Self-Verifying Execution

Matt McCutchen
MIT

Daniel Song
Rice University

Shuo Chen, Shaz Qadeer
Microsoft Research

Motivation

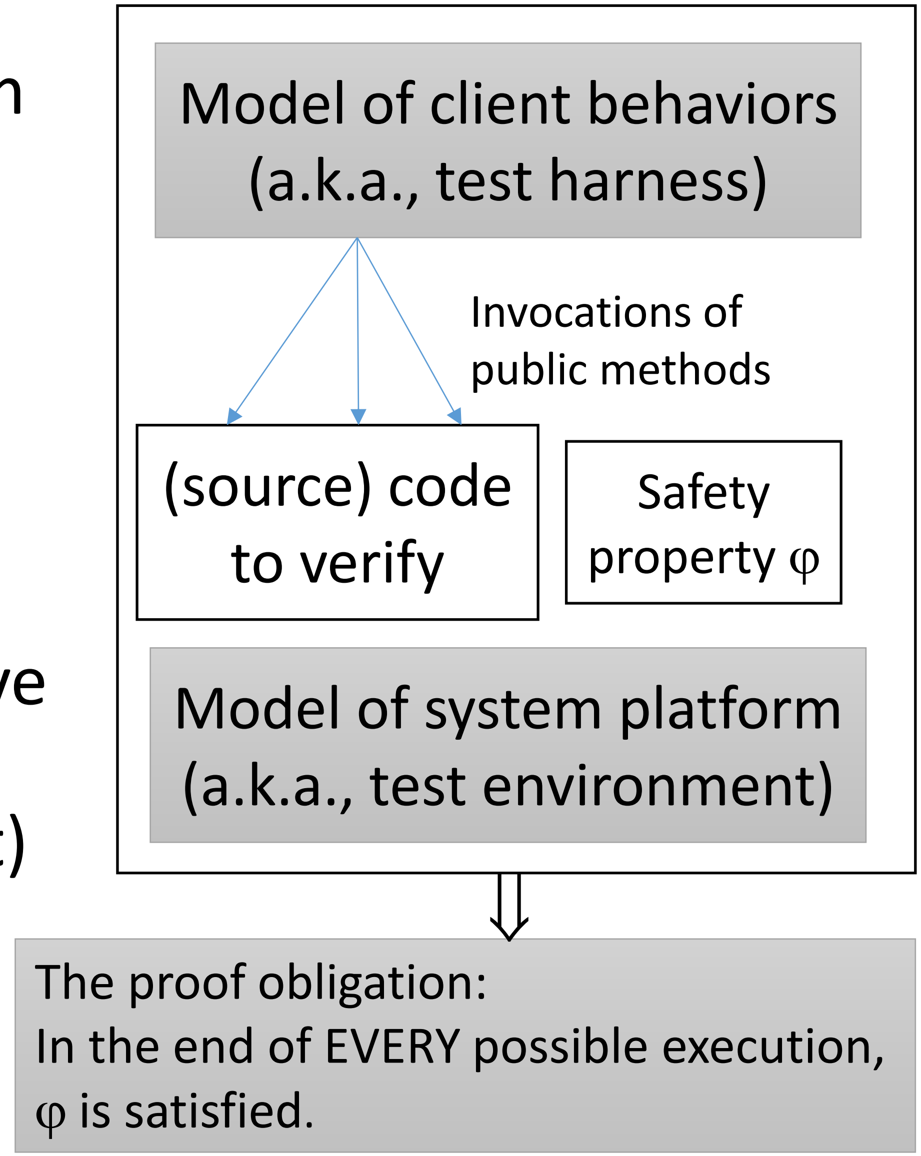
- Logic bugs are pervasive in online-service protocol implementations
  - Online-service protocols： single-sign-on, payment, authorization
  - Security consequence
    - signing into other people's accounts without password
    - making purchase without paying
    - Unauthorized access
  - This bug category is ranked as the No.4 cloud security top threat.

- Program verification
  - If adopted by real-world programmers, it would fundamentally avoid logic bugs.
  - Unfortunately, too demanding for most programmers.

- The goal of self-verifying execution (SVX)
  - To substantially lower hurdles for real-world programmers to do verification.

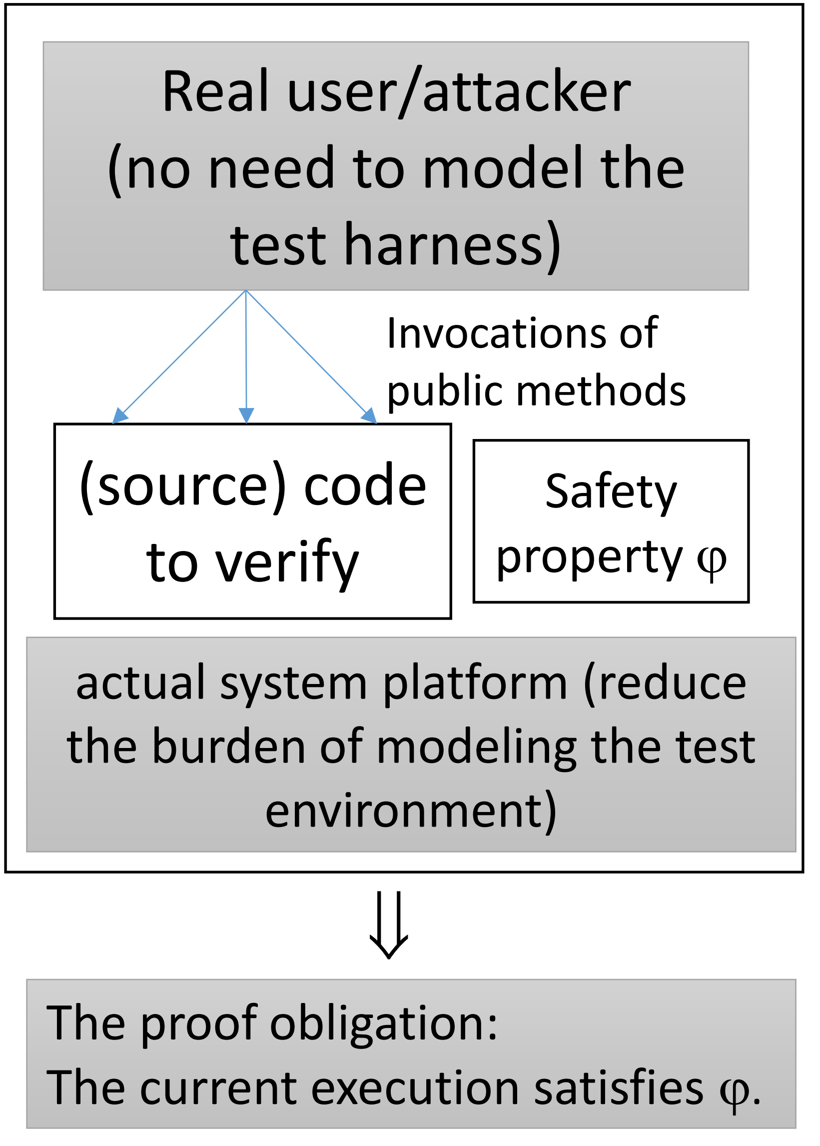
Hurdles of traditional verification approaches

- Need to model the system platform
- Need to model the client behavior
- Need to prove an inductive theorem for all possible executions (an infinite set)



Basic idea of self-verifying execution

- Every actual execution is responsible for collecting its own executed code, and symbolically proving that it satisfies φ.



Example: comparing integer constants among three websites

Alice.com

```
const int Value=10;
Message grab (Message m1)
```

Message

Untrusted client

Safety property φ:
Whenever conclude(m2) is reached, m2 must represent the website holding the biggest int.

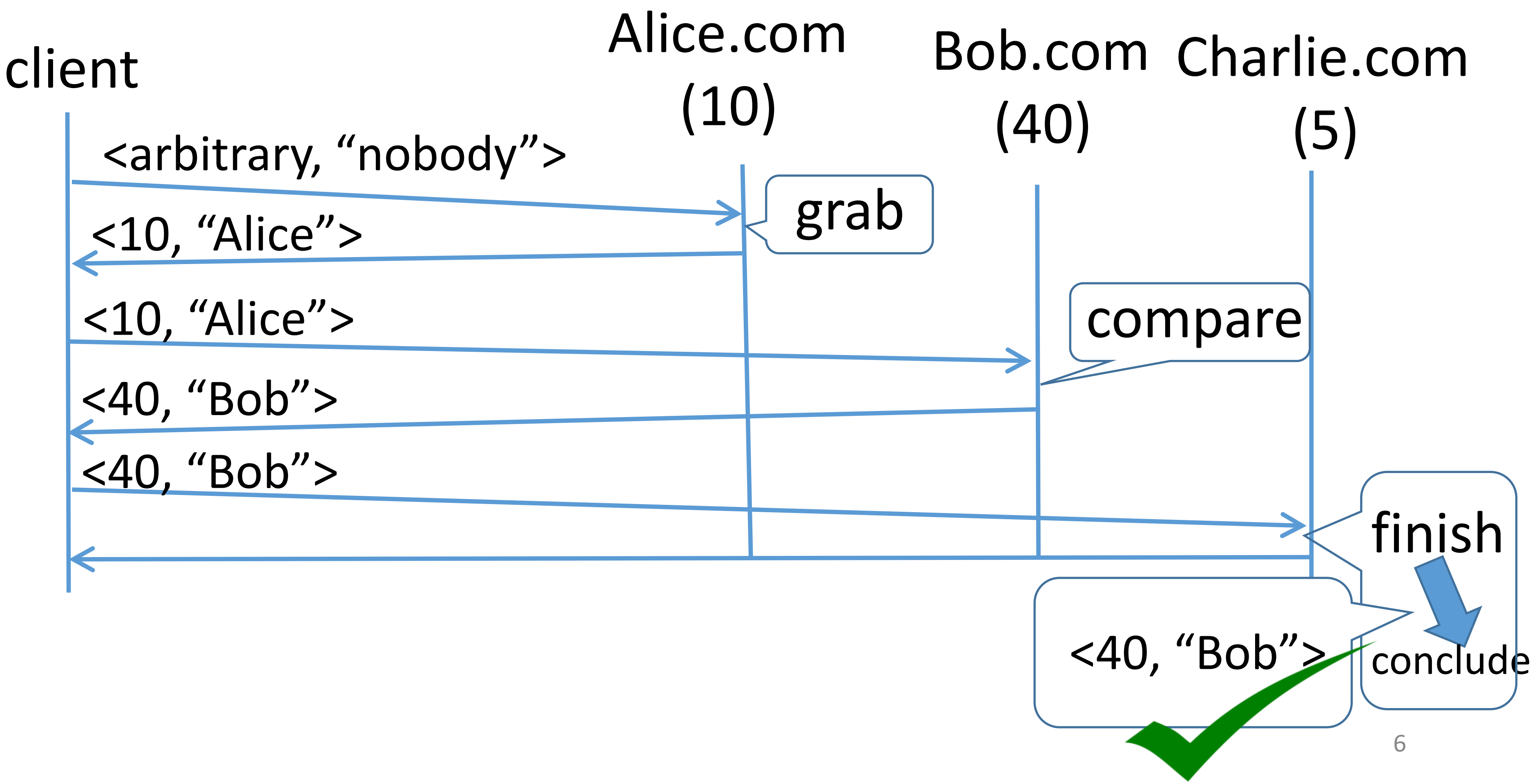
Bob.com

```
const int Value=40;
Message compare (Message m1)
{   ValidateSignature(m1);
    Message m2;
    m2 = <Value, "Bob">;
    m2 = max(m1,m2);
    m2.SignBy("Bob.com");
    return m2;
}
```

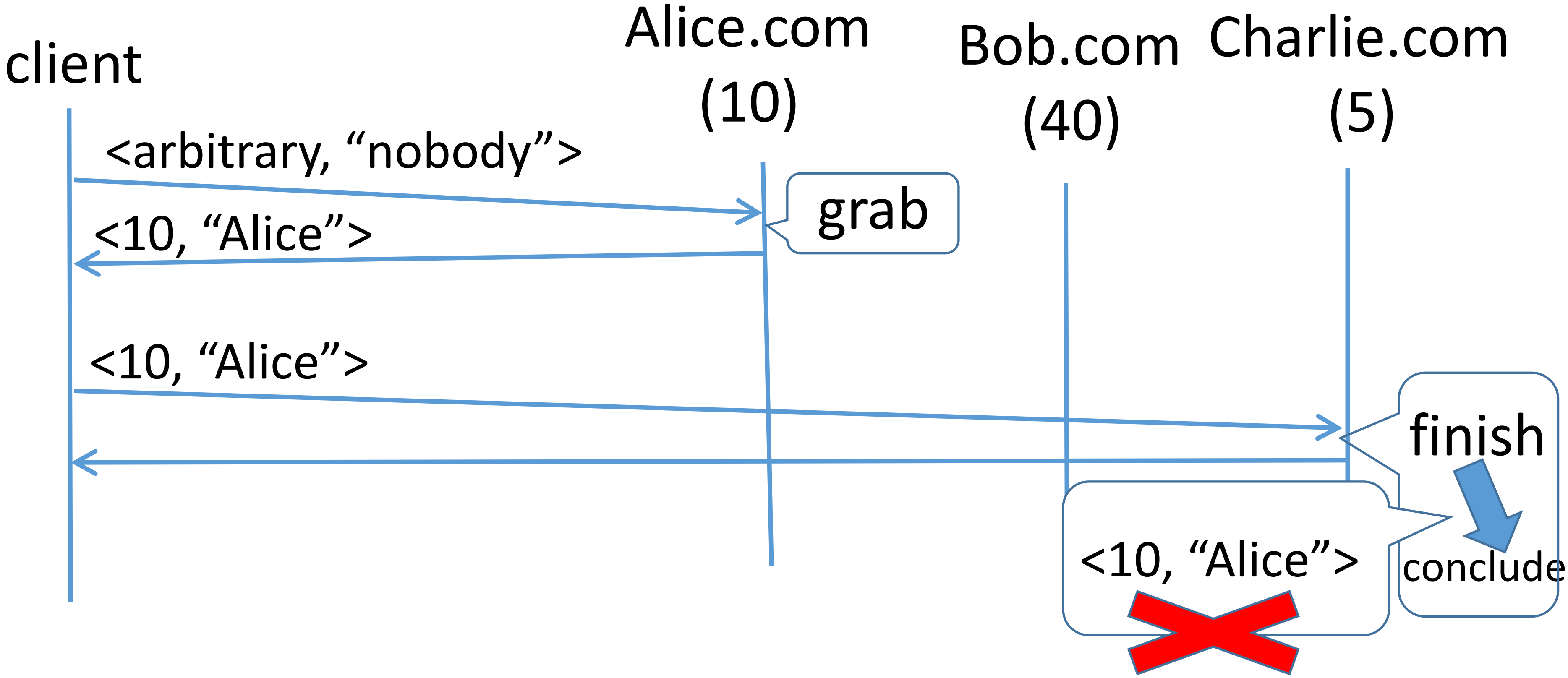
Charlie.com

```
const int Value=5;
Message finish (Message m1)
{   ValidateSignature(m1);
    Message m2;
    m2 = <Value, "Charlie">;
    m2 = max(m1,m2);
    conclude(m2);
    return m2;
}
```

The expected protocol flow

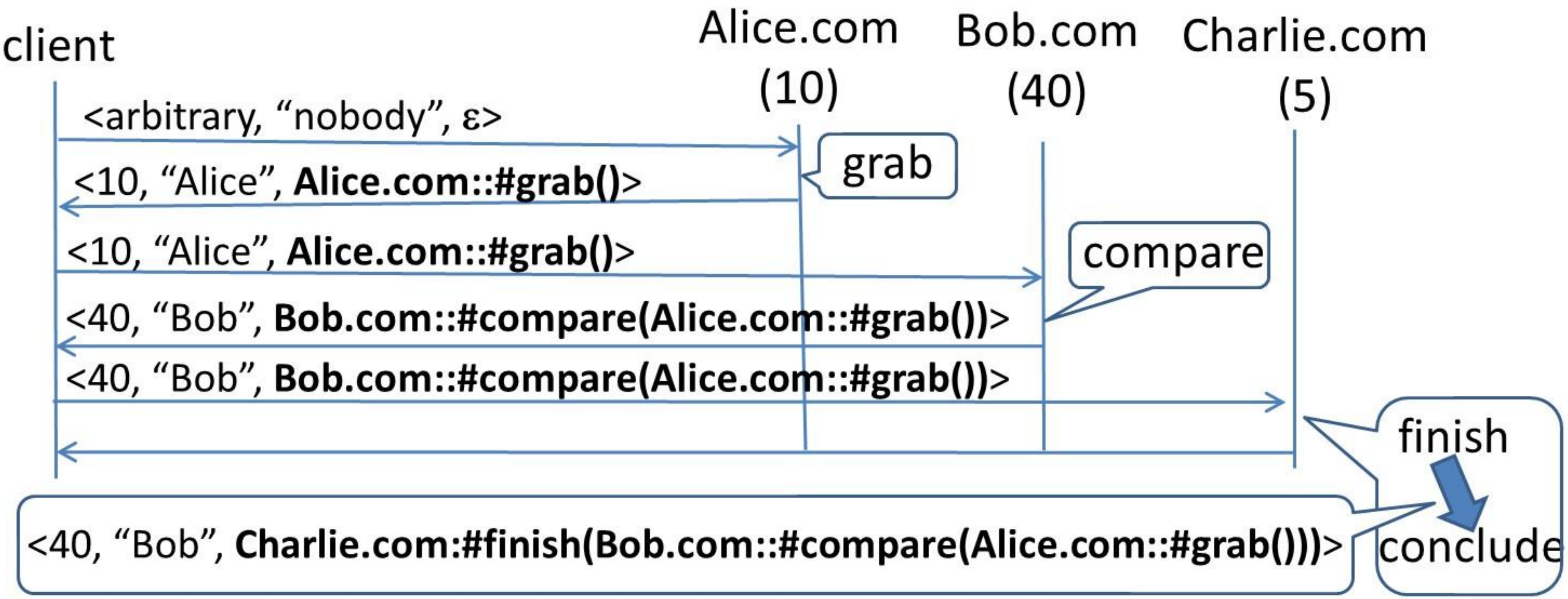


The system is vulnerable!



How SVX works

- Attach a field, namely *SymT (Symbolic Transaction)* onto every message.
- #grab, #compare and #finish are the hash values of the executed code of these methods.



Verifying an execution

- Method conclude() calls a program verifier to prove:
  The final SymT → φ
  - Charlie.com:#finish(Bob.com::#compare(Alice.com::#grab())) → φ , the execution is accepted.
  - Charlie.com:#finish(Alice.com::#grab()) ↛ φ, the execution is rejected.

- Note that the program verification is symbolic (only about code). The concrete values are ignored.
  - A middle ground between offline symbolic verification and runtime concrete checking.

The library we provide

- Only two public methods
  - RecordMe(…): to construct the SymT
  - Certify(…): to verify the execution represented by the SymT.

RecordMe() for SymT construction

Alice.com

```
int Value=10;
Message grab (Message m1)
{   Message m2;
    RecordMe(m1,m2);
    m2 = <Value, “Alice”>
    m2.SignBy(“Alice.com”);
    return m2;
}
```

Bob.com

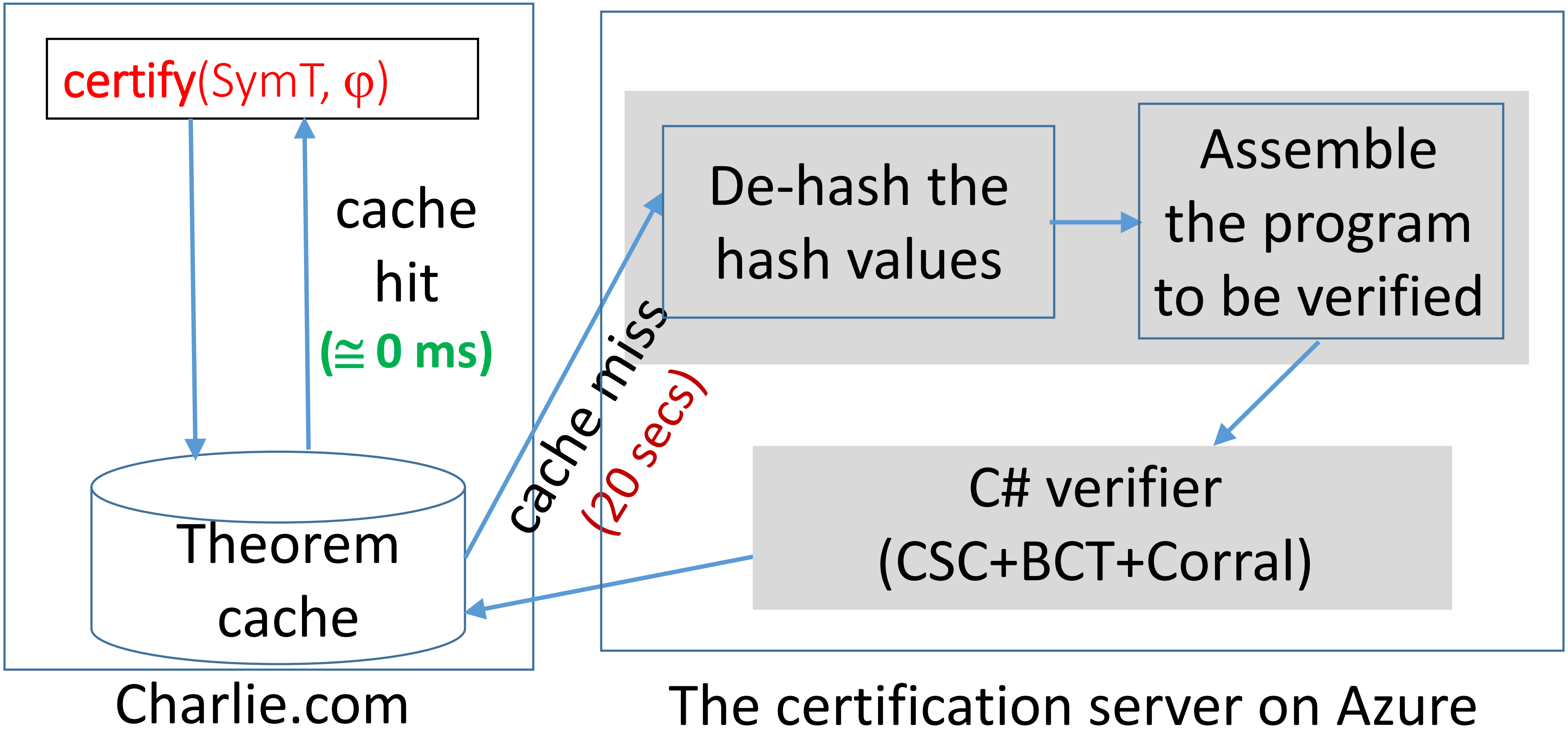
```
int Value=40;
Message compare (Message m1)
{   ValidateSignature(m1);
    Message m2;
    RecordMe(m1,m2);
    m2 = <Value, “Bob”>;
    m2 = max(m1,m2);
    m2.SignBy(“Bob.com”);
    return m2;
}
```

- **Just add a** RecordMe() **call in each message handler method.**
- RecordMe() **uses reflection to hash the current method code, and concatenates the new SymT.**

Charlie.com

```
int Value=5;
Message finish (Message m1)
{   ValidateSignature(m1);
    Message m2;
    RecordMe(m1,m2);
    m2 = <Value, “Charlie”>;
    m2 = max(m1,m2);
    if ( ! Certify(m2.SymT, φ))
        throw new Exception();
    conclude(m2);
    return m2;
}
```

Certify(): to verify SymT against φ



SVX lowers burdens for programmers

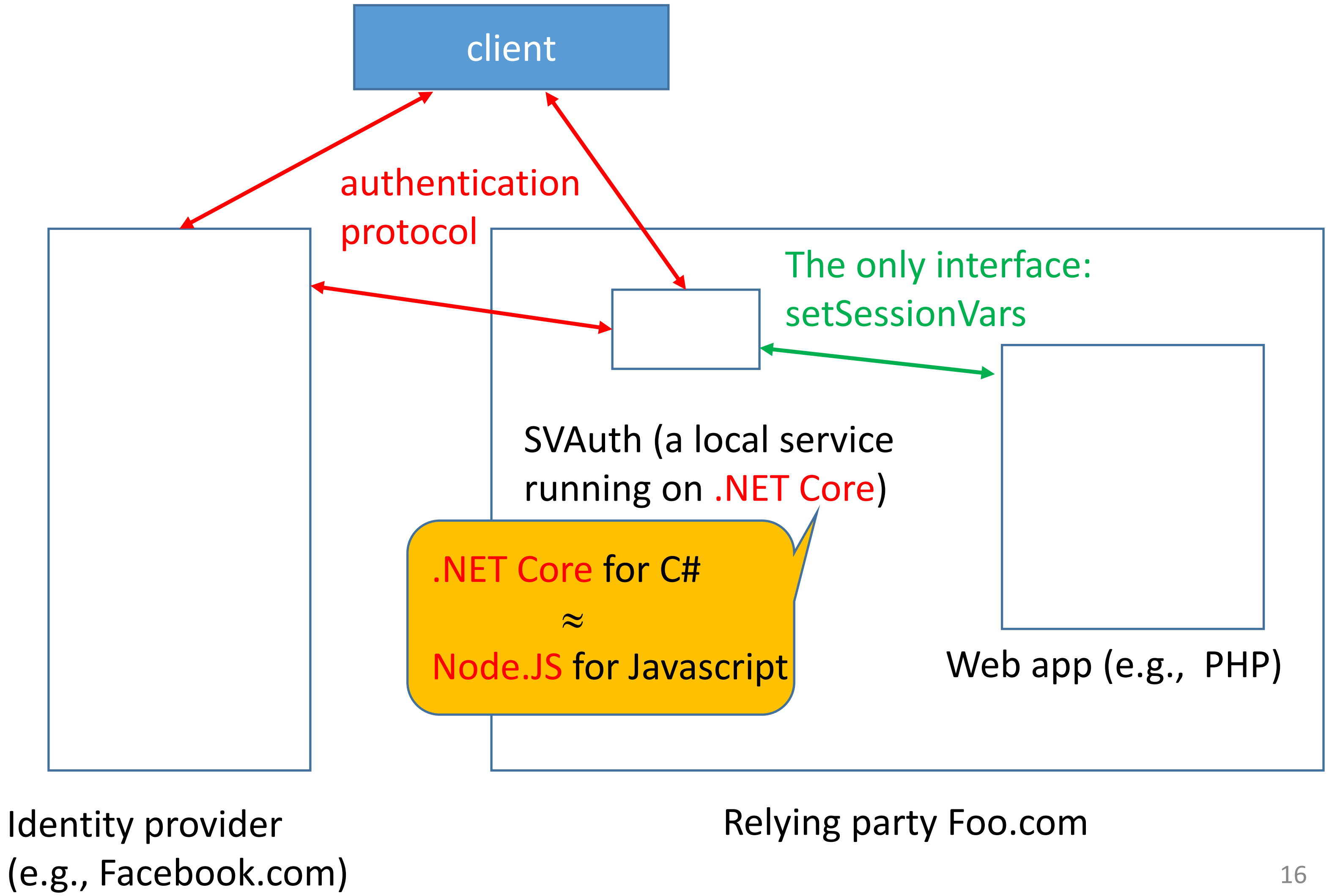
- Reduce the burden of modeling the attacker and the runtime platform
- The theorem (i.e., proof obligation) is much easier to prove automatically
- The self-verifying capability can be inherited
  - It is well known that a property proven for a base class may not carry through onto concrete classes. (Liskov Substitution Principle, LSP)
  - However, the self-verifying capability does carry through.

Focused project: SVAuth

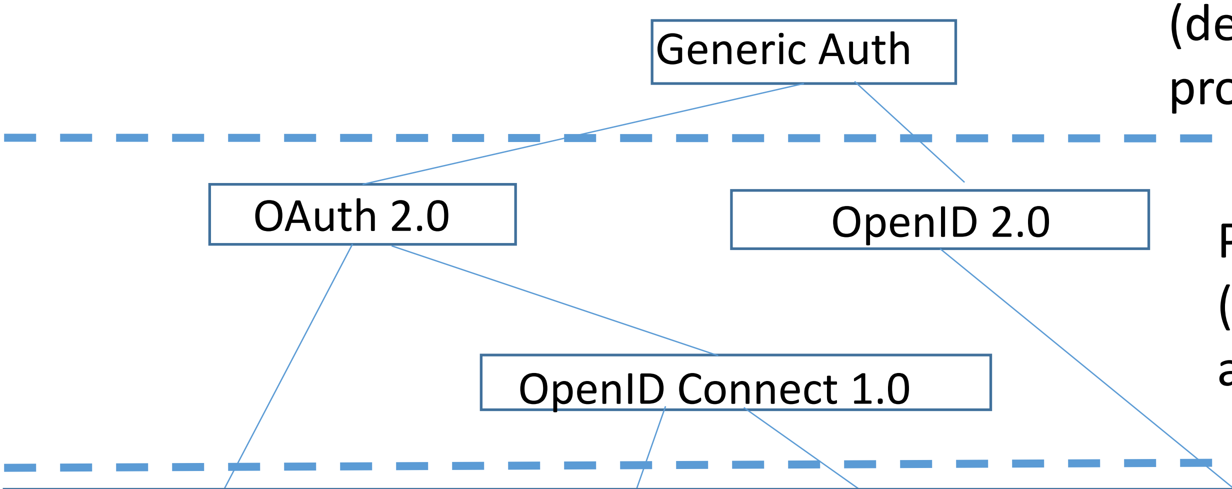
Overview of the SVAuth project

- Problem
  - Logic bugs in website sign-on mechanisms
  - Just like incorrectly installed front door locks for websites
- Vision
  - Every website’s front door lock should be installed with proven correctness.
- What is SVAuth?
  - Based on SVX, we are building relying party solutions that cover all major identity services in the world.
    - We have built solutions for 5 out of 30 major identity services

Architecture of SVAuth



SVAuth's class hierarchy



Protocol-independent level (defining a set of safety properties for all protocols)

Protocol level (where RecordMe calls are added into message handlers)

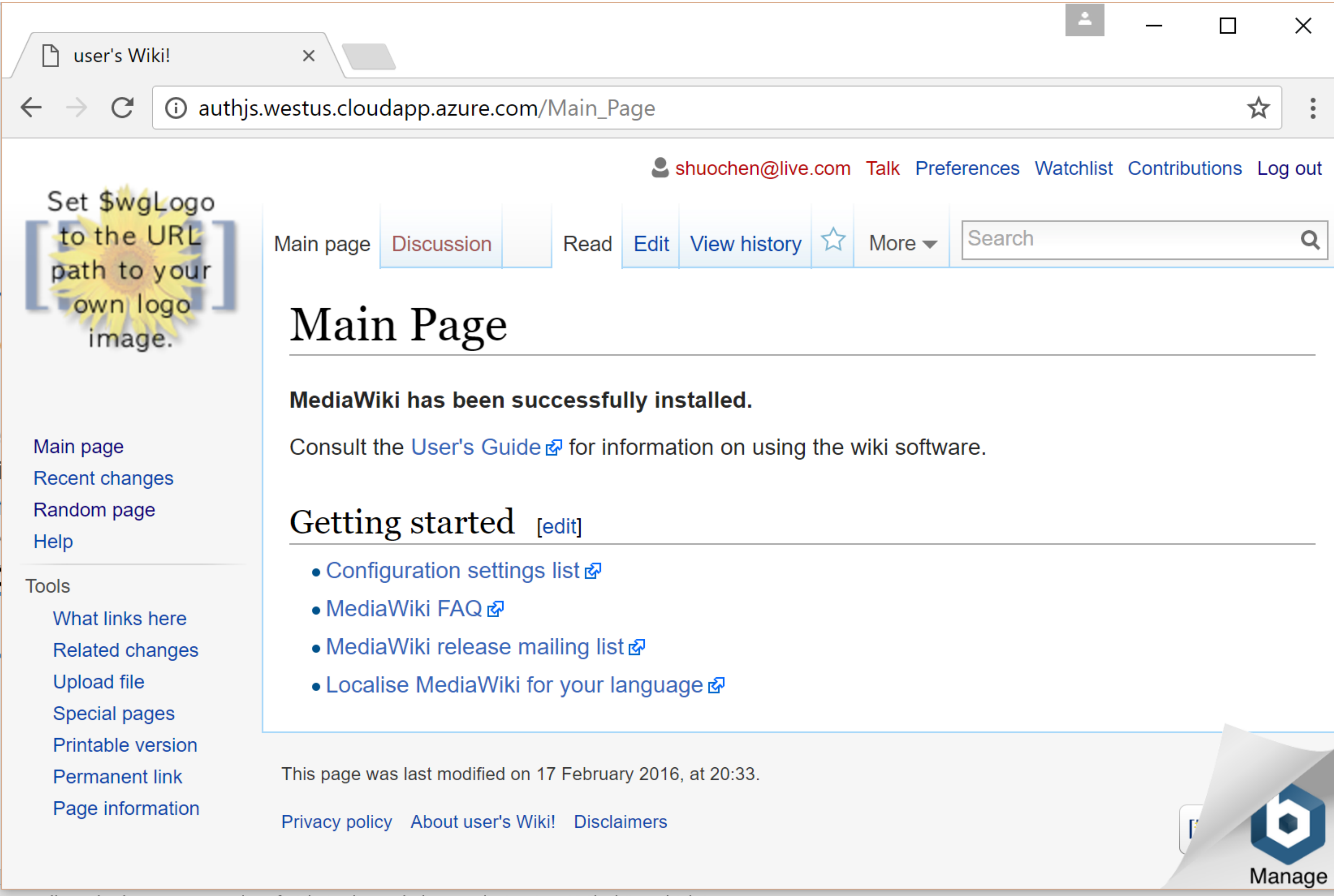
SDK level and website level:

* Programmers just need to do normal OO programming.
* The self-verifying capability will be inherited automatically.

Summary

- SVX is a practical method to do verification for online protocol implementations
- SVAuth seems promising to achieve our vision
  - Every website's front door lock should be installed with proven correctness.
- Realistic to use
  - Super easy to integrate SVAuth with real-world software
    - E.g., MediaWiki and HotCRP -- only a few lines of essential code changes.
  - A Microsoft Research website has been running MediaWiki with SVAuth for months
- You can help
  - If your website needs Facebook, Google, Yahoo, Microsoft Accounts and Microsoft Azure logins, send us email.
  - Welcome to contribute to the project (on GitHub).

backup


user's Wiki!
authjs.westus.cloudapp.azure.com/Main_Page
shuochen@live.com Talk Preferences Watchlist Contributions Log out
Set $wgLogo to the URL path to your own logo image.
Main page Discussion
Read Edit View history More
Search
Main Page
MediaWiki has been successfully installed.
Consult the User's Guide for information on using the wiki software.
Getting started [edit]
• Configuration settings list
• MediaWiki FAQ
• MediaWiki release mailing list
• Localise MediaWiki for your language
Main page
Recent changes
Random page
Help
Tools
What links here
Related changes
Upload file
Special pages
Printable version
Permanent link
Page information
This page was last modified on 17 February 2016, at 20:33.
Privacy policy About user's Wiki! Disclaimers
Manage