



# Security-Oriented Languages

Steve Zdancewic

Ph. D. students: Peng Li and Stephen Tse

*University of Pennsylvania*

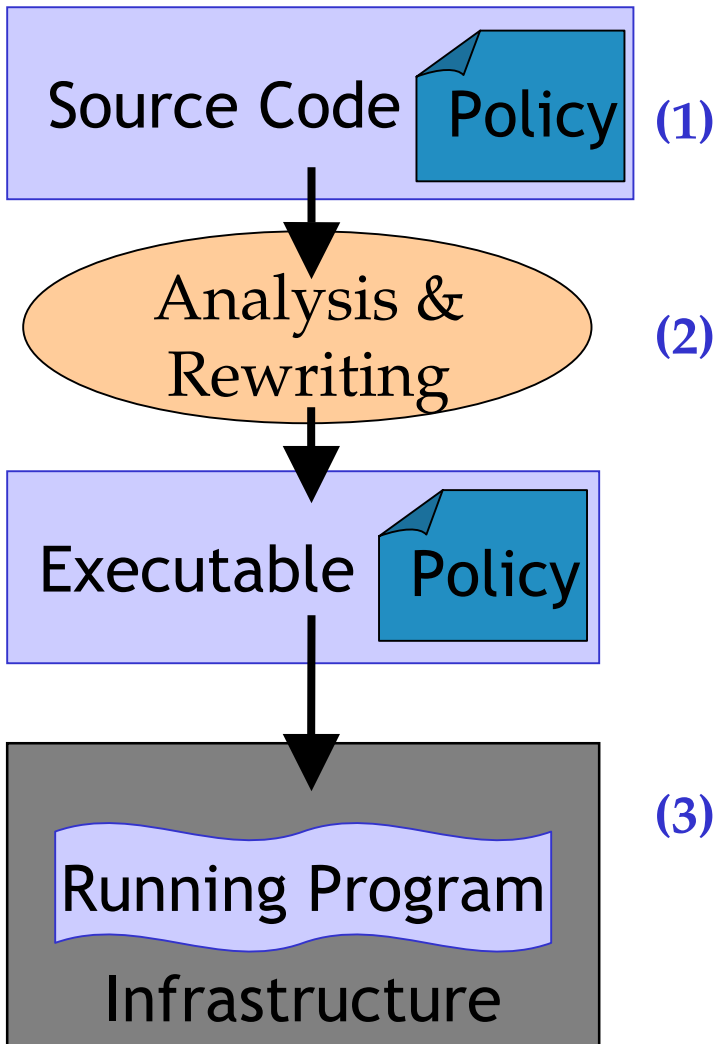
## Challenges:

1. Increasingly large and distributed applications
2. Increasingly complex and refined policies:
3. Existing mechanisms (e.g. PKI, OS) are crucial but higher levels of abstraction are needed

**Goal:** help programmers build secure distributed software that interacts well with existing **security infrastructure** such as authentication mechanisms, operating systems, and cryptography...

Security-oriented languages provide programmers with high level abstractions such as built-in notions of **principals** and **digital certificates**. Providing language-support for these abstractions means that program analyses, done statically by the compiler *before* the program is run, can detect many security errors. These constructs also provide a way for programmers to document the security policy intended for the system.

**Our approach:** Use programming-language tools and methodologies.



The figure to the left illustrates the general structure of security-oriented languages:

- (1) A source program, in addition to describing the behavior of the system, also describes an appropriate security policy.
- (2) Program analysis and transformation tools such as compiler verify that the program enforces the policy.
- (3) The executable program cooperates with the existing security infrastructures such as operating systems and PKI.

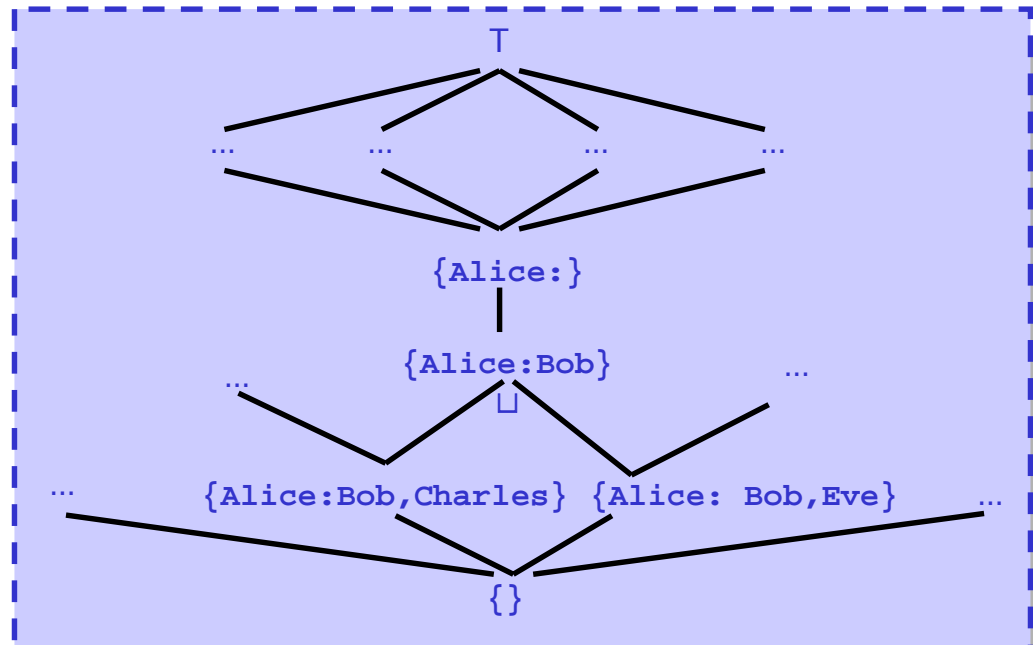
# Information-flow Policies

Our security-oriented languages are able to express strong confidentiality and integrity policies that restrict how the data is used by the system.

Policy labels in this model form a lattice (shown below). Labels higher in the lattice are more restrictive than those lower in the lattice.

Confidentiality policies, based on Myers' and Liskov's **decentralized label model**, have the form: `{owner: readers}`.

For example, the label: `{Alice: Bob, Eve}` means "Alice owns this data and she permits Bob & Eve to read it."



```

void main (cert cnet) {
    principal{ATM:} id = login();
    cert cdel = acquire id :> Delegate ATM;
    cert creq = acquire id :> Withdraw;
    Msg{ATM:} m = new Msg(id, cdel, creq);
    if (cnet #> ATM :> Declassify Net) {
        Msg{ATM:Net} x = declassify m as Msg{ATM:Net};
        Int{user:} balance = request(x); ...
    }
}

```

## Example Code

This program shows the application of our new language features. The data it manipulates are protected with confidentiality policies (in **blue**), and the language provides high-level abstractions for programming with principals and digital certificates (in **green**). The programming model is connected to existing security infrastructure, such as PKI, by dynamic policy constructs (**orange**).

# Technical Challenges

## Challenge #1: Integrate dynamic information with static analyses:

```
principal user = Runtime.getUser();  
void print (String{user:} x) {...}  
void printroot (String{root:} y) {  
    if (user == root) print(y);  
}
```

Must propagate information about the results of this dynamic test to the static analysis for the “then” branch of the “if”.

## Challenge #2: Provide guarantees about correct use or rich information-flow policies that include downgrading.

Theorem (Noninterference) : No program that passes the type-checking analysis will leak information from higher in the lattice to lower in the lattice unless the “declassify” operation is performed. Moreover, no malicious attacker can use the declassify operation to gain more information about confidential data than was intended by the policy.

We have built a prototype language called *Apollo* that demonstrates these ideas. Apollo is a vehicle for exploring security-oriented language design, theory, and implementation strategies.

### Future work:

1. Cryptographic protocol implementation.
2. Validation via machine-checked proof of soundness.
3. Support for fault tolerance.

### Benefits of security-oriented languages:

- Explicit, fine-grained policies
- Program abstractions for describing complex policies
- Ability to regulate end-to-end information flows
- Compilers and other tools can help automate security analyses.
- *Increased confidence in security of the system*

## Security-oriented Languages:

- are broadly applicable
- have many interesting open problems
- are a promising technique for improving system security

### More information:

<http://www.cis.upenn.edu/~stevez/sol/>

*Designing a Security-typed Language with Certificate-based Declassification*

*Downgrading Policies and Relaxed Noninterference*

*Advanced Control Flow in Java Card Programming - LCTES 2004*

*Translating Dependency into Parametricity - ICFP 2004*

*Enforcing Robust Declassification - CSFW 2004*

*Run-time Principals in Information-flow Type Systems - Security & Privacy 2004*

*Information Integrity Policies - FAST 2003*



## Trustworthy Infrastructure, Mechanisms, and Experimentation for Diffuse Computing

<http://www.cis.upenn.edu/group/spyce/timedc>

The SOL group is part of the TIME DC collaboration.

### Principal Investigator

**Andre Scedrov**, University of Pennsylvania

### Participants:

**Joan Feigenbaum**, Yale University

**Joseph Y. Halpern**, Cornell University

**Patrick D. Lincoln**, Consultant

**John C. Mitchell**, Stanford University

**Steve Zdancewic**, University of Pennsylvania

### Faculty Associates

**Tim Roughgarden**, Stanford University

### External Collaborators

**Cynthia Dwork**, Microsoft

**Paul Syverson**, Naval Research Laboratory

**Vitaly Shmatikov**, SRI International

### Postdoctoral Researchers

**Björn Knutsson**, University of Pennsylvania

Diffuse systems must respond to security events in order to provide robust and reliable service, but coordinating a response across a large distributed system raises particular challenges, especially when the parties involved do not trust each other completely. The TIME DC approach builds on recent work (within and beyond the ONR CIP/SW SPYCE group) exploring ways to combine multi-party computation protocols and incentive-compatible mechanisms and to develop language-enforced security methodology, including policy and programming language aspects. The proposed effort also builds on recent work in computer-virus and malicious code threat detection, monitoring, analysis, and mitigation.