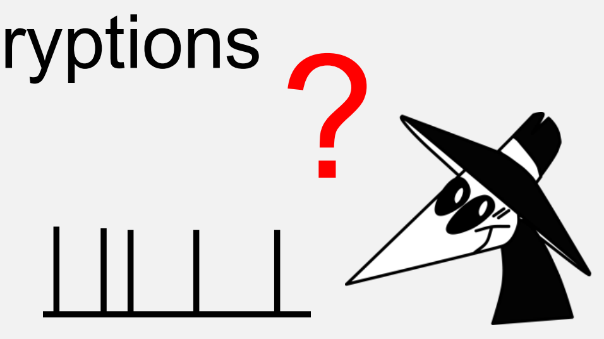
Language-Based Control and Mitigation of Timing Channels

Danfeng Zhang Cornell University
Aslan Askarov Harvard University
Andrew C. Myers Cornell University

The first language-based timing channel mitigation across language, system and architecture level

Timing channel threats

- **Network timing attacks**
  - RSA keys leaked by decryption time
  - Login/load time reveals validity of usernames, login status, size and contents of shopping cart
- **Cache timing attacks:** AES keys leaked by timing memory accesses from ~300 (!) encryptions
- **Covert channel**: transmit data by controlling timing



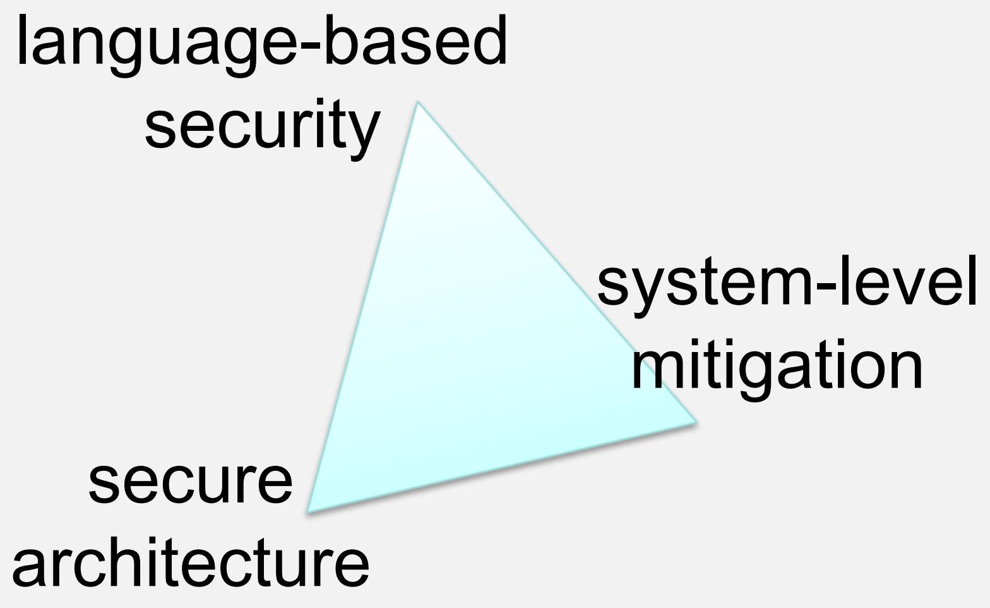
Example

```
if (secret1)
  secret2:= pulbic1;
else
  secret2:= public2;
public3:= public1;
```

Data cache affects timing

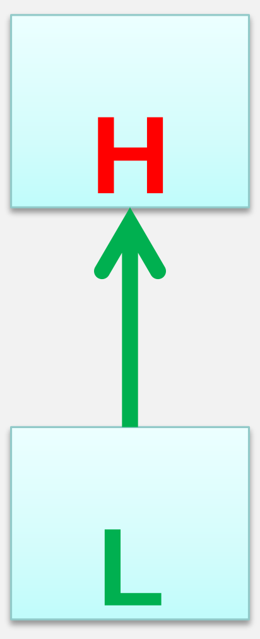
Overview

- **A language-level abstraction**
  - Bridges three levels of timing channel mitigation
- **Static analysis with proved guarantees**
- **Practical for real-world application**



Security model

- **Security policy lattice**
  - Information has label describing intended confidentiality
  - The labels form a lattice in general
- **Attacker model**
  - Sees contents of low memory (storage channel)
  - Sees timing of updates to low memory (timing channel)



A real threat for cloud computing

A language-level abstraction

Read/write labels with formal requirements

- **Machine environment:** state affecting timing but invisible at language level, e.g. data cache
- **Interface**
  - **Read label:** restricts how machine environment affects timing
  - **Write label:** restricts how machine environment is modified
  - Language implementation satisfies 3 formal requirements
- **Guidance to secure architecture design**
  - **Security:** possible to verify architecture design
  - **Performance:** avoids unnecessary overheads

All requirements are realizable on commodity HW

Type system and formal guarantees

Idea: track timing information in typing rules

$$e ::= n \mid x \mid e \; op \; e$$

$$c ::= \texttt{skip}_{[\ell_r,\ell_w]} \mid (x := e)_{[\ell_r,\ell_w]} \mid c;c \mid (\texttt{while}\; e \;\texttt{do}\; c)_{[\ell_r,\ell_w]}$$

$$\mid (\texttt{if}\; e \;\texttt{then}\; c_1 \;\texttt{else}\; c_2)_{[\ell_r,\ell_w]}$$

$$\mid (\texttt{mitigate}_\eta \; (e,\ell)\; c)_{[\ell_r,\ell_w]} \mid (\texttt{sleep}\; e)_{[\ell_r,\ell_w]}$$

- **Typing rules**

begin timing label

end timing label

$$\Gamma, pc, \tau \vdash c : \tau'$$

Qualitative result

A well-typed program without `mitigate` command leaks nothing via timing channels

`mitigate` command

Idea: improve expressiveness via dynamic control

Assume the validity of username is secret:

```
(name, pass):=input
if (exists(name))
  check(name, pass);
public:=1;
```

```
mitigate(1,S){
  (name, pass):=input
  if (exists(name))
    check(name, pass);
}
public:=1;
```

Bounded leakage via dynamic control

A well-typed program has bounded leakage

e.g. $O(\log^2 T)$

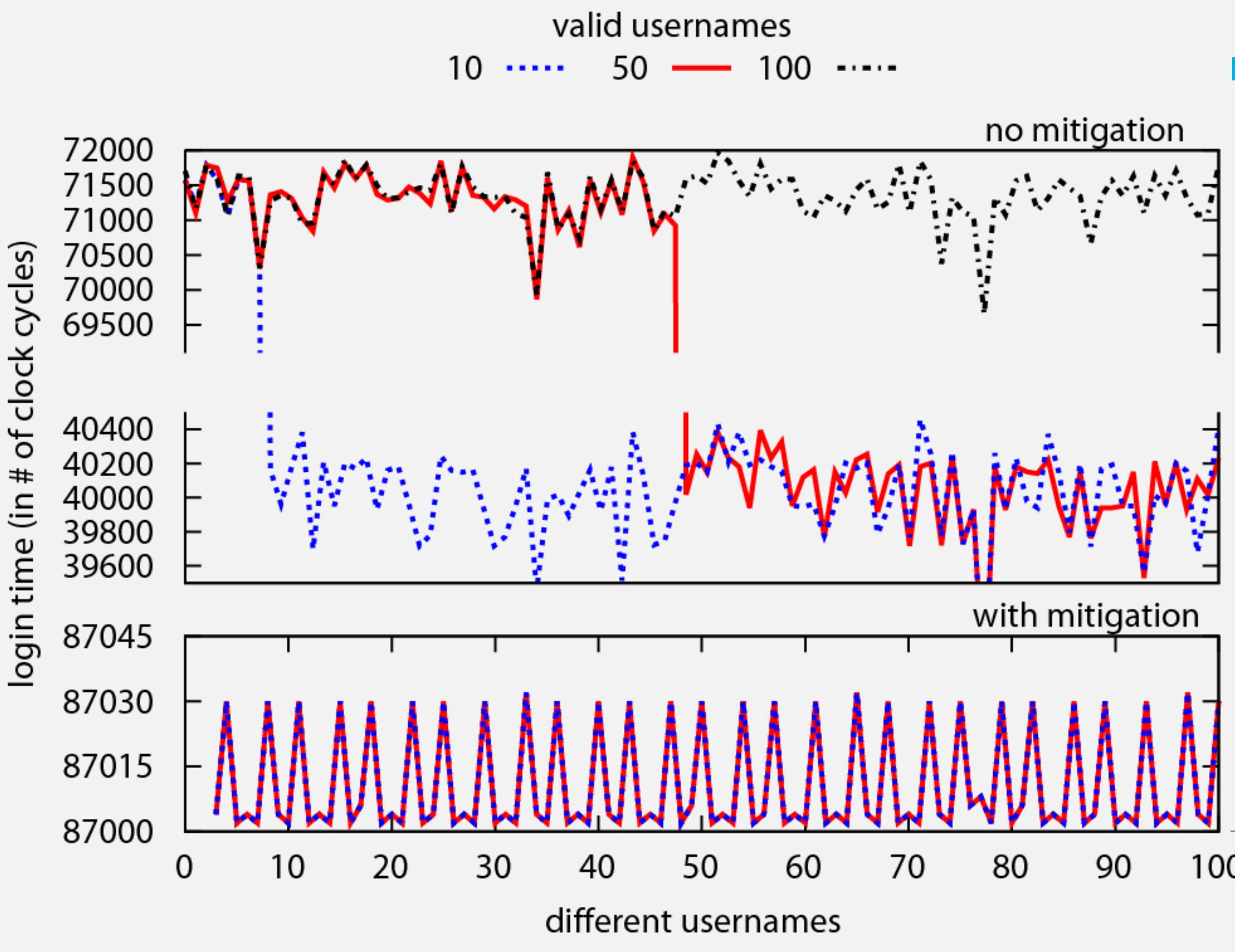
using predictive mitigation [ccs'10, ccs'11]

T: execution time of the program

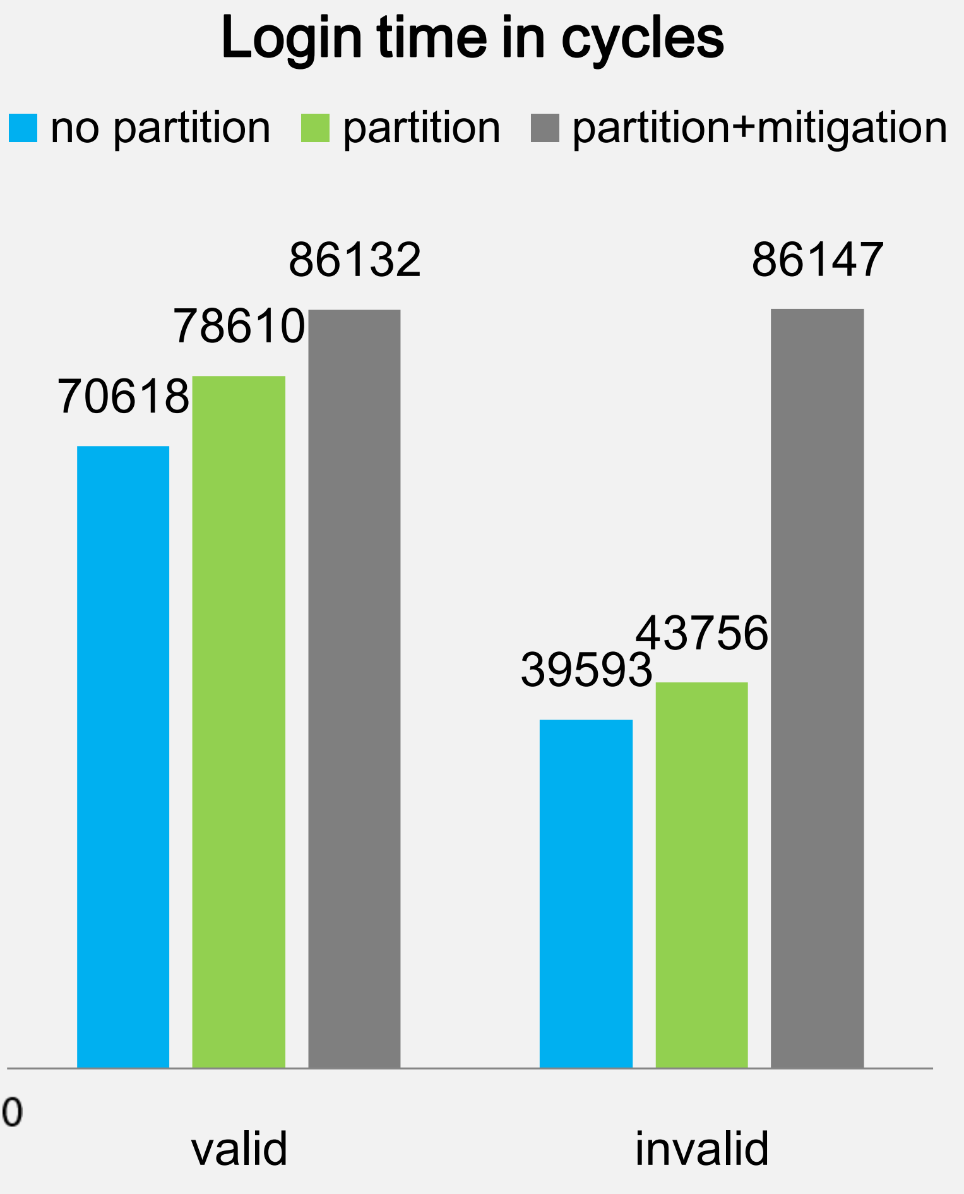
Evaluation

Implemented statically partitioned cache/TLB on SimpleScalar simulator

Web login: learn valid usernames via round-trip time
- Secret: validity of username, password pair
- Input: 100 pairs of username, password



security



performance

Cornell University