

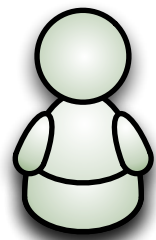
Programming secure computation with the SecreC language

Dan Bogdanov
Sharemind project lead
dan@cyber.ee



13th International School on Foundations of Security Analysis and Design
September 3rd, Bertinoro

**a simple, provably private
secure computing scheme**



Student A

Score: 25

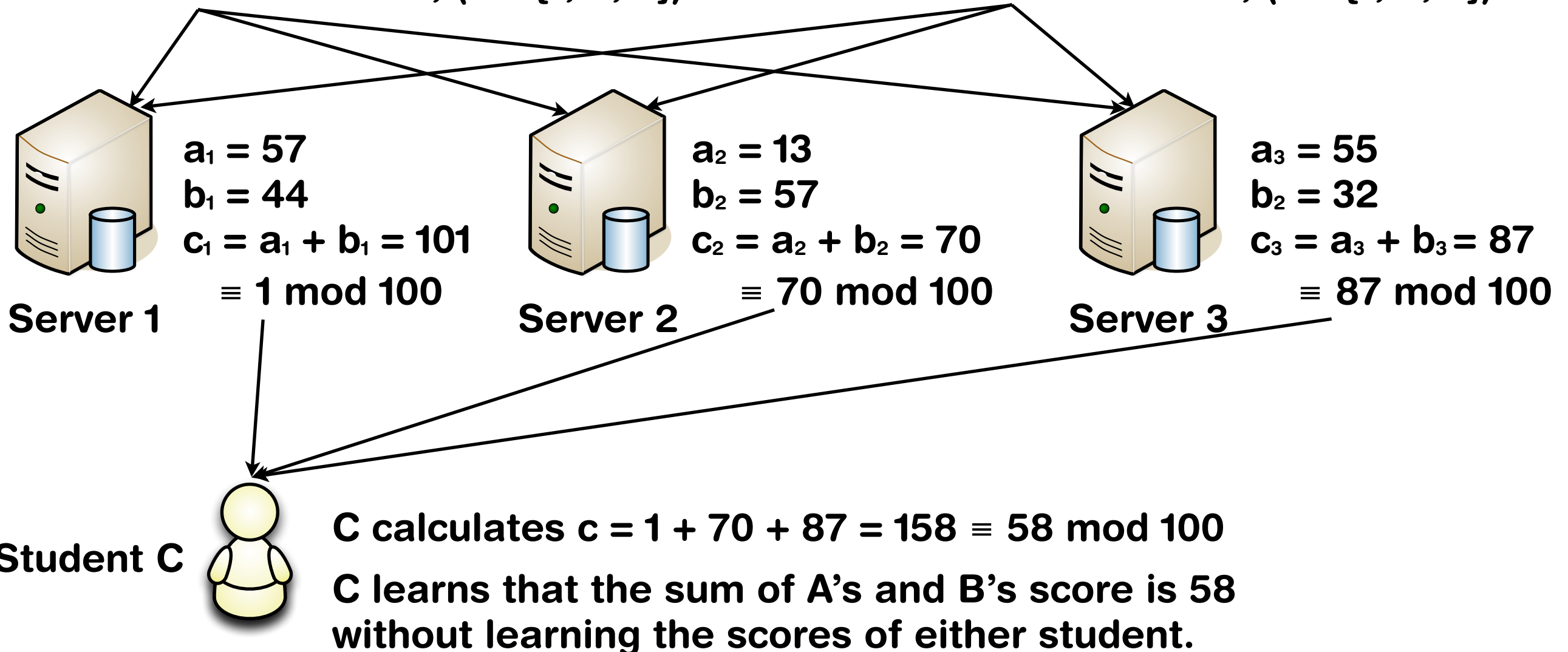


Student B

Score: 33

1. Pick random number $a_1 = 57$
2. Pick random number $a_2 = 13$
3. Find $a_3 = 25 - 57 - 13 \equiv 55 \pmod{100}$
4. Send a_k to Server k , ($k \in \{1, 2, 3\}$)

1. Pick random number $b_1 = 44$
2. Pick random number $b_2 = 57$
3. Find $b_3 = 33 - 44 - 57 \equiv 32 \pmod{100}$
4. Send b_k to Server k , ($k \in \{1, 2, 3\}$)



introducing protection domains

Protection domain kind (PDK)

- A protection domain kind (PDK) is a set of data representations, algorithms and protocols for storing and computing on protected data.
- Examples:
 - secure multiparty computation,
 - (fully) homomorphic encryption,
 - trusted hardware.

Protection domain (PD)

- A protection domain (PD) is a set of data that is protected with the same resources and for which there is a well-defined set of algorithms and protocols for computing on that data while keeping the protection.
- Examples:
 - data held by a fixed group of servers performing secure multiparty computation,
 - data encrypted under a fixed key of a homomorphic encryption scheme.

the SecreC programming language

A simple function in SecreC

```
// Import a PDK module called 'additive3pp'
import additive3pp;

// Create a domain 'private' from the PDK
domain private additive3pp;

void main () {
    // Perform secure computations using the PDK
    private int a = 2, b = 3;
    private int c = a * b;
    // We need a special function to publish a
    print (declassify (c));
}
```

Functions for any protection domain

```
// Protection domain kind polymorphism
```

```
template<domain D>
```

```
D uint sum (D uint [[1]] vector) {
```

```
    D uint result = 0;
```

```
    for (uint i = 0; i < size (vector); i++) {
```

```
        result[i] = result[i] + vector[i];
```

```
    }
```

```
    return result;
```

```
}
```

Functions for a specific PDK

```
// Specialization to a PDK
template<domain D: additive3pp>
D uint sum (D uint[[1]] vec) {
    D uint result = 0;
    __syscall ("additive3pp::sum_uint64_vec",
               __domainid (D),
               vec, result);
    return result;
}
```

practical demonstration



<https://sharemind.cyber.ee/>
sharemind@cyber.ee