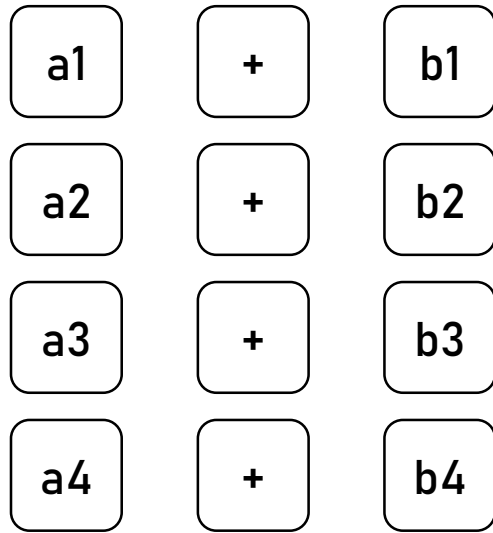


SIMD

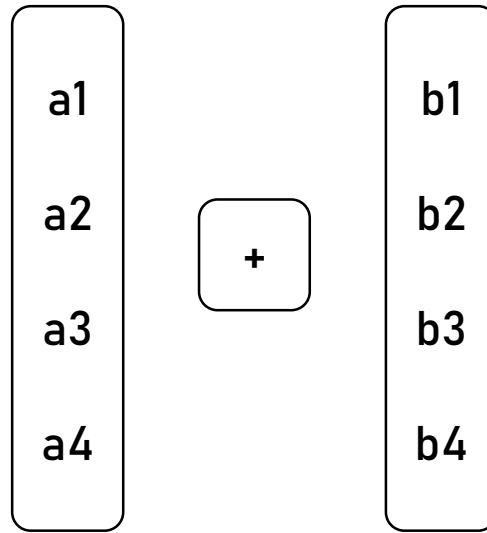
Single Instruction Multiple Data

<https://youtu.be/XPa5MyFA7-Y>

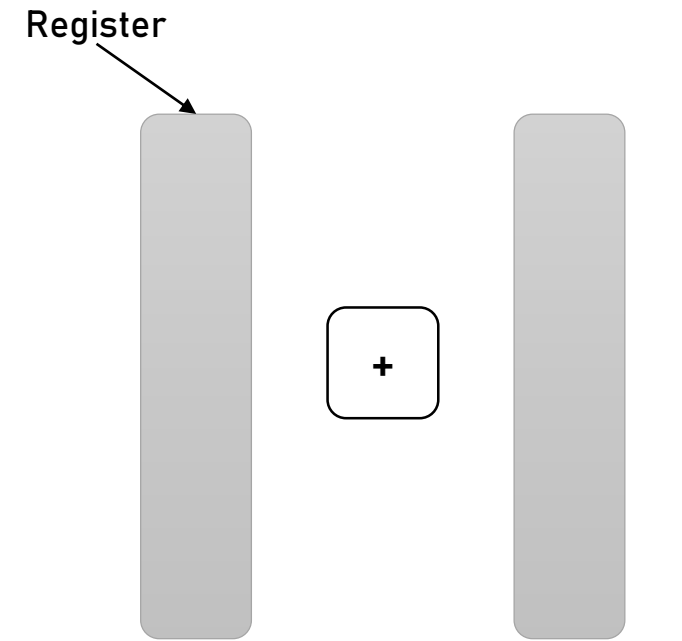
SIMD



[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

- Knight Landing, 2016
~
- Ice Lake, 2019

AVX-512 Subset	F	CD	ER	PF	4FMAPS	4VNNIW	VL	DQ	BW	IFMA	VBMI	VBMI2	VPOPCNTDQ	BITALG	VNNI	VPCLMULQDQ	GFNI	VAES					
Intel Knights Landing (2016)	Yes		Yes		No																		
Intel Knights Mill (2017)					Yes		No				Yes		No										
Intel Skylake-SP, Skylake-X (2017)			No				Yes				No												
Intel Cannon Lake (2018)											Yes		No										
Intel Cascade Lake-SP (2019)											No						Yes		No				
Intel Ice Lake (2019)											Yes												

[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



Technologies

- ☐ MMX
- ☐ SSE
- ☐ SSE2
- ☐ SSE3
- ☐ SSSE3
- ☐ SSE4.1
- ☐ SSE4.2
- ☒ AVX
- ☒ AVX2
- ☒ FMA
- ☒ AVX-512
- ☐ KNC
- ☐ SVML
- ☐ Other

Categories

- ☐ Application-Targeted
- ☐ Arithmetic
- ☐ Bit Manipulation
- ☐ Cast
- ☐ Compare
- ☐ Convert
- ☐ Counting

The Intel Intrinsics Guide is an interactive reference tool for Intel intrinsic instructions, which are C style functions that provide access to many Intel instructions - including Intel® SSE, AVX, AVX-512, and more - without the need to write assembly code. ✕

 ?

```

void _mm_2intersect_epi32 (__m128i a, __m128i b, __mmask8* k1, __mmask8* k2)          vp2intersectd
void _mm256_2intersect_epi32 (__m256i a, __m256i b, __mmask8* k1, __mmask8* k2)      vp2intersectd
void _mm512_2intersect_epi32 (__m512i a, __m512i b, __mmask16* k1, __mmask16* k2)     vp2intersectd
void _mm_2intersect_epi64 (__m128i a, __m128i b, __mmask8* k1, __mmask8* k2)          vp2intersectq
void _mm256_2intersect_epi64 (__m256i a, __m256i b, __mmask8* k1, __mmask8* k2)      vp2intersectq
void _mm512_2intersect_epi64 (__m512i a, __m512i b, __mmask8* k1, __mmask8* k2)      vp2intersectq
__m512i _mm512_4dpwssd_epi32 (__m512i src, __m512i a0, __m512i a1, __m512i a2, __m512i a3,
__m128i * b)                                vp4dpwssd
__m512i _mm512_mask_4dpwssd_epi32 (__m512i src, __mmask16 k, __m512i a0, __m512i a1, __m512i a2,
__m512i a3, __m128i * b)                    vp4dpwssd
__m512i _mm512_maskz_4dpwssd_epi32 (__mmask16 k, __m512i src, __m512i a0, __m512i a1, __m512i a2,
__m512i a3, __m128i * b)                    vp4dpwssd
__m512i _mm512_4dpwssds_epi32 (__m512i src, __m512i a0, __m512i a1, __m512i a2, __m512i a3,
__m128i * b)                                vp4dpwssds
__m512i _mm512_mask_4dpwssds_epi32 (__m512i src, __mmask16 k, __m512i a0, __m512i a1, __m512i
a2, __m512i a3, __m128i * b)                vp4dpwssds
__m512i _mm512_maskz_4dpwssds_epi32 (__mmask16 k, __m512i src, __m512i a0, __m512i a1, __m512i
a2, __m512i a3, __m128i * b)                vp4dpwssds

```

[그림] Intrinsics

Multiplication

- Multiplication

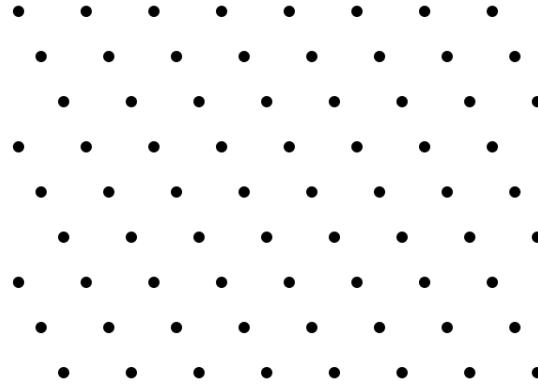
$$(x^1 + 3x^2 + 2x^3) \times (x^1 + 2x^2 + 3x^3)$$

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

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

Lattice Problem

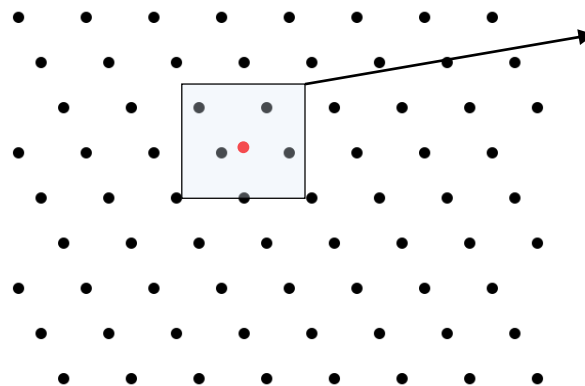
- Lattice



[그림] Vector Space

- LWE (Learning With Error)

- $B = A * S + E$



빨간 벡터가 어느 벡터로부터 왔는가?

[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 $sk = (\mathbf{s} || \mathbf{u})$

```
1:  $Seed_a \xleftarrow{\$} \{0, 1\}^{256}$   
2:  $\mathbf{a} \leftarrow \text{SHAKE256}(Seed_a, n/8)$   
3:  $\mathbf{s} \xleftarrow{\$} \text{HWT}_n(h_s), \mathbf{u} \xleftarrow{\$} \{0, 1\}^n, \mathbf{e} \leftarrow \psi_{cb}^n$   
4:  $\mathbf{b} \leftarrow -\mathbf{a} * \mathbf{s} + \mathbf{e}$   
5:  $pk \leftarrow (Seed_a || \mathbf{b}), sk \leftarrow (\mathbf{s} || \mathbf{u})$   
6: return  $pk, sk$ 
```

[그림] LizarMong KeyGen Algorithm

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

$s = \{0, 0, \dots, 0\}$
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

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

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

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

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

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

$$s = -1x^{idx}$$



$$s = 0x^{idx}$$

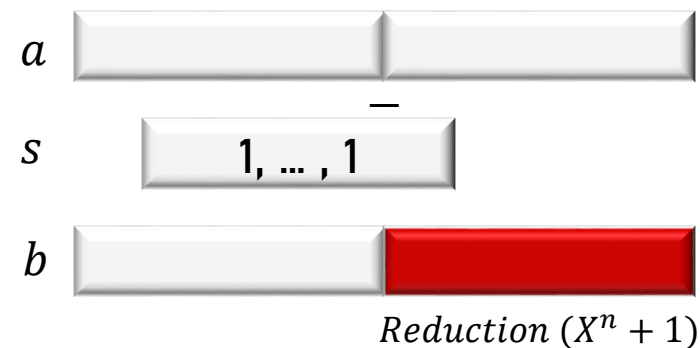
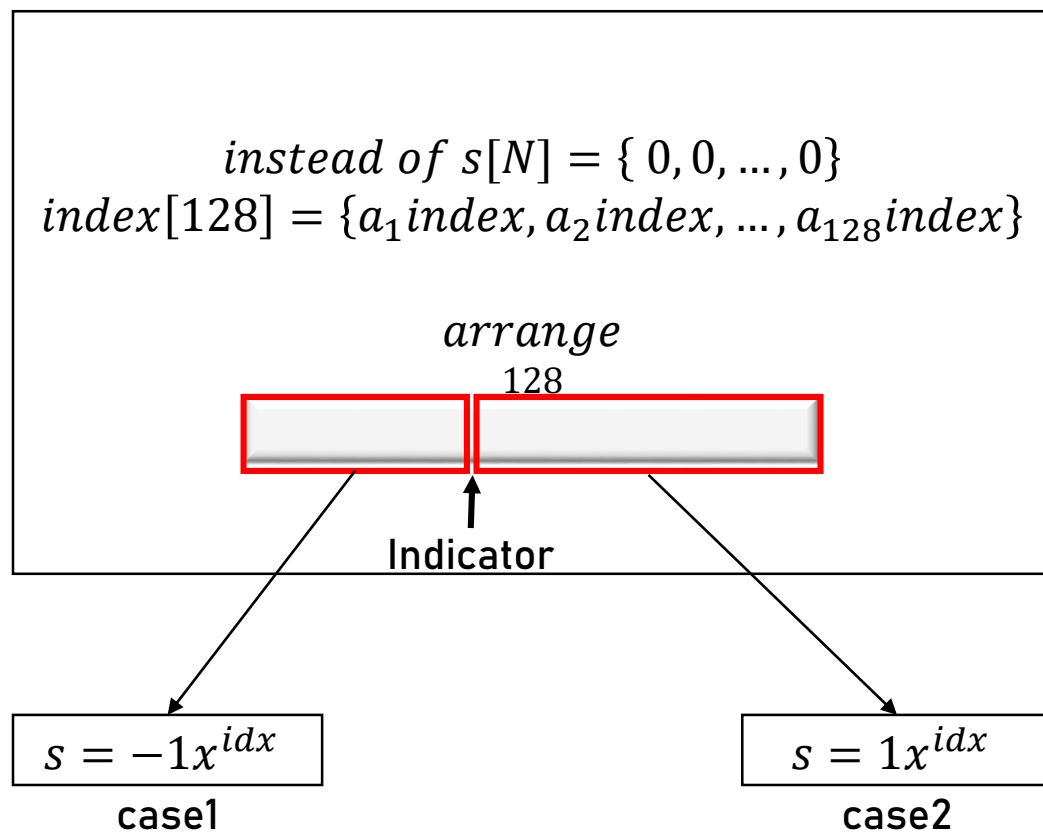


$$s = 1x^{idx}$$



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

x256	x256	x256	x256
y256	y256	y256	y256
z256	z256	z256	z256

x256	x256	x256	x256
y256	y256	y256	y256
z256	z256	z256	z256

x256	x256	x256	x256
y256	y256	y256	y256
z256	z256	z256	z256