Viaduct
an extensible, optimizing compiler for secure multi-party computation

Rolph Recto
Cornell PL Retreat 2019
viaduct

(n.) a long bridge–like structure, typically a series of arches, carrying a road or railroad across a valley or other low ground.
an extensible, optimizing compiler for secure multi-party computation

allow mutually distrusting principals to perform computations...
allow mutually distrusting principals to perform computations...

...written in a high-level specification language...

an extensible, optimizing **compiler** for secure multi-party computation
allow mutually distrusting principals to perform computations...

....written in a high-level specification language...

....compiled to protocols using a generic security analysis...

an extensible, optimizing compiler for secure multi-party computation
allow mutually distrusting principals to perform computations...

....written in a high-level specification language...

....compiled to protocols using a generic security analysis...

....that uses expensive crypto only when necessary.

an extensible, optimizing compiler for secure multi-party computation
security specification
a : \{ A \}
b : \{ B \}
decl(a<b)

computing security reqs
A & B
A
B

protocol selection
S
MPC
S

protocol instantiation
A
MPC
B

back-end compilation

compiler architecture
security specification

a : {A}
b : {B}
decl(a<b)

computing security reqs

protocol selection

protocol instantiation

back-end compilation

compiler architecture
specification language

- specifies high-level security policy
- simple imperative language with arrays, security labels, and downgrading
- type system ensures secure information flow
“dangerous” millionaire’s game

Alice and Bob check who is richer without revealing actual net worth. The poorer person loses all her money.

Alice and Bob trust each other, but cannot read the other’s private data.
FLAM label model

[A→
  • secret to A

A←
  • trusted by A

A & B
  • trusted by and secret to both A and B

A □ B
  • trusted by and visible to either A and B

security policies represented by principals and label operators

[Arden et al, CSF 2015]
"dangerous" millionaire’s game

```plaintext
a : int{A & B←}
b : int{B & A←}
b_richer : bool{A∩B}

b_richer = declassify(a < b, A∩B)
if (b_richer) {
    a = 0;
} else {
    b = 0;
}
```

- **Alice and Bob trusts, only Alice can Read**
- **Alice and Bob trusts, only Bob can Read**
- **Both Alice and Bob trust and can read**
- **Need both Alice and Bob to read**
- **Allow either Alice or Bob to read (make less secret)**
security specification

computing security reqs

protocol selection

protocol instantiation

back-end compilation

compiler architecture
computing security requirements

- what are security requirements for protocols to store variables and perform computations?
- idea: solve information flow constraints from specification program
- best solution comes as minimum authority needed for program expressions
computing security requirements

information flow labels

- classifies policies regarding the use of data
- describes who can read and influence data
- ordered by restrictiveness of data policy (flows-to)

authority labels

- classifies the power of principals in a system
- describes what a principal can read and influence
- ordered by what the amount of principal’s authority (acts-for)

*can both be represented by FLAM labels*
secure information flow

information flow lattice

least restrictive data policy

most restrictive data policy

confidentiality

integrity

secure information flow
increasing authority

1→&0←

authority lattice

weakest principal

strongest principal

confidentiality

integrity

A & B

A

B

A

0

B

1
security label constraints

- spec program induces constraints for secure information flow
  \[ x = y \implies L(y) \subseteq L(x) \]

- convert into authority constraints
  \[ L(y) \subseteq L(x) \implies L(x)\rightarrow \supseteq L(y)\rightarrow, L(y)\leftarrow \supseteq L(x)\leftarrow \]

- solve for greatest fixpoint (least authority) solution using iterative worklist algorithm
a \text{A&B+} \quad b \text{B&A+} \quad \text{b_richer.get()}

\text{tmp1 < tmp2} \quad \text{declassify}

\text{if} \quad \text{b_richer.get()}

\text{tmp1} \quad \text{tmp2}
a
A&B+

a.get()
A & B+

tmp1

tmp1 < tmp2
A & B

declassify
A & B

b
B&A+

b.get()
B & A+

tmp2

b_richer
A ⊓ B

b_richer.get()
A ⊓ B

0
A∩B

if
A∩B

0
A∩B

A & B

A & B

A & B

A & B

A & B

A & B

A & B
security specification

computing security reqs

protocol selection

protocol instantiation

back-end compilation

compiler architecture
protocol selection

- protocols must be able to “act for” program components
- derive authority of protocol from authority of protocol participants

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Authority</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single(A)</td>
<td>L(A)</td>
<td>store data or perform computation in cleartext on one host</td>
</tr>
<tr>
<td>Repl(A,B)</td>
<td>L(A) ∩ L(B)</td>
<td>replicate data or computation in cleartext on multiple hosts</td>
</tr>
<tr>
<td>MPC(A,B)</td>
<td>L(A) &amp; L(B)</td>
<td>hosts perform MPC protocol</td>
</tr>
</tbody>
</table>
protocol selection

- authority requirements satisfied by many protocols, use cost model to choose optimal
- model cost as required communication
- use directed search (Dijkstra’s) to find optimal
\( S(A) \) \( S(B) \)

\( a \) \( b \)

\( a \).get() \( b \).get()

\( S(A) \) \( S(B) \)

\( \text{Repl}(A,B) \)

\( \text{tmp1} \) < \( \text{tmp2} \)

\( \text{MPC}(A,B) \)

\( \text{declassify} \)

\( \text{MPC}(A,B) \)

\( b_{\text{richer}} \)

\( \text{Repl}(A,B) \)

\( b_{\text{richer}} \).get()

\( \text{Repl}(A,B) \)

\( \text{if} \)

\( \text{Repl}(A,B) \)

\( \text{tmp1} \)

\( \text{tmp2} \)
security specification

computing security reqs

protocol selection

protocol instantiation

back-end compilation

compiler architecture
protocol instantiation

- compile process configuration from selected protocol map
- process configuration represents a *hybrid protocol*
  - cryptographic primitives left abstract
process alice {
    int a
    bool b_richer
    send a to MPC
    res1 <- recv MPC
    b_richer = res1
    if (b_richer) {
        a = 0
    } else {} 
}

process bob {
    int b
    bool b_richer
    send b to MPC
    res2 <- recv MPC
    b_richer = res2
    if (b_richer) {} 
    else {
        b = 0
    }
}

process MPC {
    ta <- recv alice
    tb <- recv bob
    let res = ta < tb
    send res to alice
    send res to bob
}
process alice {
    int a
    bool b_richer
    send a to MPC
    res1 <- recv MPC
    b_richer = res1
    if (b_richer) {
        a = 0
    } else {
        
    }
}

process bob {
    int b
    bool b_richer
    send b to MPC
    res2 <- recv MPC
    b_richer = res2
    if (b_richer) {
    } else {
        b = 0
    }
}

process MPC {
    ta <- recv alice
    tb <- recv bob
    let res = ta < tb
    send res to alice
    send res to bob
}
security specification

computing security reqs

protocol selection

protocol instantiation

back-end compilation

compiler architecture
work in progress

- formal correctness proof for generated protocols
- backend implementation
- more protocols
- richer specification language
thanks!