# SecreC for Sharemind 3

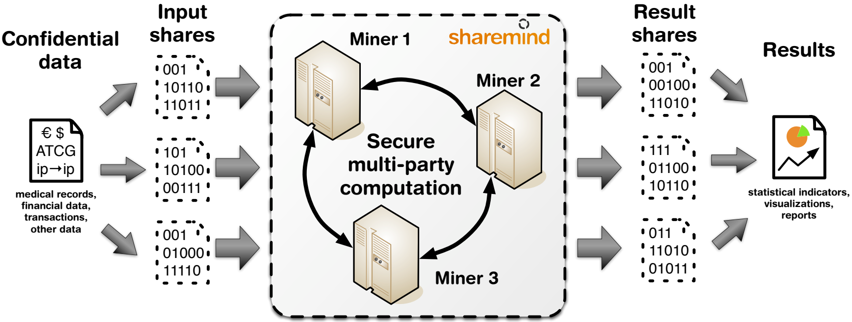
Dan Bogdanov, Peeter Laud, Jaak Randmets
Cybernetica AS

February 2, 2013

This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. Distribution Statement A (Approved for Public Release, Distribution Unlimited).

# Sharemind 2

- Sharemind is a secure multi-party computation framework.
- Sharemind 2 only supports additive 3-party secret sharing scheme.



# SecreC for Sharemind 2

- Simple high-level imperative algorithm language.
- Hides implementation details of the secret sharing scheme.
- Two visibility types: public and private.
- Private values become public only through declassify.

```
void main () {                // main function
    private uint a, b, c;     // private data
    a = b + c;                // private computation
    public uint d;            // public data
    d = declassify (a);       // private -> public
    publish (d);              // send to client
}
```

# Sharemind 3

- Various secure computation schemes.
- New underlying virtual machine.
- Complete rewrite of SecreC was in order.
- Some features of new SecreC:
  - more primitive data types
  - arbitrary dimensional arrays
  - simple module system
  - a lot of syntactic niceties
  - IR for data-flow analysis
  - protection domain polymorphism
  - etc.

# Sharemind 3

- Various secure computation schemes.
- New underlying virtual machine.
- Complete rewrite of SecreC was in order.
- Some features of new SecreC:
  - more primitive data types
  - arbitrary dimensional arrays
  - simple module system
  - a lot of syntactic niceties
  - IR for data-flow analysis
  - protection domain polymorphism
  - etc.

# Protection domains

## Definition (Protection domain)

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.

## Definition (Protection domain kind)

A protection domain kind (PDK) is a set of data representations, algorithms and protocols for storing and computing on protected data.

- Each PD belongs to a PDK, and each PDK can have several PD-s.

# Protection domains

## Example (protection domain kinds)

- A FHE system specified by its algorithms.
- A MPC system specified by its protocols.
- Public computation system.

## Example (protection domains)

- A FHE system running under a single key.
- A single physical MPC instance.
- A public machine.

- Sharemind 2 supports a single protection domain called “private” in additive 3-party protection domain kind.

# Protection domains in SecreC

- Simplest solution:
  - only public and private types
  - during program deployment map private to some protection domain
- Some issues:
  - impossible to use multiple PD-s concurrently
  - some code is PDK specific
- Better solution is to provide the ability to declare new protection domains.

# Feature : Protection domains

```
kind additive3pp;                // declare PDK
domain pd_a3p additive3pp;       // declare PD

void main () {                   // main function
    pd_a3p uint a, b, c;         // private data
    a = b + c;                   // private computation
    public uint d;               // public data
    d = declassify (a);          // private -> public
    publish (d);                 // send to client
}
```

- `declassify`, and `publish` no longer primitives.
- Module declares the PDK.

# Feature : Modules

```
import stdlib;                     // import stdlib
import additive3pp;                // import PDK
domain pd_a3p additive3pp;         // declare PD

void main () {                     // main function
    pd_a3p uint a, b, c;           // private data
    a = b + c;                     // private computation
    public uint d;                 // public data
    d = declassify (a);            // private -> public
    publish (d);                   // send to client
}
```

- Standard library declares `publish`.
- Module declares the PDK and `declassify`.

# Feature : PD monomorphic functions (1/2)

```
kind additive3pp;
domain pd_a3p additive3pp;

pd_a3p uint sum (pd_a3p uint[[1]] x) {
    pd_a3p uint s = 0;
    for (uint i = 0; i < size (x); ++ i) {
        s += x[i];
    }

    return s;
}
```

# Feature : PD monomorphic functions (2/2)

```
kind xor3pp;
domain pd_x3p xor3pp;

pd_x3p uint sum (pd_x3p uint[[1]] x) {
    pd_x3p uint s = 0;
    for (uint i = 0; i < size (x); ++ i) {
        s += x[i];
    }

    return s;
}
```

- Not very useful (apart from public PD).

# Feature : PD polymorphic functions

```
template <domain D>
D uint sum (D uint[[1]] x) {
    D uint s = 0;
    for (uint i = 0; i < size (x); ++ i) {
        s += x[i];
    }

    return s;
}
```

- Type variable `D` binds to any PD or public.
- Implemented via code duplication.
- C++ templates.

# Feature : overloading (1/2)

```
template <domain D : additive3pp>
uint declassify (D uint x) {
    // invoke system call that additive3pp
    // is known to define
}

template <domain D : xor3pp>
uint declassify (D uint x) {
    // invoke system call from xor3pp
}

uint declassify (uint x) { return x; }
```

# Feature : overloading (2/2)

```
template <domain D1 : additive3pp,
          domain D2 : additive3pp>
D1 uint reclassify (D2 uint x) {
    // invoke a system call
}

template <domain D>
D uint reclassify (D uint x) { return x; }
```

- Overload selection via ad-hoc manner.
- For example: the number of instantiated PD variables.
- Operator overloading.

# Feature : operator overloading

```
template <domain D : additive3pp>
D uint operator * (D uint x, D uint y) {
    // invoke system call
}
```

- Much of PDK functionality.

# Expected use of polymorphism

## Developer

- Standard library provides low-priority implementations.
- Each PDK is declared as a module.
  - declares the protection domain kind
  - PD polymorphic operations for that PDK
  - most operations through system calls

## End user

- Algorithm is PD polymorphic.
- Main program fixes the PD.

# Example : sorting (1/2)

```
module stdlib;

template <domain D>
D uint[[1]] sort (D uint[[1]] src) {
    uint[[2]] sn = sortnet (size (src));
    for (uint i = 0; i < shape (sn)[0]; ++ i) {
        D uint x = src[sn[i,0]];
        D uint y = src[sn[i,1]];
        D uint b = (uint) (x < y);
        src[sn[i,0]] = x*b + y*(1 - b);
        src[sn[i,1]] = x*(1 - b) + y*b;
    }

    return src;
}
```

# Example : sorting (2/2)

```
module additive3pp;

template <domain D : additive3pp>
D uint [[1]] sort (D uint[[1]] src) {

    src = shuffle (src);

    // Sort using:
    //   declassify (src[i] < src[j])

    return src;
}
```

- Can be done in $O(n \log n)$.

# Example : multiple protection domains

```
template <domain D1, domain D2>
D1 uint hamming (D2 uint[[1]] x, D2 uint[[1]] y) {
    D2 bool[[1]] neqs = (x != y);
    D1 uint[[1]] vs = (uint) reclassify (neqs);
    return sum (vs);
}
```

- Different performance depending on selected PD-s.
- If `D1 = pd_a3p`, and `D2 = pd_a3p` then 12 rounds.
- If `D1 = pd_a3p`, and `D2 = pd_x3p` then 8-9 rounds.

# Summary

## Good

- Simple solution.
- Easy to understand.

## Bad

- Ad-hoc overload resolution is not intuitive.
- Bad type errors just like in C++.
- Template interaction with modules can be strange.
  - specialization could help

# Semantics

- Formally defined type system.
- Small-step operational trace semantics.
  - labels are system calls
- Monomorphic intermediate language.
- Type-directed translation from polymorphic to monomorphic language.
- Weak bi-simulation between semantics of monomorphic and polymorphic language.
- Security of information flow.
- Models the compiler with a monomorphic IR.
- Abstract syntax, type system and semantics have been invaluable documentation for implementing and debugging the compiler.

# Future work

- Language improvements.
  - pass-by-reference
  - user defined data types
  - many small improvements
- Standard library.
- We need users and more PD-s.
- Something better than templates?