Hudson. ~~19 August~~21 September 2021. OASIS Committee Specification ~~02~~03. ~~https://docs.oasis-open.org/sam/sam-tss/v1.0/cs02/sam-tss-v1.0-cs02.html~~https://docs.oasis-open.org/sam/sam-tss/v1.0/cs03/sam-tss-v1.0-cs03.html. Latest stage: https://docs.oasis-open.org/sam/sam-tss/v1.0/sam-tss-v1.0.html.

Notices:

Table of Contents

1 Introduction

[All text is normative unless otherwise labeled]

1.1 Changes from earlier Versions

N/A

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] and [RFC8174] when, and only when, they appear in all capitals, as shown here.

2 Threshold Sharing Scheme 1

2.1 Algorithms

Shamir's secret sharing algorithm **[SHAMIR79]** provides a mechanism to be able to:

> [D]ivide data *D* into *n* pieces in such a way that *D* is easily reconstructable from any *k* pieces but even complete knowledge of *k-1* pieces reveals absolutely no information about *D*. This technique enables the construction of robust key management schemes for cryptographic systems that can function securely and reliably even when misfortunes destroy half the pieces and security breaches expose all but one of the remaining pieces.

Shamir's algorithm is a threshold secret sharing scheme. When applied to protection of cryptographic keys, the pieces are referred to as either key shares or key splits of the original key. The scheme may be referred to as either *k*-of-*n* or *m*-of-*n* where *k* or *m* refers to the minimum number of shares or splits (the threshold) required to reassemble the original key and *n* refers to the total number of shares or splits.

Shamir's scheme is based on polynomial interpolation. For interoperable systems the polynomial must be specified. The maximum number of shares or splits is determined by the specific polynomial.

In this specification, we select irreducible polynomials for interpolation over a finite field (Galois field) of the order 2^8 referred to as GF(2^8) or GF(256).

Within the finite field, we select between one of two polynomials (using the notation from **[FIPS197])**:

1. {01} {1B} (283) which is $x^8 + x^4 + x^3 + x + 1$
2. {01} {1D} (285) which is $x^8 + x^4 + x^3 + x^2 + 1$

The first polynomial (denoted throughout the rest of this document as 011B) is used within AES. The second polynomial (denoted throughout the rest of this document as 011D) is used within the RSA BSAFE range of SDKs from RSA Security and is included to enable compatibility with the broadest range of applications.

2.1.1 Field Representation and Mathematical Operations

For efficient implementation we define (for each polynomial) two tables—one table represents the exponentiation function (EXP), while the other table provides a representation of the logarithmic function (LOG). The exponentiation function is also referred to as the antilog function (ALOG). The *i*-th element of each table is denoted as EXP[*i*] and LOG[*i*] respectively.

All bytes in the threshold secret sharing algorithm are interpreted as finite field elements on which addition, subtraction, multiplication and division are defined as follows:

1. The addition operation (GF_ADD) takes two elements X and Y and returns the bitwise exclusive-or of its operands. Note that GF_ADD(0, X) returns X.
2. The subtraction operation (GF_SUB) is identical to addition, because the field has characteristic two.
3. The multiplication operation (GF_MUL) takes two elements X and Y as input and, if X or Y is zero, returns zero; otherwise, the operation returns the value EXP[(LOG[X]+LOG[Y]) modulo 255].
4. The division operation (GF_DIV) takes a dividend X and a divisor Y and computes X divided by Y. If X is zero, the operation returns zero. If Y is zero, an error condition is returned; otherwise, the operation returns the value EXP[(255-LOG[X]-LOG[Y]) modulo 255].

The exponentiation operation of raising X to the power of *i,* denoted by X^i, returns X multiplied by itself *i* times using GF_MUL is defined as GF_POW:

GF_POW(X, i) = GF_MUL(X, GF_MUL(X, … GF_MUL(X, X)))

The sum operation GF_SUM(X, *n*) takes *n* elements from the vector X (i.e. X[0], X[1], …, X[*n*-1]) and adds them together using GF_ADD operation:

GF_SUM(X, *n*) = GF_ADD(X[0], GF_ADD(X[1], GF_ADD(X[2], … GF_ADD(X[*n*-2], X[*n*-1])))).

Likewise, the product operation (GF_PROD) takes *n* elements from X and multiplies them together using the GF_MUL operation:

GF_PROD(X, *n*) = GF_MUL(X[0], GF_MUL(X[1], (GF_MUL(X[2], … GF_MUL(X[*n*-2], X[*n*-1])))))

For the specific values for the EXP and LOG tables for a given polynomial, refer to the Constants section of this specification.

2.2 Usage

2.2.1 Create Shares

The secret shares will be created from a secret, with length *L* bytes, where L ranges from zero to 65,534 (2^{16} – 2) bytes. A threshold number of shares, *m*, will be required in order to recombine the secret. The total number of shares is designated by *n*, with the constraint that $m \leq n \leq 255$.

Individual shares are calculated by applying a function *F* (~~P~~X, S) (see Appendix ~~E~~D) to ~~an *m*-~~a byte ~~vector P~~X (the share ID) and an *m*-byte input vector S:

F (~~P~~X, S) = GF_SUM(~~X~~Y[*i*~~], *m*),~~]), where ~~X~~Y[*i*] = GF_PROD(~~P~~S[*i*], ~~S[*i*])~~X^i) for *i* = 0, *m* – 1.

With input defined as:

- *n* = number of shares to be created.
- *m* = the threshold of shares required to be recombined to form the secret.
- *secret* = an *L*-byte vector containing the secret that will be split.
- *share_id*[*i*] = a unique identifier (range 1 … 255) for share *i*, otherwise known as the share ID for share *i*.
- R = a vector of random bytes of length (*m* - 1) *L*.

And with output defined as:

- *share*[*i*] = a vector of shares (*i* = 0 … *n* – 1) with each share containing *L* + 1 bytes (note: the first byte contains *share_id*[*i*]).

The pseudo code algorithm for calculating the *n* shares is:

```
CheckErrors:
  if L < 0 or L > 65,534 then
    throw an error and exit
  endif

  if m < 1 or m > 255 then
    throw an error and exit
  endif
  if n < m or n > 255 then
    throw an error and exit
  endif

  for i = 0 to n - 1
  do
    if share_id[i] not unique then
      throw an error and exit
    endif

    if share_id[i] == 0 or share_id[i] > 255 then
      throw an error and exit
    endif
  done

CalculateShares:
  for i = 0 to n - 1
  do
    share[i][0] = share_id[i]
  done

  counter = 0
  for i = 0 to L - 1
  do
    for j = 1 to m - 1
    do
      S[i][0] = secret[i]
      S[i][j] = R[counter]
      counter = counter + 1
    done
  done

  for i = 0 to n - 1
  do
    for j = 0 to ~~m~~L - 1
      do
~~        P[j] = GF_POW(share_id[i], j)~~
~~      done~~
      share[i][j + 1] = F(~~P,~~shares[i][0], S[~~i~~j])
      done
  done

F(X, Sj):
  result = 0
  for k = 0 to m - 1
  do
    result = GF_ADD(result, GF_MUL(Sj[k], GF_POW(X,k)))
  done
  return result
```

2.2.2 Recombine Shares

Recombining the shares requires *m* of the shares to be provided. If more than *m* shares are provided, a subset of *m* of the shares is selected. The method used for selecting which shares to use is arbitrary.

With the help of the function $L_{(i)}(U)$:

$L_{(i)}(U)$ = GF_PROD(GF_DIV(U[*j*], GF_SUM(U[*j*], U[*i*])), *j*) for $j = 0 \ldots m, j \neq i$, individual shares can be recombined into the secret by applying an interpolation function I (U, V) to an m-byte vector U and an m-byte input vector V:

I (U, V) = GF_SUM(X[i], m), where X[i] = GF_PROD(L$_{(i)}$(U), V[i]) for i = 0, m – 1.

The function L$_{(i)}$(U) is defined as:

L$_{(i)}$(U) = GF_PROD(GF_DIV(U[j], GF_SUM(U[j], U[i])), j) for j = 0 … m, $j \neq i$

With input defined as:

- *m* = the threshold of shares required to be recombined to form the secret.
- *share*[*i*] = a vector of shares (*i* = 0 … *m*) with each share containing *L* + 1 bytes (note: the first byte contains *share_id*[*i*]).

And with output defined as:

- *secret* = an *L*-byte vector containing the original secret.

The pseudo code algorithm for recombining the *n* shares to produce the original secret is:

```
CheckErrors:
  set length0 = length(share[0])
  for i = 0 to m - 1
  do
    if length(share[i]) not equal length0 then
      throw an error and exit
    endif
  done

RecombineShares:
  for i = 0 to m - 1
  do
    U[i] = share[i][0]
  done

  secret = empty

  for i = 0 to (length0 - 1)
  do
    for j = 0 to m - 1
    do
      V[j] = share[i][j + 1]
    done
    secret = secret concatenate I(U, V)
  done
```

2.3 Constants

The following EXP and LOG tables are used for each defined polynomial.

2.3.1 Polynomial 1

For {01} {1B} (283) (i.e., $x^8 + x^4 + x^3 + x + 1$) the following tables apply.

Note: tables are defined such that LOG[EXP[*x*]] and EXP[LOG[*x*]] return *x* for all 0x00 < *x* ≤ 0xff.

2.3.1.1 LOG

```
0x00,0xff,0x19,0x01,0x32,0x02,0x1a,0xc6,
0x4b,0xc7,0x1b,0x68,0x33,0xee,0xdf,0x03,
0x64,0x04,0xe0,0x0e,0x34,0x8d,0x81,0xef,
0x4c,0x71,0x08,0xc8,0xf8,0x69,0x1c,0xc1,
0x7d,0xc2,0x1d,0xb5,0xf9,0xb9,0x27,0x6a,
0x4d,0xe4,0xa6,0x72,0x9a,0xc9,0x09,0x78,
0x65,0x2f,0x8a,0x05,0x21,0x0f,0xe1,0x24,
0x12,0xf0,0x82,0x45,0x35,0x93,0xda,0x8e,
0x96,0x8f,0xdb,0xbd,0x36,0xd0,0xce,0x94,
0x13,0x5c,0xd2,0xf1,0x40,0x46,0x83,0x38,
0x66,0xdd,0xfd,0x30,0xbf,0x06,0x8b,0x62,
0xb3,0x25,0xe2,0x98,0x22,0x88,0x91,0x10,
0x7e,0x6e,0x48,0xc3,0xa3,0xb6,0x1e,0x42,
0x3a,0x6b,0x28,0x54,0xfa,0x85,0x3d,0xba,
0x2b,0x79,0x0a,0x15,0x9b,0x9f,0x5e,0xca,
0x4e,0xd4,0xac,0xe5,0xf3,0x73,0xa7,0x57,
0xaf,0x58,0xa8,0x50,0xf4,0xea,0xd6,0x74,
0x4f,0xae,0xe9,0xd5,0xe7,0xe6,0xad,0xe8,
0x2c,0xd7,0x75,0x7a,0xeb,0x16,0x0b,0xf5,
0x59,0xcb,0x5f,0xb0,0x9c,0xa9,0x51,0xa0,
0x7f,0x0c,0xf6,0x6f,0x17,0xc4,0x49,0xec,
0xd8,0x43,0x1f,0x2d,0xa4,0x76,0x7b,0xb7,
0xcc,0xbb,0x3e,0x5a,0xfb,0x60,0xb1,0x86,
0x3b,0x52,0xa1,0x6c,0xaa,0x55,0x29,0x9d,
0x97,0xb2,0x87,0x90,0x61,0xbe,0xdc,0xfc,
0xbc,0x95,0xcf,0xcd,0x37,0x3f,0x5b,0xd1,
0x53,0x39,0x84,0x3c,0x41,0xa2,0x6d,0x47,
0x14,0x2a,0x9e,0x5d,0x56,0xf2,0xd3,0xab,
0x44,0x11,0x92,0xd9,0x23,0x20,0x2e,0x89,
0xb4,0x7c,0xb8,0x26,0x77,0x99,0xe3,0xa5,
0x67,0x4a,0xed,0xde,0xc5,0x31,0xfe,0x18,
0x0d,0x63,0x8c,0x80,0xc0,0xf7,0x70,0x07
```

2.3.1.2 EXP

```
0x01,0x03,0x05,0x0f,0x11,0x33,0x55,0xff,
0x1a,0x2e,0x72,0x96,0xa1,0xf8,0x13,0x35,
0x5f,0xe1,0x38,0x48,0xd8,0x73,0x95,0xa4,
0xf7,0x02,0x06,0x0a,0x1e,0x22,0x66,0xaa,
0xe5,0x34,0x5c,0xe4,0x37,0x59,0xeb,0x26,
0x6a,0xbe,0xd9,0x70,0x90,0xab,0xe6,0x31,
0x53,0xf5,0x04,0x0c,0x14,0x3c,0x44,0xcc,
0x4f,0xd1,0x68,0xb8,0xd3,0x6e,0xb2,0xcd,
0x4c,0xd4,0x67,0xa9,0xe0,0x3b,0x4d,0xd7,
0x62,0xa6,0xf1,0x08,0x18,0x28,0x78,0x88,
0x83,0x9e,0xb9,0xd0,0x6b,0xbd,0xdc,0x7f,
0x81,0x98,0xb3,0xce,0x49,0xdb,0x76,0x9a,
0xb5,0xc4,0x57,0xf9,0x10,0x30,0x50,0xf0,
0x0b,0x1d,0x27,0x69,0xbb,0xd6,0x61,0xa3,
0xfe,0x19,0x2b,0x7d,0x87,0x92,0xad,0xec,
0x2f,0x71,0x93,0xae,0xe9,0x20,0x60,0xa0,
0xfb,0x16,0x3a,0x4e,0xd2,0x6d,0xb7,0xc2,
0x5d,0xe7,0x32,0x56,0xfa,0x15,0x3f,0x41,
0xc3,0x5e,0xe2,0x3d,0x47,0xc9,0x40,0xc0,
0x5b,0xed,0x2c,0x74,0x9c,0xbf,0xda,0x75,
0x9f,0xba,0xd5,0x64,0xac,0xef,0x2a,0x7e,
0x82,0x9d,0xbc,0xdf,0x7a,0x8e,0x89,0x80,
0x9b,0xb6,0xc1,0x58,0xe8,0x23,0x65,0xaf,
0xea,0x25,0x6f,0xb1,0xc8,0x43,0xc5,0x54,
0xfc,0x1f,0x21,0x63,0xa5,0xf4,0x07,0x09,
0x1b,0x2d,0x77,0x99,0xb0,0xcb,0x46,0xca,
0x45,0xcf,0x4a,0xde,0x79,0x8b,0x86,0x91,
0xa8,0xe3,0x3e,0x42,0xc6,0x51,0xf3,0x0e,
0x12,0x36,0x5a,0xee,0x29,0x7b,0x8d,0x8c,
0x8f,0x8a,0x85,0x94,0xa7,0xf2,0x0d,0x17,
0x39,0x4b,0xdd,0x7c,0x84,0x97,0xa2,0xfd,
0x1c,0x24,0x6c,0xb4,0xc7,0x52,0xf6,0x01
```

2.3.2 Polynomial 2

For {01} {1D} (285) (i.e., $x^8 + x^4 + x^3 + x^2 + 1$) the following tables apply.

Note: tables are defined such that LOG[EXP[*x*]] and EXP[LOG[*x*]] return *x* for all 0x00 < *x* ≤ 0xff.

2.3.2.1 LOG

```
0xff,0x00,0x01,0x19,0x02,0x32,0x1a,0xc6,
0x03,0xdf,0x33,0xee,0x1b,0x68,0xc7,0x4b,
0x04,0x64,0xe0,0x0e,0x34,0x8d,0xef,0x81,
0x1c,0xc1,0x69,0xf8,0xc8,0x08,0x4c,0x71,
0x05,0x8a,0x65,0x2f,0xe1,0x24,0x0f,0x21,
0x35,0x93,0x8e,0xda,0xf0,0x12,0x82,0x45,
0x1d,0xb5,0xc2,0x7d,0x6a,0x27,0xf9,0xb9,
0xc9,0x9a,0x09,0x78,0x4d,0xe4,0x72,0xa6,
0x06,0xbf,0x8b,0x62,0x66,0xdd,0x30,0xfd,
0xe2,0x98,0x25,0xb3,0x10,0x91,0x22,0x88,
0x36,0xd0,0x94,0xce,0x8f,0x96,0xdb,0xbd,
0xf1,0xd2,0x13,0x5c,0x83,0x38,0x46,0x40,
0x1e,0x42,0xb6,0xa3,0xc3,0x48,0x7e,0x6e,
0x6b,0x3a,0x28,0x54,0xfa,0x85,0xba,0x3d,
0xca,0x5e,0x9b,0x9f,0x0a,0x15,0x79,0x2b,
0x4e,0xd4,0xe5,0xac,0x73,0xf3,0xa7,0x57,
0x07,0x70,0xc0,0xf7,0x8c,0x80,0x63,0x0d,
0x67,0x4a,0xde,0xed,0x31,0xc5,0xfe,0x18,
0xe3,0xa5,0x99,0x77,0x26,0xb8,0xb4,0x7c,
0x11,0x44,0x92,0xd9,0x23,0x20,0x89,0x2e,
0x37,0x3f,0xd1,0x5b,0x95,0xbc,0xcf,0xcd,
0x90,0x87,0x97,0xb2,0xdc,0xfc,0xbe,0x61,
0xf2,0x56,0xd3,0xab,0x14,0x2a,0x5d,0x9e,
0x84,0x3c,0x39,0x53,0x47,0x6d,0x41,0xa2,
0x1f,0x2d,0x43,0xd8,0xb7,0x7b,0xa4,0x76,
0xc4,0x17,0x49,0xec,0x7f,0x0c,0x6f,0xf6,
0x6c,0xa1,0x3b,0x52,0x29,0x9d,0x55,0xaa,
0xfb,0x60,0x86,0xb1,0xbb,0xcc,0x3e,0x5a,
0xcb,0x59,0x5f,0xb0,0x9c,0xa9,0xa0,0x51,
0x0b,0xf5,0x16,0xeb,0x7a,0x75,0x2c,0xd7,
0x4f,0xae,0xd5,0xe9,0xe6,0xe7,0xad,0xe8,
0x74,0xd6,0xf4,0xea,0xa8,0x50,0x58,0xaf
```

2.3.2.2 EXP

```
0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
0x1d,0x3a,0x74,0xe8,0xcd,0x87,0x13,0x26,
0x4c,0x98,0x2d,0x5a,0xb4,0x75,0xea,0xc9,
0x8f,0x03,0x06,0x0c,0x18,0x30,0x60,0xc0,
0x9d,0x27,0x4e,0x9c,0x25,0x4a,0x94,0x35,
0x6a,0xd4,0xb5,0x77,0xee,0xc1,0x9f,0x23,
0x46,0x8c,0x05,0x0a,0x14,0x28,0x50,0xa0,
0x5d,0xba,0x69,0xd2,0xb9,0x6f,0xde,0xa1,
0x5f,0xbe,0x61,0xc2,0x99,0x2f,0x5e,0xbc,
0x65,0xca,0x89,0x0f,0x1e,0x3c,0x78,0xf0,
0xfd,0xe7,0xd3,0xbb,0x6b,0xd6,0xb1,0x7f,
0xfe,0xe1,0xdf,0xa3,0x5b,0xb6,0x71,0xe2,
0xd9,0xaf,0x43,0x86,0x11,0x22,0x44,0x88,
0x0d,0x1a,0x34,0x68,0xd0,0xbd,0x67,0xce,
0x81,0x1f,0x3e,0x7c,0xf8,0xed,0xc7,0x93,
0x3b,0x76,0xec,0xc5,0x97,0x33,0x66,0xcc,
0x85,0x17,0x2e,0x5c,0xb8,0x6d,0xda,0xa9,
0x4f,0x9e,0x21,0x42,0x84,0x15,0x2a,0x54,
0xa8,0x4d,0x9a,0x29,0x52,0xa4,0x55,0xaa,
0x49,0x92,0x39,0x72,0xe4,0xd5,0xb7,0x73,
0xe6,0xd1,0xbf,0x63,0xc6,0x91,0x3f,0x7e,
0xfc,0xe5,0xd7,0xb3,0x7b,0xf6,0xf1,0xff,
0xe3,0xdb,0xab,0x4b,0x96,0x31,0x62,0xc4,
0x95,0x37,0x6e,0xdc,0xa5,0x57,0xae,0x41,
0x82,0x19,0x32,0x64,0xc8,0x8d,0x07,0x0e,
0x1c,0x38,0x70,0xe0,0xdd,0xa7,0x53,0xa6,
0x51,0xa2,0x59,0xb2,0x79,0xf2,0xf9,0xef,
0xc3,0x9b,0x2b,0x56,0xac,0x45,0x8a,0x09,
0x12,0x24,0x48,0x90,0x3d,0x7a,0xf4,0xf5,
0xf7,0xf3,0xfb,0xeb,0xcb,0x8b,0x0b,0x16,
0x2c,0x58,0xb0,0x7d,0xfa,0xe9,0xcf,0x83,
0x1b,0x36,0x6c,0xd8,0xad,0x47,0x8e,0x01
```

2.4 Test Vectors

The following test vectors are defined for each defined polynomial.

2.4.1 Polynomial 1

For {01} {1B} (283) (i.e., x^8 + x4 + x^3 + x + 1) the following tests apply.

Note: in the test vectors, the share splits are defined without the leading share ID. The share ID associated with a share are indicated separately.

2.4.1.1 Test Vector TV011B_1

```
secret=7465737400
m=2
n=2
random=A87B3491B58BE31DBA42
split1=DC1E47E5B5 (share ID = 0x01)
split2=3F931B4D71 (share ID = 0x02)
```

2.4.1.2 Test Vector TV011B_2

```
secret=53414D5443
m=2
n=4
random=395D396C87CE75528C6691C7C0BF233AE7C96F8B
split1=6A1C7438C4 (share ID = 0x01)
split2=21FB3F8C56 (share ID = 0x02)
split3=18A606E0D1 (share ID = 0x03)
split4=B72EA9FF69 (share ID = 0x04)
```

2.4.1.3 Test Vector TV011B_3

```
secret=53414D5443
m=3
n=4
random=273A1A28AB9979BC06377C48595FF88195C2FB50
split1=4E737F9172 (share ID = 0x01)
split2=F5D5526093 (share ID = 0x02)
split3=E8E760A5A2 (share ID = 0x03)
split4=429F849E06 (share ID = 0x04)
```

2.4.1.4 Test Vector TV011B_4

```
secret=53414D5443
m=4
n=4
random=1A224C1EE9760A73A09D05774434672361D9ADF5
split1=27C094BB54 (share ID = 0x01)
split2=B969F9F40E (share ID = 0x02)
split3=7EC7CD3250 (share ID = 0x03)
split4=ABAF81828D (share ID = 0x04)
```

2.4.1.5 Test Vector TV011B_5

```
secret=546573742044617461
m=2
n=9
random=7FB4E8581EB75DC9453FE7837DDC3036F50347B820B85C88059274BA279E5EF11DF69B
841F9A2636C7AA3613EED2B8C94F8C12A3C0D99603BFA9CB7B7CFDF81927B1E01B162ACF119BD
E67FE7A6C9BDC7F
split1=2BD19B2C3EF33CBD24 (share ID = 0x01)
split2=AA16B8C41C31DBFDEB (share ID = 0x02)
split3=D5A2509C02868634AE (share ID = 0x03)
split4=B383FE0F58AE0E7D6E (share ID = 0x04)
split5=CC3716574619 53B42B (share ID = 0x05)
split6=4DF035BF64DBB4F4E4 (share ID = 0x06)
split7=3244DDE77A6CE93DA1 (share ID = 0x07)
split8=81B27282D08BBF667F (share ID = 0x08)
split9=FE069ADACE3CE2AF3A (share ID = 0x09)
```

2.4.1.6 Test Vector TV011B_6

```
secret=0102030405060708090A0B0C0D0E0F
m=3
```

```
n=5
random=EC96740540B3E1FC9A914F6E5F7CCA51DB723202C9B881004F66A28071974F36070E5A
77DA6F79355B5A3D9CB932DE31B247766B3F30E002FBFE9B415708F9811DE4C1011F13291AFF4
1D4
split1=7B73F0190E272493A03A7A8D242CE9 (share ID = 0x01)
split2=ACFE7900583B52D877665415106787 (share ID = 0x02)
split3=D68F8A1D531A7143DE562594394561 (share ID = 0x03)
split4=3F99DDF4889BE16A29E2773E106863 (share ID = 0x04)
split5=45E82EE983BAC2F180D206BF394A85 (share ID = 0x05)
```

2.4.2 Polynomial 2

For {01} {1D} (285) (i.e., $x^8 + x^4 + x^3 + x^2 + 1$) the following tests apply.

Note: in the test vectors, the share splits are defined without the leading share ID. The share ID associated with a share are indicated separately.

2.4.2.1 Test Vector TV011D_1

```
secret=7465737400
m=2
n=2
random=F3C23381F59F58B39A8B
split1=87A740F5F5 (share ID = 0x01)
split2=8FFC156BF7 (share ID = 0x02)
```

2.4.2.2 Test Vector TV011D_2

```
secret=53414D5443
m=2
n=4
random=207608930C013A33DD325324F583054DD8A0C03D
split1=733745C74F (share ID = 0x01)
split2=13AD5D6F5B (share ID = 0x02)
split3=33DB55FC57 (share ID = 0x03)
split4=D3846D2273 (share ID = 0x04)
```

2.4.2.3 Test Vector TV011D_3

```
secret=53414D5443
m=3
n=4
random=8C1592625C4AAF5341452253A3FBD2E791EFF389
split1=CAB15BA847 (share ID = 0x01)
split2=02EDC046C8 (share ID = 0x02)
```

```
split3=9B1DD6BACC (share ID = 0x03)
split4=145DF48B7E (share ID = 0x04)
```

2.4.2.4 Test Vector TV011D_4

```
secret=53414D5443
m=4
n=4
random=72B0883C96B9CBB9CBB28266F379FA4B7501A26A
split1=1952F40233 (share ID = 0x01)
split2=79FA0E08C2 (share ID = 0x02)
split3=2458371794 (share ID = 0x03)
split4=F445A9D607 (share ID = 0x04)
```

2.4.2.5 Test Vector TV011D_5

```
secret=546573742044617461
m=2
n=9
random=AFFD2B0BFA3433639CE8ECCEEA27F21AA5C1F562D2E090F2B7E7DFD62F0C48A82ADC79
876594E020D2499F6049A551D68ED675EF05557EC8FB0C87DFCE4673299A877269B8879F322DE
8F25A9A283C9DCE
split1=FB98587FDA705217FD (share ID = 0x01)
split2=17822562C92C07B244 (share ID = 0x02)
split3=B87F0E69331834D1D8 (share ID = 0x03)
split4=D2B6DF58EF94ADE52B (share ID = 0x04)
split5=7D4BF45315A09E86B7 (share ID = 0x05)
split6=9151894E06FCCB230E (share ID = 0x06)
split7=3EACA245FCC8F84092 (share ID = 0x07)
split8=45DE362CA3F9E44BF5 (share ID = 0x08)
split9=EA231D2759CDD72869 (share ID = 0x09)
```

2.4.2.6 Test Vector TV011D_6

```
secret=0102030405060708090A0B0C0D0E0F
m=3
n=5
random=024A89AC968C986577FEB024116B94F654DDDE209C3CC3E448884D31F8C8D8A8074C8B
900B062AF42C857A6D4EB2BAC11F33BB92C8A74A1FE20C2F550E902F405B43C0E8E22E4CA91B7
D55
split1=492719F98C927D6A80F4AB2BCD723F (share ID = 0x01)
split2=308738A034EB94C2F22BDE208750E5 (share ID = 0x02)
split3=78A2225DBD7FEEA07BD57E07472CD5 (share ID = 0x03)
split4=DD0E49409F86BDB9156FA6C15810D4 (share ID = 0x04)
```

```
split5=952B53BD1612C7DB9C9106E6986CE4 (share ID = 0x05)
```

3 Conformance

3.1 SAM Threshold Sharing Scheme 1 (TSS1) Implementation Conformance

An implementation is a conforming SAM TSS1 implementation if the implementation passes all applicable Test Cases in this document.

A SAM TSS1 implementation SHALL be a conforming SAM TSS1 implementation.

Appendix A. References

This appendix contains the normative and informative references that are used in this document. Normative references are specific (identified by date of publication and/or edition number or Version number) and Informative references are either specific or non-specific.

While any hyperlinks included in this appendix were valid at the time of publication, OASIS cannot guarantee their long-term validity.

A.1 Normative References

The following documents are referenced in such a way that some or all of their content constitutes requirements of this document.

[FIPS197] Advanced Encryption Standard, FIPS PUB 197, Nov 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <http://www.rfc-editor.org/info/rfc8174>.

[SHAMIR79] Shamir, Adi, "How to share a secret", Communications of the ACM, 22(11) : 612-613, <https://dl.acm.org/doi/10.1145/359168.359176>.

A.2 Informative References

The following referenced documents are not required for the application of this document but may assist the reader with regard to a particular subject area.

[Cryptol] Galois, Inc., <https://www.cryptol.net>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10. 17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

Appendix B. Acknowledgments

B.1 Participants

The following individuals were members of this Technical Committee during the creation of this document and their contributions are gratefully acknowledged:

[Participant Name, Affiliation | Individual Member]

Person	Organization	Role
Patrick Maroney	AT&T	Member
Tony Cox	Cryptsoft Pty Ltd.	Chair
Tim Hudson	Cryptsoft Pty Ltd.	Voting Member
Scott Marshall	Cryptsoft Pty Ltd.	Voting Member
Bruce Rich	Cryptsoft Pty Ltd.	Voting Member
Greg Scott	Cryptsoft Pty Ltd.	Voting Member
Jason Thatcher	Cryptsoft Pty Ltd.	Voting Member
Judith Furlong	Dell	Voting Member
Deepak Gaikwad	Dell	Voting Member
Ravi Sharda	Dell	Member
Gerald Stueve	Fornetix	Voting Member

Person	Organization	Role
Christopher Hillier	Hewlett Packard Enterprise (HPE)	Member
Andrea Caccia	Individual	Voting Member
Sander Fieten	Individual	Member
Stefan Hagen	Individual	Secretary
Tim Chevalier	NetApp	Chair
Pim van der Eijk	Sonnenglanz Consulting	Member

Appendix C. Revision History

Revisions made since the initial stage of this numbered Version of this document may be tracked here.

Revision	Date	Editor	Changes Made
CSD01	May 2, 2021	Tim Chevalier and Tim Hudson	Initial draft
CS01	August 4, 2021	TC Admin	Published as Committee Specification 01
CS01 (update)	August 11, 2021	Tony Cox	Added “Informative” notation to Appendix E
CS02	August 19, 2021	Tony Cox	Published CS
CS02 Update	September 21, 2021	Tim Chevalier and Tim Hudson	CS02 with non-material changes following further editorial review

Appendix D. Secret Sharing described in terms of matrix operations

The secret share creation process and the secret share recombination process may be described as matrix operations over GF(256).

D.1 Create Shares

Given a secret, *s*, of length *L* bytes, share creation can be expressed as matrix multiplication over GF(256):

$$P\ S = \sigma$$

where:

P is an n x m matrix,

S is an m x L matrix,

σ is an n x L matrix containing the shares,

n = number of shares to be created, and

m = the threshold of shares required to be recombined to form the secret.

D.1.1 Creating the P matrix

The P_{ij} elements contained in the n x m matrix P are given by the following formula:

P_{ij} = GF_POW(i, j – 1) for i = 1..n and j = 1..m

Where GF_POW(i, j - 1) is defined as the GF(256) element i raised to the power (j - 1). Thus, P contains the follow elements:

$P_{i,1}$ = GF_POW(i, 0x00) = 0x01

$P_{i,2}$ = GF_POW(i, 0x01) = GF_MUL(i, $P_{i,1}$) = EXP[LOG[i]] = i

$P_{i,3}$ = GF_POW(i, 0x02) = GF_MUL(i, $P_{i,2}$) = EXP[(LOG[i] + LOG[$P_{i,2}$]) % 255]

$P_{i,4}$ = GF_POW(i, 0x03) = GF_MUL(i, $P_{i,3}$) = EXP[(LOG[i] + LOG[$P_{i,3}$]) % 255]

…

$P_{i,m}$ = GF_POW(i, m - 1) = GF_MUL(i, $P_{i,m-1}$) = EXP[(LOG[i] + LOG[$P_{i,m-1}$]) % 255]

D.1.2 Creating the S matrix

The S matrix (with dimensions m x L) is created from the secret, s (length L), and a random set of bytes R of length (m – 1) L. The elements of the matrix S are:

$S_{1,j}$ = byte s[j] from secret s for j = 1, …, L

$S_{i,j}$ = byte R[k], where k = (i – 1) + (m – 1)(j – 1), for i = 2, …, m and j = 1, …, L

For example, with:

s = [53 41 4D 54 43]

$m = 4$

$n = 4$, and

R = [1A 22 4C 1E E9 76 0A 73 A0 9D 05 77 44 34 67]

The (4 x 5) matrix S would be:

53 41 4D 54 43
1A 1E 0A 9D 44
22 E9 73 05 34
4C 76 A0 77 67

D.1.3 Calculating the σ matrix

The σ matrix is calculated as the matrix product PS, with individual elements of σ calculated as the dot product over GF(128):

$$\sigma_{i,j} = P_{i,1}S_{1,j} + P_{i,2}S_{2,j} + \ldots + P_{n,m}S_{m,l}$$
$$= \mathrm{GF\_SUM}[P_{i,1}S_{1,j}, P_{i,2}S_{2,j}, \ldots, P_{n,m}S_{m,l}]$$
$$= \mathrm{GF\_SUM}[\mathrm{GF\_MUL}(P_{i,1},S_{1,j}), \mathrm{GF\_MUL}(P_{i,2}, {}_1S_{2,j}), \ldots , \mathrm{GF\_MUL}(P_{n,m}, S_{m,l})]$$

D.2 Recombine Shares

Given m share indices, where each share index, U_i, is unique and $1 \leq U_i \leq 255$, share recombination can be expressed as matrix multiplication over GF(256):

$$\mathcal{L}\,\sigma = s$$

where:

$\mathcal{L}$ is a 1 x m matrix,

σ is an m x L matrix containing shares,

m = threshold number of shares to be recombined, and

s = the secret (of length L).

The $\mathcal{L}_{ij}$ elements contained in the 1 x m matrix $\mathcal{L}$ are given by the following formula:

$$\mathcal{L}_{ij} = \mathrm{GF\_PROD}(\mathrm{GF\_DIV}(U_j, \mathrm{GF\_SUM}(U_j, U_i)) \text{ for } j = 0 \ldots m - 1, j \neq i$$

Appendix E. Cryptol code for testing secret share creation and recombination

This appendix is non-normative (informative).

Cryptol **[Cryptol]** is a programming language used for cryptography that is developed and maintained by Galois, Inc.. This appendix contains Cryptol code for creating and recombining splits for both the polynomial and the polynomial. In addition, there are separate table-based and “native” GF(256)-based implementations.

E.1 Polynomial $x^8 + x^4 + x^3 + x + 1$

Cryptol code for creating splits, checking the splits, re-combining splits, and checking the recombinations appears in the following subsections.

E.1.1 Table-based implementation

```
module secret_share_011B_table where

type GF28 = [8]

/* given an alpha^j, where alpha = mimimum primitive element (x + 1 = 3), return j */
LOG : [256][8]
LOG = [
    0x00, 0xff, 0x19, 0x01, 0x32, 0x02, 0x1a, 0xc6,
    0x4b, 0xc7, 0x1b, 0x68, 0x33, 0xee, 0xdf, 0x03,
    0x64, 0x04, 0xe0, 0x0e, 0x34, 0x8d, 0x81, 0xef,
    0x4c, 0x71, 0x08, 0xc8, 0xf8, 0x69, 0x1c, 0xc1,
    0x7d, 0xc2, 0x1d, 0xb5, 0xf9, 0xb9, 0x27, 0x6a,
    0x4d, 0xe4, 0xa6, 0x72, 0x9a, 0xc9, 0x09, 0x78,
    0x65, 0x2f, 0x8a, 0x05, 0x21, 0x0f, 0xe1, 0x24,
    0x12, 0xf0, 0x82, 0x45, 0x35, 0x93, 0xda, 0x8e,
    0x96, 0x8f, 0xdb, 0xbd, 0x36, 0xd0, 0xce, 0x94,
    0x13, 0x5c, 0xd2, 0xf1, 0x40, 0x46, 0x83, 0x38,
    0x66, 0xdd, 0xfd, 0x30, 0xbf, 0x06, 0x8b, 0x62,
    0xb3, 0x25, 0xe2, 0x98, 0x22, 0x88, 0x91, 0x10,
    0x7e, 0x6e, 0x48, 0xc3, 0xa3, 0xb6, 0x1e, 0x42,
    0x3a, 0x6b, 0x28, 0x54, 0xfa, 0x85, 0x3d, 0xba,
    0x2b, 0x79, 0x0a, 0x15, 0x9b, 0x9f, 0x5e, 0xca,
    0x4e, 0xd4, 0xac, 0xe5, 0xf3, 0x73, 0xa7, 0x57,
    0xaf, 0x58, 0xa8, 0x50, 0xf4, 0xea, 0xd6, 0x74,
    0x4f, 0xae, 0xe9, 0xd5, 0xe7, 0xe6, 0xad, 0xe8,
    0x2c, 0xd7, 0x75, 0x7a, 0xeb, 0x16, 0x0b, 0xf5,
    0x59, 0xcb, 0x5f, 0xb0, 0x9c, 0xa9, 0x51, 0xa0,
    0x7f, 0x0c, 0xf6, 0x6f, 0x17, 0xc4, 0x49, 0xec,
    0xd8, 0x43, 0x1f, 0x2d, 0xa4, 0x76, 0x7b, 0xb7,
    0xcc, 0xbb, 0x3e, 0x5a, 0xfb, 0x60, 0xb1, 0x86,
    0x3b, 0x52, 0xa1, 0x6c, 0xaa, 0x55, 0x29, 0x9d,
    0x97, 0xb2, 0x87, 0x90, 0x61, 0xbe, 0xdc, 0xfc,
    0xbc, 0x95, 0xcf, 0xcd, 0x37, 0x3f, 0x5b, 0xd1,
    0x53, 0x39, 0x84, 0x3c, 0x41, 0xa2, 0x6d, 0x47,
    0x14, 0x2a, 0x9e, 0x5d, 0x56, 0xf2, 0xd3, 0xab,
    0x44, 0x11, 0x92, 0xd9, 0x23, 0x20, 0x2e, 0x89,
    0xb4, 0x7c, 0xb8, 0x26, 0x77, 0x99, 0xe3, 0xa5,
    0x67, 0x4a, 0xed, 0xde, 0xc5, 0x31, 0xfe, 0x18,
    0x0d, 0x63, 0x8c, 0x80, 0xc0, 0xf7, 0x70, 0x07
]

/* given a j, return alpha^j, where alpha = mimimum primitive element (x + 1 = 3) */
EXP : [256][8]
EXP = [
    0x01, 0x03, 0x05, 0x0f, 0x11, 0x33, 0x55, 0xff,
    0x1a, 0x2e, 0x72, 0x96, 0xa1, 0xf8, 0x13, 0x35,
    0x5f, 0xe1, 0x38, 0x48, 0xd8, 0x73, 0x95, 0xa4,
    0xf7, 0x02, 0x06, 0x0a, 0x1e, 0x22, 0x66, 0xaa,
    0xe5, 0x34, 0x5c, 0xe4, 0x37, 0x59, 0xeb, 0x26,
    0x6a, 0xbe, 0xd9, 0x70, 0x90, 0xab, 0xe6, 0x31,
    0x53, 0xf5, 0x04, 0x0c, 0x14, 0x3c, 0x44, 0xcc,
    0x4f, 0xd1, 0x68, 0xb8, 0xd3, 0x6e, 0xb2, 0xcd,
    0x4c, 0xd4, 0x67, 0xa9, 0xe0, 0x3b, 0x4d, 0xd7,
    0x62, 0xa6, 0xf1, 0x08, 0x18, 0x28, 0x78, 0x88,
    0x83, 0x9e, 0xb9, 0xd0, 0x6b, 0xbd, 0xdc, 0x7f,
    0x81, 0x98, 0xb3, 0xce, 0x49, 0xdb, 0x76, 0x9a,
    0xb5, 0xc4, 0x57, 0xf9, 0x10, 0x30, 0x50, 0xf0,
    0x0b, 0x1d, 0x27, 0x69, 0xbb, 0xd6, 0x61, 0xa3,
    0xfe, 0x19, 0x2b, 0x7d, 0x87, 0x92, 0xad, 0xec,
    0x2f, 0x71, 0x93, 0xae, 0xe9, 0x20, 0x60, 0xa0,
    0xfb, 0x16, 0x3a, 0x4e, 0xd2, 0x6d, 0xb7, 0xc2,
    0x5d, 0xe7, 0x32, 0x56, 0xfa, 0x15, 0x3f, 0x41,
    0xc3, 0x5e, 0xe2, 0x3d, 0x47, 0xc9, 0x40, 0xc0,
    0x5b, 0xed, 0x2c, 0x74, 0x9c, 0xbf, 0xda, 0x75,
    0x9f, 0xba, 0xd5, 0x64, 0xac, 0xef, 0x2a, 0x7e,
    0x82, 0x9d, 0xbc, 0xdf, 0x7a, 0x8e, 0x89, 0x80,
    0x9b, 0xb6, 0xc1, 0x58, 0xe8, 0x23, 0x65, 0xaf,
    0xea, 0x25, 0x6f, 0xb1, 0xc8, 0x43, 0xc5, 0x54,
    0xfc, 0x1f, 0x21, 0x63, 0xa5, 0xf4, 0x07, 0x09,
    0x1b, 0x2d, 0x77, 0x99, 0xb0, 0xcb, 0x46, 0xca,
    0x45, 0xcf, 0x4a, 0xde, 0x79, 0x8b, 0x86, 0x91,
    0xa8, 0xe3, 0x3e, 0x42, 0xc6, 0x51, 0xf3, 0x0e,
    0x12, 0x36, 0x5a, 0xee, 0x29, 0x7b, 0x8d, 0x8c,
    0x8f, 0x8a, 0x85, 0x94, 0xa7, 0xf2, 0x0d, 0x17,
    0x39, 0x4b, 0xdd, 0x7c, 0x84, 0x97, 0xa2, 0xfd,
    0x1c, 0x24, 0x6c, 0xb4, 0xc7, 0x52, 0xf6, 0x01
]

GF_ADD : (GF28, GF28) -> GF28
GF_ADD (x, y) = x ^ y

GF_SUB : (GF28, GF28) -> GF28
GF_SUB (x, y) = x ^ y

GF_MUL : (GF28, GF28) -> GF28
GF_MUL (x, y) = z
    where z = if x == 0 then  0
              else if y == 0 then 0
                   else EXP @ ((toInteger (LOG @ x) + toInteger (LOG @ y)) % 255)

GF_POW : (GF28, [8]) -> GF28
GF_POW (n, k) = pows ! 0
    where pows = [1] # [ if bit then GF_MUL (n, sq x)
                                else sq x
                       | x <- pows
                       | bit <- k
                       ]
          sq x = GF_MUL (x, x)

GF_DIV : (GF28, GF28) -> GF28
GF_DIV (x, y) = div x
  where div i = if x == 0 then 0
                else EXP @ (((toInteger(LOG @ x)) - toInteger (LOG @ y)) % 255)

GF_SUM : {n} (fin n) => [n]GF28 -> GF28
GF_SUM ps = accum ! 0
  where accum = [zero] # [ GF_ADD(p, a) | p <- ps | a <- accum ]

GF_PROD : {n} (fin n) => [n]GF28 -> GF28
GF_PROD ps = accum ! 0
  where accum = [1] # [ GF_MUL(p, a) | p <- ps | a <- accum ]

GF_DOT_PROD : {n} (fin n) => ([n]GF28, [n]GF28) -> GF28
GF_DOT_PROD (xs, ys) = GF_SUM [ GF_MUL (x, y) | x <- xs
                                              | y <- ys
                              ]
```

```
GF_VEC_MUL : {n, m} (fin n) => ([n]GF28, [m][n]GF28) -> [m]GF28
GF_VEC_MUL (v, ms) = [ GF_DOT_PROD(v, m) | m <- ms ]

GF_MAT_MUL : {n, m, k} (fin m) => ([n][m]GF28, [m][k]GF28) -> [n][k]GF28
GF_MAT_MUL (xss, yss) = [ GF_VEC_MUL(xs, yss') | xs <- xss ]
        where yss' = transpose yss

// Test test vectors for Polynomial 1 (x^^8 + x^^4 + x^^3 + x + 1)

/*
 * Test vector TV011B_1
 * secret = 74 65 73 74 00
 * random = A8 7B 34 91 B5
 *
 * l = 5
 * m = 2
 * n = 2
 *
 * split1 = DC 1E 47 E5 B5
 * split2 = 3F 93 1B 4D 71
 */

TV011B_TV1_P : [2][2]GF28
TV011B_TV1_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ]
]

TV011B_TV1_SR : [2][5]GF28
TV011B_TV1_SR = [
    [ 0x74, 0x65, 0x73, 0x74, 0x00 ],
    [ 0xA8, 0x7B, 0x34, 0x91, 0xB5 ]
]

TV011B_TV1_SPLITS : [2][5]GF28
TV011B_TV1_SPLITS = [
    [ 0xDC, 0x1E, 0x47, 0xE5, 0xB5 ],
    [ 0x3F, 0x93, 0x1B, 0x4D, 0x71 ]
]

/* test split */
property TV011B_TV1_split_passes_test = [ GF_MAT_MUL(TV011B_TV1_P, TV011B_TV1_SR) ==
TV011B_TV1_SPLITS ] == ~zero

TV011B_TV1_1_2_R : [1][2]GF28
TV011B_TV1_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x02, 0x01)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011B_TV1_1_2_SPLITS : [2][5]GF28
TV011B_TV1_1_2_SPLITS = [
    [ 0xDC, 0x1E, 0x47, 0xE5, 0xB5 ],
    [ 0x3F, 0x93, 0x1B, 0x4D, 0x71 ]
]

TV011B_TV1_SECRET : [1][5]GF28
TV011B_TV1_SECRET = [
    [ 0x74, 0x65, 0x73, 0x74, 0x00 ]
]

/* test recombine */
property TV011B_TV1_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011B_TV1_1_2_R,
TV011B_TV1_1_2_SPLITS) == TV011B_TV1_SECRET ] == ~zero


/*
 * Test vector TV011B_2
 * secret = 53 41 4D 54 43
 * random = 39 5D 39 6C 87
 *
 * l = 5
 * m = 2
 * n = 4
 *
 * split1 = 6A 1C 74 38 C4
 * split2 = 21 FB 3F 8C 56
 * split3 = 18 A6 06 E0 D1
 * split4 = B7 2E A9 FF 69
 */

TV011B_TV2_P : [4][2]GF28
TV011B_TV2_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01) ]
]

TV011B_TV2_SR : [2][5]GF28
TV011B_TV2_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x39, 0x5D, 0x39, 0x6C, 0x87 ]
]

TV011B_TV2_SPLITS : [4][5]GF28
TV011B_TV2_SPLITS = [
    [ 0x6A, 0x1C, 0x74, 0x38, 0xC4 ],
    [ 0x21, 0xFB, 0x3F, 0x8C, 0x56 ],
    [ 0x18, 0xA6, 0x06, 0xE0, 0xD1 ],
    [ 0xB7, 0x2E, 0xA9, 0xFF, 0x69 ]
]

/* test split */
property TV011B_TV2_split_passes_test = [ GF_MAT_MUL(TV011B_TV2_P, TV011B_TV2_SR) ==
TV011B_TV2_SPLITS ] == ~zero

TV011B_TV2_1_2_R : [1][2]GF28
TV011B_TV2_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011B_TV2_1_4_R : [1][2]GF28
TV011B_TV2_1_4_R = [
    [ GF_DIV(0x04, GF_ADD(0x01, 0x04)), GF_DIV(0x01, GF_ADD(0x01, 0x04)) ]
]

TV011B_TV2_3_4_R : [1][2]GF28
TV011B_TV2_3_4_R = [
    [ GF_DIV(0x04, GF_ADD(0x03, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04)) ]
]

TV011B_TV2_1_2_SPLITS : [2][5]GF28
TV011B_TV2_1_2_SPLITS = [
    [ 0x6A, 0x1C, 0x74, 0x38, 0xC4 ],
    [ 0x21, 0xFB, 0x3F, 0x8C, 0x56 ]
]

TV011B_TV2_1_4_SPLITS : [2][5]GF28
TV011B_TV2_1_4_SPLITS = [
    [ 0x6A, 0x1C, 0x74, 0x38, 0xC4 ],
    [ 0xB7, 0x2E, 0xA9, 0xFF, 0x69 ]
]

TV011B_TV2_3_4_SPLITS : [2][5]GF28
TV011B_TV2_3_4_SPLITS = [
    [ 0x18, 0xA6, 0x06, 0xE0, 0xD1 ],
    [ 0xB7, 0x2E, 0xA9, 0xFF, 0x69 ]
]

TV011B_TV2_SECRET : [1][5]GF28
TV011B_TV2_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
```

```
]

/* test recombines */
property TV011B_TV2_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011B_TV2_1_2_R,
TV011B_TV2_1_2_SPLITS) == TV011B_TV2_SECRET ] == ~zero
property TV011B_TV2_recombine_1_4_passes_test = [ GF_MAT_MUL(TV011B_TV2_1_4_R,
TV011B_TV2_1_4_SPLITS) == TV011B_TV2_SECRET ] == ~zero
property TV011B_TV2_recombine_3_4_passes_test = [ GF_MAT_MUL(TV011B_TV2_3_4_R,
TV011B_TV2_3_4_SPLITS) == TV011B_TV2_SECRET ] == ~zero

/*
 * Test vector TV011B_3
 * secret = 53 41 4D 54 43
 * random = 27 3A 1A 28 AB 99 79 BC 06 37
 *
 * l = 5
 * m = 3
 * n = 4
 *
 * split1 = 4E 73 7F 91 72
 * split2 = F5 D5 52 60 93
 * split3 = E8 E7 60 A5 A2
 * split4 = 42 9F 84 9E 06
 */

TV011B_TV3_P : [4][3]GF28
TV011B_TV3_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02) ]
]

TV011B_TV3_SR : [3][5]GF28
TV011B_TV3_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x27, 0x1A, 0xAB, 0x79, 0x06 ],
    [ 0x3A, 0x28, 0x99, 0xBC, 0x37 ]
]

TV011B_TV3_SPLITS : [4][5]GF28
TV011B_TV3_SPLITS = [
    [ 0x4E, 0x73, 0x7F, 0x91, 0x72 ],
    [ 0xF5, 0xD5, 0x52, 0x60, 0x93 ],
    [ 0xE8, 0xE7, 0x60, 0xA5, 0xA2 ],
    [ 0x42, 0x9F, 0x84, 0x9E, 0x06 ]
]

/* test split */
property TV011B_TV3_split_passes_test = [ GF_MAT_MUL(TV011B_TV3_P, TV011B_TV3_SR) ==
TV011B_TV3_SPLITS ] == ~zero

TV011B_TV3_1_2_3_R : [1][3]GF28
TV011B_TV3_1_2_3_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03))]
    ]
]

TV011B_TV3_1_2_4_R : [1][3]GF28
TV011B_TV3_1_2_4_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x02, GF_ADD(0x02, 0x04))]
    ]
]

TV011B_TV3_1_3_4_R : [1][3]GF28
TV011B_TV3_1_3_4_R = [
    [ GF_PROD[GF_DIV(0x03, GF_ADD(0x01, 0x03)), GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011B_TV3_1_2_3_SPLITS : [3][5]GF28
TV011B_TV3_1_2_3_SPLITS = [
    [ 0x4E, 0x73, 0x7F, 0x91, 0x72 ],
    [ 0xF5, 0xD5, 0x52, 0x60, 0x93 ],
    [ 0xE8, 0xE7, 0x60, 0xA5, 0xA2 ]
]

TV011B_TV3_1_2_4_SPLITS : [3][5]GF28
TV011B_TV3_1_2_4_SPLITS = [
    [ 0x4E, 0x73, 0x7F, 0x91, 0x72 ],
    [ 0xF5, 0xD5, 0x52, 0x60, 0x93 ],
    [ 0x42, 0x9F, 0x84, 0x9E, 0x06 ]
]

TV011B_TV3_1_3_4_SPLITS : [3][5]GF28
TV011B_TV3_1_3_4_SPLITS = [
    [ 0x4E, 0x73, 0x7F, 0x91, 0x72 ],
    [ 0xE8, 0xE7, 0x60, 0xA5, 0xA2 ],
    [ 0x42, 0x9F, 0x84, 0x9E, 0x06 ]
]

TV011B_TV3_SECRET : [1][5]GF28
TV011B_TV3_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]

/* test recombines */
property TV011B_TV3_recombine_1_2_3_passes_test = [ GF_MAT_MUL(TV011B_TV3_1_2_3_R,
TV011B_TV3_1_2_3_SPLITS) == TV011B_TV3_SECRET ] == ~zero
property TV011B_TV3_recombine_1_2_4_passes_test = [ GF_MAT_MUL(TV011B_TV3_1_2_4_R,
TV011B_TV3_1_2_4_SPLITS) == TV011B_TV3_SECRET ] == ~zero
property TV011B_TV3_recombine_1_3_4_passes_test = [ GF_MAT_MUL(TV011B_TV3_1_3_4_R,
TV011B_TV3_1_3_4_SPLITS) == TV011B_TV3_SECRET ] == ~zero

/*
 * Test vector TV011B_4
 * secret = 53 41 4D 54 43
 * random = 1A 22 4C 1E E9 76 0A 73 A0 9D 05 77 44 34 67
 *
 * l = 5
 * m = 4
 * n = 4
 *
 * split1 = 27 C0 94 BB 54
 * split2 = B9 69 F9 F4 0E
 * split3 = 7E C7 CD 32 50
 * split4 = AB AF 81 82 8D
 */

TV011B_TV4_P : [4][4]GF28
TV011B_TV4_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02), GF_POW(0x01, 0x03) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02), GF_POW(0x02, 0x03) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02), GF_POW(0x03, 0x03) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02), GF_POW(0x04, 0x03) ]
]

TV011B_TV4_SR : [4][5]GF28
TV011B_TV4_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x1A, 0x1E, 0x0A, 0x9D, 0x44 ],
    [ 0x22, 0xE9, 0x73, 0x05, 0x34 ],
    [ 0x4C, 0x76, 0xA0, 0x77, 0x67 ]
]

TV011B_TV4_SPLITS : [4][5]GF28
```

```
TV011B_TV4_SPLITS = [
    [ 0x27, 0xC0, 0x94, 0xBB, 0x54 ],
    [ 0xB9, 0x69, 0xF9, 0xF4, 0x0E ],
    [ 0x7E, 0xC7, 0xCD, 0x32, 0x50 ],
    [ 0xAB, 0xAF, 0x81, 0x82, 0x8D ]
]

/* test split */
property TV011B_TV4_split_passes_test = [ GF_MAT_MUL(TV011B_TV4_P, TV011B_TV4_SR) ==
TV011B_TV4_SPLITS ] == ~zero

TV011B_TV4_1_2_3_4_R : [1][4]GF28
TV011B_TV4_1_2_3_4_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03)),
GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03)),
GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03)),
GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x02, GF_ADD(0x02, 0x04)),
GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011B_TV4_1_2_3_4_SPLITS : [4][5]GF28
TV011B_TV4_1_2_3_4_SPLITS = [
    [ 0x27, 0xC0, 0x94, 0xBB, 0x54 ],
    [ 0xB9, 0x69, 0xF9, 0xF4, 0x0E ],
    [ 0x7E, 0xC7, 0xCD, 0x32, 0x50 ],
    [ 0xAB, 0xAF, 0x81, 0x82, 0x8D ]
]

TV011B_TV4_SECRET : [1][5]GF28
TV011B_TV4_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]

/* test recombines */
property TV011B_TV4_recombine_1_2_3_4_passes_test = [ GF_MAT_MUL(TV011B_TV4_1_2_3_4_R,
TV011B_TV4_1_2_3_4_SPLITS) == TV011B_TV4_SECRET ] == ~zero

/*
 * Test vector TV011B_5
 * secret = 54 65 73 74 20 44 61 74 61
 * random = 7F B4 E8 58 1E B7 5D C9 45
 *
 * l = 9
 * m = 2
 * n = 9
 *
 * split1 = 2B D1 9B 2C 3E F3 3C BD 24
 * split2 = AA 16 B8 C4 1C 31 DB FD EB
 * split3 = D5 A2 50 9C 02 86 86 34 AE
 * split4 = B3 83 FE 0F 58 AE 0E 7D 6E
 * split5 = CC 37 16 57 46 19 53 B4 2B
 * split6 = 4D F0 35 BF 64 DB B4 F4 E4
 * split7 = 32 44 DD E7 7A 6C E9 3D A1
 * split8 = 81 B2 72 82 D0 8B BF 66 7F
 * split9 = FE 06 9A DA CE 3C E2 AF 3A
 */

TV011B_TV5_P : [9][2]GF28
TV011B_TV5_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01) ],
    [ GF_POW(0x05, 0x00), GF_POW(0x05, 0x01) ],
    [ GF_POW(0x06, 0x00), GF_POW(0x06, 0x01) ],
    [ GF_POW(0x07, 0x00), GF_POW(0x07, 0x01) ],
    [ GF_POW(0x08, 0x00), GF_POW(0x08, 0x01) ],
    [ GF_POW(0x09, 0x00), GF_POW(0x09, 0x01) ]
]

TV011B_TV5_SR : [2][9]GF28
TV011B_TV5_SR = [
    [ 0x54, 0x65, 0x73, 0x74, 0x20, 0x44, 0x61, 0x74, 0x61 ],
    [ 0x7F, 0xB4, 0xE8, 0x58, 0x1E, 0xB7, 0x5D, 0xC9, 0x45 ]
]

TV011B_TV5_SPLITS : [9][9]GF28
TV011B_TV5_SPLITS = [
    [ 0x2B, 0xD1, 0x9B, 0x2C, 0x3E, 0xF3, 0x3C, 0xBD, 0x24 ],
    [ 0xAA, 0x16, 0xB8, 0xC4, 0x1C, 0x31, 0xDB, 0xFD, 0xEB ],
    [ 0xD5, 0xA2, 0x50, 0x9C, 0x02, 0x86, 0x86, 0x34, 0xAE ],
    [ 0xB3, 0x83, 0xFE, 0x0F, 0x58, 0xAE, 0x0E, 0x7D, 0x6E ],
    [ 0xCC, 0x37, 0x16, 0x57, 0x46, 0x19, 0x53, 0xB4, 0x2B ],
    [ 0x4D, 0xF0, 0x35, 0xBF, 0x64, 0xDB, 0xB4, 0xF4, 0xE4 ],
    [ 0x32, 0x44, 0xDD, 0xE7, 0x7A, 0x6C, 0xE9, 0x3D, 0xA1 ],
    [ 0x81, 0xB2, 0x72, 0x82, 0xD0, 0x8B, 0xBF, 0x66, 0x7F ],
    [ 0xFE, 0x06, 0x9A, 0xDA, 0xCE, 0x3C, 0xE2, 0xAF, 0x3A ]
]

/* test split */
property TV011B_TV5_split_passes_test = [ GF_MAT_MUL(TV011B_TV5_P, TV011B_TV5_SR) ==
TV011B_TV5_SPLITS ] == ~zero

TV011B_TV5_1_2_R : [1][2]GF28
TV011B_TV5_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011B_TV5_8_9_R : [1][2]GF28
TV011B_TV5_8_9_R = [
    [ GF_DIV(0x09, GF_ADD(0x08, 0x09)), GF_DIV(0x08, GF_ADD(0x08, 0x09)) ]
]

TV011B_TV5_1_2_SPLITS : [2][9]GF28
TV011B_TV5_1_2_SPLITS = [
    [ 0x2B, 0xD1, 0x9B, 0x2C, 0x3E, 0xF3, 0x3C, 0xBD, 0x24 ],
    [ 0xAA, 0x16, 0xB8, 0xC4, 0x1C, 0x31, 0xDB, 0xFD, 0xEB ]
]

TV011B_TV5_8_9_SPLITS : [2][9]GF28
TV011B_TV5_8_9_SPLITS = [
    [ 0x81, 0xB2, 0x72, 0x82, 0xD0, 0x8B, 0xBF, 0x66, 0x7F ],
    [ 0xFE, 0x06, 0x9A, 0xDA, 0xCE, 0x3C, 0xE2, 0xAF, 0x3A ]
]

TV011B_TV5_SECRET : [1][9]GF28
TV011B_TV5_SECRET = [
    [ 0x54, 0x65, 0x73, 0x74, 0x20, 0x44, 0x61, 0x74, 0x61 ]
]

/* test recombines */
property TV011B_TV5_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011B_TV5_1_2_R,
TV011B_TV5_1_2_SPLITS) == TV011B_TV5_SECRET ] == ~zero
property TV011B_TV5_recombine_8_9_passes_test = [ GF_MAT_MUL(TV011B_TV5_8_9_R,
TV011B_TV5_8_9_SPLITS) == TV011B_TV5_SECRET ] == ~zero

/*
 * Test vector TV011B_6
 * secret = 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
 * random = EC 96 74 05 40 B3 E1 FC 9A 91 4F 6E 5F 7C CA 51 DB 72 32 02 C9 B8 81 00 4F
66 A2 80 71 97
 *
 * l = 15
 * m = 3
 * n = 5
 *
 * split1 = 7B 73 F0 19 0E 27 24 93 A0 3A 7A 8D 24 2C E9
 * split2 = AC FE 79 00 58 3B 52 D8 77 66 54 15 10 67 87
 * split3 = D6 8F 8A 1D 53 1A 71 43 DE 56 25 94 39 45 61
 * split4 = 3F 99 DD F4 88 9B E1 6A 29 E2 77 3E 10 68 63
 * split5 = 45 E8 2E E9 83 BA C2 F1 80 D2 06 BF 39 4A 85
 */

TV011B_TV6_P : [5][3]GF28
TV011B_TV6_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02) ],
    [ GF_POW(0x05, 0x00), GF_POW(0x05, 0x01), GF_POW(0x05, 0x02) ]
]

TV011B_TV6_SR : [3][15]GF28
TV011B_TV6_SR = [
    [ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D,
0x0E, 0x0F ],
    [ 0xEC, 0x74, 0x40, 0xE1, 0x9A, 0x4F, 0x5F, 0xCA, 0xDB, 0x32, 0xC9, 0x81, 0x4F,
0xA2, 0x71 ],
    [ 0x96, 0x05, 0xB3, 0xFC, 0x91, 0x6E, 0x7C, 0x51, 0x72, 0x02, 0xB8, 0x00, 0x66,
0x80, 0x97 ]
]

TV011B_TV6_SPLITS : [5][15]GF28
TV011B_TV6_SPLITS = [
    [ 0x7B, 0x73, 0xF0, 0x19, 0x0E, 0x27, 0x24, 0x93, 0xA0, 0x3A, 0x7A, 0x8D, 0x24,
0x2C, 0xE9 ],
    [ 0xAC, 0xFE, 0x79, 0x00, 0x58, 0x3B, 0x52, 0xD8, 0x77, 0x66, 0x54, 0x15, 0x10,
0x67, 0x87 ],
    [ 0xD6, 0x8F, 0x8A, 0x1D, 0x53, 0x1A, 0x71, 0x43, 0xDE, 0x56, 0x25, 0x94, 0x39,
0x45, 0x61 ],
    [ 0x3F, 0x99, 0xDD, 0xF4, 0x88, 0x9B, 0xE1, 0x6A, 0x29, 0xE2, 0x77, 0x3E, 0x10,
0x68, 0x63 ],
    [ 0x45, 0xE8, 0x2E, 0xE9, 0x83, 0xBA, 0xC2, 0xF1, 0x80, 0xD2, 0x06, 0xBF, 0x39,
0x4A, 0x85 ]
]

/* test split */
property TV011B_TV6_split_passes_test = [ GF_MAT_MUL(TV011B_TV6_P, TV011B_TV6_SR) ==
TV011B_TV6_SPLITS ] == ~zero

TV011B_TV6_1_2_3_R : [1][3]GF28
TV011B_TV6_1_2_3_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03))]
    ]
]

TV011B_TV6_2_3_4_R : [1][3]GF28
TV011B_TV6_2_3_4_R = [
    [ GF_PROD[GF_DIV(0x03, GF_ADD(0x02, 0x03)), GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x02, GF_ADD(0x02, 0x03)), GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x02, GF_ADD(0x02, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011B_TV6_1_2_3_SPLITS : [3][15]GF28
TV011B_TV6_1_2_3_SPLITS = [
    [ 0x7B, 0x73, 0xF0, 0x19, 0x0E, 0x27, 0x24, 0x93, 0xA0, 0x3A, 0x7A, 0x8D, 0x24,
0x2C, 0xE9 ],
    [ 0xAC, 0xFE, 0x79, 0x00, 0x58, 0x3B, 0x52, 0xD8, 0x77, 0x66, 0x54, 0x15, 0x10,
0x67, 0x87 ],
    [ 0xD6, 0x8F, 0x8A, 0x1D, 0x53, 0x1A, 0x71, 0x43, 0xDE, 0x56, 0x25, 0x94, 0x39,
0x45, 0x61 ]
]

TV011B_TV6_2_3_4_SPLITS : [3][15]GF28
TV011B_TV6_2_3_4_SPLITS = [
    [ 0xAC, 0xFE, 0x79, 0x00, 0x58, 0x3B, 0x52, 0xD8, 0x77, 0x66, 0x54, 0x15, 0x10,
0x67, 0x87 ],
    [ 0xD6, 0x8F, 0x8A, 0x1D, 0x53, 0x1A, 0x71, 0x43, 0xDE, 0x56, 0x25, 0x94, 0x39,
0x45, 0x61 ],
    [ 0x3F, 0x99, 0xDD, 0xF4, 0x88, 0x9B, 0xE1, 0x6A, 0x29, 0xE2, 0x77, 0x3E, 0x10,
0x68, 0x63 ]
]

TV011B_TV6_SECRET : [1][15]GF28
TV011B_TV6_SECRET = [
    [ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D,
0x0E, 0x0F ]
]

/* test recombines */
property TV011B_TV6_recombine_1_2_3_passes_test = [ GF_MAT_MUL(TV011B_TV6_1_2_3_R,
TV011B_TV6_1_2_3_SPLITS) == TV011B_TV6_SECRET ] == ~zero
property TV011B_TV6_recombine_2_3_4_passes_test = [ GF_MAT_MUL(TV011B_TV6_2_3_4_R,
TV011B_TV6_2_3_4_SPLITS) == TV011B_TV6_SECRET ] == ~zero
```

E.1.2 “Native” GF(256)-based implementation

```
module secret_share_011B_gf28 where

type GF28 = [8]

irreducible = <| x^^8 + x^^4 + x^^3 + x + 1 |>

GF_ADD : (GF28, GF28) -> GF28
GF_ADD (x, y) = x ^ y

GF_SUB : (GF28, GF28) -> GF28
GF_SUB (x, y) = x ^ y

GF_MUL : (GF28, GF28) -> GF28
GF_MUL (x, y) = pmod(pmult x y) irreducible

GF_POW : (GF28, [8]) -> GF28
GF_POW (n, k) = pow k
  where   sq x  = GF_MUL (x, x)
          odd x = x ! 0
          pow i = if i == 0 then 1
                  else if odd i
                       then GF_MUL(n, sq (pow (i >> 1)))
                       else sq (pow (i >> 1))

GF_INV : GF28 -> GF28
GF_INV x = GF_POW (x, 254)

GF_DIV : (GF28, GF28) -> GF28
GF_DIV (x, y) = div x
  where   inv yi  = GF_INV (yi)
          div i = if x == 0 then 0
                  else GF_MUL(x, inv y)

GF_SUM : {n} (fin n) => [n]GF28 -> GF28
GF_SUM ps = accum ! 0
  where accum = [zero] # [ GF_ADD(p, a) | p <- ps | a <- accum ]

GF_PROD : {n} (fin n) => [n]GF28 -> GF28
GF_PROD ps = accum ! 0
  where accum = [1] # [ GF_MUL(p, a) | p <- ps | a <- accum ]

GF_DOT_PROD : {n} (fin n) => ([n]GF28, [n]GF28) -> GF28
GF_DOT_PROD (xs, ys) = GF_SUM [ GF_MUL (x, y) | x <- xs
                                              | y <- ys
                              ]
```

```
GF_VEC_MUL : {n, m} (fin n) => ([n]GF28, [m][n]GF28) -> [m]GF28
GF_VEC_MUL (v, ms) = [ GF_DOT_PROD(v, m) | m <- ms ]

GF_MAT_MUL : {n, m, k} (fin m) => ([n][m]GF28, [m][k]GF28) -> [n][k]GF28
GF_MAT_MUL (xss, yss) = [ GF_VEC_MUL(xs, yss') | xs <- xss ]
        where yss' = transpose yss

// Test test vectors for Polynomial 1 (x^^8 + x^^4 + x^^3 + x + 1)

/*
 * Test vector TV011B_1
 * secret = 74 65 73 74 00
 * random = A8 7B 34 91 B5
 *
 * l = 5
 * m = 2
 * n = 2
 *
 * split1 = DC 1E 47 E5 B5
 * split2 = 3F 93 1B 4D 71
 */

TV011B_TV1_P : [2][2]GF28
TV011B_TV1_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ]
]

TV011B_TV1_SR : [2][5]GF28
TV011B_TV1_SR = [
    [ 0x74, 0x65, 0x73, 0x74, 0x00 ],
    [ 0xA8, 0x7B, 0x34, 0x91, 0xB5 ]
]

TV011B_TV1_SPLITS : [2][5]GF28
TV011B_TV1_SPLITS = [
    [ 0xDC, 0x1E, 0x47, 0xE5, 0xB5 ],
    [ 0x3F, 0x93, 0x1B, 0x4D, 0x71 ]
]

/* test split */
property TV011B_TV1_split_passes_test = [ GF_MAT_MUL(TV011B_TV1_P, TV011B_TV1_SR) ==
TV011B_TV1_SPLITS ] == ~zero

TV011B_TV1_1_2_R : [1][2]GF28
TV011B_TV1_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x02, 0x01)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011B_TV1_1_2_SPLITS : [2][5]GF28
TV011B_TV1_1_2_SPLITS = [
    [ 0xDC, 0x1E, 0x47, 0xE5, 0xB5 ],
    [ 0x3F, 0x93, 0x1B, 0x4D, 0x71 ]
]

TV011B_TV1_SECRET : [1][5]GF28
TV011B_TV1_SECRET = [
    [ 0x74, 0x65, 0x73, 0x74, 0x00 ]
]

/* test recombine */
property TV011B_TV1_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011B_TV1_1_2_R,
TV011B_TV1_1_2_SPLITS) == TV011B_TV1_SECRET ] == ~zero

/*
 * Test vector TV011B_2
 * secret = 53 41 4D 54 43
 * random = 39 5D 39 6C 87
 *
 * l = 5
 * m = 2
 * n = 4
 *
 * split1 = 6A 1C 74 38 C4
 * split2 = 21 FB 3F 8C 56
 * split3 = 18 A6 06 E0 D1
 * split4 = B7 2E A9 FF 69
 */

TV011B_TV2_P : [4][2]GF28
TV011B_TV2_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01) ]
]

TV011B_TV2_SR : [2][5]GF28
TV011B_TV2_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x39, 0x5D, 0x39, 0x6C, 0x87 ]
]

TV011B_TV2_SPLITS : [4][5]GF28
TV011B_TV2_SPLITS = [
    [ 0x6A, 0x1C, 0x74, 0x38, 0xC4 ],
    [ 0x21, 0xFB, 0x3F, 0x8C, 0x56 ],
    [ 0x18, 0xA6, 0x06, 0xE0, 0xD1 ],
    [ 0xB7, 0x2E, 0xA9, 0xFF, 0x69 ]
]

/* test split */
property TV011B_TV2_split_passes_test = [ GF_MAT_MUL(TV011B_TV2_P, TV011B_TV2_SR) ==
TV011B_TV2_SPLITS ] == ~zero

TV011B_TV2_1_2_R : [1][2]GF28
TV011B_TV2_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011B_TV2_1_4_R : [1][2]GF28
TV011B_TV2_1_4_R = [
    [ GF_DIV(0x04, GF_ADD(0x01, 0x04)), GF_DIV(0x01, GF_ADD(0x01, 0x04)) ]
]

TV011B_TV2_3_4_R : [1][2]GF28
TV011B_TV2_3_4_R = [
    [ GF_DIV(0x04, GF_ADD(0x03, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04)) ]
]

TV011B_TV2_1_2_SPLITS : [2][5]GF28
TV011B_TV2_1_2_SPLITS = [
    [ 0x6A, 0x1C, 0x74, 0x38, 0xC4 ],
    [ 0x21, 0xFB, 0x3F, 0x8C, 0x56 ]
]

TV011B_TV2_1_4_SPLITS : [2][5]GF28
TV011B_TV2_1_4_SPLITS = [
    [ 0x6A, 0x1C, 0x74, 0x38, 0xC4 ],
    [ 0xB7, 0x2E, 0xA9, 0xFF, 0x69 ]
]

TV011B_TV2_3_4_SPLITS : [2][5]GF28
TV011B_TV2_3_4_SPLITS = [
    [ 0x18, 0xA6, 0x06, 0xE0, 0xD1 ],
    [ 0xB7, 0x2E, 0xA9, 0xFF, 0x69 ]
]

TV011B_TV2_SECRET : [1][5]GF28
TV011B_TV2_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]
```

```
/* test recombines */
property TV011B_TV2_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011B_TV2_1_2_R,
TV011B_TV2_1_2_SPLITS) == TV011B_TV2_SECRET ] == ~zero
property TV011B_TV2_recombine_1_4_passes_test = [ GF_MAT_MUL(TV011B_TV2_1_4_R,
TV011B_TV2_1_4_SPLITS) == TV011B_TV2_SECRET ] == ~zero
property TV011B_TV2_recombine_3_4_passes_test = [ GF_MAT_MUL(TV011B_TV2_3_4_R,
TV011B_TV2_3_4_SPLITS) == TV011B_TV2_SECRET ] == ~zero

/*
 * Test vector TV011B_3
 * secret = 53 41 4D 54 43
 * random = 27 3A 1A 28 AB 99 79 BC 06 37
 *
 * l = 5
 * m = 3
 * n = 4
 *
 * split1 = 4E 73 7F 91 72
 * split2 = F5 D5 52 60 93
 * split3 = E8 E7 60 A5 A2
 * split4 = 42 9F 84 9E 06
 */

TV011B_TV3_P : [4][3]GF28
TV011B_TV3_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02) ]
]

TV011B_TV3_SR : [3][5]GF28
TV011B_TV3_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x27, 0x1A, 0xAB, 0x79, 0x06 ],
    [ 0x3A, 0x28, 0x99, 0xBC, 0x37 ]
]

TV011B_TV3_SPLITS : [4][5]GF28
TV011B_TV3_SPLITS = [
    [ 0x4E, 0x73, 0x7F, 0x91, 0x72 ],
    [ 0xF5, 0xD5, 0x52, 0x60, 0x93 ],
    [ 0xE8, 0xE7, 0x60, 0xA5, 0xA2 ],
    [ 0x42, 0x9F, 0x84, 0x9E, 0x06 ]
]

/* test split */
property TV011B_TV3_split_passes_test = [ GF_MAT_MUL(TV011B_TV3_P, TV011B_TV3_SR) ==
TV011B_TV3_SPLITS ] == ~zero

TV011B_TV3_1_2_3_R : [1][3]GF28
TV011B_TV3_1_2_3_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03))]
    ]
]

TV011B_TV3_1_2_4_R : [1][3]GF28
TV011B_TV3_1_2_4_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x02, GF_ADD(0x02, 0x04))]
    ]
]

TV011B_TV3_1_3_4_R : [1][3]GF28
TV011B_TV3_1_3_4_R = [
    [ GF_PROD[GF_DIV(0x03, GF_ADD(0x01, 0x03)), GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011B_TV3_1_2_3_SPLITS : [3][5]GF28
TV011B_TV3_1_2_3_SPLITS = [
    [ 0x4E, 0x73, 0x7F, 0x91, 0x72 ],
    [ 0xF5, 0xD5, 0x52, 0x60, 0x93 ],
    [ 0xE8, 0xE7, 0x60, 0xA5, 0xA2 ]
]

TV011B_TV3_1_2_4_SPLITS : [3][5]GF28
TV011B_TV3_1_2_4_SPLITS = [
    [ 0x4E, 0x73, 0x7F, 0x91, 0x72 ],
    [ 0xF5, 0xD5, 0x52, 0x60, 0x93 ],
    [ 0x42, 0x9F, 0x84, 0x9E, 0x06 ]
]

TV011B_TV3_1_3_4_SPLITS : [3][5]GF28
TV011B_TV3_1_3_4_SPLITS = [
    [ 0x4E, 0x73, 0x7F, 0x91, 0x72 ],
    [ 0xE8, 0xE7, 0x60, 0xA5, 0xA2 ],
    [ 0x42, 0x9F, 0x84, 0x9E, 0x06 ]
]

TV011B_TV3_SECRET : [1][5]GF28
TV011B_TV3_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]

/* test recombines */
property TV011B_TV3_recombine_1_2_3_passes_test = [ GF_MAT_MUL(TV011B_TV3_1_2_3_R,
TV011B_TV3_1_2_3_SPLITS) == TV011B_TV3_SECRET ] == ~zero
property TV011B_TV3_recombine_1_2_4_passes_test = [ GF_MAT_MUL(TV011B_TV3_1_2_4_R,
TV011B_TV3_1_2_4_SPLITS) == TV011B_TV3_SECRET ] == ~zero
property TV011B_TV3_recombine_1_3_4_passes_test = [ GF_MAT_MUL(TV011B_TV3_1_3_4_R,
TV011B_TV3_1_3_4_SPLITS) == TV011B_TV3_SECRET ] == ~zero

/*
 * Test vector TV011B_4
 * secret = 53 41 4D 54 43
 * random = 1A 22 4C 1E E9 76 0A 73 A0 9D 05 77 44 34 67
 *
 * l = 5
 * m = 4
 * n = 4
 *
 * split1 = 27 C0 94 BB 54
 * split2 = B9 69 F9 F4 0E
 * split3 = 7E C7 CD 32 50
 * split4 = AB AF 81 82 8D
 */

TV011B_TV4_P : [4][4]GF28
TV011B_TV4_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02), GF_POW(0x01, 0x03) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02), GF_POW(0x02, 0x03) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02), GF_POW(0x03, 0x03) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02), GF_POW(0x04, 0x03) ]
]

TV011B_TV4_SR : [4][5]GF28
TV011B_TV4_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x1A, 0x1E, 0x0A, 0x9D, 0x44 ],
    [ 0x22, 0xE9, 0x73, 0x05, 0x34 ],
    [ 0x4C, 0x76, 0xA0, 0x77, 0x67 ]
]

TV011B_TV4_SPLITS : [4][5]GF28
TV011B_TV4_SPLITS = [
    [ 0x27, 0xC0, 0x94, 0xBB, 0x54 ],
    [ 0xB9, 0x69, 0xF9, 0xF4, 0x0E ],
    [ 0x7E, 0xC7, 0xCD, 0x32, 0x50 ],
    [ 0xAB, 0xAF, 0x81, 0x82, 0x8D ]
]

/* test split */
property TV011B_TV4_split_passes_test = [ GF_MAT_MUL(TV011B_TV4_P, TV011B_TV4_SR) ==
TV011B_TV4_SPLITS ] == ~zero

TV011B_TV4_1_2_3_4_R : [1][4]GF28
TV011B_TV4_1_2_3_4_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03)),
GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03)),
GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03)),
GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x02, GF_ADD(0x02, 0x04)),
GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011B_TV4_1_2_3_4_SPLITS : [4][5]GF28
TV011B_TV4_1_2_3_4_SPLITS = [
    [ 0x27, 0xC0, 0x94, 0xBB, 0x54 ],
    [ 0xB9, 0x69, 0xF9, 0xF4, 0x0E ],
    [ 0x7E, 0xC7, 0xCD, 0x32, 0x50 ],
    [ 0xAB, 0xAF, 0x81, 0x82, 0x8D ]
]

TV011B_TV4_SECRET : [1][5]GF28
TV011B_TV4_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]

/* test recombines */
property TV011B_TV4_recombine_1_2_3_4_passes_test = [ GF_MAT_MUL(TV011B_TV4_1_2_3_4_R,
TV011B_TV4_1_2_3_4_SPLITS) == TV011B_TV4_SECRET ] == ~zero

/*
 * Test vector TV011B_5
 * secret = 54 65 73 74 20 44 61 74 61
 * random = 7F B4 E8 58 1E B7 5D C9 45
 *
 * l = 9
 * m = 2
 * n = 9
 *
 * split1 = 2B D1 9B 2C 3E F3 3C BD 24
 * split2 = AA 16 B8 C4 1C 31 DB FD EB
 * split3 = D5 A2 50 9C 02 86 86 34 AE
 * split4 = B3 83 FE 0F 58 AE 0E 7D 6E
 * split5 = CC 37 16 57 46 19 53 B4 2B
 * split6 = 4D F0 35 BF 64 DB B4 F4 E4
 * split7 = 32 44 DD E7 7A 6C E9 3D A1
 * split8 = 81 B2 72 82 D0 8B BF 66 7F
 * split9 = FE 06 9A DA CE 3C E2 AF 3A
 */

TV011B_TV5_P : [9][2]GF28
TV011B_TV5_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01) ],
    [ GF_POW(0x05, 0x00), GF_POW(0x05, 0x01) ],
    [ GF_POW(0x06, 0x00), GF_POW(0x06, 0x01) ],
    [ GF_POW(0x07, 0x00), GF_POW(0x07, 0x01) ],
    [ GF_POW(0x08, 0x00), GF_POW(0x08, 0x01) ],
    [ GF_POW(0x09, 0x00), GF_POW(0x09, 0x01) ]
```

```
]

TV011B_TV5_SR : [2][9]GF28
TV011B_TV5_SR = [
    [ 0x54, 0x65, 0x73, 0x74, 0x20, 0x44, 0x61, 0x74, 0x61 ],
    [ 0x7F, 0xB4, 0xE8, 0x58, 0x1E, 0xB7, 0x5D, 0xC9, 0x45 ]
]

TV011B_TV5_SPLITS : [9][9]GF28
TV011B_TV5_SPLITS = [
    [ 0x2B, 0xD1, 0x9B, 0x2C, 0x3E, 0xF3, 0x3C, 0xBD, 0x24 ],
    [ 0xAA, 0x16, 0xB8, 0xC4, 0x1C, 0x31, 0xDB, 0xFD, 0xEB ],
    [ 0xD5, 0xA2, 0x50, 0x9C, 0x02, 0x86, 0x86, 0x34, 0xAE ],
    [ 0xB3, 0x83, 0xFE, 0x0F, 0x58, 0xAE, 0x0E, 0x7D, 0x6E ],
    [ 0xCC, 0x37, 0x16, 0x57, 0x46, 0x19, 0x53, 0xB4, 0x2B ],
    [ 0x4D, 0xF0, 0x35, 0xBF, 0x64, 0xDB, 0xB4, 0xF4, 0xE4 ],
    [ 0x32, 0x44, 0xDD, 0xE7, 0x7A, 0x6C, 0xE9, 0x3D, 0xA1 ],
    [ 0x81, 0xB2, 0x72, 0x82, 0xD0, 0x8B, 0xBF, 0x66, 0x7F ],
    [ 0xFE, 0x06, 0x9A, 0xDA, 0xCE, 0x3C, 0xE2, 0xAF, 0x3A ]
]

/* test split */
property TV011B_TV5_split_passes_test = [ GF_MAT_MUL(TV011B_TV5_P, TV011B_TV5_SR) ==
TV011B_TV5_SPLITS ] == ~zero

TV011B_TV5_1_2_R : [1][2]GF28
TV011B_TV5_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011B_TV5_8_9_R : [1][2]GF28
TV011B_TV5_8_9_R = [
    [ GF_DIV(0x09, GF_ADD(0x08, 0x09)), GF_DIV(0x08, GF_ADD(0x08, 0x09)) ]
]

TV011B_TV5_1_2_SPLITS : [2][9]GF28
TV011B_TV5_1_2_SPLITS = [
    [ 0x2B, 0xD1, 0x9B, 0x2C, 0x3E, 0xF3, 0x3C, 0xBD, 0x24 ],
    [ 0xAA, 0x16, 0xB8, 0xC4, 0x1C, 0x31, 0xDB, 0xFD, 0xEB ]
]

TV011B_TV5_8_9_SPLITS : [2][9]GF28
TV011B_TV5_8_9_SPLITS = [
    [ 0x81, 0xB2, 0x72, 0x82, 0xD0, 0x8B, 0xBF, 0x66, 0x7F ],
    [ 0xFE, 0x06, 0x9A, 0xDA, 0xCE, 0x3C, 0xE2, 0xAF, 0x3A ]
]

TV011B_TV5_SECRET : [1][9]GF28
TV011B_TV5_SECRET = [
    [ 0x54, 0x65, 0x73, 0x74, 0x20, 0x44, 0x61, 0x74, 0x61 ]
]

/* test recombines */
property TV011B_TV5_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011B_TV5_1_2_R,
TV011B_TV5_1_2_SPLITS) == TV011B_TV5_SECRET ] == ~zero
property TV011B_TV5_recombine_8_9_passes_test = [ GF_MAT_MUL(TV011B_TV5_8_9_R,
TV011B_TV5_8_9_SPLITS) == TV011B_TV5_SECRET ] == ~zero

/*
 * Test vector TV011B_6
 * secret = 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
 * random = EC 96 74 05 40 B3 E1 FC 9A 91 4F 6E 5F 7C CA 51 DB 72 32 02 C9 B8 81 00 4F
66 A2 80 71 97
 *
 * l = 15
 * m = 3
 * n = 5
 *
 * split1 = 7B 73 F0 19 0E 27 24 93 A0 3A 7A 8D 24 2C E9
 * split2 = AC FE 79 00 58 3B 52 D8 77 66 54 15 10 67 87
 * split3 = D6 8F 8A 1D 53 1A 71 43 DE 56 25 94 39 45 61
 * split4 = 3F 99 DD F4 88 9B E1 6A 29 E2 77 3E 10 68 63
 * split5 = 45 E8 2E E9 83 BA C2 F1 80 D2 06 BF 39 4A 85
 */

TV011B_TV6_P : [5][3]GF28
TV011B_TV6_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02) ],
    [ GF_POW(0x05, 0x00), GF_POW(0x05, 0x01), GF_POW(0x05, 0x02) ]
]

TV011B_TV6_SR : [3][15]GF28
TV011B_TV6_SR = [
    [ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D,
0x0E, 0x0F ],
    [ 0xEC, 0x74, 0x40, 0xE1, 0x9A, 0x4F, 0x5F, 0xCA, 0xDB, 0x32, 0xC9, 0x81, 0x4F,
0xA2, 0x71 ],
    [ 0x96, 0x05, 0xB3, 0xFC, 0x91, 0x6E, 0x7C, 0x51, 0x72, 0x02, 0xB8, 0x00, 0x66,
0x80, 0x97 ]
]

TV011B_TV6_SPLITS : [5][15]GF28
TV011B_TV6_SPLITS = [
    [ 0x7B, 0x73, 0xF0, 0x19, 0x0E, 0x27, 0x24, 0x93, 0xA0, 0x3A, 0x7A, 0x8D, 0x24,
0x2C, 0xE9 ],
    [ 0xAC, 0xFE, 0x79, 0x00, 0x58, 0x3B, 0x52, 0xD8, 0x77, 0x66, 0x54, 0x15, 0x10,
0x67, 0x87 ],
    [ 0xD6, 0x8F, 0x8A, 0x1D, 0x53, 0x1A, 0x71, 0x43, 0xDE, 0x56, 0x25, 0x94, 0x39,
0x45, 0x61 ],
    [ 0x3F, 0x99, 0xDD, 0xF4, 0x88, 0x9B, 0xE1, 0x6A, 0x29, 0xE2, 0x77, 0x3E, 0x10,
0x68, 0x63 ],
    [ 0x45, 0xE8, 0x2E, 0xE9, 0x83, 0xBA, 0xC2, 0xF1, 0x80, 0xD2, 0x06, 0xBF, 0x39,
0x4A, 0x85 ]
]

/* test split */
property TV011B_TV6_split_passes_test = [ GF_MAT_MUL(TV011B_TV6_P, TV011B_TV6_SR) ==
TV011B_TV6_SPLITS ] == ~zero

TV011B_TV6_1_2_3_R : [1][3]GF28
TV011B_TV6_1_2_3_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03))]
    ]
]

TV011B_TV6_2_3_4_R : [1][3]GF28
TV011B_TV6_2_3_4_R = [
    [ GF_PROD[GF_DIV(0x03, GF_ADD(0x02, 0x03)), GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x02, GF_ADD(0x02, 0x03)), GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x02, GF_ADD(0x02, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011B_TV6_1_2_3_SPLITS : [3][15]GF28
TV011B_TV6_1_2_3_SPLITS = [
    [ 0x7B, 0x73, 0xF0, 0x19, 0x0E, 0x27, 0x24, 0x93, 0xA0, 0x3A, 0x7A, 0x8D, 0x24,
0x2C, 0xE9 ],
    [ 0xAC, 0xFE, 0x79, 0x00, 0x58, 0x3B, 0x52, 0xD8, 0x77, 0x66, 0x54, 0x15, 0x10,
0x67, 0x87 ],
    [ 0xD6, 0x8F, 0x8A, 0x1D, 0x53, 0x1A, 0x71, 0x43, 0xDE, 0x56, 0x25, 0x94, 0x39,
0x45, 0x61 ]
]

TV011B_TV6_2_3_4_SPLITS : [3][15]GF28
TV011B_TV6_2_3_4_SPLITS = [
    [ 0xAC, 0xFE, 0x79, 0x00, 0x58, 0x3B, 0x52, 0xD8, 0x77, 0x66, 0x54, 0x15, 0x10,
0x67, 0x87 ],
    [ 0xD6, 0x8F, 0x8A, 0x1D, 0x53, 0x1A, 0x71, 0x43, 0xDE, 0x56, 0x25, 0x94, 0x39,
0x45, 0x61 ],
    [ 0x3F, 0x99, 0xDD, 0xF4, 0x88, 0x9B, 0xE1, 0x6A, 0x29, 0xE2, 0x77, 0x3E, 0x10,
0x68, 0x63 ]
]

TV011B_TV6_SECRET : [1][15]GF28
TV011B_TV6_SECRET = [
    [ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D,
0x0E, 0x0F ]
]

/* test recombines */
property TV011B_TV6_recombine_1_2_3_passes_test = [ GF_MAT_MUL(TV011B_TV6_1_2_3_R,
TV011B_TV6_1_2_3_SPLITS) == TV011B_TV6_SECRET ] == ~zero
property TV011B_TV6_recombine_2_3_4_passes_test = [ GF_MAT_MUL(TV011B_TV6_2_3_4_R,
TV011B_TV6_2_3_4_SPLITS) == TV011B_TV6_SECRET ] == ~zero
```

E.2 Polynomial

Cryptol code for creating splits, checking the splits, re-combining splits, and checking the recombinations appears in the following subsections.

E.2.1 Table-based implementation

```
module secret_share_011D_table where

type GF28 = [8]

/* given an alpha^j, where alpha = mimimum primitive element (x = 2), return j */
LOG : [256][8]
LOG = [
    0xff, 0x00, 0x01, 0x19, 0x02, 0x32, 0x1a, 0xc6,
    0x03, 0xdf, 0x33, 0xee, 0x1b, 0x68, 0xc7, 0x4b,
    0x04, 0x64, 0xe0, 0x0e, 0x34, 0x8d, 0xef, 0x81,
    0x1c, 0xc1, 0x69, 0xf8, 0xc8, 0x08, 0x4c, 0x71,
    0x05, 0x8a, 0x65, 0x2f, 0xe1, 0x24, 0x0f, 0x21,
    0x35, 0x93, 0x8e, 0xda, 0xf0, 0x12, 0x82, 0x45,
    0x1d, 0xb5, 0xc2, 0x7d, 0x6a, 0x27, 0xf9, 0xb9,
    0xc9, 0x9a, 0x09, 0x78, 0x4d, 0xe4, 0x72, 0xa6,
    0x06, 0xbf, 0x8b, 0x62, 0x66, 0xdd, 0x30, 0xfd,
    0xe2, 0x98, 0x25, 0xb3, 0x10, 0x91, 0x22, 0x88,
    0x36, 0xd0, 0x94, 0xce, 0x8f, 0x96, 0xdb, 0xbd,
    0xf1, 0xd2, 0x13, 0x5c, 0x83, 0x38, 0x46, 0x40,
    0x1e, 0x42, 0xb6, 0xa3, 0xc3, 0x48, 0x7e, 0x6e,
    0x6b, 0x3a, 0x28, 0x54, 0xfa, 0x85, 0xba, 0x3d,
    0xca, 0x5e, 0x9b, 0x9f, 0x0a, 0x15, 0x79, 0x2b,
    0x4e, 0xd4, 0xe5, 0xac, 0x73, 0xf3, 0xa7, 0x57,
    0x07, 0x70, 0xc0, 0xf7, 0x8c, 0x80, 0x63, 0x0d,
    0x67, 0x4a, 0xde, 0xed, 0x31, 0xc5, 0xfe, 0x18,
    0xe3, 0xa5, 0x99, 0x77, 0x26, 0xb8, 0xb4, 0x7c,
    0x11, 0x44, 0x92, 0xd9, 0x23, 0x20, 0x89, 0x2e,
    0x37, 0x3f, 0xd1, 0x5b, 0x95, 0xbc, 0xcf, 0xcd,
    0x90, 0x87, 0x97, 0xb2, 0xdc, 0xfc, 0xbe, 0x61,
    0xf2, 0x56, 0xd3, 0xab, 0x14, 0x2a, 0x5d, 0x9e,
    0x84, 0x3c, 0x39, 0x53, 0x47, 0x6d, 0x41, 0xa2,
    0x1f, 0x2d, 0x43, 0xd8, 0xb7, 0x7b, 0xa4, 0x76,
    0xc4, 0x17, 0x49, 0xec, 0x7f, 0x0c, 0x6f, 0xf6,
    0x6c, 0xa1, 0x3b, 0x52, 0x29, 0x9d, 0x55, 0xaa,
    0xfb, 0x60, 0x86, 0xb1, 0xbb, 0xcc, 0x3e, 0x5a,
    0xcb, 0x59, 0x5f, 0xb0, 0x9c, 0xa9, 0xa0, 0x51,
    0x0b, 0xf5, 0x16, 0xeb, 0x7a, 0x75, 0x2c, 0xd7,
    0x4f, 0xae, 0xd5, 0xe9, 0xe6, 0xe7, 0xad, 0xe8,
    0x74, 0xd6, 0xf4, 0xea, 0xa8, 0x50, 0x58, 0xaf
]

/* given a j, return alpha^j, where alpha = mimimum primitive element (x = 2) */
EXP : [256][8]
EXP = [
    0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
    0x1d, 0x3a, 0x74, 0xe8, 0xcd, 0x87, 0x13, 0x26,
    0x4c, 0x98, 0x2d, 0x5a, 0xb4, 0x75, 0xea, 0xc9,
    0x8f, 0x03, 0x06, 0x0c, 0x18, 0x30, 0x60, 0xc0,
    0x9d, 0x27, 0x4e, 0x9c, 0x25, 0x4a, 0x94, 0x35,
    0x6a, 0xd4, 0xb5, 0x77, 0xee, 0xc1, 0x9f, 0x23,
    0x46, 0x8c, 0x05, 0x0a, 0x14, 0x28, 0x50, 0xa0,
    0x5d, 0xba, 0x69, 0xd2, 0xb9, 0x6f, 0xde, 0xa1,
    0x5f, 0xbe, 0x61, 0xc2, 0x99, 0x2f, 0x5e, 0xbc,
    0x65, 0xca, 0x89, 0x0f, 0x1e, 0x3c, 0x78, 0xf0,
    0xfd, 0xe7, 0xd3, 0xbb, 0x6b, 0xd6, 0xb1, 0x7f,
    0xfe, 0xe1, 0xdf, 0xa3, 0x5b, 0xb6, 0x71, 0xe2,
    0xd9, 0xaf, 0x43, 0x86, 0x11, 0x22, 0x44, 0x88,
    0x0d, 0x1a, 0x34, 0x68, 0xd0, 0xbd, 0x67, 0xce,
    0x81, 0x1f, 0x3e, 0x7c, 0xf8, 0xed, 0xc7, 0x93,
    0x3b, 0x76, 0xec, 0xc5, 0x97, 0x33, 0x66, 0xcc,
    0x85, 0x17, 0x2e, 0x5c, 0xb8, 0x6d, 0xda, 0xa9,
    0x4f, 0x9e, 0x21, 0x42, 0x84, 0x15, 0x2a, 0x54,
    0xa8, 0x4d, 0x9a, 0x29, 0x52, 0xa4, 0x55, 0xaa,
    0x49, 0x92, 0x39, 0x72, 0xe4, 0xd5, 0xb7, 0x73,
    0xe6, 0xd1, 0xbf, 0x63, 0xc6, 0x91, 0x3f, 0x7e,
    0xfc, 0xe5, 0xd7, 0xb3, 0x7b, 0xf6, 0xf1, 0xff,
    0xe3, 0xdb, 0xab, 0x4b, 0x96, 0x31, 0x62, 0xc4,
    0x95, 0x37, 0x6e, 0xdc, 0xa5, 0x57, 0xae, 0x41,
    0x82, 0x19, 0x32, 0x64, 0xc8, 0x8d, 0x07, 0x0e,
    0x1c, 0x38, 0x70, 0xe0, 0xdd, 0xa7, 0x53, 0xa6,
    0x51, 0xa2, 0x59, 0xb2, 0x79, 0xf2, 0xf9, 0xef,
    0xc3, 0x9b, 0x2b, 0x56, 0xac, 0x45, 0x8a, 0x09,
    0x12, 0x24, 0x48, 0x90, 0x3d, 0x7a, 0xf4, 0xf5,
    0xf7, 0xf3, 0xfb, 0xeb, 0xcb, 0x8b, 0x0b, 0x16,
    0x2c, 0x58, 0xb0, 0x7d, 0xfa, 0xe9, 0xcf, 0x83,
    0x1b, 0x36, 0x6c, 0xd8, 0xad, 0x47, 0x8e, 0x01
]

GF_ADD : (GF28, GF28) -> GF28
GF_ADD (x, y) = x ^ y

GF_SUB : (GF28, GF28) -> GF28
GF_SUB (x, y) = x ^ y

GF_MUL : (GF28, GF28) -> GF28
GF_MUL (x, y) = z
    where z = if x == 0 then  0
              else if y == 0 then 0
                   else EXP @ ((toInteger (LOG @ x) + toInteger (LOG @ y)) % 255)

GF_POW : (GF28, [8]) -> GF28
GF_POW (n, k) = pows ! 0
    where pows = [1] # [ if bit then GF_MUL (n, sq x)
                                else sq x
                       | x <- pows
                       | bit <- k
                       ]
          sq x = GF_MUL (x, x)

GF_DIV : (GF28, GF28) -> GF28
GF_DIV (x, y) = div x
  where div i = if x == 0 then 0
                else EXP @ (((toInteger(LOG @ x)) - toInteger (LOG @ y)) % 255)

GF_SUM : {n} (fin n) => [n]GF28 -> GF28
GF_SUM ps = accum ! 0
  where accum = [zero] # [ GF_ADD(p, a) | p <- ps | a <- accum ]

GF_PROD : {n} (fin n) => [n]GF28 -> GF28
GF_PROD ps = accum ! 0
  where accum = [1] # [ GF_MUL(p, a) | p <- ps | a <- accum ]

GF_DOT_PROD : {n} (fin n) => ([n]GF28, [n]GF28) -> GF28
GF_DOT_PROD (xs, ys) = GF_SUM [ GF_MUL (x, y) | x <- xs
                                              | y <- ys
                              ]

GF_VEC_MUL : {n, m} (fin n) => ([n]GF28, [m][n]GF28) -> [m]GF28
GF_VEC_MUL (v, ms) = [ GF_DOT_PROD(v, m) | m <- ms ]

GF_MAT_MUL : {n, m, k} (fin m) => ([n][m]GF28, [m][k]GF28) -> [n][k]GF28
GF_MAT_MUL (xss, yss) = [ GF_VEC_MUL(xs, yss') | xs <- xss ]
        where yss' = transpose yss

// Test test vectors for Polynomial 2 (x^^8 + x^^4 + x^^3 + x^^2 + 1)

/*
 * Test vector TV011D_1
 * secret = 74 65 73 74 00
 * random = F3 C2 33 81 F5
 *
 * l = 5
 * m = 2
 * n = 2
 *
 * split1 = 87 A7 40 F5 F5
 * split2 = 8F FC 15 6B F7
 */

TV011D_TV1_P : [2][2]GF28
TV011D_TV1_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ]
]

TV011D_TV1_SR : [2][5]GF28
TV011D_TV1_SR = [
    [ 0x74, 0x65, 0x73, 0x74, 0x00 ],
    [ 0xF3, 0xC2, 0x33, 0x81, 0xF5 ]
]

TV011D_TV1_SPLITS : [2][5]GF28
TV011D_TV1_SPLITS = [
    [ 0x87, 0xA7, 0x40, 0xF5, 0xF5 ],
    [ 0x8F, 0xFC, 0x15, 0x6B, 0xF7 ]
]

/* test split */
property TV011D_TV1_split_passes_test = [ GF_MAT_MUL(TV011D_TV1_P, TV011D_TV1_SR) ==
TV011D_TV1_SPLITS ] == ~zero

TV011D_TV1_1_2_R : [1][2]GF28
TV011D_TV1_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x02, 0x01)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011D_TV1_1_2_SPLITS : [2][5]GF28
TV011D_TV1_1_2_SPLITS = [
    [ 0x87, 0xA7, 0x40, 0xF5, 0xF5 ],
    [ 0x8F, 0xFC, 0x15, 0x6B, 0xF7 ]
]

TV011D_TV1_SECRET : [1][5]GF28
TV011D_TV1_SECRET = [
    [0x74, 0x65, 0x73, 0x74, 0x00 ]
]

/* test recombine */
property TV011D_TV1_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011D_TV1_1_2_R,
TV011D_TV1_1_2_SPLITS) == TV011D_TV1_SECRET ] == ~zero
```

```
/*
 * Test vector TV011D_2
 * secret = 53 41 4D 54 43
 * random = 20 76 08 93 0C
 *
 * l = 5
 * m = 2
 * n = 4
 *
 * split1 = 73 37 45 C7 4F
 * split2 = 13 AD 5D 6F 5B
 * split3 = 33 DB 55 FC 57
 * split4 = D3 84 6D 22 73
 */

TV011D_TV2_P : [4][2]GF28
TV011D_TV2_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01) ]
]

TV011D_TV2_SR : [2][5]GF28
TV011D_TV2_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x20, 0x76, 0x08, 0x93, 0x0C ]
]

TV011D_TV2_SPLITS : [4][5]GF28
TV011D_TV2_SPLITS = [
    [ 0x73, 0x37, 0x45, 0xC7, 0x4F ],
    [ 0x13, 0xAD, 0x5D, 0x6F, 0x5B ],
    [ 0x33, 0xDB, 0x55, 0xFC, 0x57 ],
    [ 0xD3, 0x84, 0x6D, 0x22, 0x73 ]
]

/* test split */
property TV011D_TV2_split_passes_test = [ GF_MAT_MUL(TV011D_TV2_P, TV011D_TV2_SR) ==
TV011D_TV2_SPLITS ] == ~zero

TV011D_TV2_1_2_R : [1][2]GF28
TV011D_TV2_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011D_TV2_1_4_R : [1][2]GF28
TV011D_TV2_1_4_R = [
    [ GF_DIV(0x04, GF_ADD(0x01, 0x04)), GF_DIV(0x01, GF_ADD(0x01, 0x04)) ]
]

TV011D_TV2_3_4_R : [1][2]GF28
TV011D_TV2_3_4_R = [
    [ GF_DIV(0x04, GF_ADD(0x03, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04)) ]
]

TV011D_TV2_1_2_SPLITS : [2][5]GF28
TV011D_TV2_1_2_SPLITS = [
    [ 0x73, 0x37, 0x45, 0xC7, 0x4F ],
    [ 0x13, 0xAD, 0x5D, 0x6F, 0x5B ]
]

TV011D_TV2_1_4_SPLITS : [2][5]GF28
TV011D_TV2_1_4_SPLITS = [
    [ 0x73, 0x37, 0x45, 0xC7, 0x4F ],
    [ 0xD3, 0x84, 0x6D, 0x22, 0x73 ]
]

TV011D_TV2_3_4_SPLITS : [2][5]GF28
TV011D_TV2_3_4_SPLITS = [
    [ 0x33, 0xDB, 0x55, 0xFC, 0x57 ],
    [ 0xD3, 0x84, 0x6D, 0x22, 0x73 ]
]

TV011D_TV2_SECRET : [1][5]GF28
TV011D_TV2_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]

/* test recombines */
property TV011D_TV2_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011D_TV2_1_2_R,
TV011D_TV2_1_2_SPLITS) == TV011D_TV2_SECRET ] == ~zero
property TV011D_TV2_recombine_1_4_passes_test = [ GF_MAT_MUL(TV011D_TV2_1_4_R,
TV011D_TV2_1_4_SPLITS) == TV011D_TV2_SECRET ] == ~zero
property TV011D_TV2_recombine_3_4_passes_test = [ GF_MAT_MUL(TV011D_TV2_3_4_R,
TV011D_TV2_3_4_SPLITS) == TV011D_TV2_SECRET ] == ~zero

/*
 * Test vector TV011D_3
 * secret = 53 41 4D 54 43
 * random = 8C 15 92 62 5C 4A AF 53 41 45
 *
 * l = 5
 * m = 3
 * n = 4
 *
 * split1 = CA B1 5B A8 47
 * split2 = 02 ED C0 46 C8
 * split3 = 9B 1D D6 BA CC
 * split4 = 14 5D F4 8B 7E
 */

TV011D_TV3_P : [4][3]GF28
TV011D_TV3_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02) ]
]

TV011D_TV3_SR : [3][5]GF28
TV011D_TV3_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x8C, 0x92, 0x5C, 0xAF, 0x41 ],
    [ 0x15, 0x62, 0x4A, 0x53, 0x45 ]
]

TV011D_TV3_SPLITS : [4][5]GF28
TV011D_TV3_SPLITS = [
    [ 0xCA, 0xB1, 0x5B, 0xA8, 0x47 ],
    [ 0x02, 0xED, 0xC0, 0x46, 0xC8 ],
    [ 0x9B, 0x1D, 0xD6, 0xBA, 0xCC ],
    [ 0x14, 0x5D, 0xF4, 0x8B, 0x7E ]
]

/* test split */
property TV011D_TV3_split_passes_test = [ GF_MAT_MUL(TV011D_TV3_P, TV011D_TV3_SR) ==
TV011D_TV3_SPLITS ] == ~zero

TV011D_TV3_1_2_3_R : [1][3]GF28
TV011D_TV3_1_2_3_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03))]
    ]
]

TV011D_TV3_1_2_4_R : [1][3]GF28
TV011D_TV3_1_2_4_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x02, GF_ADD(0x02, 0x04))]
    ]
]

TV011D_TV3_1_3_4_R : [1][3]GF28
TV011D_TV3_1_3_4_R = [
    [ GF_PROD[GF_DIV(0x03, GF_ADD(0x01, 0x03)), GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011D_TV3_1_2_3_SPLITS : [3][5]GF28
TV011D_TV3_1_2_3_SPLITS = [
    [ 0xCA, 0xB1, 0x5B, 0xA8, 0x47 ],
    [ 0x02, 0xED, 0xC0, 0x46, 0xC8 ],
    [ 0x9B, 0x1D, 0xD6, 0xBA, 0xCC ]
]

TV011D_TV3_1_2_4_SPLITS : [3][5]GF28
TV011D_TV3_1_2_4_SPLITS = [
    [ 0xCA, 0xB1, 0x5B, 0xA8, 0x47 ],
    [ 0x02, 0xED, 0xC0, 0x46, 0xC8 ],
    [ 0x14, 0x5D, 0xF4, 0x8B, 0x7E ]
]

TV011D_TV3_1_3_4_SPLITS : [3][5]GF28
TV011D_TV3_1_3_4_SPLITS = [
    [ 0xCA, 0xB1, 0x5B, 0xA8, 0x47 ],
    [ 0x9B, 0x1D, 0xD6, 0xBA, 0xCC ],
    [ 0x14, 0x5D, 0xF4, 0x8B, 0x7E ]
]

TV011D_TV3_SECRET : [1][5]GF28
TV011D_TV3_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]

/* test recombines */
property TV011D_TV3_recombine_1_2_3_passes_test = [ GF_MAT_MUL(TV011D_TV3_1_2_3_R,
TV011D_TV3_1_2_3_SPLITS) == TV011D_TV3_SECRET ] == ~zero
property TV011D_TV3_recombine_1_2_4_passes_test = [ GF_MAT_MUL(TV011D_TV3_1_2_4_R,
TV011D_TV3_1_2_4_SPLITS) == TV011D_TV3_SECRET ] == ~zero
property TV011D_TV3_recombine_1_3_4_passes_test = [ GF_MAT_MUL(TV011D_TV3_1_3_4_R,
TV011D_TV3_1_3_4_SPLITS) == TV011D_TV3_SECRET ] == ~zero

/*
 * Test vector TV011D_4
 * secret = 53 41 4D 54 43
 * random = 72 B0 88 3C 96 B9 CB B9 CB B2 82 66 F3 79 FA
 *
 * l = 5
 * m = 4
 * n = 4
 *
 * split1 = 19 52 F4 02 33
 * split2 = 79 FA 0E 08 C2
 * split3 = 24 58 37 17 94
 * split4 = F4 45 A9 D6 07
 */

TV011D_TV4_P : [4][4]GF28
TV011D_TV4_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02), GF_POW(0x01, 0x03) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02), GF_POW(0x02, 0x03) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02), GF_POW(0x03, 0x03) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02), GF_POW(0x04, 0x03) ]
]

TV011D_TV4_SR : [4][5]GF28
TV011D_TV4_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x72, 0x3C, 0xCB, 0xB2, 0xF3 ],
    [ 0xB0, 0x96, 0xB9, 0x82, 0x79 ],
    [ 0x88, 0xB9, 0xCB, 0x66, 0xFA ]
]

TV011D_TV4_SPLITS : [4][5]GF28
TV011D_TV4_SPLITS = [
    [ 0x19, 0x52, 0xF4, 0x02, 0x33 ],
    [ 0x79, 0xFA, 0x0E, 0x08, 0xC2 ],
    [ 0x24, 0x58, 0x37, 0x17, 0x94 ],
    [ 0xF4, 0x45, 0xA9, 0xD6, 0x07 ]
]

/* test split */
property TV011D_TV4_split_passes_test = [ GF_MAT_MUL(TV011D_TV4_P, TV011D_TV4_SR) ==
TV011D_TV4_SPLITS ] == ~zero

TV011D_TV4_1_2_3_4_R : [1][4]GF28
TV011D_TV4_1_2_3_4_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03)),
GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03)),
GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03)),
GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x02, GF_ADD(0x02, 0x04)),
GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011D_TV4_1_2_3_4_SPLITS : [4][5]GF28
TV011D_TV4_1_2_3_4_SPLITS = [
    [ 0x19, 0x52, 0xF4, 0x02, 0x33 ],
    [ 0x79, 0xFA, 0x0E, 0x08, 0xC2 ],
    [ 0x24, 0x58, 0x37, 0x17, 0x94 ],
    [ 0xF4, 0x45, 0xA9, 0xD6, 0x07 ]
]

TV011D_TV4_SECRET : [1][5]GF28
TV011D_TV4_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]

/* test recombines */
property TV011D_TV4_recombine_1_2_3_4_passes_test = [ GF_MAT_MUL(TV011D_TV4_1_2_3_4_R,
TV011D_TV4_1_2_3_4_SPLITS) == TV011D_TV4_SECRET ] == ~zero

/*
 * Test vector TV011D_5
 * secret = 54 65 73 74 20 44 61 74 61
 * random = AF FD 2B 0B FA 34 33 63 9C
 *
 * l = 9
 * m = 2
 * n = 9
 *
 * split1 = FB 98 58 7F DA 70 52 17 FD
 * split2 = 17 82 25 62 C9 2C 07 B2 44
 * split3 = B8 7F 0E 69 33 18 34 D1 D8
 * split4 = D2 B6 DF 58 EF 94 AD E5 2B
 * split5 = 7D 4B F4 53 15 A0 9E 86 B7
 * split6 = 91 51 89 4E 06 FC CB 23 0E
 * split7 = 3E AC A2 45 FC C8 F8 40 92
 * split8 = 45 DE 36 2C A3 F9 E4 4B F5
 * split9 = EA 23 1D 27 59 CD D7 28 69
 */

TV011D_TV5_P : [9][2]GF28
TV011D_TV5_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01) ],
    [ GF_POW(0x05, 0x00), GF_POW(0x05, 0x01) ],
    [ GF_POW(0x06, 0x00), GF_POW(0x06, 0x01) ],
    [ GF_POW(0x07, 0x00), GF_POW(0x07, 0x01) ],
    [ GF_POW(0x08, 0x00), GF_POW(0x08, 0x01) ],
    [ GF_POW(0x09, 0x00), GF_POW(0x09, 0x01) ]
]

TV011D_TV5_SR : [2][9]GF28
TV011D_TV5_SR = [
    [ 0x54, 0x65, 0x73, 0x74, 0x20, 0x44, 0x61, 0x74, 0x61 ],
    [ 0xAF, 0xFD, 0x2B, 0x0B, 0xFA, 0x34, 0x33, 0x63, 0x9C ]
]

TV011D_TV5_SPLITS : [9][9]GF28
TV011D_TV5_SPLITS = [
    [ 0xFB, 0x98, 0x58, 0x7F, 0xDA, 0x70, 0x52, 0x17, 0xFD ],
    [ 0x17, 0x82, 0x25, 0x62, 0xC9, 0x2C, 0x07, 0xB2, 0x44 ],
    [ 0xB8, 0x7F, 0x0E, 0x69, 0x33, 0x18, 0x34, 0xD1, 0xD8 ],
    [ 0xD2, 0xB6, 0xDF, 0x58, 0xEF, 0x94, 0xAD, 0xE5, 0x2B ],
    [ 0x7D, 0x4B, 0xF4, 0x53, 0x15, 0xA0, 0x9E, 0x86, 0xB7 ],
    [ 0x91, 0x51, 0x89, 0x4E, 0x06, 0xFC, 0xCB, 0x23, 0x0E ],
    [ 0x3E, 0xAC, 0xA2, 0x45, 0xFC, 0xC8, 0xF8, 0x40, 0x92 ],
    [ 0x45, 0xDE, 0x36, 0x2C, 0xA3, 0xF9, 0xE4, 0x4B, 0xF5 ],
    [ 0xEA, 0x23, 0x1D, 0x27, 0x59, 0xCD, 0xD7, 0x28, 0x69 ]
]

/* test split */
property TV011D_TV5_split_passes_test = [ GF_MAT_MUL(TV011D_TV5_P, TV011D_TV5_SR) ==
TV011D_TV5_SPLITS ] == ~zero

TV011D_TV5_1_2_R : [1][2]GF28
TV011D_TV5_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011D_TV5_8_9_R : [1][2]GF28
TV011D_TV5_8_9_R = [
    [ GF_DIV(0x09, GF_ADD(0x08, 0x09)), GF_DIV(0x08, GF_ADD(0x08, 0x09)) ]
]

TV011D_TV5_1_2_SPLITS : [2][9]GF28
TV011D_TV5_1_2_SPLITS = [
    [ 0xFB, 0x98, 0x58, 0x7F, 0xDA, 0x70, 0x52, 0x17, 0xFD ],
    [ 0x17, 0x82, 0x25, 0x62, 0xC9, 0x2C, 0x07, 0xB2, 0x44 ]
]

TV011D_TV5_8_9_SPLITS : [2][9]GF28
TV011D_TV5_8_9_SPLITS = [
    [ 0x45, 0xDE, 0x36, 0x2C, 0xA3, 0xF9, 0xE4, 0x4B, 0xF5 ],
    [ 0xEA, 0x23, 0x1D, 0x27, 0x59, 0xCD, 0xD7, 0x28, 0x69 ]
]

TV011D_TV5_SECRET : [1][9]GF28
TV011D_TV5_SECRET = [
    [ 0x54, 0x65, 0x73, 0x74, 0x20, 0x44, 0x61, 0x74, 0x61 ]
]

/* test recombines */
property TV011D_TV5_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011D_TV5_1_2_R,
TV011D_TV5_1_2_SPLITS) == TV011D_TV5_SECRET ] == ~zero
property TV011D_TV5_recombine_8_9_passes_test = [ GF_MAT_MUL(TV011D_TV5_8_9_R,
TV011D_TV5_8_9_SPLITS) == TV011D_TV5_SECRET ] == ~zero

/*
 * Test vector TV011D_6
 * secret = 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
 * random = 02 4A 89 AC 96 8C 98 65 77 FE B0 24 11 6B 94 F6 54 DD DE 20 9C 3C C3 E4 48
88 4D 31 F8 C8
```

```
 *
 * l = 15
 * m = 3
 * n = 5
 *
 * split1 = 49 27 19 F9 8C 92 7D 6A 80 F4 AB 2B CD 72 3F
 * split2 = 30 87 38 A0 34 EB 94 C2 F2 2B DE 20 87 50 E5
 * split3 = 78 A2 22 5D BD 7F EE A0 7B D5 7E 07 47 2C D5
 * split4 = DD 0E 49 40 9F 86 BD B9 15 6F A6 C1 58 10 D4
 * split5 = 95 2B 53 BD 16 12 C7 DB 9C 91 06 E6 98 6C E4
 */

TV011D_TV6_P : [5][3]GF28
TV011D_TV6_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02) ],
    [ GF_POW(0x05, 0x00), GF_POW(0x05, 0x01), GF_POW(0x05, 0x02) ]
]

TV011D_TV6_SR : [3][15]GF28
TV011D_TV6_SR = [
    [ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D,
0x0E, 0x0F ],
    [ 0x02, 0x89, 0x96, 0x98, 0x77, 0xB0, 0x11, 0x94, 0x54, 0xDE, 0x9C, 0xC3, 0x48,
0x4D, 0xF8 ],
    [ 0x4A, 0xAC, 0x8C, 0x65, 0xFE, 0x24, 0x6B, 0xF6, 0xDD, 0x20, 0x3C, 0xE4, 0x88,
0x31, 0xC8 ]
]

TV011D_TV6_SPLITS : [5][15]GF28
TV011D_TV6_SPLITS = [
    [ 0x49, 0x27, 0x19, 0xF9, 0x8C, 0x92, 0x7D, 0x6A, 0x80, 0xF4, 0xAB, 0x2B, 0xCD,
0x72, 0x3F ],
    [ 0x30, 0x87, 0x38, 0xA0, 0x34, 0xEB, 0x94, 0xC2, 0xF2, 0x2B, 0xDE, 0x20, 0x87,
0x50, 0xE5 ],
    [ 0x78, 0xA2, 0x22, 0x5D, 0xBD, 0x7F, 0xEE, 0xA0, 0x7B, 0xD5, 0x7E, 0x07, 0x47,
0x2C, 0xD5 ],
    [ 0xDD, 0x0E, 0x49, 0x40, 0x9F, 0x86, 0xBD, 0xB9, 0x15, 0x6F, 0xA6, 0xC1, 0x58,
0x10, 0xD4 ],
    [ 0x95, 0x2B, 0x53, 0xBD, 0x16, 0x12, 0xC7, 0xDB, 0x9C, 0x91, 0x06, 0xE6, 0x98,
0x6C, 0xE4 ]
]

/* test split */
property TV011D_TV6_split_passes_test = [ GF_MAT_MUL(TV011D_TV6_P, TV011D_TV6_SR) ==
TV011D_TV6_SPLITS ] == ~zero

TV011D_TV6_1_2_3_R : [1][3]GF28
TV011D_TV6_1_2_3_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03))]
    ]
]

TV011D_TV6_2_3_4_R : [1][3]GF28
TV011D_TV6_2_3_4_R = [
    [ GF_PROD[GF_DIV(0x03, GF_ADD(0x02, 0x03)), GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x02, GF_ADD(0x02, 0x03)), GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x02, GF_ADD(0x02, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011D_TV6_1_2_3_SPLITS : [3][15]GF28
TV011D_TV6_1_2_3_SPLITS = [
    [ 0x49, 0x27, 0x19, 0xF9, 0x8C, 0x92, 0x7D, 0x6A, 0x80, 0xF4, 0xAB, 0x2B, 0xCD,
0x72, 0x3F ],
    [ 0x30, 0x87, 0x38, 0xA0, 0x34, 0xEB, 0x94, 0xC2, 0xF2, 0x2B, 0xDE, 0x20, 0x87,
0x50, 0xE5 ],
    [ 0x78, 0xA2, 0x22, 0x5D, 0xBD, 0x7F, 0xEE, 0xA0, 0x7B, 0xD5, 0x7E, 0x07, 0x47,
0x2C, 0xD5 ]
]

TV011D_TV6_2_3_4_SPLITS : [3][15]GF28
TV011D_TV6_2_3_4_SPLITS = [
    [ 0x30, 0x87, 0x38, 0xA0, 0x34, 0xEB, 0x94, 0xC2, 0xF2, 0x2B, 0xDE, 0x20, 0x87,
0x50, 0xE5 ],
    [ 0x78, 0xA2, 0x22, 0x5D, 0xBD, 0x7F, 0xEE, 0xA0, 0x7B, 0xD5, 0x7E, 0x07, 0x47,
0x2C, 0xD5 ],
    [ 0xDD, 0x0E, 0x49, 0x40, 0x9F, 0x86, 0xBD, 0xB9, 0x15, 0x6F, 0xA6, 0xC1, 0x58,
0x10, 0xD4 ]
]

TV011D_TV6_SECRET : [1][15]GF28
TV011D_TV6_SECRET = [
    [ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D,
0x0E, 0x0F ]
]

/* test recombines */
property TV011D_TV6_recombine_1_2_3_passes_test = [ GF_MAT_MUL(TV011D_TV6_1_2_3_R,
TV011D_TV6_1_2_3_SPLITS) == TV011D_TV6_SECRET ] == ~zero
property TV011D_TV6_recombine_2_3_4_passes_test = [ GF_MAT_MUL(TV011D_TV6_2_3_4_R,
TV011D_TV6_2_3_4_SPLITS) == TV011D_TV6_SECRET ] == ~zero
```

E.2.2 “Native” GF(256)-based implementation

```
module secret_share_011D_gf28 where

type GF28 = [8]

irreducible = <| x^^8 + x^^4 + x^^3 + x^^2 + 1 |>

GF_ADD : (GF28, GF28) -> GF28
GF_ADD (x, y) = x ^ y

GF_SUB : (GF28, GF28) -> GF28
GF_SUB (x, y) = x ^ y

GF_MUL : (GF28, GF28) -> GF28
GF_MUL (x, y) = pmod(pmult x y) irreducible

GF_POW : (GF28, [8]) -> GF28
GF_POW (n, k) = pow k
  where   sq x  = GF_MUL (x, x)
          odd x = x ! 0
          pow i = if i == 0 then 1
                  else if odd i
                       then GF_MUL(n, sq (pow (i >> 1)))
                       else sq (pow (i >> 1))

GF_INV : GF28 -> GF28
GF_INV x = GF_POW (x, 254)

GF_DIV : (GF28, GF28) -> GF28
GF_DIV (x, y) = div x
  where   inv yi  = GF_INV (yi)
          div i = if x == 0 then 0
                  else GF_MUL(x, inv y)

GF_SUM : {n} (fin n) => [n]GF28 -> GF28
GF_SUM ps = accum ! 0
  where accum = [zero] # [ GF_ADD(p, a) | p <- ps | s <- accum ]

GF_PROD : {n} (fin n) => [n]GF28 -> GF28
GF_PROD ps = accum ! 0
  where accum = [1] # [ GF_MUL(p, a) | p <- ps | a <- accum ]

GF_DOT_PROD : {n} (fin n) => ([n]GF28, [n]GF28) -> GF28
GF_DOT_PROD (xs, ys) = GF_SUM [ GF_MUL (x, y) | x <- xs
                                              | y <- ys
                              ]

GF_VEC_MUL : {n, m} (fin n) => ([n]GF28, [m][n]GF28) -> [m]GF28
GF_VEC_MUL (v, ms) = [ GF_DOT_PROD(v, m) | m <- ms ]

GF_MAT_MUL : {n, m, k} (fin m) => ([n][m]GF28, [m][k]GF28) -> [n][k]GF28
GF_MAT_MUL (xss, yss) = [ GF_VEC_MUL(xs, yss') | xs <- xss ]
        where yss' = transpose yss

// Test test vectors for Polynomial 2 (x^^8 + x^^4 + x^^3 + x^^2 + 1)

/*
 * Test vector TV011D_1
 * secret = 74 65 73 74 00
 * random = F3 C2 33 81 F5
 *
 * l = 5
 * m = 2
 * n = 2
 *
 * split1 = 87 A7 40 F5 F5
 * split2 = 8F FC 15 6B F7
 */

TV011D_TV1_P : [2][2]GF28
TV011D_TV1_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ]
]

TV011D_TV1_SR : [2][5]GF28
TV011D_TV1_SR = [
    [ 0x74, 0x65, 0x73, 0x74, 0x00 ],
    [ 0xF3, 0xC2, 0x33, 0x81, 0xF5 ]
]

TV011D_TV1_SPLITS : [2][5]GF28
TV011D_TV1_SPLITS = [
    [ 0x87, 0xA7, 0x40, 0xF5, 0xF5 ],
    [ 0x8F, 0xFC, 0x15, 0x6B, 0xF7 ]
]

/* test split */
property TV011D_TV1_passes_test = [ GF_MAT_MUL(TV011D_TV1_P, TV011D_TV1_SR) ==
TV011D_TV1_SPLITS ] == ~zero

TV011D_TV1_1_2_R : [1][2]GF28
TV011D_TV1_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x02, 0x01)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011D_TV1_1_2_SPLITS : [2][5]GF28
TV011D_TV1_1_2_SPLITS = [
    [ 0x87, 0xA7, 0x40, 0xF5, 0xF5 ],
    [ 0x8F, 0xFC, 0x15, 0x6B, 0xF7 ]
]

TV011D_TV1_SECRET : [1][5]GF28
TV011D_TV1_SECRET = [
    [0x74, 0x65, 0x73, 0x74, 0x00 ]
]

/* test recombine */
property TV011D_TV1_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011D_TV1_1_2_R,
TV011D_TV1_1_2_SPLITS) == TV011D_TV1_SECRET ] == ~zero
```

```
/*
 * Test vector TV011D_2
 * secret = 53 41 4D 54 43
 * random = 20 76 08 93 0C
 *
 * l = 5
 * m = 2
 * n = 4
 *
 * split1 = 73 37 45 C7 4F
 * split2 = 13 AD 5D 6F 5B
 * split3 = 33 DB 55 FC 57
 * split4 = D3 84 6D 22 73
 */

TV011D_TV2_P : [4][2]GF28
TV011D_TV2_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01) ]
]

TV011D_TV2_SR : [2][5]GF28
TV011D_TV2_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x20, 0x76, 0x08, 0x93, 0x0C ]
]

TV011D_TV2_SPLITS : [4][5]GF28
TV011D_TV2_SPLITS = [
    [ 0x73, 0x37, 0x45, 0xC7, 0x4F ],
    [ 0x13, 0xAD, 0x5D, 0x6F, 0x5B ],
    [ 0x33, 0xDB, 0x55, 0xFC, 0x57 ],
    [ 0xD3, 0x84, 0x6D, 0x22, 0x73 ]
]

/* test split */
property TV011D_TV2_passes_test = [ GF_MAT_MUL(TV011D_TV2_P, TV011D_TV2_SR) ==
TV011D_TV2_SPLITS ] == ~zero

TV011D_TV2_1_2_R : [1][2]GF28
TV011D_TV2_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011D_TV2_1_4_R : [1][2]GF28
TV011D_TV2_1_4_R = [
    [ GF_DIV(0x04, GF_ADD(0x01, 0x04)), GF_DIV(0x01, GF_ADD(0x01, 0x04)) ]
]

TV011D_TV2_3_4_R : [1][2]GF28
TV011D_TV2_3_4_R = [
    [ GF_DIV(0x04, GF_ADD(0x03, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04)) ]
]

TV011D_TV2_1_2_SPLITS : [2][5]GF28
TV011D_TV2_1_2_SPLITS = [
    [ 0x73, 0x37, 0x45, 0xC7, 0x4F ],
    [ 0x13, 0xAD, 0x5D, 0x6F, 0x5B ]
]

TV011D_TV2_1_4_SPLITS : [2][5]GF28
TV011D_TV2_1_4_SPLITS = [
    [ 0x73, 0x37, 0x45, 0xC7, 0x4F ],
    [ 0xD3, 0x84, 0x6D, 0x22, 0x73 ]
]

TV011D_TV2_3_4_SPLITS : [2][5]GF28
TV011D_TV2_3_4_SPLITS = [
    [ 0x33, 0xDB, 0x55, 0xFC, 0x57 ],
    [ 0xD3, 0x84, 0x6D, 0x22, 0x73 ]
]

TV011D_TV2_SECRET : [1][5]GF28
TV011D_TV2_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]

/* test recombines */
property TV011D_TV2_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011D_TV2_1_2_R,
TV011D_TV2_1_2_SPLITS) == TV011D_TV2_SECRET ] == ~zero
property TV011D_TV2_recombine_1_4_passes_test = [ GF_MAT_MUL(TV011D_TV2_1_4_R,
TV011D_TV2_1_4_SPLITS) == TV011D_TV2_SECRET ] == ~zero
property TV011D_TV2_recombine_3_4_passes_test = [ GF_MAT_MUL(TV011D_TV2_3_4_R,
TV011D_TV2_3_4_SPLITS) == TV011D_TV2_SECRET ] == ~zero

/*
 * Test vector TV011D_3
 * secret = 53 41 4D 54 43
 * random = 8C 15 92 62 5C 4A AF 53 41 45
 *
 * l = 5
 * m = 3
 * n = 4
 *
 * split1 = CA B1 5B A8 47
 * split2 = 02 ED C0 46 C8
 * split3 = 9B 1D D6 BA CC
 * split4 = 14 5D F4 8B 7E
 */

TV011D_TV3_P : [4][3]GF28
TV011D_TV3_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02) ]
]

TV011D_TV3_SR : [3][5]GF28
TV011D_TV3_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x8C, 0x92, 0x5C, 0xAF, 0x41 ],
    [ 0x15, 0x62, 0x4A, 0x53, 0x45 ]
]

TV011D_TV3_SPLITS : [4][5]GF28
TV011D_TV3_SPLITS = [
    [ 0xCA, 0xB1, 0x5B, 0xA8, 0x47 ],
    [ 0x02, 0xED, 0xC0, 0x46, 0xC8 ],
    [ 0x9B, 0x1D, 0xD6, 0xBA, 0xCC ],
    [ 0x14, 0x5D, 0xF4, 0x8B, 0x7E ]
]

/* test split */
property TV011D_TV3_split_passes_test = [ GF_MAT_MUL(TV011D_TV3_P, TV011D_TV3_SR) ==
TV011D_TV3_SPLITS ] == ~zero

TV011D_TV3_1_2_3_R : [1][3]GF28
TV011D_TV3_1_2_3_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03))]
    ]
]

TV011D_TV3_1_2_4_R : [1][3]GF28
TV011D_TV3_1_2_4_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x02, GF_ADD(0x02, 0x04))]
    ]
]

TV011D_TV3_1_3_4_R : [1][3]GF28
TV011D_TV3_1_3_4_R = [
    [ GF_PROD[GF_DIV(0x03, GF_ADD(0x01, 0x03)), GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011D_TV3_1_2_3_SPLITS : [3][5]GF28
TV011D_TV3_1_2_3_SPLITS = [
    [ 0xCA, 0xB1, 0x5B, 0xA8, 0x47 ],
    [ 0x02, 0xED, 0xC0, 0x46, 0xC8 ],
    [ 0x9B, 0x1D, 0xD6, 0xBA, 0xCC ]
]

TV011D_TV3_1_2_4_SPLITS : [3][5]GF28
TV011D_TV3_1_2_4_SPLITS = [
    [ 0xCA, 0xB1, 0x5B, 0xA8, 0x47 ],
    [ 0x02, 0xED, 0xC0, 0x46, 0xC8 ],
    [ 0x14, 0x5D, 0xF4, 0x8B, 0x7E ]
]

TV011D_TV3_1_3_4_SPLITS : [3][5]GF28
TV011D_TV3_1_3_4_SPLITS = [
    [ 0xCA, 0xB1, 0x5B, 0xA8, 0x47 ],
    [ 0x9B, 0x1D, 0xD6, 0xBA, 0xCC ],
    [ 0x14, 0x5D, 0xF4, 0x8B, 0x7E ]
]

TV011D_TV3_SECRET : [1][5]GF28
TV011D_TV3_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]

/* test recombines */
property TV011D_TV3_recombine_1_2_3_passes_test = [ GF_MAT_MUL(TV011D_TV3_1_2_3_R,
TV011D_TV3_1_2_3_SPLITS) == TV011D_TV3_SECRET ] == ~zero
property TV011D_TV3_recombine_1_2_4_passes_test = [ GF_MAT_MUL(TV011D_TV3_1_2_4_R,
TV011D_TV3_1_2_4_SPLITS) == TV011D_TV3_SECRET ] == ~zero
property TV011D_TV3_recombine_1_3_4_passes_test = [ GF_MAT_MUL(TV011D_TV3_1_3_4_R,
TV011D_TV3_1_3_4_SPLITS) == TV011D_TV3_SECRET ] == ~zero

/*
 * Test vector TV011D_4
 * secret = 53 41 4D 54 43
 * random = 72 B0 88 3C 96 B9 CB B9 CB B2 82 66 F3 79 FA
 *
 * l = 5
 * m = 4
 * n = 4
 *
 * split1 = 19 52 F4 02 33
 * split2 = 79 FA 0E 08 C2
 * split3 = 24 58 37 17 94
 * split4 = F4 45 A9 D6 07
 */

TV011D_TV4_P : [4][4]GF28
TV011D_TV4_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02), GF_POW(0x01, 0x03) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02), GF_POW(0x02, 0x03) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02), GF_POW(0x03, 0x03) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02), GF_POW(0x04, 0x03) ]
]

TV011D_TV4_SR : [4][5]GF28
TV011D_TV4_SR = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ],
    [ 0x72, 0x3C, 0xCB, 0xB2, 0xF3 ],
    [ 0xB0, 0x96, 0xB9, 0x82, 0x79 ],
    [ 0x88, 0xB9, 0xCB, 0x66, 0xFA ]
]

TV011D_TV4_SPLITS : [4][5]GF28
TV011D_TV4_SPLITS = [
    [ 0x19, 0x52, 0xF4, 0x02, 0x33 ],
    [ 0x79, 0xFA, 0x0E, 0x08, 0xC2 ],
    [ 0x24, 0x58, 0x37, 0x17, 0x94 ],
    [ 0xF4, 0x45, 0xA9, 0xD6, 0x07 ]
]

/* test split */
property TV011D_TV4_split_passes_test = [ GF_MAT_MUL(TV011D_TV4_P, TV011D_TV4_SR) ==
TV011D_TV4_SPLITS ] == ~zero

TV011D_TV4_1_2_3_4_R : [1][4]GF28
TV011D_TV4_1_2_3_4_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03)),
GF_DIV(0x04, GF_ADD(0x01, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03)),
GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03)),
GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x04)), GF_DIV(0x02, GF_ADD(0x02, 0x04)),
GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011D_TV4_1_2_3_4_SPLITS : [4][5]GF28
TV011D_TV4_1_2_3_4_SPLITS = [
    [ 0x19, 0x52, 0xF4, 0x02, 0x33 ],
    [ 0x79, 0xFA, 0x0E, 0x08, 0xC2 ],
    [ 0x24, 0x58, 0x37, 0x17, 0x94 ],
    [ 0xF4, 0x45, 0xA9, 0xD6, 0x07 ]
]

TV011D_TV4_SECRET : [1][5]GF28
TV011D_TV4_SECRET = [
    [ 0x53, 0x41, 0x4D, 0x54, 0x43 ]
]

/* test recombines */
property TV011D_TV4_recombine_1_2_3_4_passes_test = [ GF_MAT_MUL(TV011D_TV4_1_2_3_4_R,
TV011D_TV4_1_2_3_4_SPLITS) == TV011D_TV4_SECRET ] == ~zero

/*
 * Test vector TV011D_5
 * secret = 54 65 73 74 20 44 61 74 61
 * random = AF FD 2B 0B FA 34 33 63 9C
 *
 * l = 9
 * m = 2
 * n = 9
 *
 * split1 = FB 98 58 7F DA 70 52 17 FD
 * split2 = 17 82 25 62 C9 2C 07 B2 44
 * split3 = B8 7F 0E 69 33 18 34 D1 D8
 * split4 = D2 B6 DF 58 EF 94 AD E5 2B
 * split5 = 7D 4B F4 53 15 A0 9E 86 B7
 * split6 = 91 51 89 4E 06 FC CB 23 0E
 * split7 = 3E AC A2 45 FC C8 F8 40 92
 * split8 = 45 DE 36 2C A3 F9 E4 4B F5
 * split9 = EA 23 1D 27 59 CD D7 28 69
 */

TV011D_TV5_P : [9][2]GF28
TV011D_TV5_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01) ],
    [ GF_POW(0x05, 0x00), GF_POW(0x05, 0x01) ],
    [ GF_POW(0x06, 0x00), GF_POW(0x06, 0x01) ],
    [ GF_POW(0x07, 0x00), GF_POW(0x07, 0x01) ],
    [ GF_POW(0x08, 0x00), GF_POW(0x08, 0x01) ],
    [ GF_POW(0x09, 0x00), GF_POW(0x09, 0x01) ]
]

TV011D_TV5_SR : [2][9]GF28
TV011D_TV5_SR = [
    [ 0x54, 0x65, 0x73, 0x74, 0x20, 0x44, 0x61, 0x74, 0x61 ],
    [ 0xAF, 0xFD, 0x2B, 0x0B, 0xFA, 0x34, 0x33, 0x63, 0x9C ]
]

TV011D_TV5_SPLITS : [9][9]GF28
TV011D_TV5_SPLITS = [
    [ 0xFB, 0x98, 0x58, 0x7F, 0xDA, 0x70, 0x52, 0x17, 0xFD ],
    [ 0x17, 0x82, 0x25, 0x62, 0xC9, 0x2C, 0x07, 0xB2, 0x44 ],
    [ 0xB8, 0x7F, 0x0E, 0x69, 0x33, 0x18, 0x34, 0xD1, 0xD8 ],
    [ 0xD2, 0xB6, 0xDF, 0x58, 0xEF, 0x94, 0xAD, 0xE5, 0x2B ],
    [ 0x7D, 0x4B, 0xF4, 0x53, 0x15, 0xA0, 0x9E, 0x86, 0xB7 ],
    [ 0x91, 0x51, 0x89, 0x4E, 0x06, 0xFC, 0xCB, 0x23, 0x0E ],
    [ 0x3E, 0xAC, 0xA2, 0x45, 0xFC, 0xC8, 0xF8, 0x40, 0x92 ],
    [ 0x45, 0xDE, 0x36, 0x2C, 0xA3, 0xF9, 0xE4, 0x4B, 0xF5 ],
    [ 0xEA, 0x23, 0x1D, 0x27, 0x59, 0xCD, 0xD7, 0x28, 0x69 ]
]

/* test split */
property TV011D_TV5_split_passes_test = [ GF_MAT_MUL(TV011D_TV5_P, TV011D_TV5_SR) ==
TV011D_TV5_SPLITS ] == ~zero

TV011D_TV5_1_2_R : [1][2]GF28
TV011D_TV5_1_2_R = [
    [ GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x01, GF_ADD(0x01, 0x02)) ]
]

TV011D_TV5_8_9_R : [1][2]GF28
TV011D_TV5_8_9_R = [
    [ GF_DIV(0x09, GF_ADD(0x08, 0x09)), GF_DIV(0x08, GF_ADD(0x08, 0x09)) ]
]

TV011D_TV5_1_2_SPLITS : [2][9]GF28
TV011D_TV5_1_2_SPLITS = [
    [ 0xFB, 0x98, 0x58, 0x7F, 0xDA, 0x70, 0x52, 0x17, 0xFD ],
    [ 0x17, 0x82, 0x25, 0x62, 0xC9, 0x2C, 0x07, 0xB2, 0x44 ]
]

TV011D_TV5_8_9_SPLITS : [2][9]GF28
TV011D_TV5_8_9_SPLITS = [
    [ 0x45, 0xDE, 0x36, 0x2C, 0xA3, 0xF9, 0xE4, 0x4B, 0xF5 ],
    [ 0xEA, 0x23, 0x1D, 0x27, 0x59, 0xCD, 0xD7, 0x28, 0x69 ]
]

TV011D_TV5_SECRET : [1][9]GF28
TV011D_TV5_SECRET = [
    [ 0x54, 0x65, 0x73, 0x74, 0x20, 0x44, 0x61, 0x74, 0x61 ]
]

/* test recombines */
property TV011D_TV5_recombine_1_2_passes_test = [ GF_MAT_MUL(TV011D_TV5_1_2_R,
TV011D_TV5_1_2_SPLITS) == TV011D_TV5_SECRET ] == ~zero
property TV011D_TV5_recombine_8_9_passes_test = [ GF_MAT_MUL(TV011D_TV5_8_9_R,
TV011D_TV5_8_9_SPLITS) == TV011D_TV5_SECRET ] == ~zero

/*
 * Test vector TV011D_6
 * secret = 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
 * random = 02 4A 89 AC 96 8C 98 65 77 FE B0 24 11 6B 94 F6 54 DD DE 20 9C 3C C3 E4 48
88 4D 31 F8 C8
```

```
 *
 * l = 15
 * m = 3
 * n = 5
 *
 * split1 = 49 27 19 F9 8C 92 7D 6A 80 F4 AB 2B CD 72 3F
 * split2 = 30 87 38 A0 34 EB 94 C2 F2 2B DE 20 87 50 E5
 * split3 = 78 A2 22 5D BD 7F EE A0 7B D5 7E 07 47 2C D5
 * split4 = DD 0E 49 40 9F 86 BD B9 15 6F A6 C1 58 10 D4
 * split5 = 95 2B 53 BD 16 12 C7 DB 9C 91 06 E6 98 6C E4
 */

TV011D_TV6_P : [5][3]GF28
TV011D_TV6_P = [
    [ GF_POW(0x01, 0x00), GF_POW(0x01, 0x01), GF_POW(0x01, 0x02) ],
    [ GF_POW(0x02, 0x00), GF_POW(0x02, 0x01), GF_POW(0x02, 0x02) ],
    [ GF_POW(0x03, 0x00), GF_POW(0x03, 0x01), GF_POW(0x03, 0x02) ],
    [ GF_POW(0x04, 0x00), GF_POW(0x04, 0x01), GF_POW(0x04, 0x02) ],
    [ GF_POW(0x05, 0x00), GF_POW(0x05, 0x01), GF_POW(0x05, 0x02) ]
]

TV011D_TV6_SR : [3][15]GF28
TV011D_TV6_SR = [
    [ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D,
0x0E, 0x0F ],
    [ 0x02, 0x89, 0x96, 0x98, 0x77, 0xB0, 0x11, 0x94, 0x54, 0xDE, 0x9C, 0xC3, 0x48,
0x4D, 0xF8 ],
    [ 0x4A, 0xAC, 0x8C, 0x65, 0xFE, 0x24, 0x6B, 0xF6, 0xDD, 0x20, 0x3C, 0xE4, 0x88,
0x31, 0xC8 ]
]

TV011D_TV6_SPLITS : [5][15]GF28
TV011D_TV6_SPLITS = [
    [ 0x49, 0x27, 0x19, 0xF9, 0x8C, 0x92, 0x7D, 0x6A, 0x80, 0xF4, 0xAB, 0x2B, 0xCD,
0x72, 0x3F ],
    [ 0x30, 0x87, 0x38, 0xA0, 0x34, 0xEB, 0x94, 0xC2, 0xF2, 0x2B, 0xDE, 0x20, 0x87,
0x50, 0xE5 ],
    [ 0x78, 0xA2, 0x22, 0x5D, 0xBD, 0x7F, 0xEE, 0xA0, 0x7B, 0xD5, 0x7E, 0x07, 0x47,
0x2C, 0xD5 ],
    [ 0xDD, 0x0E, 0x49, 0x40, 0x9F, 0x86, 0xBD, 0xB9, 0x15, 0x6F, 0xA6, 0xC1, 0x58,
0x10, 0xD4 ],
    [ 0x95, 0x2B, 0x53, 0xBD, 0x16, 0x12, 0xC7, 0xDB, 0x9C, 0x91, 0x06, 0xE6, 0x98,
0x6C, 0xE4 ]
]

/* test split */
property TV011D_TV6_split_passes_test = [ GF_MAT_MUL(TV011D_TV6_P, TV011D_TV6_SR) ==
TV011D_TV6_SPLITS ] == ~zero

TV011D_TV6_1_2_3_R : [1][3]GF28
TV011D_TV6_1_2_3_R = [
    [ GF_PROD[GF_DIV(0x02, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x01, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x02)), GF_DIV(0x03, GF_ADD(0x02, 0x03))],
      GF_PROD[GF_DIV(0x01, GF_ADD(0x01, 0x03)), GF_DIV(0x02, GF_ADD(0x02, 0x03))]
    ]
]

TV011D_TV6_2_3_4_R : [1][3]GF28
TV011D_TV6_2_3_4_R = [
    [ GF_PROD[GF_DIV(0x03, GF_ADD(0x02, 0x03)), GF_DIV(0x04, GF_ADD(0x02, 0x04))],
      GF_PROD[GF_DIV(0x02, GF_ADD(0x02, 0x03)), GF_DIV(0x04, GF_ADD(0x03, 0x04))],
      GF_PROD[GF_DIV(0x02, GF_ADD(0x02, 0x04)), GF_DIV(0x03, GF_ADD(0x03, 0x04))]
    ]
]

TV011D_TV6_1_2_3_SPLITS : [3][15]GF28
TV011D_TV6_1_2_3_SPLITS = [
    [ 0x49, 0x27, 0x19, 0xF9, 0x8C, 0x92, 0x7D, 0x6A, 0x80, 0xF4, 0xAB, 0x2B, 0xCD,
0x72, 0x3F ],
    [ 0x30, 0x87, 0x38, 0xA0, 0x34, 0xEB, 0x94, 0xC2, 0xF2, 0x2B, 0xDE, 0x20, 0x87,
0x50, 0xE5 ],
    [ 0x78, 0xA2, 0x22, 0x5D, 0xBD, 0x7F, 0xEE, 0xA0, 0x7B, 0xD5, 0x7E, 0x07, 0x47,
0x2C, 0xD5 ]
]

TV011D_TV6_2_3_4_SPLITS : [3][15]GF28
TV011D_TV6_2_3_4_SPLITS = [
    [ 0x30, 0x87, 0x38, 0xA0, 0x34, 0xEB, 0x94, 0xC2, 0xF2, 0x2B, 0xDE, 0x20, 0x87,
0x50, 0xE5 ],
    [ 0x78, 0xA2, 0x22, 0x5D, 0xBD, 0x7F, 0xEE, 0xA0, 0x7B, 0xD5, 0x7E, 0x07, 0x47,
0x2C, 0xD5 ],
    [ 0xDD, 0x0E, 0x49, 0x40, 0x9F, 0x86, 0xBD, 0xB9, 0x15, 0x6F, 0xA6, 0xC1, 0x58,
0x10, 0xD4 ]
]

TV011D_TV6_SECRET : [1][15]GF28
TV011D_TV6_SECRET = [
    [ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D,
0x0E, 0x0F ]
]

/* test recombines */
property TV011D_TV6_recombine_1_2_3_passes_test = [ GF_MAT_MUL(TV011D_TV6_1_2_3_R,
TV011D_TV6_1_2_3_SPLITS) == TV011D_TV6_SECRET ] == ~zero
property TV011D_TV6_recombine_2_3_4_passes_test = [ GF_MAT_MUL(TV011D_TV6_2_3_4_R,
TV011D_TV6_2_3_4_SPLITS) == TV011D_TV6_SECRET ] == ~zero
```

Appendix F. Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website: [https://www.oasis-open.org/policies-guidelines/ipr].

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. OASIS AND ITS MEMBERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THIS DOCUMENT OR ANY PART THEREOF.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specifications, OASIS Standards, or Approved Errata).

[OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this deliverable.]

[OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this OASIS Standards Final Deliverable. OASIS may include such claims on its website, but disclaims any obligation to do so.]

[OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this OASIS Standards Final Deliverable or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Standards Final Deliverable, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.]

The name "OASIS" is a trademark of OASIS, the owner and developer of this document, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, documents, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for above guidance.