Vector Homomorphic Encryption Accelerator

CSEE4840 Final Project

Lanxiang Hu, Liqin Zhang, Enze Chen

Outline

- Introduction to Homomorphic Encryption Scheme.
- System Block Diagram.
- Theory.
    - Cryptographical operations.
    - Encrypted domain operations.
- Hardware Implementation and Simulation.
- Software Implementation.
- Hardware-Software Interface.
- Challenge and Conclusion.

Introduction to Homomorphic Encryption Scheme

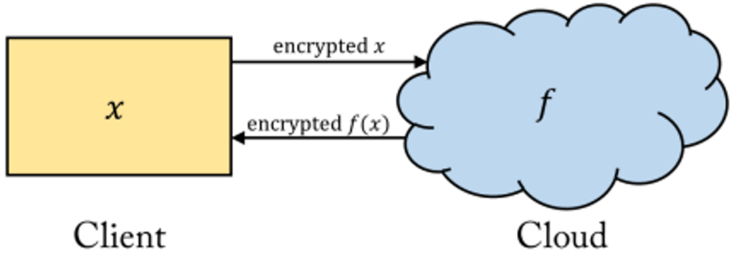


Fig1. Most Homomorphic Encryption schemes: The cloud has access to the function f, and the client sends encrypted x to the cloud for computation.

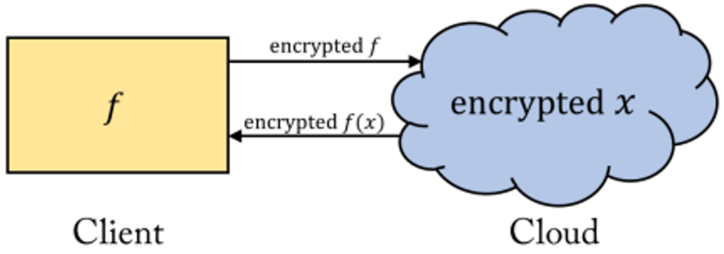
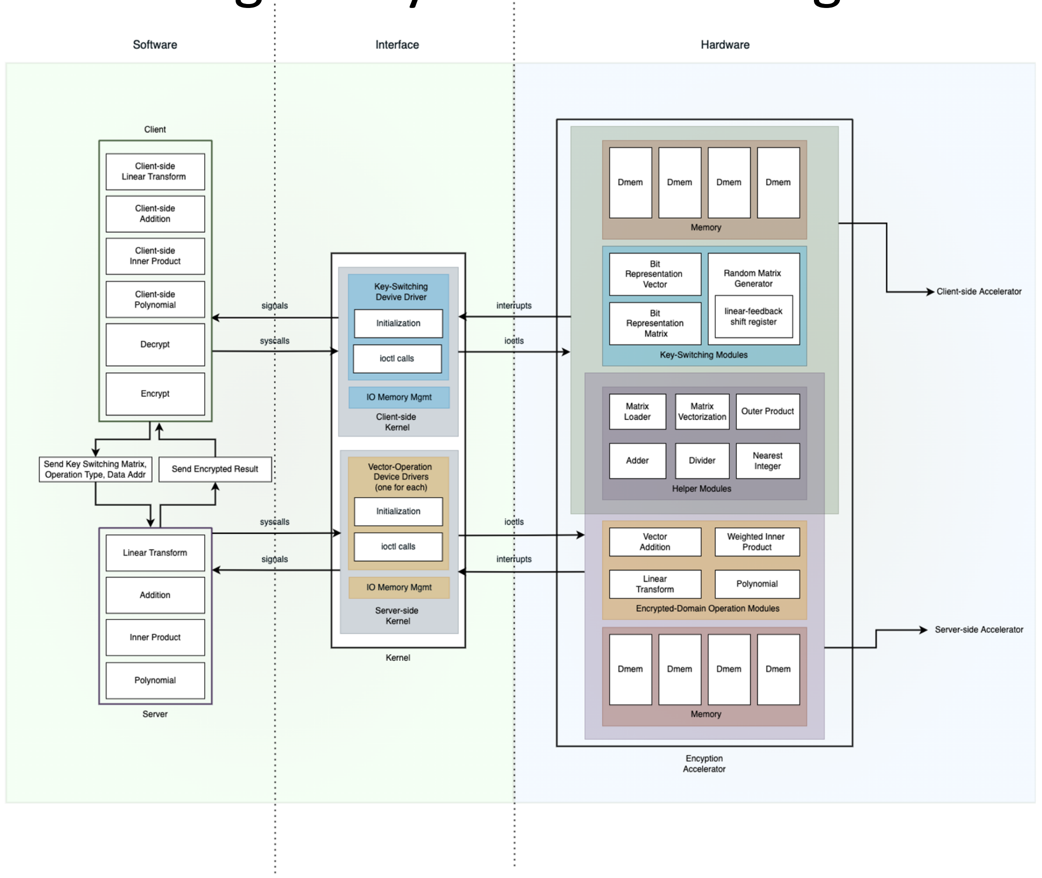


Fig2. The scheme used in our project. The cloud computes f(x) without knowing either x or f(·)

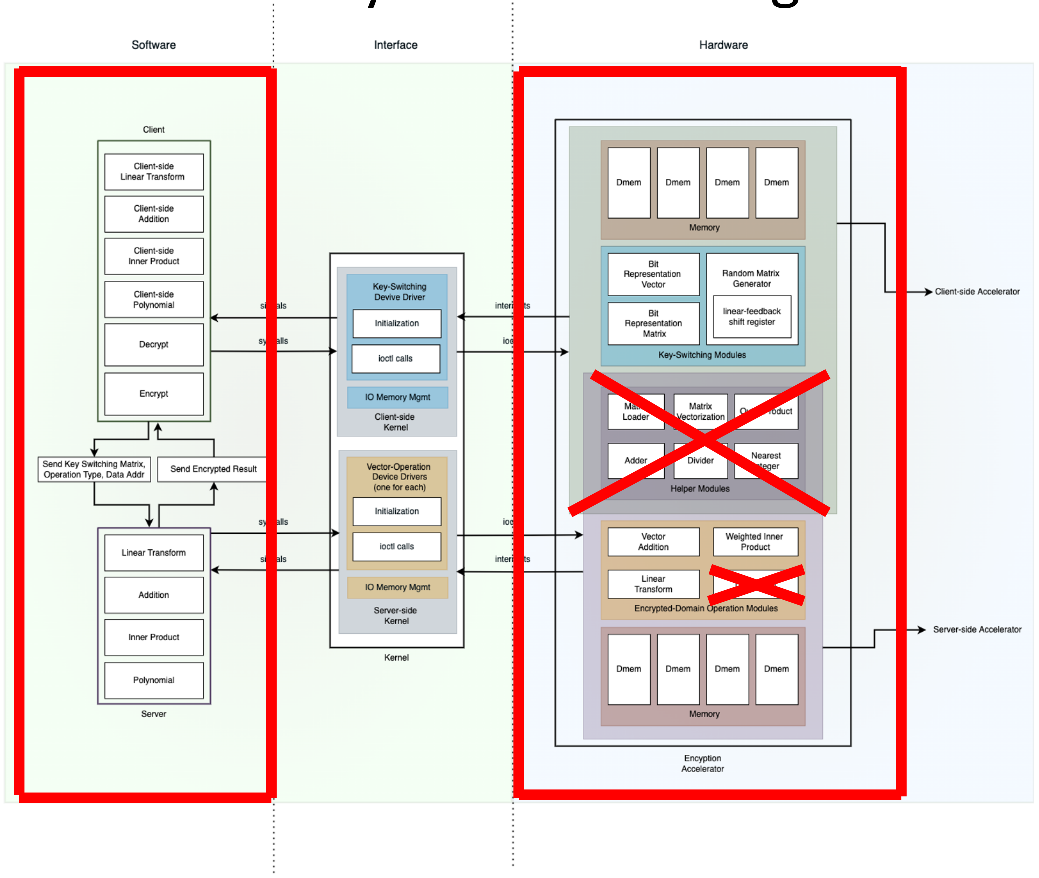
System Block Diagram

$$\begin{tikzcd}
\text{User-space functions \& API} \arrow[d, shift right] \\
\text{Kernel-space functions} \\ \text{Device drivers} \arrow[u, shift right] \arrow[d, shift right] \\
\text{Hardware} \\ \text{(Computational units)} \arrow[u, shift right]
\end{tikzcd}$$

Original System Block Diagram



Final System Block Diagram



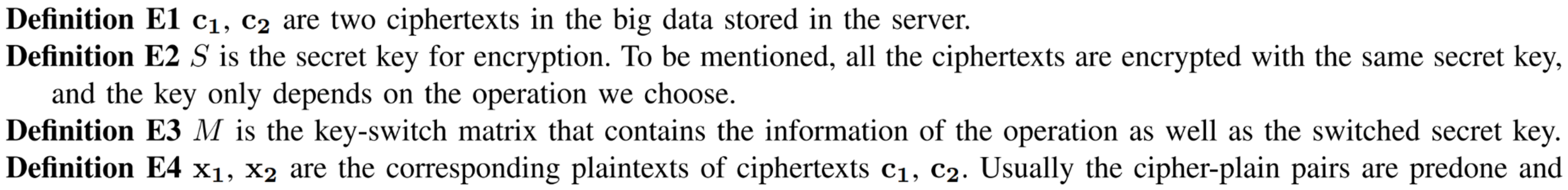
Theory — Overview

- **Cryptographical Operations:**
  - Encryption: c = E(x), choose c such that Sc = wx + e, S is secret key.
  - Decryption: x = D(c), decrypt c with S, x = int(Sc/w).
  - Key Switching:
    - Switching Secret key from S to a new key S' = [I,T] such that Sc = S'c'.
    - Return Key Switching Matrix M. Key Switching Matrix M encodes computational details.
    - Send M to server.
    - Server simply uses M to carry out computation by performing linear transformation to c.

- **Encrypted Domain (Integer Vector) Operations:**
  - Addition of two vectors.
  - Linear Transformation.
  - Weighted inner product.

Theory — Cryptography

- Security: Arithmetic Logic Units perform computations in encrypted domain, and the results can be only be decrypted by the client with the secret key.
- Application Scenarios: Without direct access to ciphertext in the cloud, the client can get computational results while the cloud server is agnostic about computational details.



Definition E1 $\mathbf{c_1}$, $\mathbf{c_2}$ are two ciphertexts in the big data stored in the server.

Definition E2 S is the secret key for encryption. To be mentioned, all the ciphertexts are encrypted with the same secret key, and the key only depends on the operation we choose.

Definition E3 M is the key-switch matrix that contains the information of the operation as well as the switched secret key.

Definition E4 $\mathbf{x_1}$, $\mathbf{x_2}$ are the corresponding plaintexts of ciphertexts $\mathbf{c_1}$, $\mathbf{c_2}$. Usually the cipher-plain pairs are predone and

Theory — Encrypted Domain Operations

ADDITION

$$S(\mathbf{c_1} + \mathbf{c_2}) = w(\mathbf{x_1} + \mathbf{x_2}) + (\mathbf{e_1} + \mathbf{e_2})$$

SOLUTION

Client

$$\text{Keep S the same. Send } \mathbf{c_1}, \mathbf{c_2}$$

Server

$$\mathbf{c}' = \mathbf{c_1} + \mathbf{c_2}$$

Theory — Encrypted Domain Operations

LINEAR TRANSFORMATION

$$(GS)\,\mathbf{c} = wG\mathbf{x} + G\mathbf{e}$$

SOLUTION

Client

$$\text{Create M for } s' = GS\text{, send } \mathbf{c}\text{, M}$$

Server

$$\mathbf{c}' = M\mathbf{c}$$

Theory — Encrypted Domain Operations

WEIGHTED INNER PRODUCT

$$h = \mathbf{x}_1^T H \mathbf{x}_2$$

$$\mathbf{x}_1^T H \mathbf{x}_2 = \text{vec}\left(M\right)^T \text{vec}\left(\mathbf{x}_1 \mathbf{x}_2^T\right)$$

$$\text{vec}\left(S^T H S\right)^T \left\lfloor \frac{\text{vec}\left(\mathbf{c}_1 \mathbf{c}_2^T\right)}{w} \right\rceil = w \mathbf{x}_1 H \mathbf{x}_2 + e$$

SOLUTION

Client $\text{Create } \mathbf{M} \text{ for } S' = \text{vec}\left(S^T H S\right)^T. \text{ Send } \mathbf{M}, \omega, \mathbf{c_1}, \mathbf{c_2}$

Server $\mathbf{c}'' = M \left\lfloor \frac{\text{vec}\left(\mathbf{c}_1 \mathbf{c}_2^T\right)}{w} \right\rceil$

Theory — Encrypted Domain Operations

POLYNOMIAL

$$\mathbf{x}_p = [1, x_1, x_2, ..., x_n]^T$$

$$\mathbf{c}' := [w, c_1, ..., c_n]^T \qquad S' := \begin{bmatrix} 1 & 0 \\ 0 & S \end{bmatrix} \qquad h = \mathbf{x_p}^T H \mathbf{x_p}$$

SOLUTION

Client $\quad$ Create M for $S'' = \text{vec}\left(S'^T H S'\right)^T$. Send M,$\omega$,$\mathbf{c}'$

Server

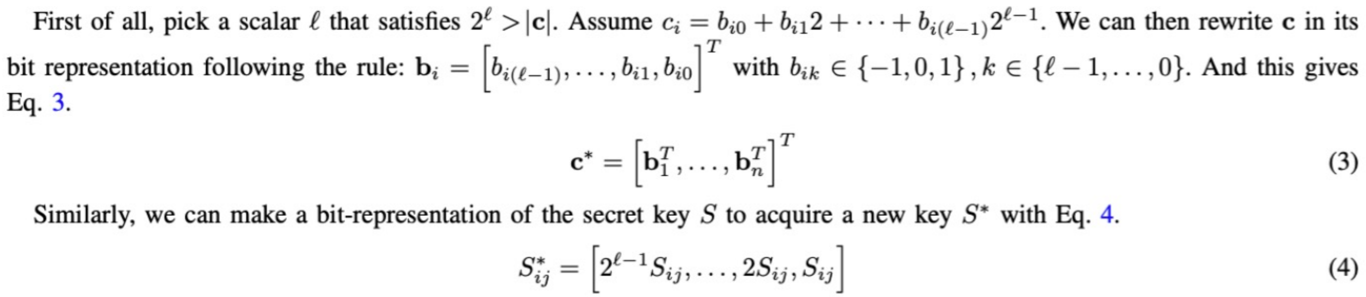
$$\mathbf{c}'' = M \left\lfloor \frac{\text{vec}\left(\mathbf{c}'\mathbf{c}'^T\right)}{w} \right\rceil$$

Hardware Implementation and Simulation

- Key-Switching Modules:
  - Take bit-representation of a vector (at most 8-element wide, 32-bit each).
  - Take bit-representation of a matrix (at most 8 by 8 in size, 32-bit each).
  - Get a random matrix with integer entries (at most 8 by 8 in size, 16-bit each).
  - Get a noise matrix with small integer entries (at most 8 by 8 in size, 4-bit each).
- Encrypted-Domain Computational Modules:
  - Vector addition (at most 16-element wide, 32-bit each).
  - Linear Transformation (supports linear operator of at most 16 by 16 in size, 32-bit each).
  - Weighted Inner Product (supports linear operator of at most 16 by 16 in size, 32-bit each).

Implementation and Simulation — Key-Switching Modules

BIT REPRESETATION OF VECTOR : convert a vector into its bit representation.



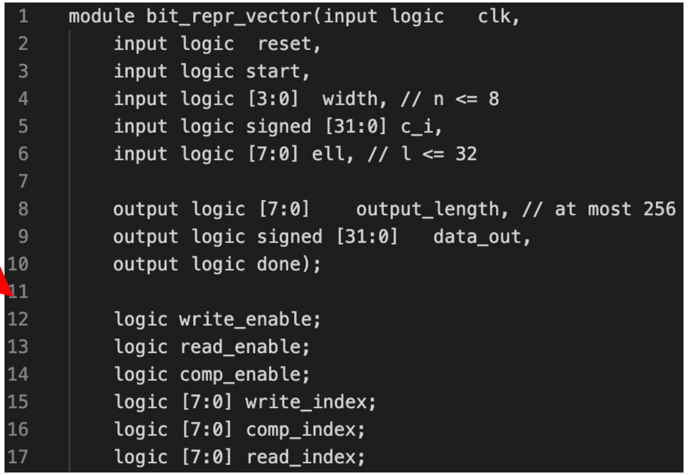
First of all, pick a scalar ℓ that satisfies $2^\ell > |\mathbf{c}|$. Assume $c_i = b_{i0} + b_{i1}2 + \cdots + b_{i(\ell-1)}2^{\ell-1}$. We can then rewrite $\mathbf{c}$ in its bit representation following the rule: $\mathbf{b}_i = \left[b_{i(\ell-1)}, \ldots, b_{i1}, b_{i0}\right]^T$ with $b_{ik} \in \{-1, 0, 1\}, k \in \{\ell-1, \ldots, 0\}$. And this gives Eq. 3.

$$\mathbf{c}^* = \left[\mathbf{b}_1^T, \ldots, \mathbf{b}_n^T\right]^T \tag{3}$$

Similarly, we can make a bit-representation of the secret key S to acquire a new key S^* with Eq. 4.

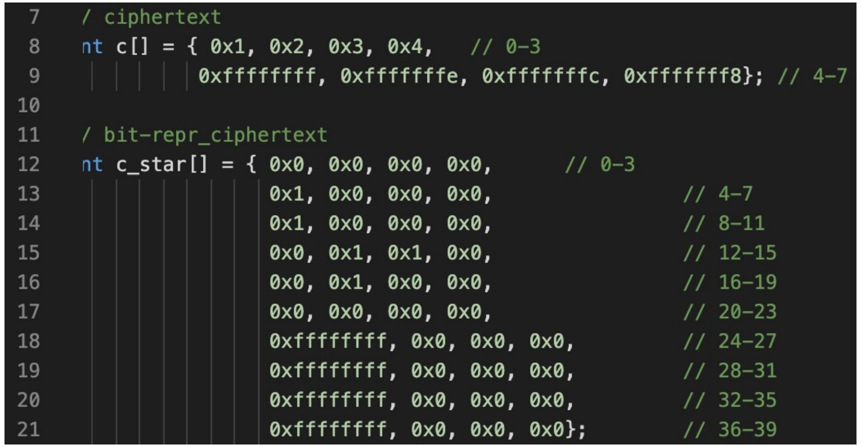
$$S_{ij}^* = \left[2^{\ell-1}S_{ij}, \ldots, 2S_{ij}, S_{ij}\right] \tag{4}$$

Moore machine

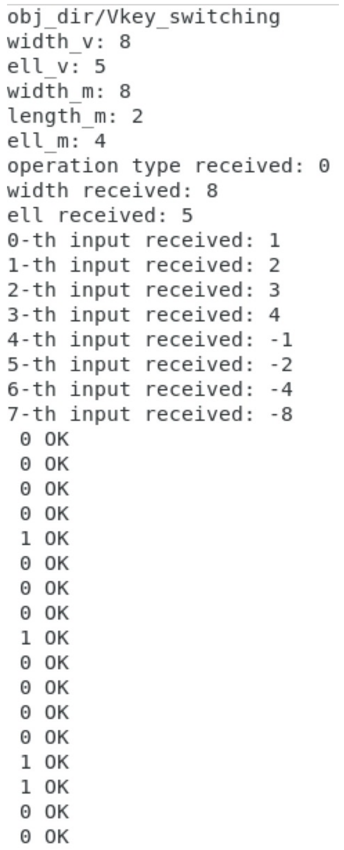


```
module bit_repr_vector(input logic   clk,
    input logic  reset,
    input logic start,
    input logic [3:0]  width, // n <= 8
    input logic signed [31:0] c_i,
    input logic [7:0] ell, // l <= 32

    output logic [7:0]    output_length, // at most 256
    output logic signed [31:0]   data_out,
    output logic done);

    logic write_enable;
    logic read_enable;
    logic comp_enable;
    logic [7:0] write_index;
    logic [7:0] comp_index;
    logic [7:0] read_index;
```

Implementation and Simulation — Key-Switching Modules



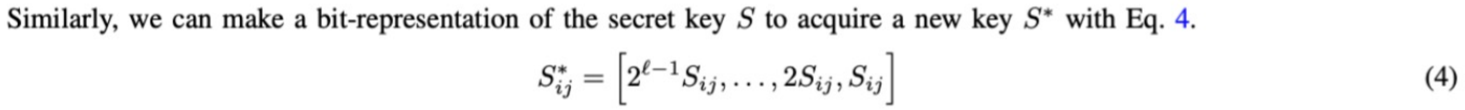
```
/ ciphertext
nt c[] = { 0x1, 0x2, 0x3, 0x4,   // 0-3
                   0xffffffff, 0xfffffffe, 0xfffffffc, 0xfffffff8}; // 4-7

/ bit-repr_ciphertext
nt c_star[] = { 0x0, 0x0, 0x0, 0x0,      // 0-3
                0x1, 0x0, 0x0, 0x0,                // 4-7
                0x1, 0x0, 0x0, 0x0,                // 8-11
                0x0, 0x1, 0x1, 0x0,                // 12-15
                0x0, 0x1, 0x0, 0x0,                // 16-19
                0x0, 0x0, 0x0, 0x0,                // 20-23
                0xffffffff, 0x0, 0x0, 0x0,         // 24-27
                0xffffffff, 0x0, 0x0, 0x0,         // 28-31
                0xffffffff, 0x0, 0x0, 0x0,         // 32-35
                0xffffffff, 0x0, 0x0, 0x0};        // 36-39
```



```
obj_dir/Vkey_switching
width_v: 8
ell_v: 5
width_m: 8
length_m: 2
ell_m: 4
operation type received: 0
width received: 8
ell received: 5
0-th input received: 1
1-th input received: 2
2-th input received: 3
3-th input received: 4
4-th input received: -1
5-th input received: -2
6-th input received: -4
7-th input received: -8
 0 OK
 0 OK
 0 OK
 0 OK
 1 OK
 0 OK
 0 OK
 0 OK
 1 OK
 0 OK
 0 OK
 0 OK
 0 OK
 1 OK
 1 OK
 0 OK
 0 OK
```

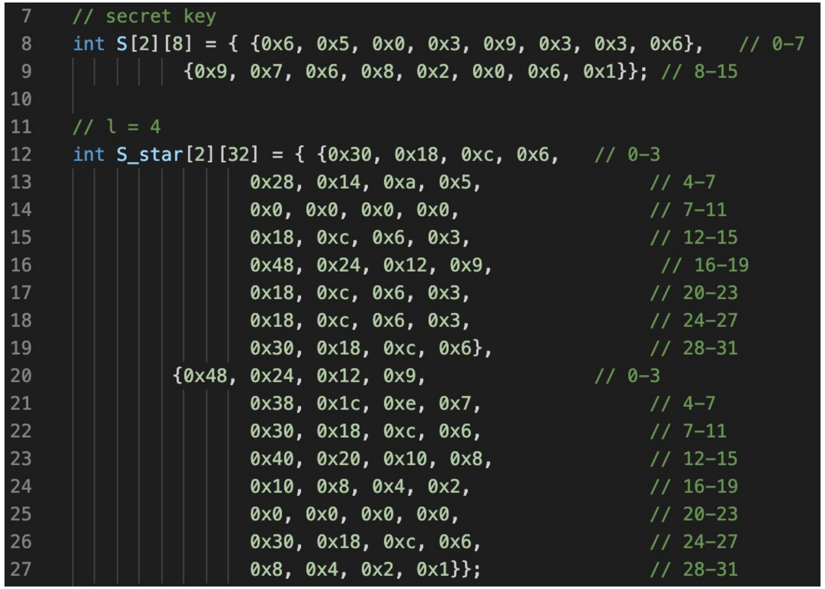
Implementation and Simulation — Key-Switching Modules

BIT REPRESETATION OF MATRIX : convert a matrix into its bit representation.

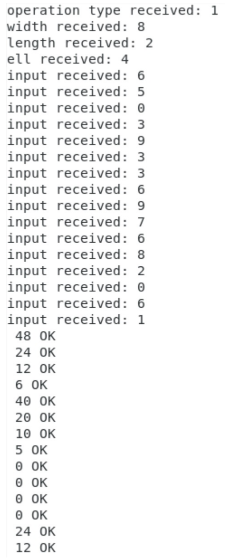


Similarly, we can make a bit-representation of the secret key S to acquire a new key S^* with Eq. 4.

$$S^*_{ij} = \left[2^{\ell-1}S_{ij}, \ldots, 2S_{ij}, S_{ij}\right] \tag{4}$$



```
// secret key
int S[2][8] = { {0x6, 0x5, 0x0, 0x3, 0x9, 0x3, 0x3, 0x6},   // 0-7
                {0x9, 0x7, 0x6, 0x8, 0x2, 0x0, 0x6, 0x1}}; // 8-15

// l = 4
int S_star[2][32] = { {0x30, 0x18, 0xc, 0x6,   // 0-3
                       0x28, 0x14, 0xa, 0x5,             // 4-7
                       0x0, 0x0, 0x0, 0x0,               // 7-11
                       0x18, 0xc, 0x6, 0x3,              // 12-15
                       0x48, 0x24, 0x12, 0x9,             // 16-19
                       0x18, 0xc, 0x6, 0x3,              // 20-23
                       0x18, 0xc, 0x6, 0x3,              // 24-27
                       0x30, 0x18, 0xc, 0x6},            // 28-31
                      {0x48, 0x24, 0x12, 0x9,            // 0-3
                       0x38, 0x1c, 0xe, 0x7,             // 4-7
                       0x30, 0x18, 0xc, 0x6,             // 7-11
                       0x40, 0x20, 0x10, 0x8,            // 12-15
                       0x10, 0x8, 0x4, 0x2,              // 16-19
                       0x0, 0x0, 0x0, 0x0,               // 20-23
                       0x30, 0x18, 0xc, 0x6,             // 24-27
                       0x8, 0x4, 0x2, 0x1}};             // 28-31
```

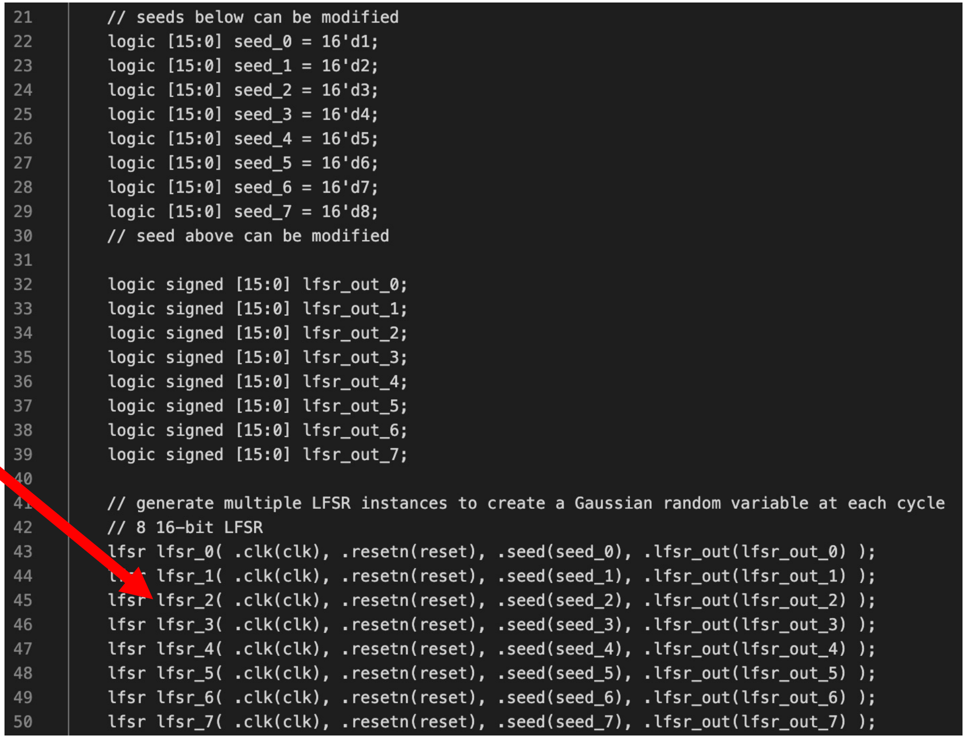


```
operation type received: 1
width received: 8
length received: 2
ell received: 4
input received: 6
input received: 5
input received: 0
input received: 3
input received: 9
input received: 3
input received: 3
input received: 6
input received: 9
input received: 7
input received: 6
input received: 8
input received: 2
input received: 0
input received: 6
input received: 1
 48 OK
 24 OK
 12 OK
 6 OK
 40 OK
 20 OK
 10 OK
 5 OK
 0 OK
 0 OK
 0 OK
 0 OK
 24 OK
 12 OK
```

Implementation and Simulation — Key-Switching Modules

GET RANDOM MATRIX: get an integer-valued random matrix.

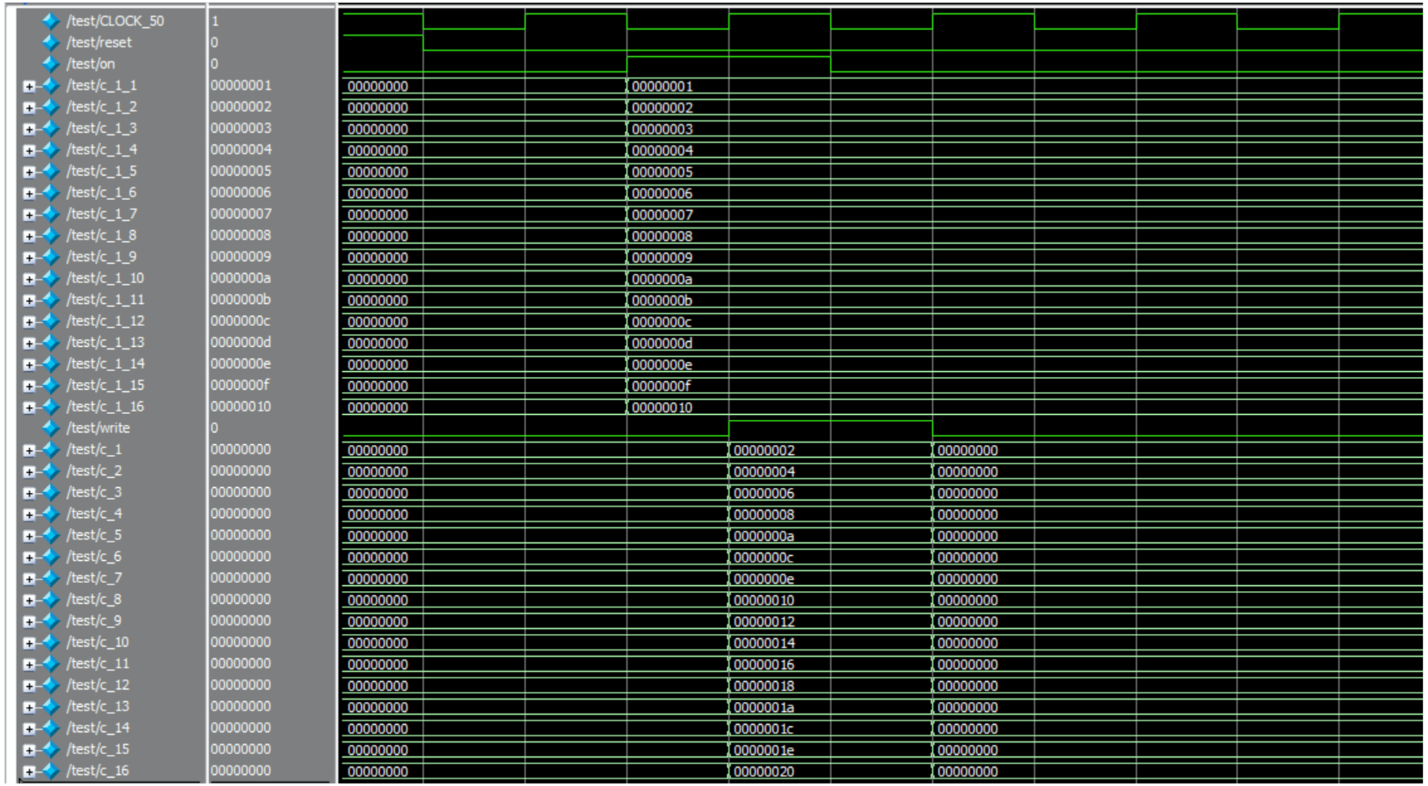
Moore machine with LFSR pseudorandom number generator.



```
// seeds below can be modified
logic [15:0] seed_0 = 16'd1;
logic [15:0] seed_1 = 16'd2;
logic [15:0] seed_2 = 16'd3;
logic [15:0] seed_3 = 16'd4;
logic [15:0] seed_4 = 16'd5;
logic [15:0] seed_5 = 16'd6;
logic [15:0] seed_6 = 16'd7;
logic [15:0] seed_7 = 16'd8;
// seed above can be modified

logic signed [15:0] lfsr_out_0;
logic signed [15:0] lfsr_out_1;
logic signed [15:0] lfsr_out_2;
logic signed [15:0] lfsr_out_3;
logic signed [15:0] lfsr_out_4;
logic signed [15:0] lfsr_out_5;
logic signed [15:0] lfsr_out_6;
logic signed [15:0] lfsr_out_7;

// generate multiple LFSR instances to create a Gaussian random variable at each cycle
// 8 16-bit LFSR
lfsr lfsr_0( .clk(clk), .resetn(reset), .seed(seed_0), .lfsr_out(lfsr_out_0) );
lfsr lfsr_1( .clk(clk), .resetn(reset), .seed(seed_1), .lfsr_out(lfsr_out_1) );
lfsr lfsr_2( .clk(clk), .resetn(reset), .seed(seed_2), .lfsr_out(lfsr_out_2) );
lfsr lfsr_3( .clk(clk), .resetn(reset), .seed(seed_3), .lfsr_out(lfsr_out_3) );
lfsr lfsr_4( .clk(clk), .resetn(reset), .seed(seed_4), .lfsr_out(lfsr_out_4) );
lfsr lfsr_5( .clk(clk), .resetn(reset), .seed(seed_5), .lfsr_out(lfsr_out_5) );
lfsr lfsr_6( .clk(clk), .resetn(reset), .seed(seed_6), .lfsr_out(lfsr_out_6) );
lfsr lfsr_7( .clk(clk), .resetn(reset), .seed(seed_7), .lfsr_out(lfsr_out_7) );
```

Implementation and Simulation — Encrypted Domain Operations

ADDITION : Each time add n elements of input.

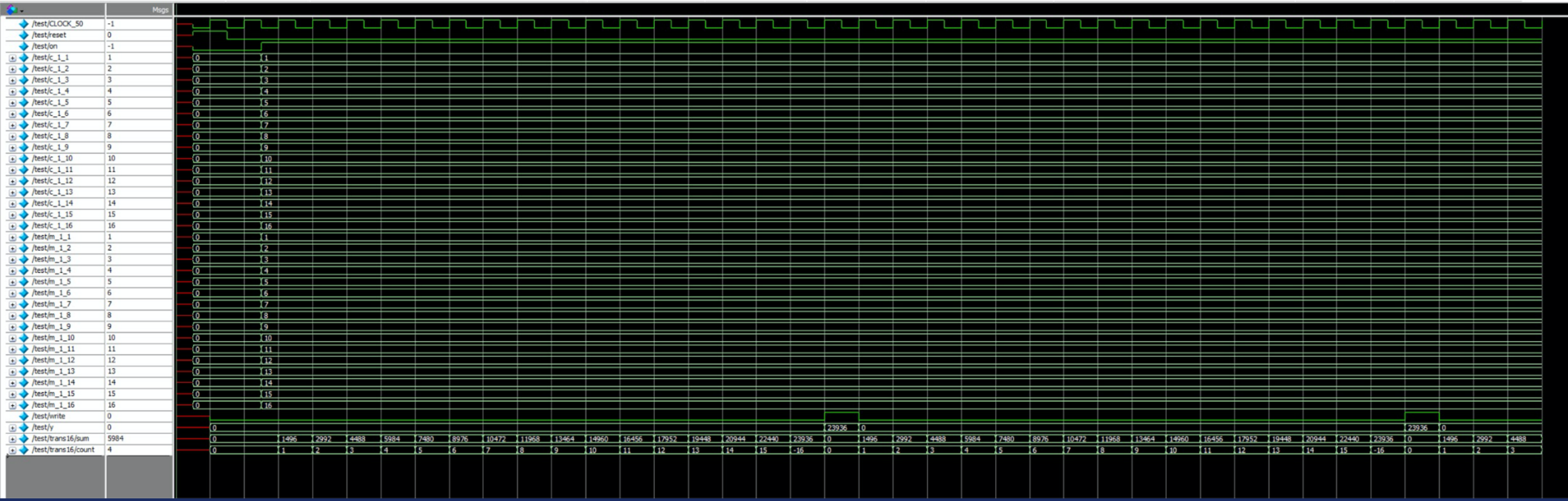


```
/test/CLOCK_50   1
/test/reset      0
/test/on         0
/test/c_1_1      00000001   00000000  00000001
/test/c_1_2      00000002   00000000  00000002
/test/c_1_3      00000003   00000000  00000003
/test/c_1_4      00000004   00000000  00000004
/test/c_1_5      00000005   00000000  00000005
/test/c_1_6      00000006   00000000  00000006
/test/c_1_7      00000007   00000000  00000007
/test/c_1_8      00000008   00000000  00000008
/test/c_1_9      00000009   00000000  00000009
/test/c_1_10     0000000a   00000000  0000000a
/test/c_1_11     0000000b   00000000  0000000b
/test/c_1_12     0000000c   00000000  0000000c
/test/c_1_13     0000000d   00000000  0000000d
/test/c_1_14     0000000e   00000000  0000000e
/test/c_1_15     0000000f   00000000  0000000f
/test/c_1_16     00000010   00000000  00000010
/test/write      0
/test/c_1        00000000   00000000  00000002  00000000
/test/c_2        00000000   00000000  00000004  00000000
/test/c_3        00000000   00000000  00000006  00000000
/test/c_4        00000000   00000000  00000008  00000000
/test/c_5        00000000   00000000  0000000a  00000000
/test/c_6        00000000   00000000  0000000c  00000000
/test/c_7        00000000   00000000  0000000e  00000000
/test/c_8        00000000   00000000  00000010  00000000
/test/c_9        00000000   00000000  00000012  00000000
/test/c_10       00000000   00000000  00000014  00000000
/test/c_11       00000000   00000000  00000016  00000000
/test/c_12       00000000   00000000  00000018  00000000
/test/c_13       00000000   00000000  0000001a  00000000
/test/c_14       00000000   00000000  0000001c  00000000
/test/c_15       00000000   00000000  0000001e  00000000
/test/c_16       00000000   00000000  00000020  00000000
```

Implementation and Simulation — Encrypted Domain Operations

LINEAR TRANSFORMATION :Each time do inner product of n elements of c and M, sum the result after a whole line has been calculated. Do several epochs until whole M is scanned.

https://drive.google.com/file/d/1cqC6TUnnxU2AczAaUOS2lxPEVCOCCsqM/view?usp=sharing



	Msgs
/test/CLOCK_50	-1
/test/reset	0
/test/on	-1
/test/c_1_1	1
/test/c_1_2	2
/test/c_1_3	3
/test/c_1_4	4
/test/c_1_5	5
/test/c_1_6	6
/test/c_1_7	7
/test/c_1_8	8
/test/c_1_9	9
/test/c_1_10	10
/test/c_1_11	11
/test/c_1_12	12
/test/c_1_13	13
/test/c_1_14	14
/test/c_1_15	15
/test/c_1_16	16
/test/m_1_1	1
/test/m_1_2	2
/test/m_1_3	3
/test/m_1_4	4
/test/m_1_5	5
/test/m_1_6	6
/test/m_1_7	7
/test/m_1_8	8
/test/m_1_9	9
/test/m_1_10	10
/test/m_1_11	11
/test/m_1_12	12
/test/m_1_13	13
/test/m_1_14	14
/test/m_1_15	15
/test/m_1_16	16
/test/write	0
/test/y	0
/test/trans16/sum	5984
/test/trans16/count	4

```
/test/y:            0 23936 0 23936 0
/test/trans16/sum:  0 1496 2992 4488 5984 7480 8976 10472 11968 13464 14960 16456 17952 19448 20944 22440 23936 0 1496 2992 4488 5984 7480 8976 10472 11968 13464 14960 16456 17952 19448 20944 22440 23936 0 1496 2992 4488
/test/trans16/count: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 -16 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 -16 0 1 2 3
```

Implementation and Simulation — Encrypted Domain Operations

‘Batch Size’ = n : Deal with n elements of input vectors at a time.

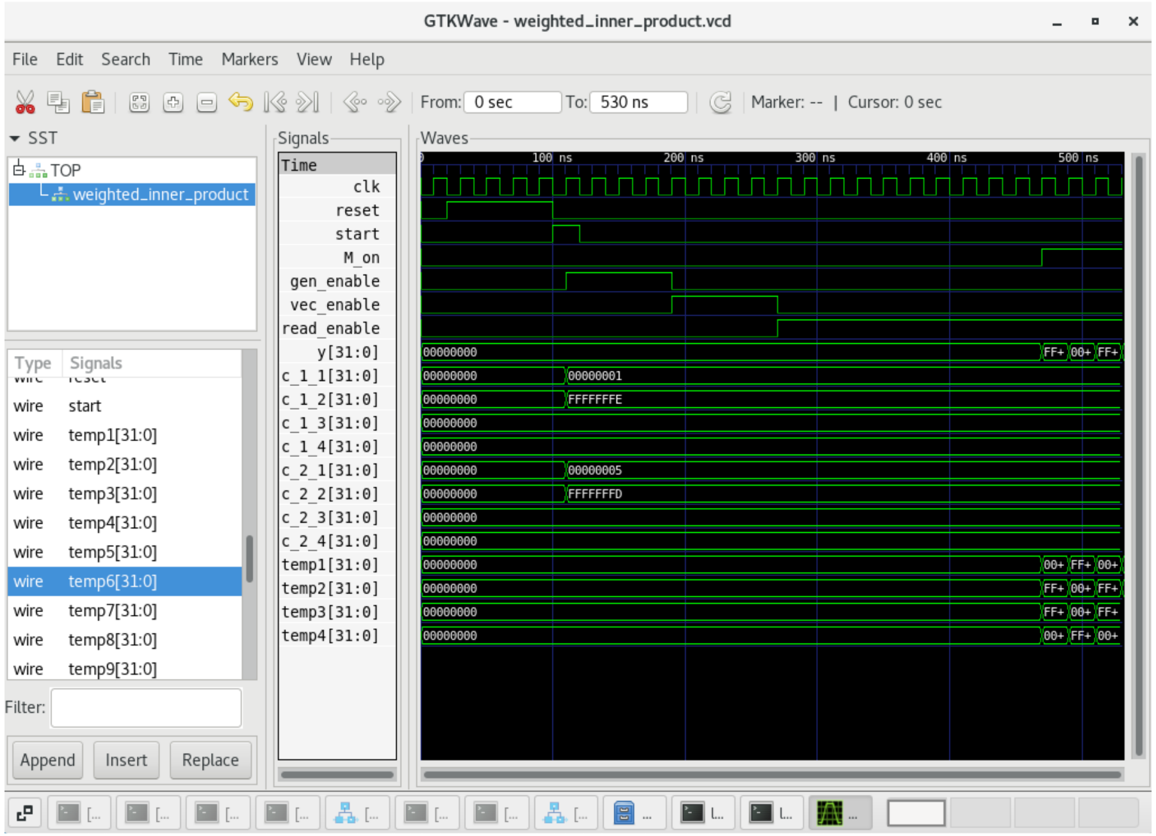
WEIGHTED INNER PRODUCT :

STAGE 1: Load input c1 and input c2. Do outer product of c1 and c2.

STAGE 2: Vectorize the output in STAGE 1. Divide the vector by w.

STAGE 3: Do linear transformation of the output, using same theory as in LINEAR TRANSFORMATION.

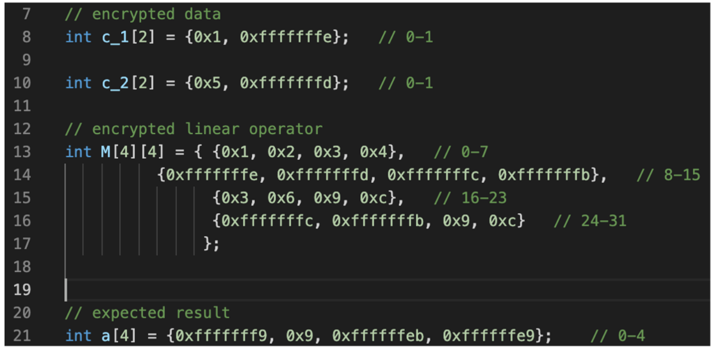
Implementation and Simulation — Encrypted Domain Operations



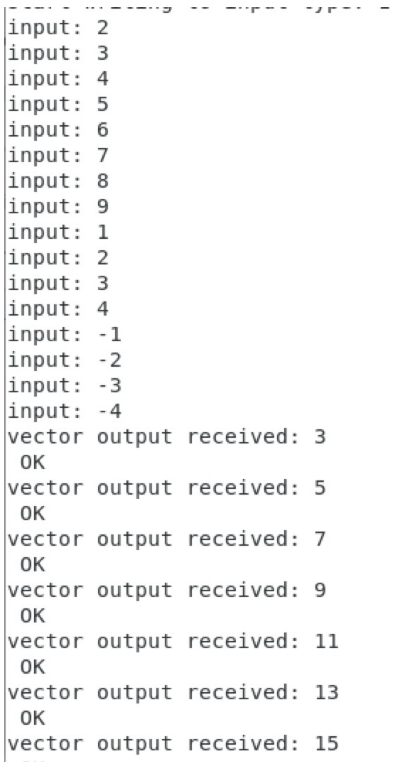
From: 0 sec To: 530 ns Marker: -- | Cursor: 0 sec

Type	Signals
wire	reset
wire	start
wire	temp1[31:0]
wire	temp2[31:0]
wire	temp3[31:0]
wire	temp4[31:0]
wire	temp5[31:0]
wire	temp6[31:0]
wire	temp7[31:0]
wire	temp8[31:0]
wire	temp9[31:0]

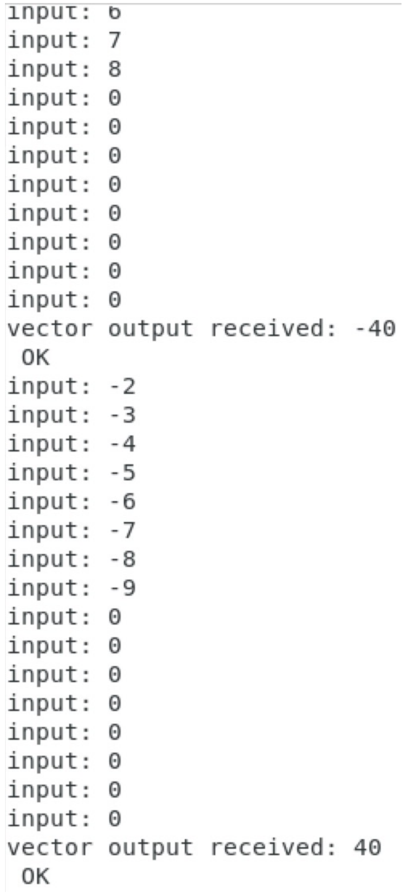
```
Time
clk
reset
start
M_on
gen_enable
vec_enable
read_enable
y[31:0]       00000000                         FF+ 00+ FF+
c_1_1[31:0]   00000000  00000001
c_1_2[31:0]   00000000  FFFFFFFE
c_1_3[31:0]   00000000
c_1_4[31:0]   00000000
c_2_1[31:0]   00000000  00000005
c_2_2[31:0]   00000000  FFFFFFFD
c_2_3[31:0]   00000000
c_2_4[31:0]   00000000
temp1[31:0]   00000000                         00+ FF+ 00+
temp2[31:0]   00000000                         FF+ 00+ FF+
temp3[31:0]   00000000                         FF+ 00+ FF+
temp4[31:0]   00000000                         00+ FF+ 00+
```



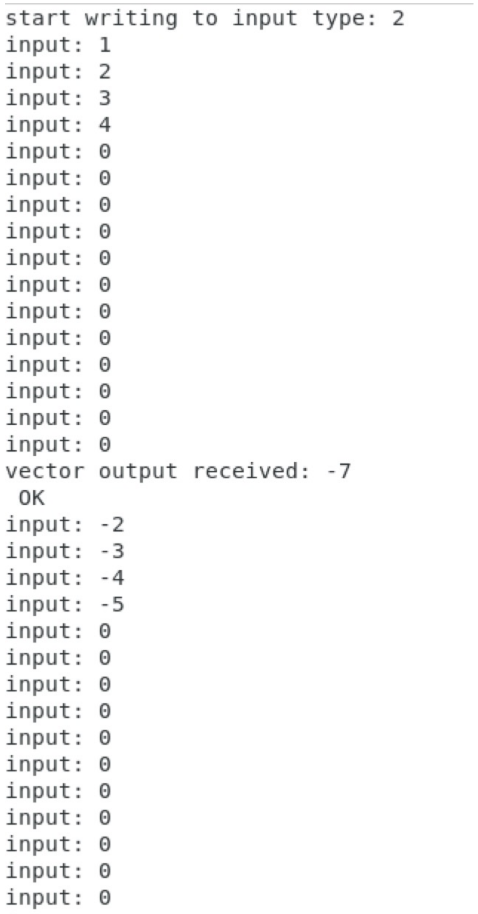
```
// encrypted data
int c_1[2] = {0x1, 0xfffffffe};   // 0-1

int c_2[2] = {0x5, 0xfffffffd};   // 0-1

// encrypted linear operator
int M[4][4] = { {0x1, 0x2, 0x3, 0x4},   // 0-7
                {0xfffffffe, 0xfffffffd, 0xfffffffc, 0xfffffffb},   // 8-15
                  {0x3, 0x6, 0x9, 0xc},   // 16-23
                  {0xfffffffc, 0xfffffffb, 0x9, 0xc}   // 24-31
                };


// expected result
int a[4] = {0xfffffff9, 0x9, 0xffffffeb, 0xffffffe9};   // 0-4
```



```
input: 2
input: 3
input: 4
input: 5
input: 6
input: 7
input: 8
input: 9
input: 1
input: 2
input: 3
input: 4
input: -1
input: -2
input: -3
input: -4
vector output received: 3
 OK
vector output received: 5
 OK
vector output received: 7
 OK
vector output received: 9
 OK
vector output received: 11
 OK
vector output received: 13
 OK
vector output received: 15
```



```
input: 6
input: 7
input: 8
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
vector output received: -40
 OK
input: -2
input: -3
input: -4
input: -5
input: -6
input: -7
input: -8
input: -9
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
vector output received: 40
 OK
```



```
start writing to input type: 2
input: 1
input: 2
input: 3
input: 4
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
vector output received: -7
 OK
input: -2
input: -3
input: -4
input: -5
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
input: 0
```

Software Implementation

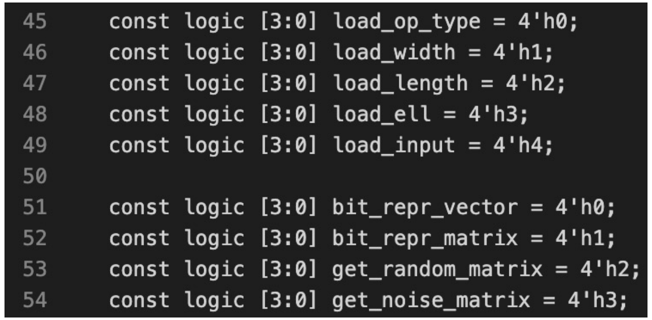
- Matrix operation library.
- Client-side operation library.
- Server-side operation library.
- Syscall library.
- Kernel code for device driver.

Software Implementation

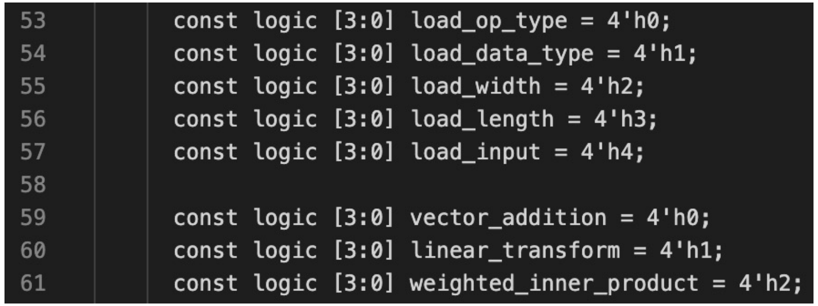
- Matrix operation library.
- Client-side operation library.
- Server-side operation library.
- Syscall library.
- Kernel code for device drivers.

Hardware-Software Interface

- AXI master-slave pair:
  - 32-bit write and read data.
  - Addresses control types of operation each write/read corresponds.
- As a results, control signals need to be sent off by the user as well (more complicated software coding).

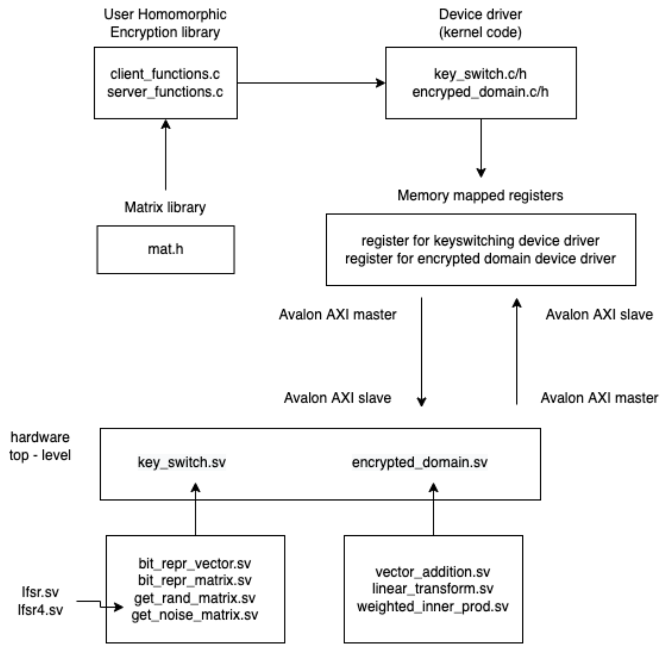


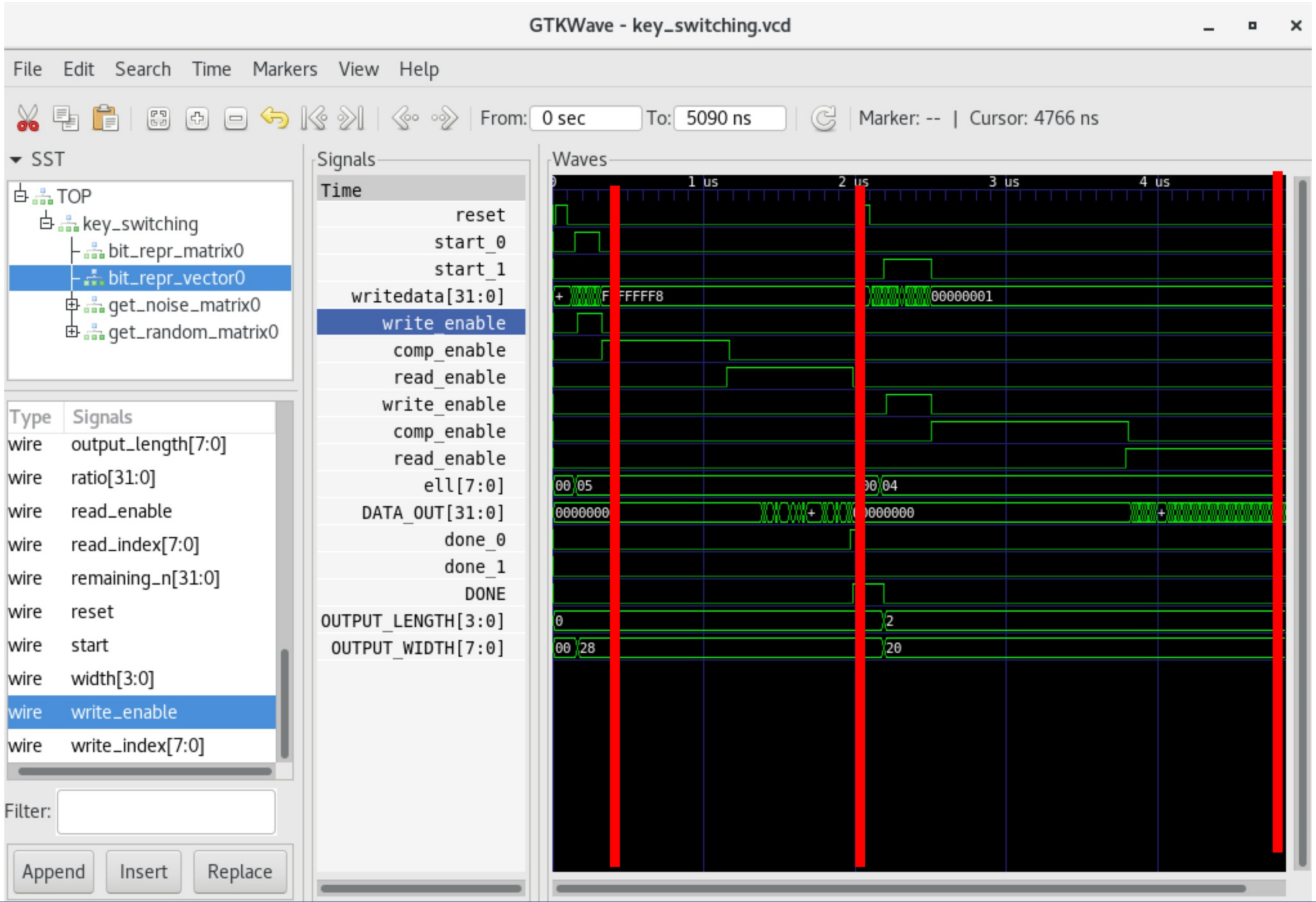
```
    const logic [3:0] load_op_type = 4'h0;
    const logic [3:0] load_width = 4'h1;
    const logic [3:0] load_length = 4'h2;
    const logic [3:0] load_ell = 4'h3;
    const logic [3:0] load_input = 4'h4;

    const logic [3:0] bit_repr_vector = 4'h0;
    const logic [3:0] bit_repr_matrix = 4'h1;
    const logic [3:0] get_random_matrix = 4'h2;
    const logic [3:0] get_noise_matrix = 4'h3;
```



```
            const logic [3:0] load_op_type = 4'h0;
            const logic [3:0] load_data_type = 4'h1;
            const logic [3:0] load_width = 4'h2;
            const logic [3:0] load_length = 4'h3;
            const logic [3:0] load_input = 4'h4;

            const logic [3:0] vector_addition = 4'h0;
            const logic [3:0] linear_transform = 4'h1;
            const logic [3:0] weighted_inner_product = 4'h2;
```

Hardware-Software Interface




GTKWave - key_switching.vcd

File Edit Search Time Markers View Help

From: 0 sec To: 5090 ns Marker: -- | Cursor: 4766 ns

SST
- TOP
  - key_switching
    - bit_repr_matrix0
    - bit_repr_vector0
    - get_noise_matrix0
    - get_random_matrix0

Type	Signals
wire	output_length[7:0]
wire	ratio[31:0]
wire	read_enable
wire	read_index[7:0]
wire	remaining_n[31:0]
wire	reset
wire	start
wire	width[3:0]
wire	write_enable
wire	write_index[7:0]

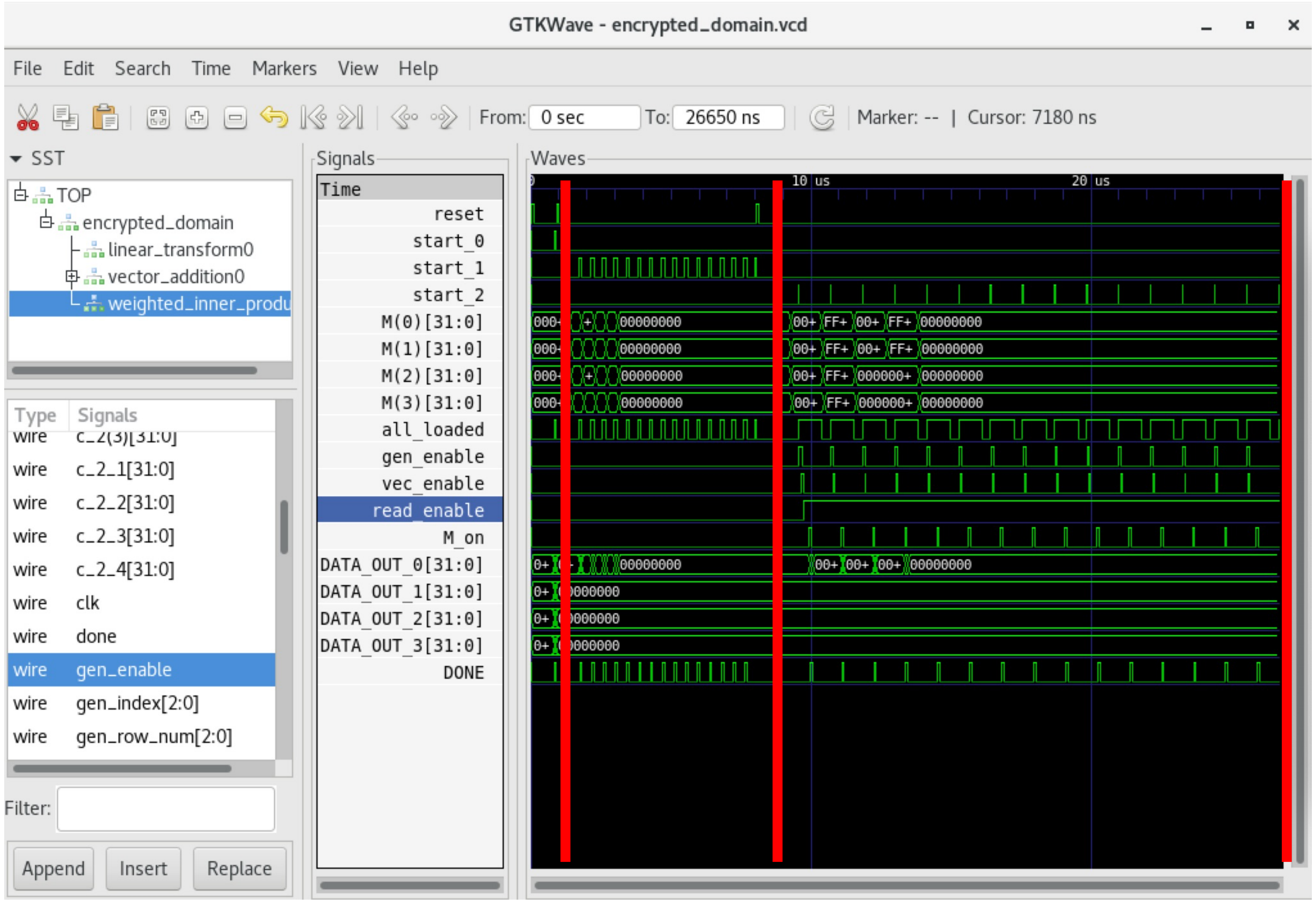
Filter:

Append Insert Replace

Signals: Time, reset, start_0, start_1, writedata[31:0], write_enable, comp_enable, read_enable, write_enable, comp_enable, read_enable, ell[7:0], DATA_OUT[31:0], done_0, done_1, DONE, OUTPUT_LENGTH[3:0], OUTPUT_WIDTH[7:0]

Vector Homomorphic Encryption Accelerator

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science



Type	Signals
wire	c_2(3)[31:0]
wire	c_2_1[31:0]
wire	c_2_2[31:0]
wire	c_2_3[31:0]
wire	c_2_4[31:0]
wire	clk
wire	done
wire	gen_enable
wire	gen_index[2:0]
wire	gen_row_num[2:0]

```
Time
reset
start_0
start_1
start_2
M(0)[31:0]
M(1)[31:0]
M(2)[31:0]
M(3)[31:0]
all_loaded
gen_enable
vec_enable
read_enable
M_on
DATA_OUT_0[31:0]
DATA_OUT_1[31:0]
DATA_OUT_2[31:0]
DATA_OUT_3[31:0]
DONE
```

Challenges

- Dimension scalability.
- Intermediate data caching.
- Interplay among different top-level external and submodule internal control signals.
- User-friendliness.
  - Need to manage control signal manually.

Lesson Learned

- Simplify pipeline logic for Avalon bus communication.
- Might be easier to use shared SDRAM rather than implementing memory blocks from scratch so that intermediate results can be more easily cached with SDRAM on hardware.
- Figure out what to put on registers is important and might make life much easier and avoid evoking too many syscalls.