```
#include <iostream>
#include <vector>
#include <abycore/aby/abyparty.h>
#include <abycore/sharing/sharing.h>
#include <abycore/circuit/arithmeticcircuits.h>
#include <abycore/circuit/booleancircuits.h>
#include <abycore/circuit/circuit.h>
#include <abycore/aby/abyparty.h>
#include <ENCRYPTO_utils/crypto/crypto.h>
#include <ENCRYPTO_utils/parse_options.h>

int32_t read_test_options(int32_t *argcp, char ***argvp,
                          e_role *role, uint32_t *bitlen,
                          uint32_t *nvals, uint32_t *secparam,
                          std::string *address,
                          uint16_t *port,
                          int32_t *test_op) {

    uint32_t int_role = 0;
    parsing_ctx options[] = {
        {(void*)&int_role, T_NUM, "r", "Role: 0/1", true, false},
        {(void*)nvals, T_NUM, "n", "Number of elements", false,
false},
        {(void*)bitlen, T_NUM, "b", "Bit-length, default 32", false,
false},
        {(void*)secparam, T_NUM, "s", "Symmetric Security Bits,
default: 128", false, false},
        {(void*)address, T_STR, "a", "IP address, default:
localhost", false, false},
        {(void*)port, T_NUM, "p", "Port, default: 7766", false,
false},
        {(void*)test_op, T_NUM, "t", "Single test (leave out for all
operations), default: off", false, false}
    };

```

```
30      if (!parse_options(argcp, argvp, options,
    sizeof(options)/sizeof(parsing_ctx))) {
31          print_usage(*argvp[0], options,
    sizeof(options)/sizeof(parsing_ctx));
32          std::cout << "Exiting" << std::endl;
33          exit(0);
34      }
35
36      *role = (e_role)int_role;
37
38      return 1;
39  }
40
41  share* build_inner_product_circuit(ArithmeticCircuit *ac,
    std::vector<share*> &a, std::vector<share*> &b) {
42      assert(a.size() == b.size());
43      share *result = ac->PutCONSGate(0, a[0]->get_bitlength());
44      for(size_t i = 0; i < a.size(); ++i) {
45          share *product = ac->PutMULGate(a[i], b[i]);
46          result = ac->PutADDGate(result, product);
47      }
48      return result;
49  }
50
51  int main(int argc, char** argv) {
52      e_role role;
53      uint32_t bitlen = 32, nvals = 10, secparam = 128, nthreads = 1;
54      std::string address = "127.0.0.1";
55      uint16_t port = 7766;
56      int32_t test_op = -1;
57
58      read_test_options(&argc, &argv, &role, &bitlen, &nvals,
    &secparam, &address, &port, &test_op);
59
60      seclvl seclvl = get_sec_lvl(secparam);
61      e_mt_gen_alg mt_alg = MT_OT;
```

```
62
63      ABYParty* party = new ABYParty(role, address, port, seclvl,
    bitlen, nthreads, mt_alg);
64      std::vector<Sharing*>& sharings = party->GetSharings();
65
66      ArithmeticCircuit* ac = (ArithmeticCircuit*) sharings[S_ARITH]-
    >GetCircuitBuildRoutine();
67
68      std::vector<uint32_t> a_vals(nvals, 0), b_vals(nvals, 0);
69
70      // Initialize the vectors with some values (for example
    purposes)
71      if (role == SERVER) {
72          for (uint32_t i = 0; i < nvals; ++i) {
73              a_vals[i] = i + 1;
74              b_vals[i] = nvals - i;
75          }
76      }
77
78      std::vector<share*> s_a, s_b;
79
80      for (uint32_t i = 0; i < nvals; ++i) {
81          s_a.push_back(ac->PutSharedINGate(a_vals[i], bitlen));
82          s_b.push_back(ac->PutSharedINGate(b_vals[i], bitlen));
83      }
84
85      share* s_result = build_inner_product_circuit(ac, s_a, s_b);
86      s_result = ac->PutOUTGate(s_result, ALL);
87
88      party->ExecCircuit();
89
90      uint32_t result = s_result->get_clear_value<uint32_t>();
91      if (role == SERVER) {
92          std::cout << "Inner product result: " << result <<
    std::endl;
93      }
```

```
94
95      delete party;
96
97      return 0;
98  }
```

2.内积计算

```
1   #include <aby/ABYParty.h>
2   #include <ENCRYPTO_utils/crypto/crypto.h>
3   #include <iostream>
4   #include <vector>
5
6   // Function to calculate dot product securely
7   share* SecureDotProduct(ABYParty* party, const
    std::vector<uint32_t>& a, const std::vector<uint32_t>& b, uint32_t
    bitlen) {
8       // Number of elements in the vectors
9       size_t n = a.size();
10
11      // Get the sharing type (Boolean sharing)
12      share_type sharing = S_BOOL;
13
14      // Get the circuit for boolean sharing
15      Circuit* circ = party->GetCircuitBuildRoutine();
16
17      // Create shares for the input vectors
18      std::vector<share*> a_shares, b_shares;
19      for(size_t i = 0; i < n; ++i) {
20          a_shares.push_back(party->PutINGate(a[i], bitlen, ALICE));
21          b_shares.push_back(party->PutINGate(b[i], bitlen, BOB));
```

```
    }

    // Compute the product of each pair of elements
    std::vector<share*> product_shares;
    for(size_t i = 0; i < n; ++i) {
        product_shares.push_back(circ->PutMULGate(a_shares[i],
b_shares[i]));
    }

    // Sum the products to get the dot product
    share* result = product_shares[0];
    for(size_t i = 1; i < n; ++i) {
        result = circ->PutADDGate(result, product_shares[i]);
    }

    // Reveal the result to both parties
    result = party->PutOUTGate(result, ALL);

    return result;
}

int main(int argc, char** argv) {
    // Set bit length of inputs
    uint32_t bitlen = 32;

    // Create an ABYParty object with id, role, and other parameters
    e_role role = (argv[1][0] == '1') ? SERVER : CLIENT;
    ABYParty* party = new ABYParty(role, "127.0.0.1", 7766, LT,
bitlen, 1, S_BOOL);

    // Input vectors
    std::vector<uint32_t> a = {1, 2, 3, 4};
    std::vector<uint32_t> b = {5, 6, 7, 8};

    // Execute the secure dot product computation
    share* result_share = SecureDotProduct(party, a, b, bitlen);
```

```
56
57      // Execute the protocol
58      party->ExecCircuit();
59
60      // Get the result
61      uint32_t result;
62      result_share->get_clear_value(&result);
63
64      // Print the result
65      std::cout << "Dot Product: " << result << std::endl;
66
67      delete party;
68      return 0;
69  }
```

3.实现内积计算并比较大小

```
1  #include <aby/abyparty.h>
2  #include <aby/circuit/share.h>
3  #include <aby/circuit/arithmeticcircuits.h>
4  #include <aby/circuit/booleancircuits.h>
5
6  int main(int argc, char** argv) {
7      // 设置ABY框架的基本配置
8      e_role role = (strcmp(argv[1], "SERVER") == 0) ? SERVER :
   CLIENT;
9      uint16_t port = 7766;
10     std::string address = "127.0.0.1";
11
12     seclvl seclvl = get_sec_lvl(128);
13     e_mt_gen_alg mt_alg = MT_OT;
14     uint32_t bitlen = 32;
15     uint32_t nthreads = 1;
```

```
    e_sharing sharing = S_ARITH;

    ABYParty* party = new ABYParty(role, address, port, seclvl, bitlen, nthreads, mt_alg);
    std::vector<Sharing*>& sharings = party->GetSharings();
    ArithmeticCircuit* ac = (ArithmeticCircuit*) sharings[S_ARITH]->GetCircuitBuildRoutine();
    BooleanCircuit* bc = (BooleanCircuit*) sharings[S_BOOL]->GetCircuitBuildRoutine();

    // 定义输入向量
    std::vector<uint32_t> vec1 = {1, 2, 3}; // 假设这是第一个参与方的输入
    std::vector<uint32_t> vec2 = {4, 5, 6}; // 假设这是第二个参与方的输入
    uint32_t vec_size = vec1.size();

    // 将输入转换为共享值
    std::vector<share*> s_vec1, s_vec2;
    for (uint32_t i = 0; i < vec_size; i++) {
        s_vec1.push_back(ac->PutINGate(vec1[i], bitlen, CLIENT));
        s_vec2.push_back(ac->PutINGate(vec2[i], bitlen, SERVER));
    }

    // 计算内积
    share* s_inner_product = ac->PutCONSGate(0, bitlen);
    for (uint32_t i = 0; i < vec_size; i++) {
        share* temp = ac->PutMULGate(s_vec1[i], s_vec2[i]);
        s_inner_product = ac->PutADDGate(s_inner_product, temp);
    }

    // 比较大小
    share* s_threshold = ac->PutCONSGate(50, bitlen); // 假设我们要比较内积是否大于50
    share* s_comparison = bc->PutGTGate(s_inner_product, s_threshold);

    // 输出结果
```

```
    s_comparison = bc->PutOUTGate(s_comparison, ALL);
    party->ExecCircuit();

    // 获取并打印结果
    uint32_t result = s_comparison->get_clear_value<uint32_t>();
    if (result) {
        std::cout << "Inner product is greater than threshold" << std::endl;
    } else {
        std::cout << "Inner product is not greater than threshold" << std::endl;
    }

    delete party;
    return 0;
}
```

4.内积计算支持小数和负数

```
#include <abycore/aby/abyparty.h>
#include <abycore/sharing/sharing.h>
#include <vector>

share* InnerProduct(share* a[], share* b[], uint32_t n, ABYParty* party, BooleanCircuit* circ) {
    share* result = circ->PutCONSGate(0, 32);
    for (uint32_t i = 0; i < n; i++) {
        share* prod = circ->PutMULGate(a[i], b[i]);
        result = circ->PutADDGate(result, prod);
    }
    return result;
}

```

```
14 share* Compare(share* a, share* b, BooleanCircuit* circ) {
15     return circ->PutGTGate(a, b);
16 }
17
18 int main(int argc, char** argv) {
19     // 解析命令行参数
20     uint32_t role = (argv[1][0] == '0') ? SERVER : CLIENT;
21     std::string address = "127.0.0.1";
22     uint16_t port = 7766;
23     seclvl seclvl = get_sec_lvl(128);
24     uint32_t bitlen = 32;
25     uint32_t nthreads = 1;
26     e_mt_gen_alg mt_alg = MT_OT;
27     ABYParty* party = new ABYParty(role, address, port, seclvl,
   bitlen, nthreads, mt_alg);
28
29     // 创建电路
30     std::vector<Sharing*>& sharings = party->GetSharings();
31     BooleanCircuit* circ = (BooleanCircuit*)sharings[S_BOOL]-
   >GetCircuitBuildRoutine();
32
33     // 定义向量大小
34     uint32_t n = 3;
35     // 创建共享输入
36     share* a[n];
37     share* b[n];
38
39     if (role == SERVER) {
40         std::vector<float> input_a = {1.5, -2.0, 3.5};
41         for (uint32_t i = 0; i < n; i++) {
42             a[i] = circ->PutINGate(*(uint32_t*)&input_a[i], 32,
   SERVER);
43         }
44         for (uint32_t i = 0; i < n; i++) {
45             b[i] = circ->PutDummyINGate(32);
46         }
```

```
    } else { // CLIENT
        std::vector<float> input_b = {-1.5, 2.0, -3.0};
        for (uint32_t i = 0; i < n; i++) {
            a[i] = circ->PutDummyINGate(32);
        }
        for (uint32_t i = 0; i < n; i++) {
            b[i] = circ->PutINGate(*(uint32_t*)&input_b[i], 32,
CLIENT);
        }
    }

    // 计算内积
    share* inner_product = InnerProduct(a, b, n, party, circ);

    // 比较大小
    share* comparison_result = Compare(inner_product, circ-
>PutCONSGate(0, 32), circ);

    // 输出结果
    comparison_result = circ->PutOUTGate(comparison_result, ALL);
    party->ExecCircuit();

    // 获取结果
    uint32_t result;
    comparison_result->get_clear_value(&result);

    std::cout << "Comparison Result: " << result << std::endl;

    delete party;
    return 0;
}
```