Modular arithmetic for FNT

Vianney Rancurel[1]
vianney.rancurel@scality.com

Lâm Pham-Sy[2]
lam.pham-sy@scality.com

January 20, 2020

[1]Scality in San Francisco, CA, 94104.
[2]Scality France.

0.1 Some notations

0.1.1 Element representation

Every element $e \in FNT(q = 2^p + 1)$ is represented by an integer x of p bits and an extra-bit b such that

$$e := (b, x) = \begin{cases} (0,\, e) & \text{if } e < q - 1 \\ (1,\, 0) & \text{if } e = q - 1 \end{cases} \tag{1}$$

0.1.2 BitMap and Vector

Note, we focus on the field $FNT(q = 2^p + 1)$ whose elements are on the value range from 0 to 2^p.

Let $h := 2^p - 1$ denote maximal value for $p-$ bit integers.

A vector $\vec{v}$ consists of m packed $p-$bit integers. A bitmap of $\vec{v}$ is an m-bit integer b whose ith bit is the extra-bit of the ith packed integer of $\vec{v}$.

A vector of m elements of $FNT(q = 2^p + 1)$ can be represented by a vector $\vec{v}$ and a bitmap b.

Example 1. $S := \big(e_1, e_2, ..., e_m\big)$ can be represented by a vector $\vec{v}$ and a bitmap b such that

$$e_i = (b_i, v_i), \forall i$$

0.2 Basic functions

- Create a vector from a bitmap with a same values

  Given a bitmap $b = b_1 b_2 ... b_m$, the following function will return a vector $\vec{v}$ whose ith packed integer is equal to b_i.

  $$\vec{v} = \texttt{to\_vector}(b)$$

  $$v_i = b_i, \forall i$$

- Create a mask from a bitmap

  Given a bitmap $b = b_1 b_2 ... b_m$, the following function will return a mask vector $\vec{v}$ whose ith packed integer is equal to h if $b_i = 1$, and equal to 0 otherwise.

  $$\vec{v} = \texttt{to\_mask}(b)$$

  $$v_i = \begin{cases} h & \text{if } b_i = 1 \\ 0 & \text{otherwise} \end{cases}$$

- Create a bitmap from a vector

  Given a vector $\vec{v}$, the following function will return a bitmap b whose ith bit is equal to 1 if $v_i > 0$, and equal to 0 if $v_i = 0$.

$$b = \texttt{to\_bitmap}(\vec{v})$$

$$b_i = \begin{cases} 1 & \text{if } v_i \text{ ¿ } 0 \\ 0 & \text{otherwise} \end{cases}$$

0.3 Modular increment

Given a vector $\vec{x}$ and a bitmap b, we increment each element.

$$(\vec{y}, b_y) = \texttt{mod\_inc}(\vec{x}, b_x)$$

Data: A vector and a bit-map $\vec{x}, b_x$
Result: Element-wise increment modulo q: $(\vec{y}, b_y) = \texttt{mod\_inc}(\vec{x}, b_x)$
`/* mask for elements equal` h `*/`
$\vec{m} = \texttt{vector\_cmpeq}(\vec{x}, \vec{h})$;
`/* output bitmap will be 1 if input element is` h `*/`
$b_y = \texttt{to\_bitmap}(\vec{m})$;
`/* output` $y_i = x_i + 1 - b_x[i]$ `*/`
$\vec{y} = \texttt{vector\_add}(\vec{x}, \texttt{to\_vector}(\neg b)$;

Algorithm 1: Modular increment

0.4 Modular decrement

Given a vector $\vec{x}$ and a bitmap b, we decrement each element.

$$(\vec{y}, b_y) = \texttt{mod\_dec}(\vec{x}, b_x)$$

0.5 Modular subtraction of bounded inputs

Given two vectors $\vec{x}$ and $\vec{y}$ representing two vectors whose elements are less than h, it does element-wise subtraction of the two vectors.

$$(\vec{z}, b_z) = \texttt{mod\_sub\_bounded}(\vec{x}, \vec{y})$$

Data: A vector and a bit-map $\vec{x}, b_x$
Result: Element-wise decrement modulo q: $(\vec{y}, b_y) = \texttt{mod\_dec}(\vec{x}, b_x)$

```
/* output bitmap equals to 1 if input bitmap and vector
   element equal to 0                                           */
```

$\vec{m} = \texttt{vector\_cmpeq}(\vec{x}, 0)$;
$b_y = (\neg b_x) \wedge \texttt{to\_bitmap}(\vec{m})$;

/* output $y[i] = h$ if input bitmap is 1, otherwise $y[i]$ = $\texttt{max}(0, x[i] - 1)$, it covers also the case input bitmap is 1 */
/* subtract using saturation */

$\vec{y} = \texttt{vector\_sub\_sat}(\vec{x}, \vec{1})$;
if $b_x > 0$ **then**
    $\vec{y} = \texttt{vector\_or}(\vec{y}, \texttt{to\_mask}(b_x))$;

Algorithm 2: Modular decrement

Data: Two vectors $\vec{x}, \vec{y}$
Result: Element-wise subtraction modulo q:
$(\vec{z}, b_z) = \texttt{mod\_sub\_bounded}(\vec{x}, \vec{y})$

/* $x_i \geq y_i \rightarrow z_i = x_i - y_i$ */
/* $x_i < y_i \rightarrow z_i = q + x_i - y_i$ */
/* do subtraction */

$\vec{z_1} = \texttt{vector\_sub}(\vec{x}, \vec{y})$;

/* for i s.t. $x_i < y_i$, z_i is modulo to $(q-1)$ $\rightarrow$ increment z_i */

$\vec{m} = \texttt{vector\_cmpgt}(\vec{y}, \vec{x})$;
$(\vec{z_2}, b_z) = \texttt{mod\_inc}(\texttt{vector\_and}(\vec{z_1}, \vec{m}), 0)$;
$\vec{z} = \texttt{vector\_blendv}(\vec{z_1}, \vec{z_2}, \vec{m})$;

Algorithm 3: Modular subtraction of bounded inputs

0.6 Modular subtraction

Given two vectors $\vec{x}$ and $\vec{y}$ and their bitmap b_x, b_y, it does element-wise subtraction of the two vectors.

$$(\vec{z}, b_z) = \texttt{mod\_sub}(\vec{x}, b_x, \vec{y}, b_y)$$

Data: Two vectors of FNT elements $(\vec{x}, b_x), (\vec{y}, b_y)$
Result: Element-wise subtraction modulo q:
$(\vec{z}, b_z) = \texttt{mod\_sub}(\vec{x}, b_x, \vec{y}, b_y)$

/* do subtraction of vectors */
$(\vec{z}, b_z) = \texttt{mod\_sub\_bounded}(\vec{x}, \vec{y})$;
/* focus on i elements where $b_x[i] = 1, b_y[i] = 0$, we had $z[i] = q - y[i] \rightarrow$ decrement it as we want $z[i] = (q-1) - y[i]$ */
$c_1 = b_x \oplus (b_x \wedge b_y)$;
if $c_1 > 0$ **then**
  $(\vec{z_1}, b_1) = \texttt{mod\_dec}(\vec{z}, b_z)$;
  /* update output */
  $b_z = (b_z \wedge \neg c_1) \vee (b_1 \wedge c_1)$;
  $\vec{z} = \texttt{vector\_blendv}(\vec{z}, \vec{z_1}, \texttt{to\_mask}(c_1))$;
/* focus on i elements where $b_x[i] = 0, b_y[i] = 1$, we had $z[i] = x[i] \rightarrow$ increment it as we want $z[i] = x[i] - (q-1)$ */
$c_2 = b_y \oplus (b_x \wedge b_y)$;
if $c_2 > 0$ **then**
  $(\vec{z_2}, b_2) = \texttt{mod\_inc}(\vec{z}, b_z)$;
  /* update output */
  $b_z = (b_z \wedge \neg c_2) \vee (b_2 \wedge c_2)$;
  $\vec{z} = \texttt{vector\_blendv}(\vec{z}, \vec{z_2}, \texttt{to\_mask}(c_2))$;

Algorithm 4: Modular subtraction

0.7 Modular addition of bounded inputs

Given two vectors $\vec{x}$ and $\vec{y}$ representing two vectors whose elements are less than h, it does element-wise addition of the two vectors.

$$(\vec{z}, b_z) = \texttt{mod\_add\_bounded}(\vec{x}, \vec{y})$$

0.8 Modular addition

Given two vectors $\vec{x}$ and $\vec{y}$ and their bitmap b_x, b_y, it does element-wise addition of the two vectors.

$$(\vec{z}, b_z) = \texttt{mod\_add}(\vec{x}, b_x, \vec{y}, b_y)$$

Data: Two vectors $\vec{x}, \vec{y}$
Result: Element-wise addition modulo q:
$(\vec{z}, b_z) = \texttt{mod\_add\_bounded}(\vec{x}, \vec{y})$
/* do normal addition */
$\vec{z} = \texttt{vector\_add}(\vec{x}, \vec{y})$;
/* for i s.t. $x[i] + y[i] \leq h \rightarrow z[i] = (x[i] + h[i])\%q$ */
/* for i s.t. $x[i] + y[i] > h \rightarrow z[i] = (x[i] + h[i]) - (q - 1)$, as $h = q + 2 \rightarrow z[i] < x[i]$ (and $y[i]$). We want however obtain $(x[i] + y[i])\%q = (z[i] + q - 1)\%q \equiv (z[i] - 1)\%q \rightarrow$ we need decrement $z[i]$ */
$\vec{m} = \texttt{vector\_cmpgt}(\vec{x}, \vec{z})$;
/* for i s.t. $z[i] \geq x[i]$, we set z[i] = 1 before decrement it. Result b_z will be bitmap of the operation */
$(\vec{z_1}, b_z) = \texttt{mod\_dec}(\texttt{vector\_blendv}(\vec{1}, \vec{z}, \vec{m}), 0)$;
$\vec{z} = \texttt{vector\_blendv}(\vec{z}, \vec{z_1}, \vec{m})$;

Algorithm 5: Modular addition of bounded inputs

Data: Two vectors of FNT elements $(\vec{x}, b_x), (\vec{y}, b_y)$
Result: Element-wise addition modulo q: $(\vec{z}, b_z) = \texttt{mod\_add}(\vec{x}, b_x, \vec{y}, b_y)$
/* do addition of vectors */
$(\vec{z}, b_z) = \texttt{mod\_add\_bounded}(\vec{x}, \vec{y})$;
/* for i s.t. $b_x[i] = b_y[i] = 1 \rightarrow$ we set $z[i] = q - 2 \equiv h$ */
$c_1 = b_x \wedge b_y$;
if $c_1 > 0$ **then**
  /* as $z[i] = 0$ */
  $\vec{z} = \texttt{vector\_or}(\vec{z}, \texttt{to\_mask}(c_1))$;
/* for i s.t. $b_x[i] \neq b_y[i] \rightarrow$ we need to add $q - 1$ to $z[i]$, equivalently decrement it */
$c_2 = b_x \oplus b_y$;
if $c_2 > 0$ **then**
  $\vec{m} = \texttt{to\_mask}(c_2)$;
  $(\vec{z_1}, b_1) = \texttt{mod\_dec}(\vec{z}, b_z)$;
  /* update results */
  $b_z = (b_z \wedge \neg c_2) \vee (b_1 \wedge c_2)$;
  $\vec{z} = \texttt{vector\_blendv}(\vec{z}, \vec{z_1}, \vec{m})$;

Algorithm 6: Modular addition

0.9 Modular negation

Given a vector $\vec{x}$ and a bitmap b, we negate each element

$$\vec{y}, b_y) = \texttt{mod\_neg}(\vec{x}, b_x)$$

There are three cases.

- Case1: $x[i] = 0 \rightarrow y[i] = b_x[i], b_y[i] = 0$
- Case2: $x[i] = 1 \rightarrow y[i] = 0, b_y[i] = 1$
- Case3: otherwise, $b_y[i] = 0, y[i] = q - x[i]$. As $x[i] > 1$, we have

  $$(1 - x[i])\%(q-1) \equiv (q - x[i])\%(q-1) = (q - x[i])\%q$$

  Hence, $y[i] = (1 - x[i])\%(q-1)$

Data: A vector and a bit-map $\vec{x}, b_x$
Result: Element-wise negate modulo q: $(\vec{y}, b_y) = \texttt{mod\_neg}(\vec{x}, b_x)$
/* For Case1 and Case2 */
$\vec{y1} = \texttt{to\_vector}(b_x)$;
$b_y = \texttt{to\_bitmap}(\texttt{vector\_cmpeq}(\vec{x}, \vec{1}))$;
/* For Case3 */
$\vec{m} = \texttt{vector\_cmpgt}(\vec{x}, \vec{1})$;
$\vec{y_2} = \texttt{vector\_sub}(\vec{1}, \vec{x})$;
/* get results */
$\vec{y} = \texttt{vector\_or}(\vec{y_1}, \texttt{vector\_and}(\vec{y_2}, \vec{m}))$;

Algorithm 7: Modular negation

0.10 Modular multiplication of bounded inputs

Given two vectors $\vec{x}$ and $\vec{y}$ representing two vectors whose elements are less than h, it does element-wise multiplication of the two vectors.

$$(\vec{z}, b_z) = \texttt{mod\_mul\_bounded}(\vec{x}, \vec{y})$$

0.11 Modular multiplication

Given two vectors $\vec{x}$ and $\vec{y}$ and their bitmap b_x, b_y, it does element-wise multiplication of the two vectors.

$$(\vec{z}, b_z) = \texttt{mod\_mul}(\vec{x}, b_x, \vec{y}, b_y)$$

Data: Two vectors $\vec{x}, \vec{y}$
Result: Element-wise multiplication modulo q:
$(\vec{z}, b_z) =$ `mod_mul_bounded`$(\vec{x}, \vec{y})$

```
/* do multiplication and keep low and high parts */
```
$\vec{lo} =$ `vector_mul_lo`$(\vec{x}, \vec{y})$;
$\vec{hi} =$ `vector_mul_hi`$(\vec{x}, \vec{y})$;
/* subtract $\vec{lo}$ to $\vec{hi}$ to obtain result */
$(\vec{z}, b_z) =$ `mod_sub_bounded`$(\vec{lo}, \vec{hi})$;

Algorithm 8: Modular multiplication of bounded inputs

Data: Two vectors of FNT elements $(\vec{x}, b_x), (\vec{y}, b_y)$
Result: Element-wise multiplication modulo q:
$(\vec{z}, b_z) =$ `mod_mul`$(\vec{x}, b_x, \vec{y}, b_y)$

```
/* do multiplication of vectors */
```
$(\vec{z}, b_z) =$ `mod_mul_bounded`$(\vec{x}, \vec{y})$;
/* for i s.t. $b_x[i] = b_y[i] = 1 \rightarrow$ we set $z[i] = 1$ */
$c_1 = b_x \wedge b_y$;
if $c_1 > 0$ **then**
  /* as $z[i] = 0$ */
  $\vec{z} =$ `vector_or`$(\vec{z},$ `to_vector`$(c_1))$;
/* for i s.t. $b_x[i] \neq b_y[i] \rightarrow z[i] = -\mathtt{max}(x[i], y[i])$ */
$c_2 = b_x \oplus b_y$;
if $c_2 > 0$ **then**
  $\vec{m} =$ `to_mask`(c_2);
  $(\vec{z_1}, b_1) =$ `mod_neg`(`vector_max`$(\vec{x}, \vec{y}), 0)$;
  /* update results */
  $b_z = (b_z \wedge \neg c_2) \vee (b_1 \wedge c_2)$;
  $\vec{z} =$ `vector_blendv`$(\vec{z}, \vec{z_1}, \vec{m})$;

Algorithm 9: Modular multiplication