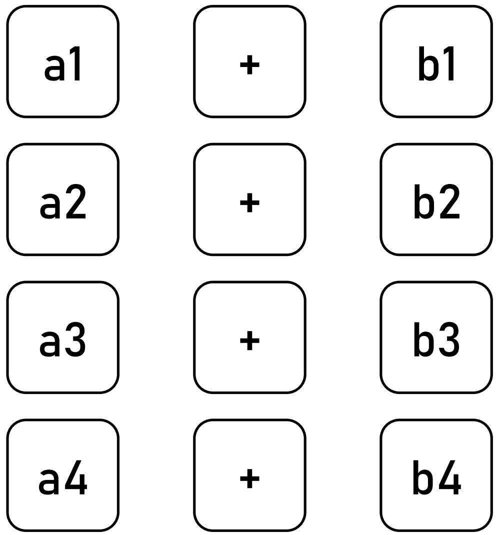
SIMD

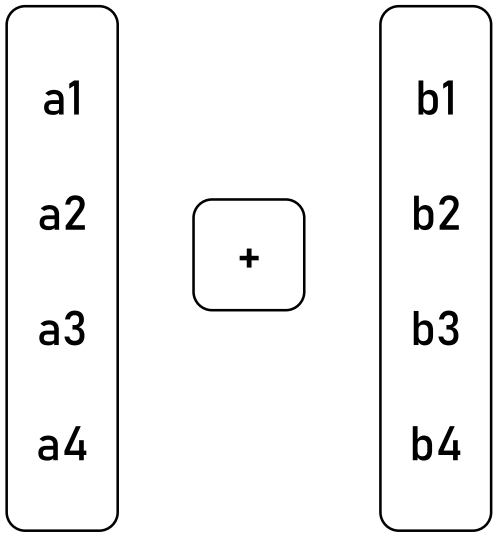
Single Instruction Multiple Data

https://youtu.be/XPa5MyFA7-Y

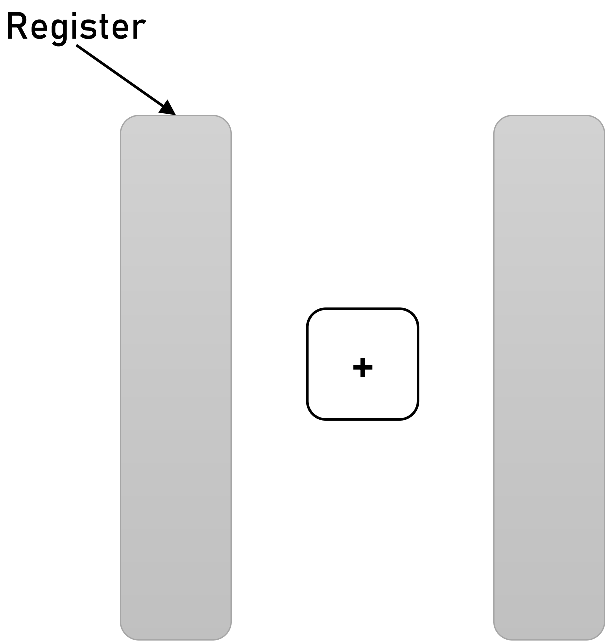
SIMD



[그림] Original Operation



[그림] SIMD Operation



[그림] AVX (Advanced Vector eXtensions)

AVX

- AVX
  - 128~256bit
  - Sandy Bridge, 2011 ~
- AVX2
  - 256bit
  - Haswell, 2013 ~
- AVX-512
  - 256~512bit
  - Knght Landing, 2016 ~
  - Ice Lake, 2019

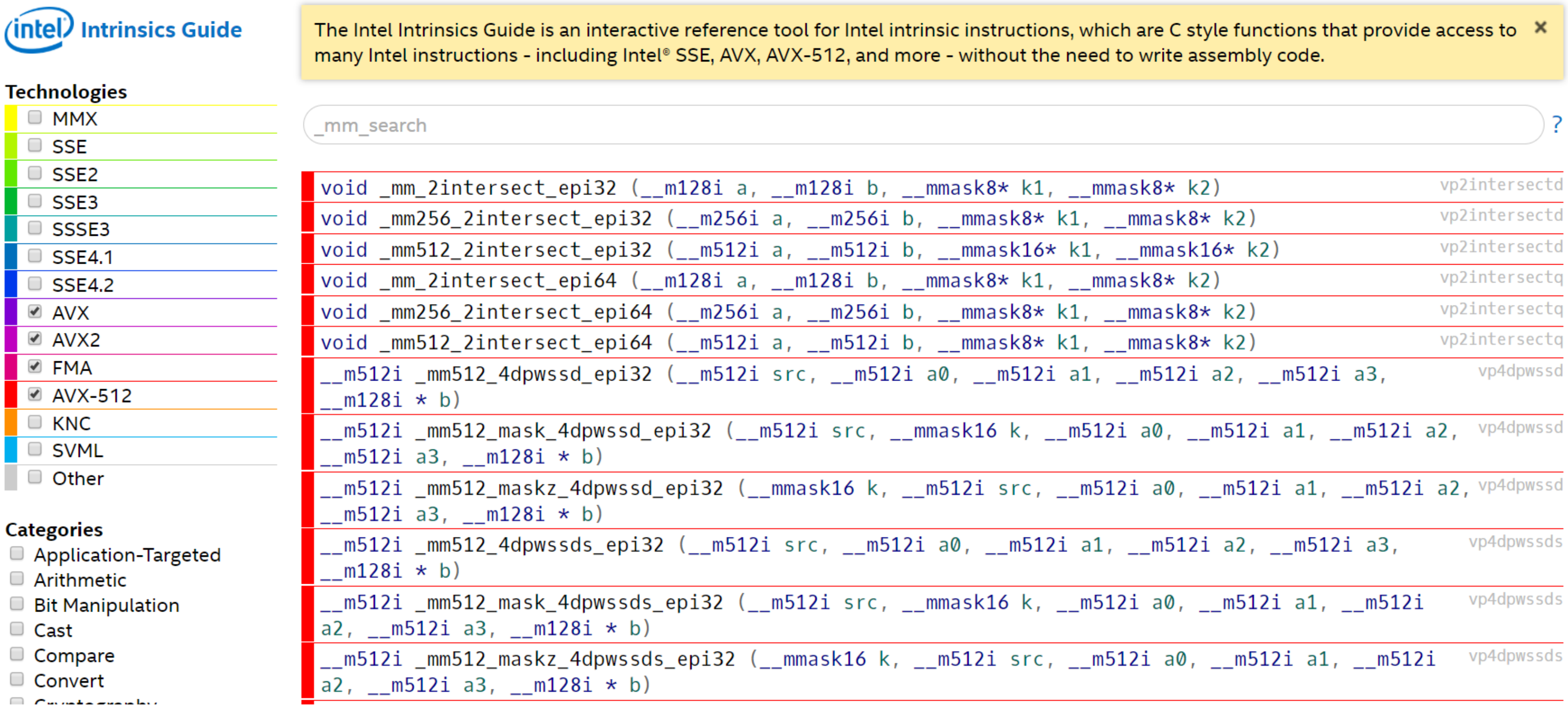
<table>
<tr><th>AVX-512 Subset</th><th>F</th><th>CD</th><th>ER</th><th>PF</th><th>4FMAPS</th><th>4VNNIW</th><th>VL</th><th>DQ</th><th>BW</th><th>IFMA</th><th>VBMI</th><th>VBMI2</th><th>VPOPCNTDQ</th><th>BITALG</th><th>VNNI</th><th>VPCLMULQDQ</th><th>GFNI</th><th>VAES</th></tr>
<tr><td>Intel Knights Landing (2016)</td><td colspan="2" rowspan="6">Yes</td><td colspan="2" rowspan="2">Yes</td><td colspan="14">No</td></tr>
<tr><td>Intel Knights Mill (2017)</td><td colspan="2">Yes</td><td colspan="6">No</td><td>Yes</td><td colspan="5">No</td></tr>
<tr><td>Intel Skylake-SP, Skylake-X (2017)</td><td colspan="4" rowspan="4">No</td><td colspan="3" rowspan="4">Yes</td><td colspan="9">No</td></tr>
<tr><td>Intel Cannon Lake (2018)</td><td colspan="2">Yes</td><td colspan="7">No</td></tr>
<tr><td>Intel Cascade Lake-SP (2019)</td><td colspan="5">No</td><td>Yes</td><td colspan="3">No</td></tr>
<tr><td>Intel Ice Lake (2019)</td><td colspan="9">Yes</td></tr>
</table>

[그림] Processors supporting AVX-512

AVX

Instruction
VBROADCASTSS, VBROADCASTSD
VPBROADCASTB, VPBROADCASTW, VPBROADCASTD, VPBROADCASTQ
VBROADCASTI128
VINSERTI128
VEXTRACTI128
VGATHERDPD, VGATHERQPD, VGATHERDPS, VGATHERQPS
VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ

[그림] AVX2 Instructions



[그림] Intrinsics

Multiplication

- Multiplication

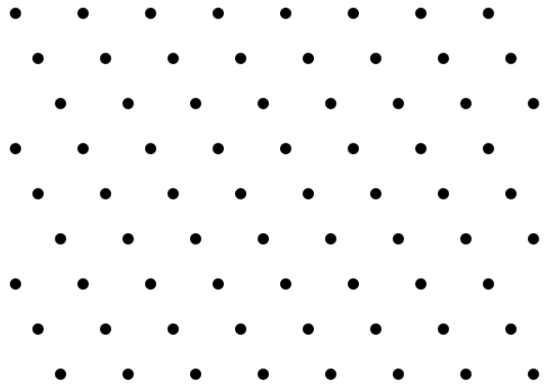
$$(x^1 + 3x^2 + 2x^3) \times (x^1 + 2x^2 + 3x^3)$$

- Multiplication in Ring $(X^n + 1)$

$$(x^1 + 3x^2 + 2x^3) \times (x^1 + 2x^2 + 3x^3)$$

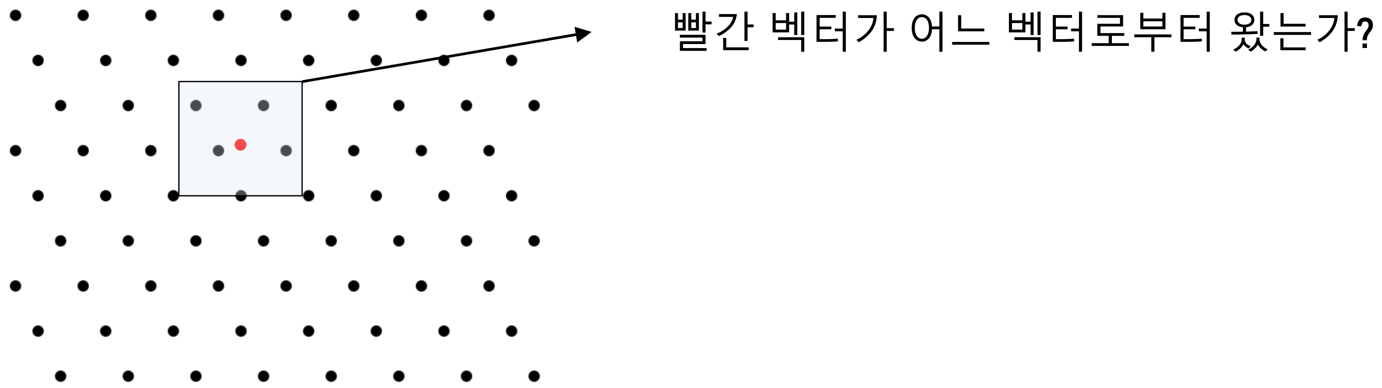
Lattice Problem

- Lattice



[그림] Vector Space

- LWE (Learning With Error)
  - $B = \boxed{A * S} + E$



[그림] a Vector in a Space

Example: LizarMong

Algorithm 4 IND-CCA2-KEM.KeyGen

Input: The set of public *parameters*

Output: Public Key $pk = (Seed_a || \mathbf{b})$, Private Key $sk = (\mathbf{s} || \mathbf{u})$

$Seed_a \xleftarrow{\$} \{0,1\}^{256}$
$\mathbf{a} \leftarrow \texttt{SHAKE256}(Seed_a, n/8)$
$\mathbf{s} \xleftarrow{\$} HWT_n(h_s)$, $\mathbf{u} \xleftarrow{\$} \{0,1\}^n$, $\mathbf{e} \leftarrow \psi_{cb}^n$
$\mathbf{b} \leftarrow -\mathbf{a} * \mathbf{s} + \mathbf{e}$
$pk \leftarrow (Seed_a || \mathbf{b})$, $sk \leftarrow (\mathbf{s} || \mathbf{u})$
return pk, sk

[그림] LizarMong KeyGen Algorithm

$$a = \{ r_1, r_2, \ldots, r_n \}$$

$$s = \{ 0, 0, \ldots, 0 \}$$
$$sample\ a\ from\ \{-1, 0, 1\}$$
$$set\ a\ into\ the\ s\ randomly$$

$$s = \begin{bmatrix} 0 & 0 & 0 & & -1 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & & 0 & 1 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 0 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & & 0 & 0 & 0 \end{bmatrix}$$

Example: LizarMong

$a = \{ r_1, r_2, \dots, r_n\}$

$a[N] = \{ r_1, r_2, \dots, r_n\}$

$s = \{ 0, 0, \dots, 0\}$
$sample\ 128\ a_i\ from\ \{-1, 0, 1\}$
$set\ a\ into\ the\ s\ randomly$

$$s = \begin{bmatrix} 0 & 0 & 0 & & a_3 & 0 & 0 \\ 0 & a_4 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & a_{20} & & 0 & a_{34} & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 0 & a_{67} & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & & a_{128} & 0 & 0 \end{bmatrix}$$

$instead\ of\ s[N] = \{ 0, 0, \dots, 0\}$
$index[128] = \{a_1 index, a_2 index, \dots, a_{128} index\}$

Example: LizarMong

$a \times s$ case study

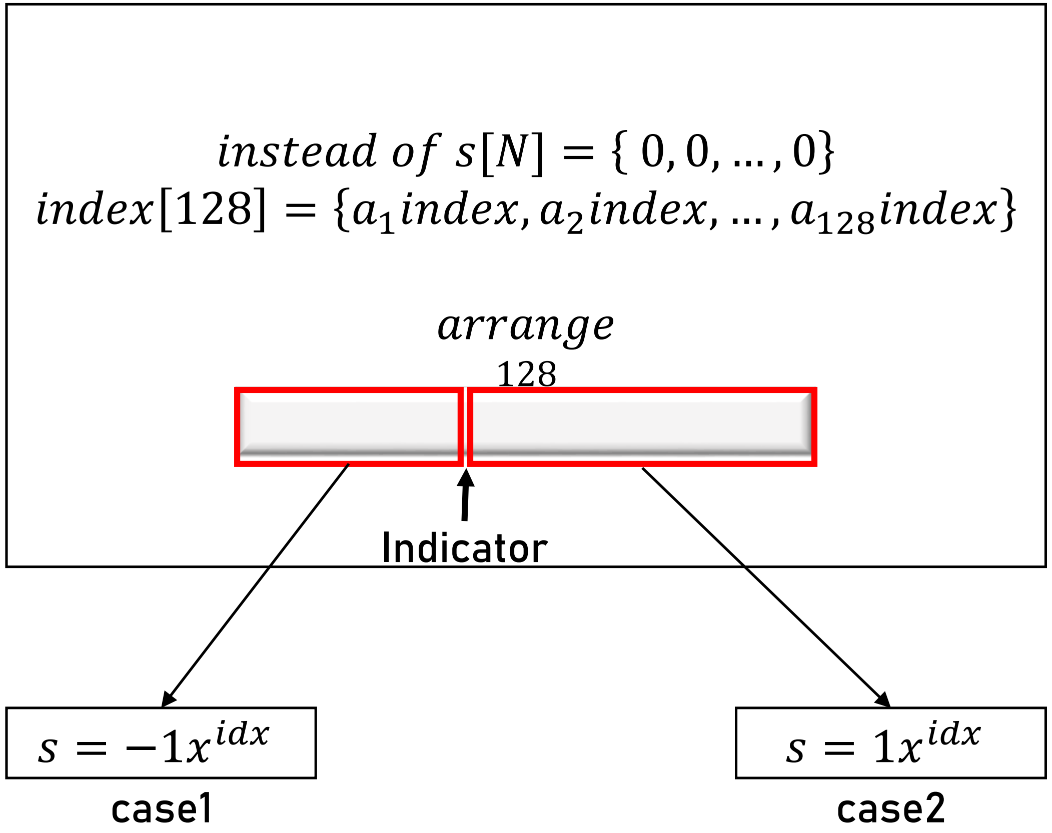
$s = -1x^{idx}$

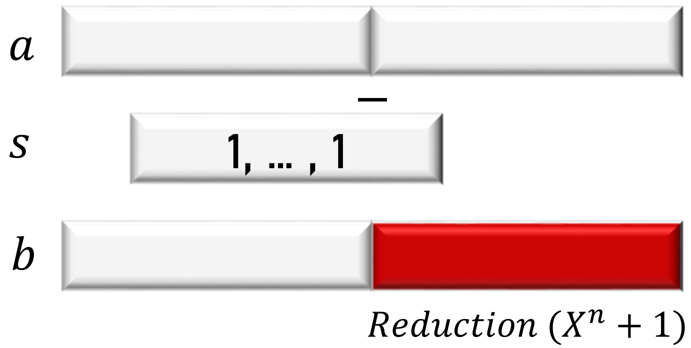
$s = 0x^{idx}$

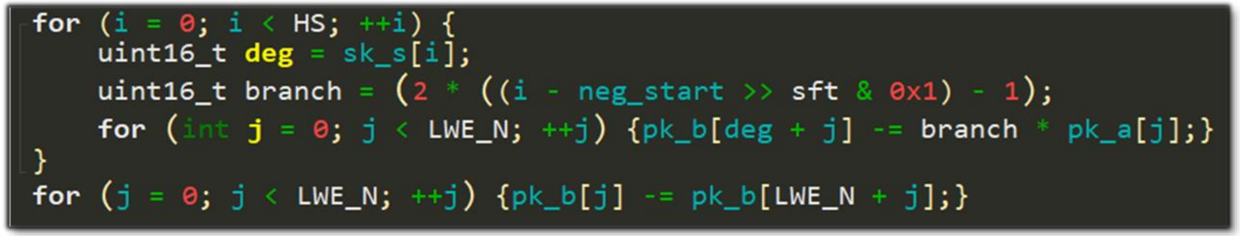
$s = 1x^{idx}$

Example: LizarMong

$a \times s$ case study







```
for (i = 0; i < HS; ++i) {
    uint16_t deg = sk_s[i];
    uint16_t branch = (2 * ((i - neg_start >> sft & 0x1) - 1);
    for (int j = 0; j < LWE_N; ++j) {pk_b[deg + j] -= branch * pk_a[j];}
}
for (j = 0; j < LWE_N; ++j) {pk_b[j] -= pk_b[LWE_N + j];}
```

[그림] Multiplication Implementation

Example: LizarMong

```
for (i = 0; i < neg_start; ++i){
        deg = sk_s[i];
        for ( j = 0; j < LWE_N; j+=32) {
        x256 = _mm256_loadu_si256((__m256i*) &pk_b[deg+j]);
        y256 = _mm256_loadu_si256((__m256i*) &pk_a[j]);
        z256 = _mm256_sub_epi8(x256, y256);
        _mm256_storeu_si256((__m256i*)&pk_b[deg+j], z256);
        }

    }
    for (i = neg_start; i < HS; ++i){
        deg = sk_s[i];
        for ( j = 0; j < LWE_N; j+=32) {
         x256 = _mm256_loadu_si256((__m256i*) &pk_b[deg+j]);
         y256 = _mm256_loadu_si256((__m256i*) &pk_a[j]);
         z256 = _mm256_add_epi8(x256, y256);
        _mm256_storeu_si256((__m256i*)&pk_b[deg+j], z256);
        }
    }


    for (j = 0; j < LWE_N; j+=32){
        x256 = _mm256_loadu_si256((__m256i*) &pk_b[j]);
        y256 = _mm256_loadu_si256((__m256i*) &pk_b[LWE_N+j]);
        z256 = _mm256_sub_epi8(x256, y256);
        _mm256_storeu_si256((__m256i*)&pk_b[j], z256);
    }
```

[그림] AVX Implementation