

Summer School "Secure Implementation of Cryptographic Software" – 27.08.17 – 03.09.17

---

# Approaches to Reliable Side-Channel Security

---

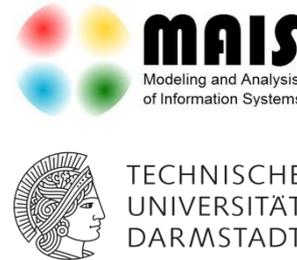
Alexandra Weber

MAIS group, TU Darmstadt, Germany

# Introduction

## About me

I am a PhD candidate at the chair „Modeling and Analysis of Information Systems“ of Prof. Heiko Mantel at TU Darmstadt, Germany



## My research

- mission: (machine) provable security guarantees
- current focus: security of cryptographic algorithms against side channels
- approach: systematic and proactive, using program analysis

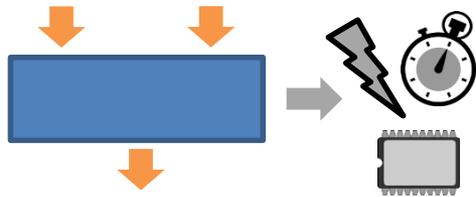
## I would like to thank my sponsors



# Approaches to Reliable Side-Channel Security

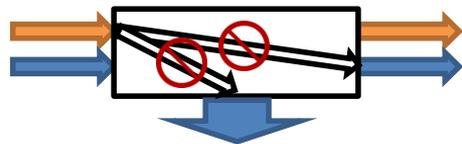
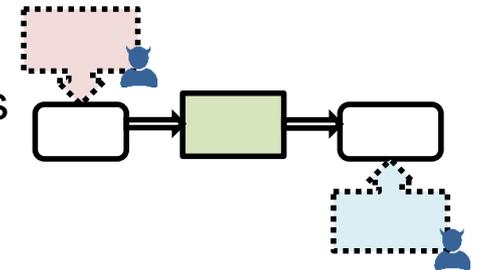
## Agenda

---



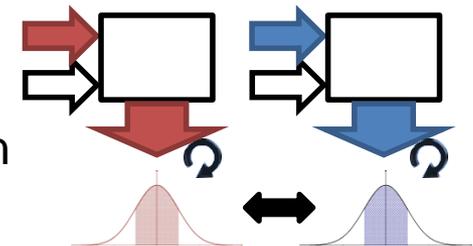
### 1. Basic Concepts

Approach 1: information theory and reachability analysis

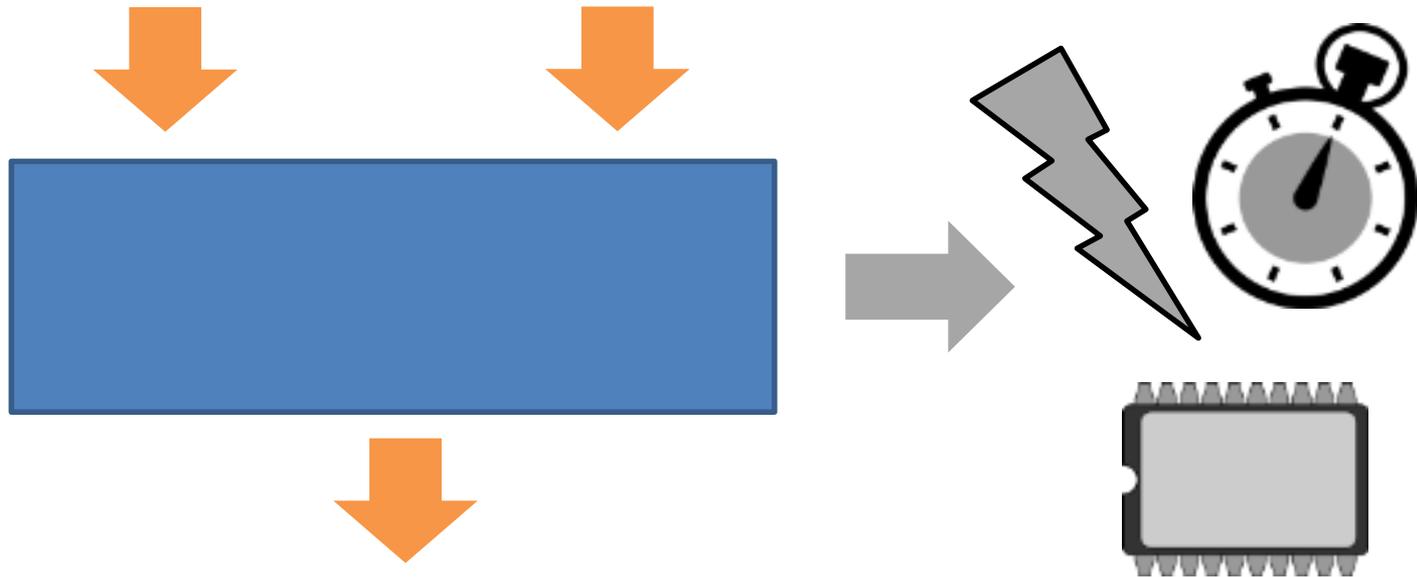


Approach 2: security type system

Approach 3: experimental evaluation



# Basic Concepts

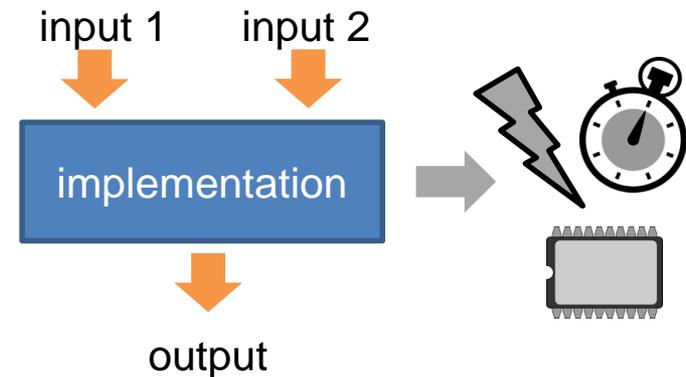


# Basic Concepts

## Side Channels I

### Physical Characteristics of Program Execution

- examples:
  - running time
  - power consumption
  - interaction with the cache



### Leakage through Execution Characteristics

- side-channel vulnerability:  
characteristics depends on secret information
- attacker observes characteristics, e.g.:
  - measures time over network
  - accesses cache from VM on same cloud server



## Basic Concepts

# Side Channels II

---

### Implementation-Level Vulnerabilities

- not always visible in the algorithm
- refinement to implementation can introduce vulnerabilities
- examples:
  - through control flow  
(e.g., square-and-multiply for modular exponentiation)
  - through use of data structures  
(e.g., lookup-tables for AES)

## Basic Concepts

# Example: Square-and-Multiply I

---

### Modular Exponentiation

- $z = x^y \text{ mod } N$  (remainder in division by  $N$ )
- e.g., decryption with RSA:  $message \leftarrow ciphertext^{secret\ key} \text{ mod } N$

### Square-and-Multiply

- example:  $x^{21}$
- binary representation of exponent:  $21_{10} = 10101_2$
- 1  $\rightarrow$  square-and-multiply:  $temp = (temp^2) * temp$
- 0  $\rightarrow$  square:  $temp = temp^2$

- $x^{21} = \left( \left( \left( \left( 1^2 * x \right)^2 * x \right)^2 * x \right)^2 * x \right)^2 * x$

## Basic Concepts

# Example: Square-and-Multiply II

---

## Square-and-Multiply Implementation of Modular Exponentiation

```
square-and-multiply(x, exponent)
  res = 1
  for (i=0; i<|exponent|; i++){
    res=res*res mod N;
    if (exponent[i]=1){
      res=res*x mod N
    }
  }
}
```

## Basic Concepts

# Example: Square-and-Multiply II

---

## Square-and-Multiply Implementation of Modular Exponentiation

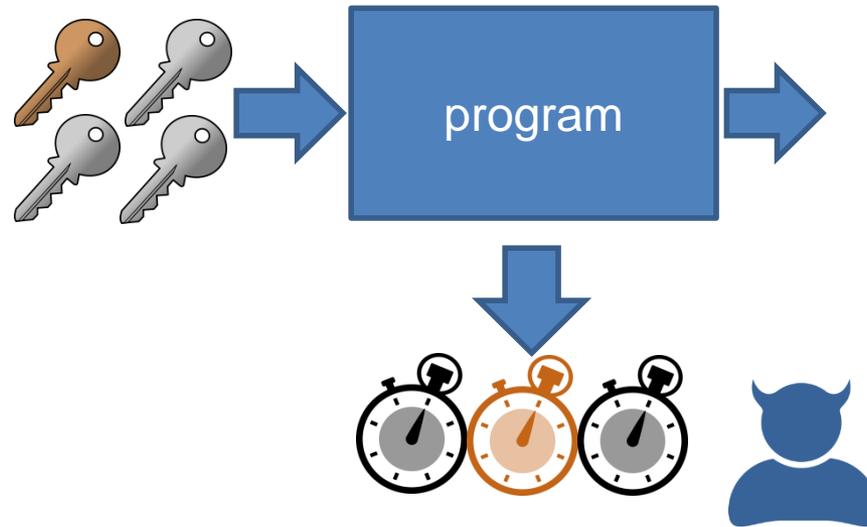
```
square-and-multiply(x, exponent)
  res = 1
  for (i=0; i<|exponent|; i++){
    res=res*res mod N;
    if (exponent[i]=1){
      res=res*x mod N
    }
  }
```



- $res=res*x \bmod N$  is only executed for 1-bits in exponent
- running time depends on Hamming weight of exponent

# Side-Channel Security

---



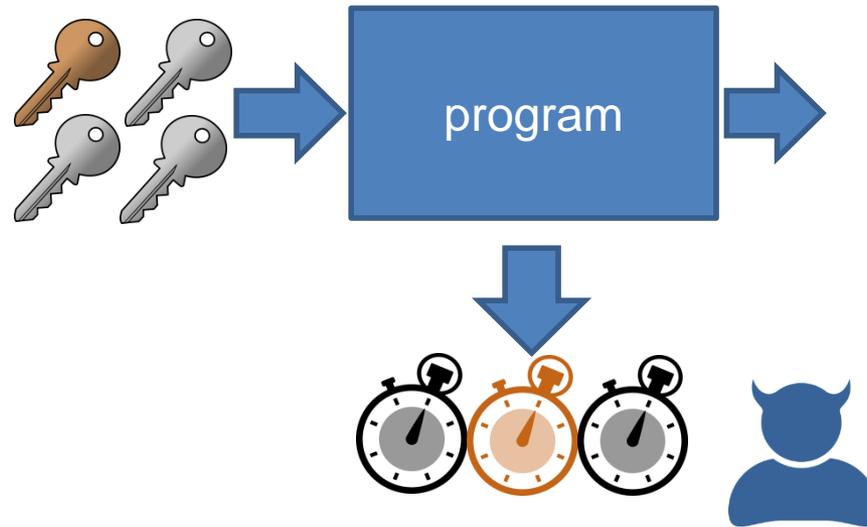
**How much information does the program leak through side channels in the worst case?**

**Is the program secure against side channels?**

**What do theoretical security guarantees for the program mean in practice?**

# Overview Approach 1: Information Theory and Reachability

---



**How much information does the program leak through side channels in the worst case?**

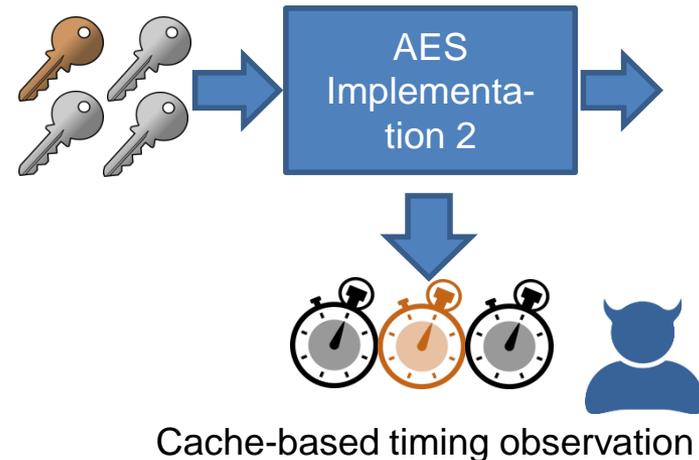
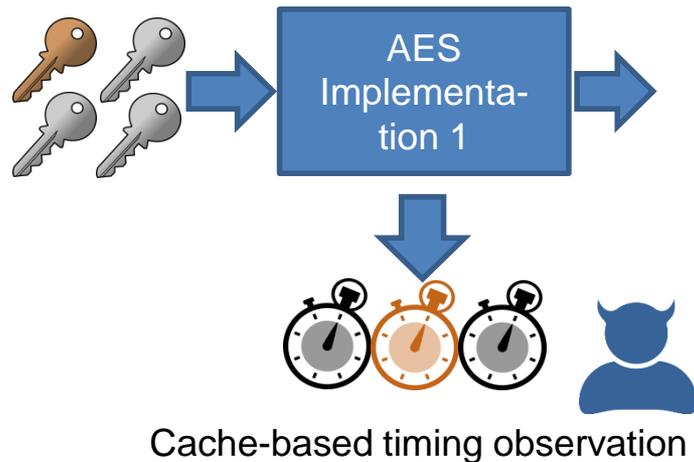
Information Theory:      leakage  $\leq |O|$ ,  $O$  hard to compute

Abstract Interpretation:       $O \leq \bar{O}$ ,  $\bar{O}$  can be computed

$\Rightarrow$  leakage  $\leq |\bar{O}|$

# Overview Application 1: Cache Side Channels in AES Implementations

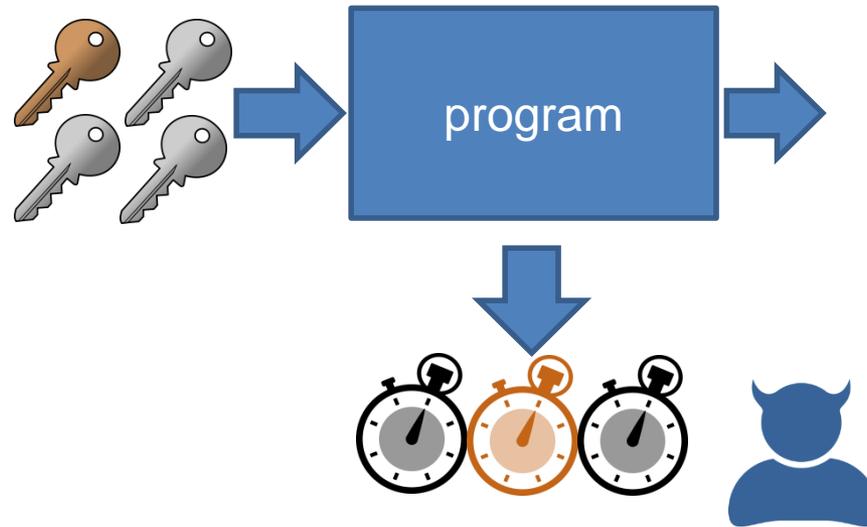
---



- How do different AES implementations compare in terms of leakage?
- How do implementation details influence the side-channel leakage?
- Use information theory and abstract interpretation to compute leakage bounds for different implementations in different scenarios.
- Interpret the leakage bounds with respect to implementation details.

# Overview Approach 2: Security Type System

---

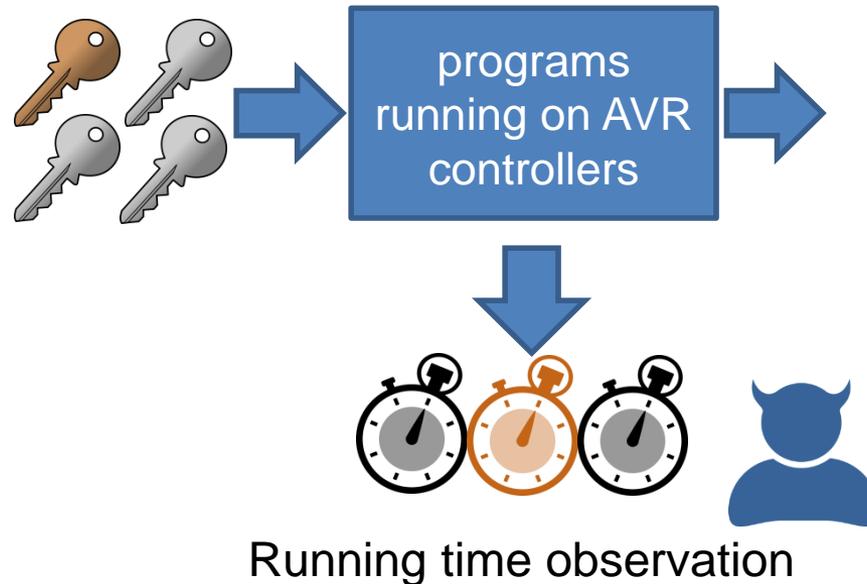


**Is the program secure against side channels?**

- Domain assignment: label secret information with „H“ and public information with „L“.
- Define (syntactic) check where secret information is propagated.
- Define a formal model how syntax influences running time.
- Prove that the check only succeeds for programs without leakage.

# Overview Application 2: Timing Side Channels in IoT Implementations

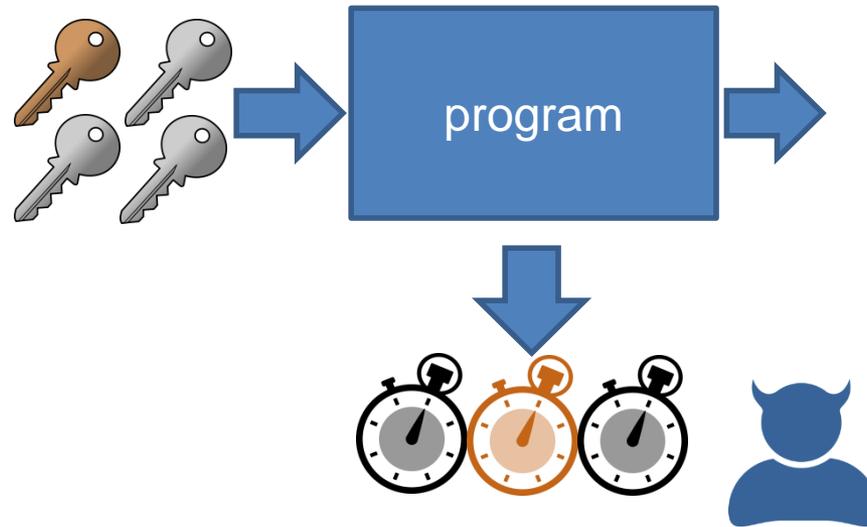
---



- check for AVR assembly programs against timing side channels
- formal semantics of AVR assembly  
(models time taken to execute given syntax)
- proof that the syntactic check has no false negatives
- automation in a tool
- evaluation on implementations from a crypto library for AVR processors

# Overview Approach 3: Experimental Evaluation

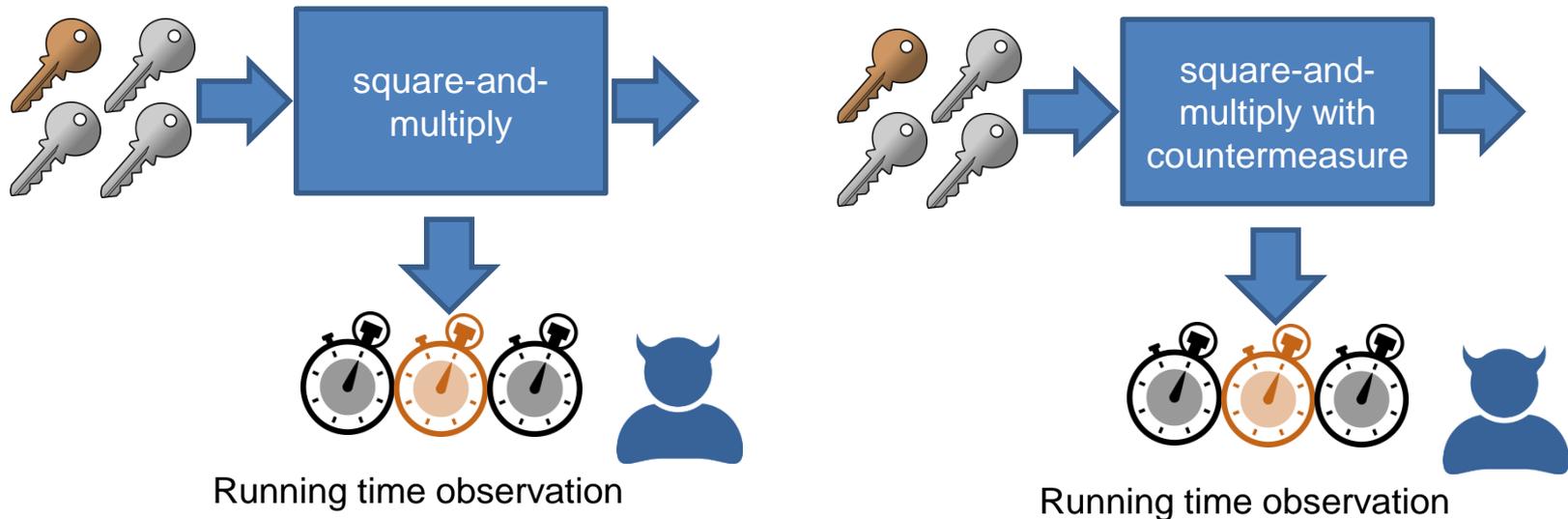
---



**What do theoretical security guarantees for the program mean in practice?**

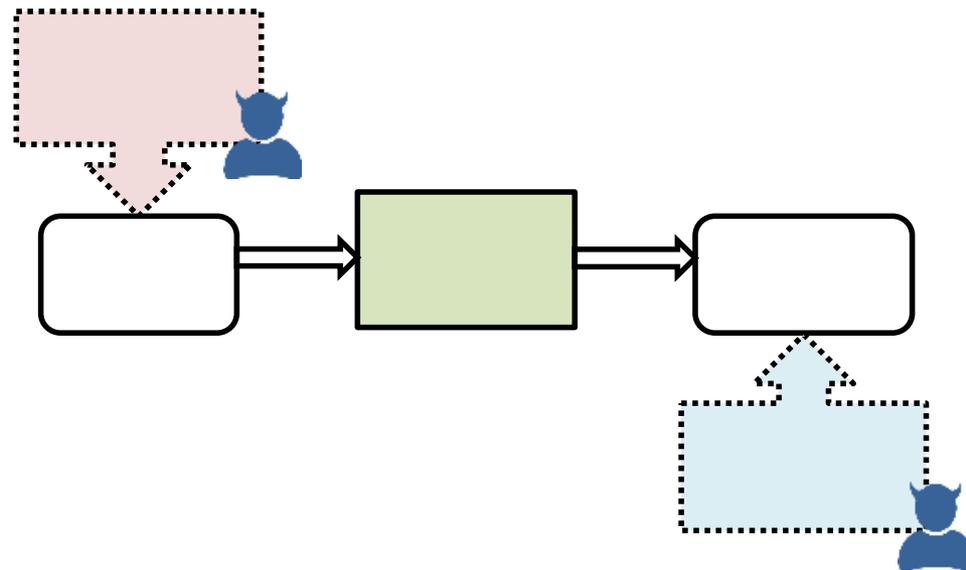
- Information Theory: leakage  $\leq$  channel capacity.
- Statistics: channel capacity can be approximated if enough sample observations are available.
- Obtain samples from experiments using program executions.

# Overview Application 3: Evaluation of Side-Channel Mitigations



- Approximate channel capacity of square-and-multiply.
- Approximate channel capacity of square-and-multiply with countermeasure.
- Quantify effectiveness of countermeasure by reduction of capacity.
- Compare across different countermeasures.

# Approach 1: Information Theory and Reachability Analysis

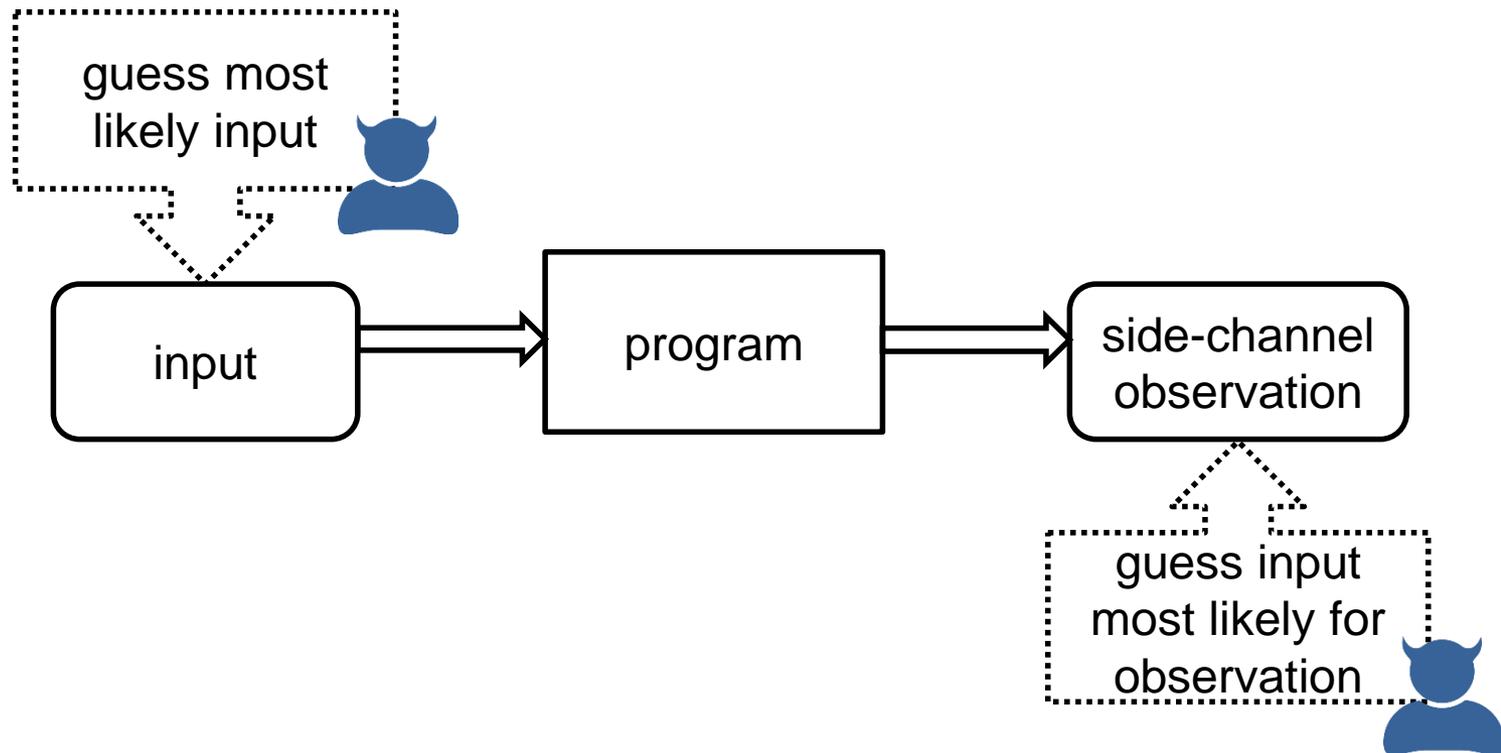


# Approach 1: Information Theory and Reachability Analysis

## Information Theoretic Channels

---

### Program as a Channel

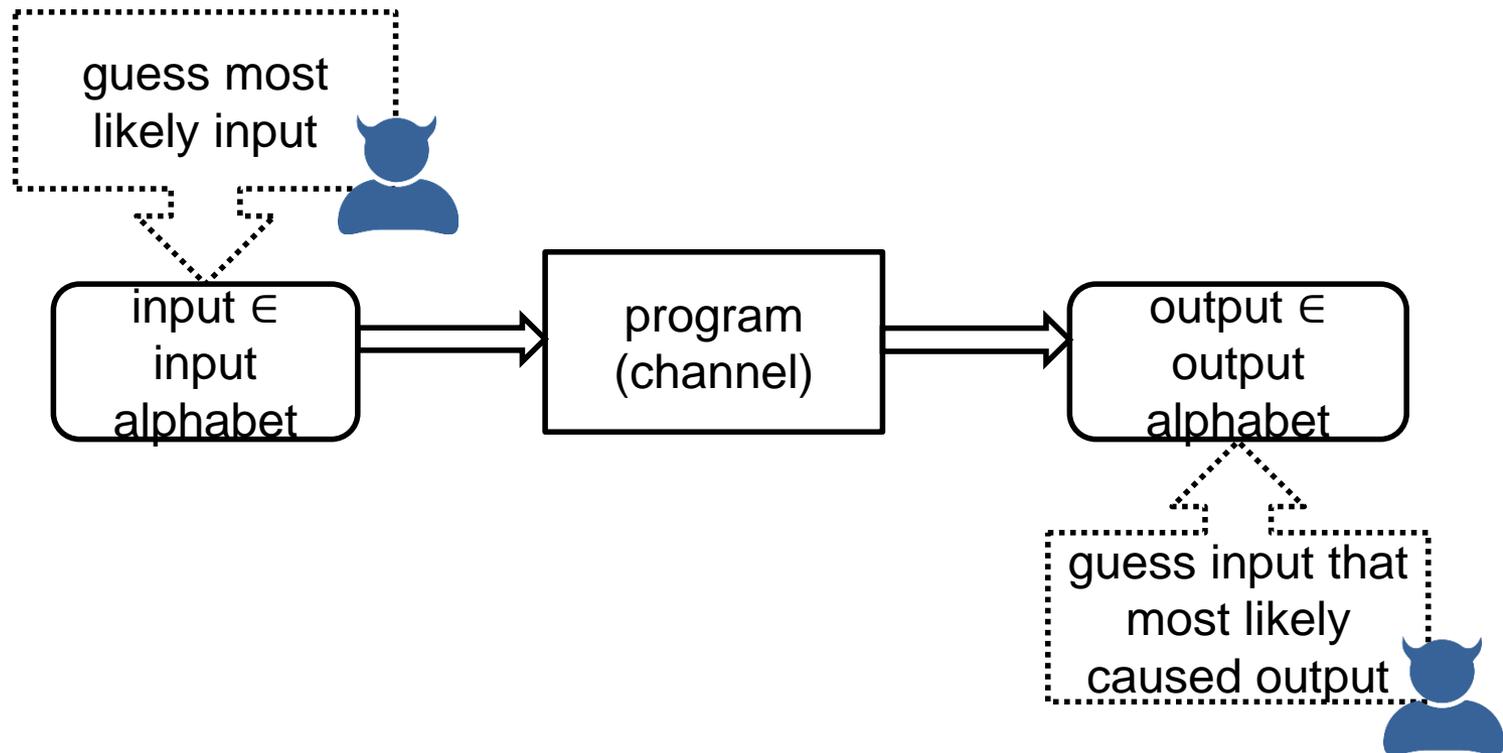


# Approach 1: Information Theory and Reachability Analysis

## Information Theoretic Channels

---

### Program as a Channel

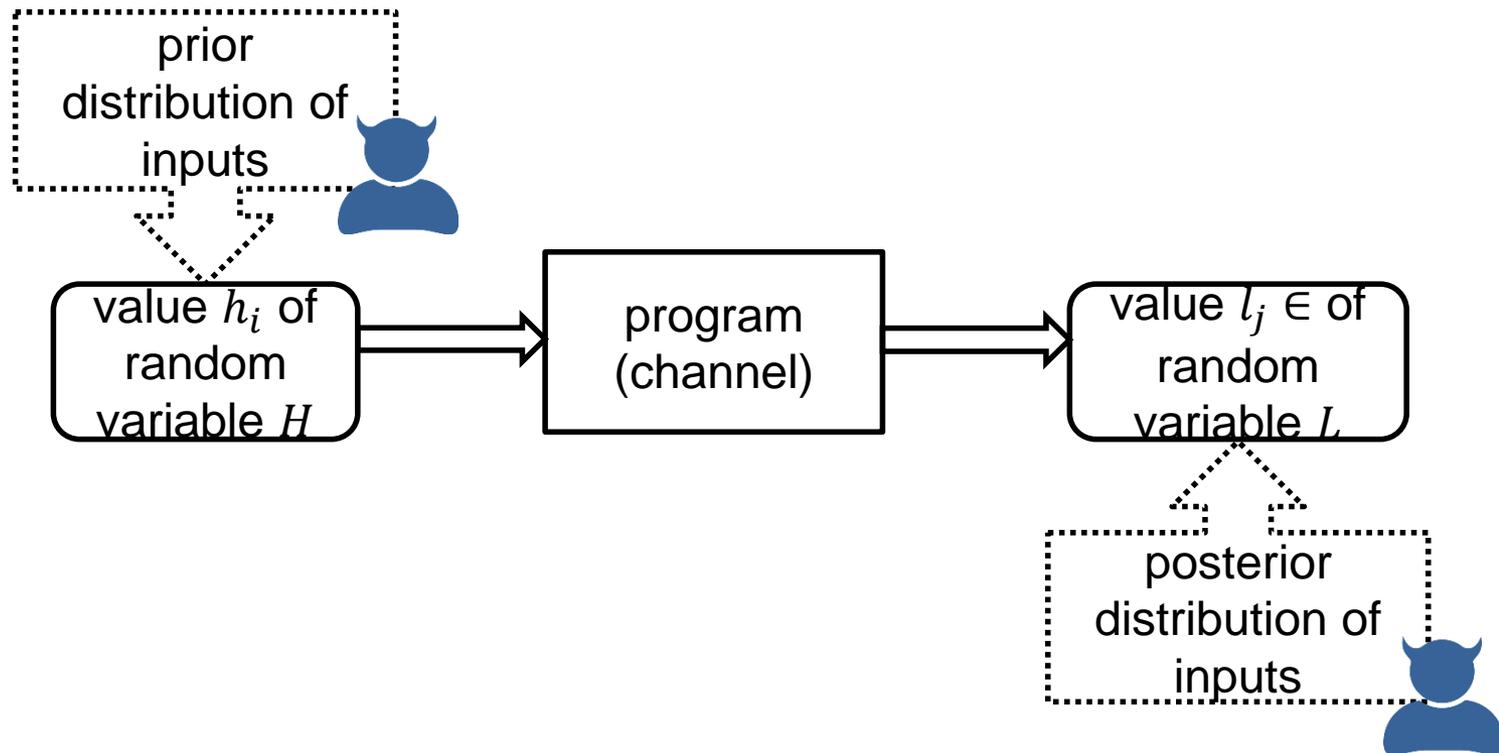


# Approach 1: Information Theory and Reachability Analysis

## Information Theoretic Channels

---

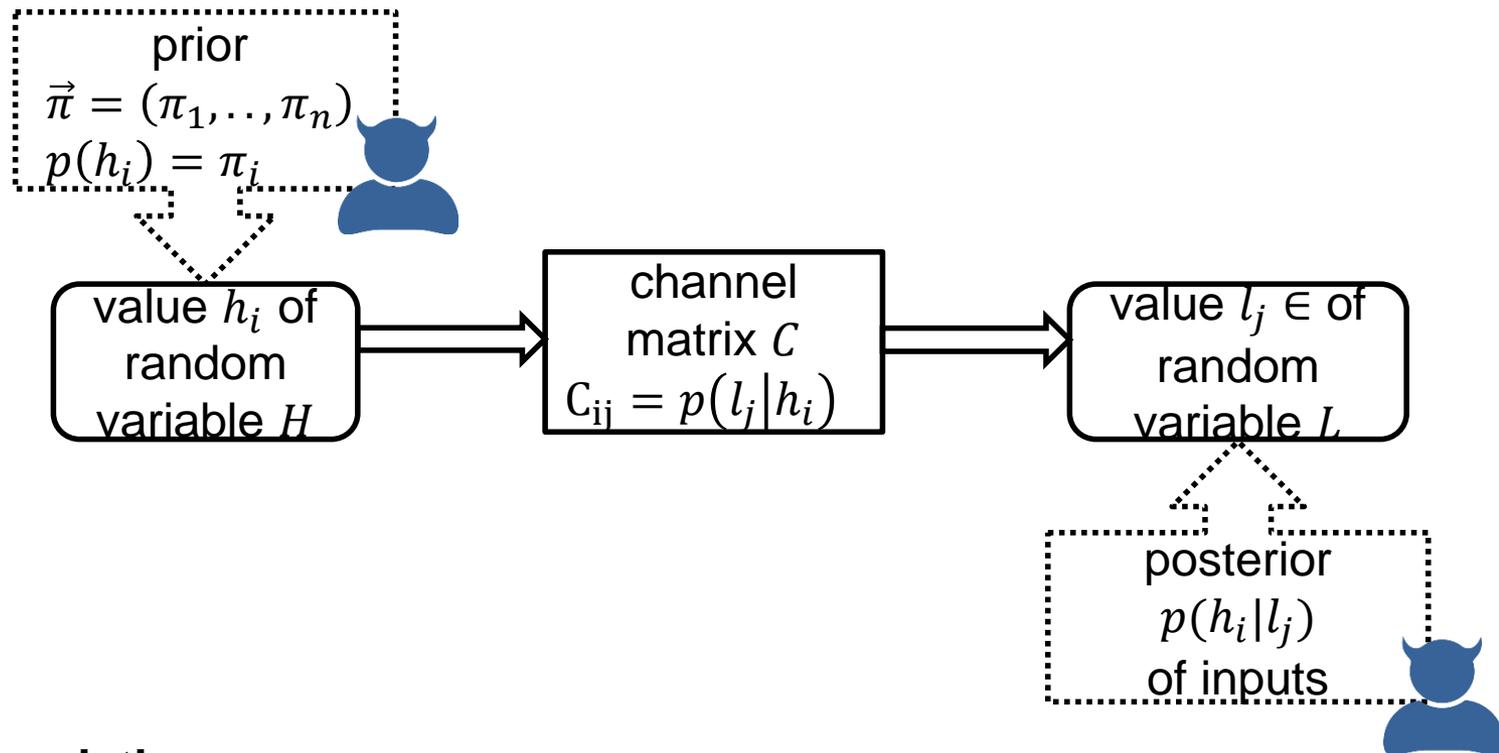
### Program as a Channel



# Approach 1: Information Theory and Reachability Analysis

## Information Theoretic Channels

### Program as a Channel



### Abbreviations

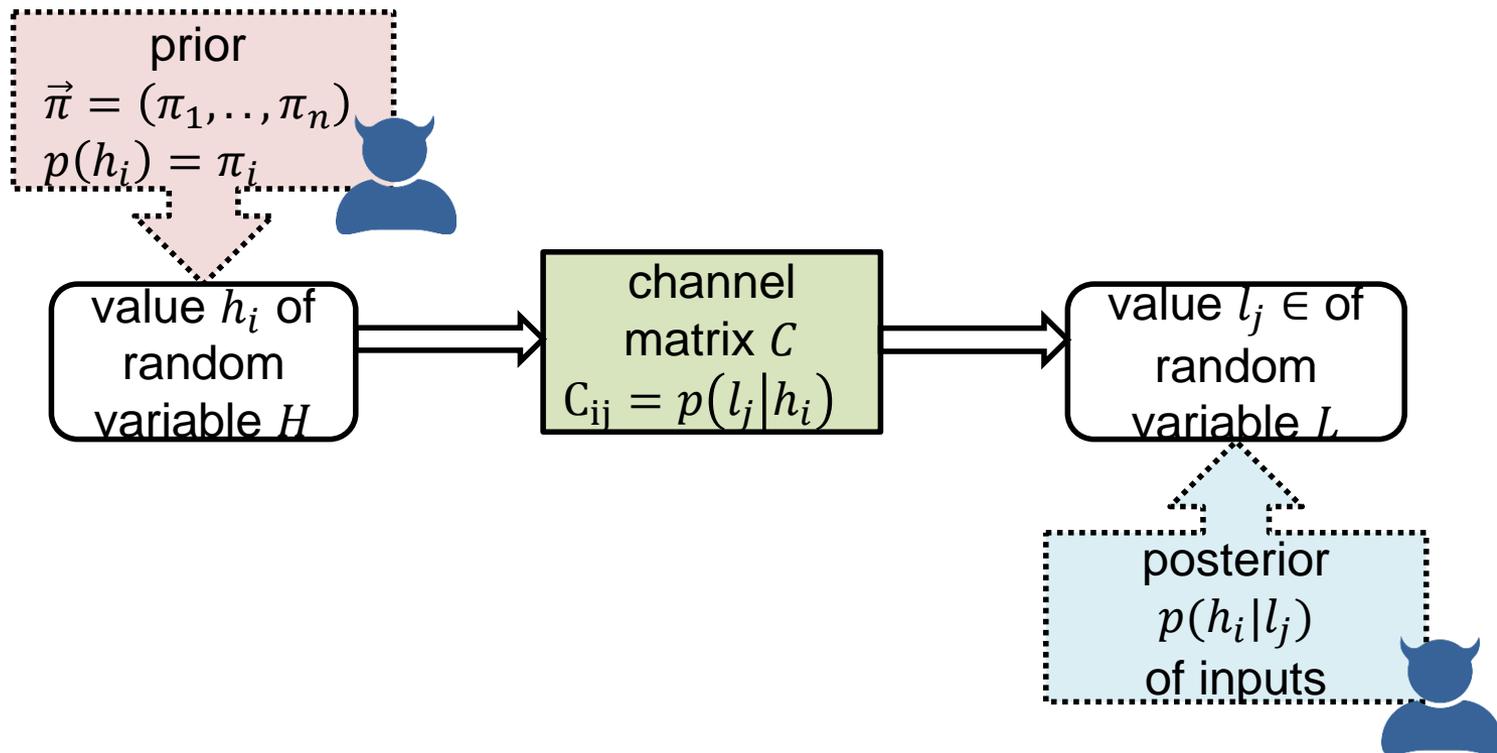
$$p(h_i) = p(H = h_i), p(l_i) = p(L = l_i)$$

$$p(h_i | l_j) = p(H = h_i | L = l_j), p(l_j | h_i) = p(L = l_j | H = h_i)$$

# Approach 1: Information Theory and Reachability Analysis

## Information Theoretic Channels

### Program as a Channel



# Approach 1: Information Theory and Reachability Analysis

## Information in a Random Variable

### Optimal Encoding of Values

- input to program is a value  $h_i$  of random variable  $H$
- we want to send an infinite list of values sampled from  $H$
- we want to encode them optimally
- inputs that occur the most often get the shortest encoding
- then encoding of input  $h_i$  requires  $\log\left(\frac{1}{p(h_i)}\right)$  bits

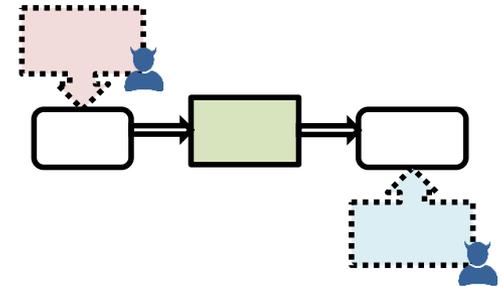
Prior distribution	Encoding Length	Intuition
$p(h_i) = \frac{1}{n}$	$\log\left(\frac{1}{p(h_i)}\right) = \log(n)$	need $\log(n)$ bits to represent any message drawn from uniform distribution
$p(h_i) = \begin{cases} 0, & h_i \neq 15 \\ 1, & h_i = 15 \end{cases}$	$\log\left(\frac{1}{p(h_i)}\right) = \begin{cases} 0, & h_i = 15 \\ \perp, & h_i \neq 15 \end{cases}$	need 0 bits to represent any message if every message is "15"

# Approach 1: Information Theory and Reachability Analysis

## Information in a Random Variable

### Shannon Entropy

- information in random variable
- optimal average encoding length
- reminder:  $-\log(A) = \log\left(\frac{1}{A}\right)$



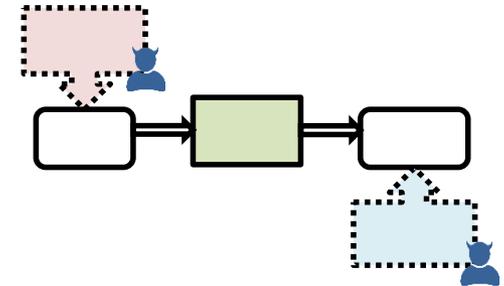
$$H_1(H) = \sum_i p(h_i) * \log\left(\frac{1}{p(h_i)}\right) = - \sum_i p(h_i) * \log p(h_i)$$

# Approach 1: Information Theory and Reachability Analysis

## Information in a Random Variable

### Conditional Shannon Entropy

- information in one random variable conditioned on the value of another
- weighted average of entropies for possible conditions



$$H_1(H|L) = - \sum_j p(l_j) H_1(H|L = l_j) = \sum_{ij} p(h_i, l_i) * \log p(h_i|l_i)$$

### More Information on Shannon Entropy

- Shannon, Claude Elwood. "A mathematical theory of communication." ACM SIGMOBILE Mobile Computing and Communications Review 5.1 (2001): 3-55.

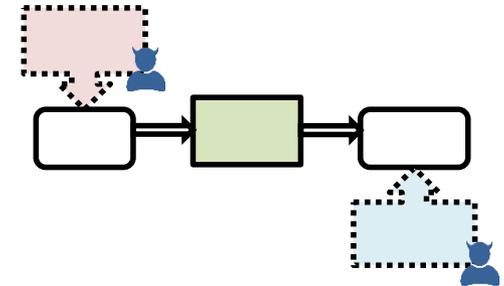
# Approach 1: Information Theory and Reachability Analysis

## Mutual Information

---

### Mutual Information

- amount of information that one random variable contains about another
- decrease in encoding length through conditioning



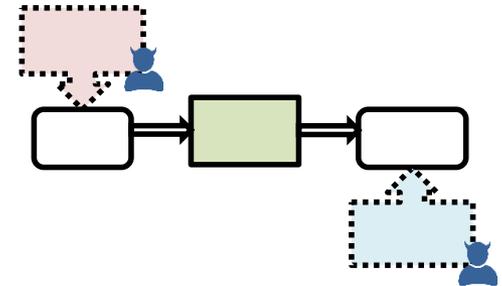
$$I_1(H; L) = H_1(H) - H_1(H|L)$$

# Approach 1: Information Theory and Reachability Analysis

## Leakage

### Leakage of a Program

- amount of information that output contains about input
- depends on prior distribution of inputs  $\vec{\pi}$ 
  - if the input is always “15“, the output does not reveal anything new



$$L_1(\vec{\pi}, C) = H_1(H) - H_1(H|L)$$

### Worst-Case Leakage of a Program

- channel capacity
- leakage for worst-case prior

$$ML_1 = \max_{\vec{\pi}} L_1(\vec{\pi}, C)$$

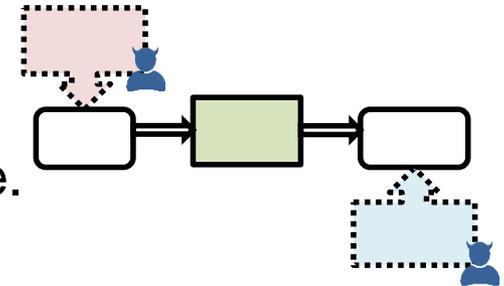
# Approach 1: Information Theory and Reachability Analysis

## Bounding the Worst-Case Leakage

### Leakage Bound

Consider a channel with channel matrix  $C$  from random variable  $H$  with prior  $\vec{\pi}$  to random variable.

$$\text{Then, } L_1(\vec{\pi}, C) \leq L_\infty(\vec{\pi}, C) \leq \log_2(|L|)$$



### Why?

- [Köpf/Smith, CSF 2010, Thm. 1 (notation adapted)]:

The min-entropy channel capacity of  $C$  is

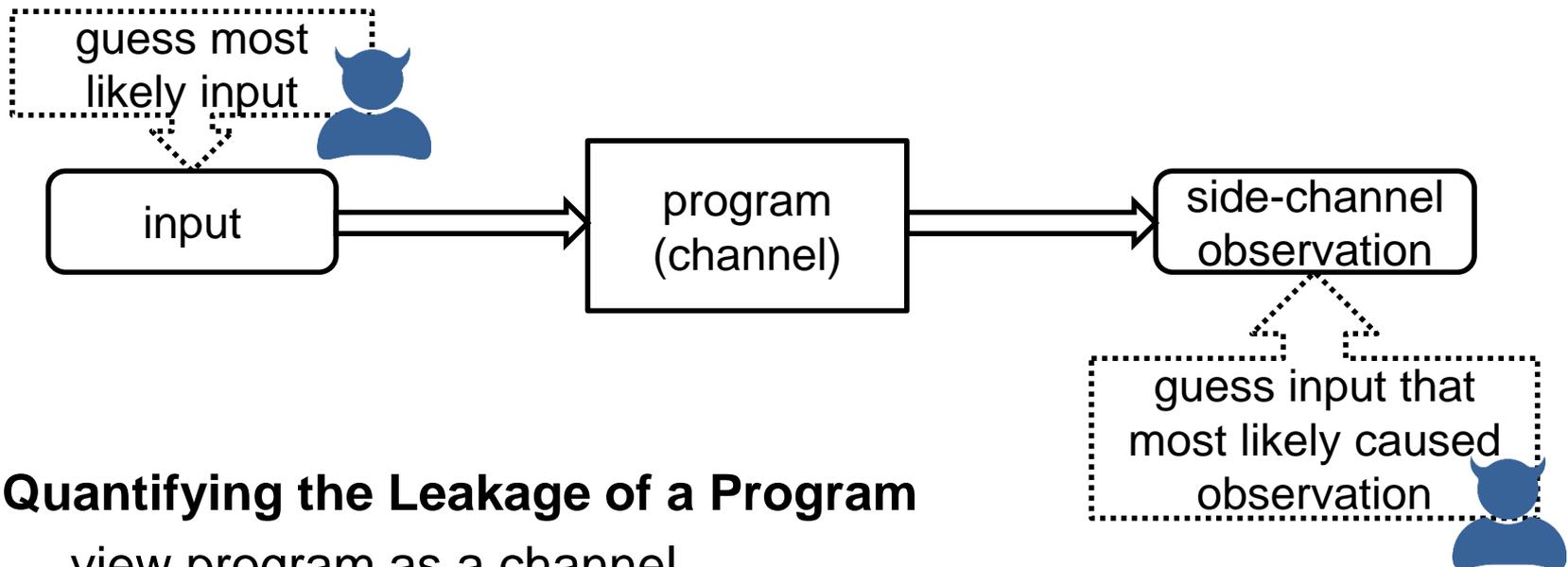
$$ML_\infty(\vec{\pi}, C) = \log \sum_{l_j \in L} \max_{h_i \in H} p(l_j | h_i)$$

and it is realized by a uniform prior distribution on  $H$

- $\log \sum_{l_j \in L} \max_{h_i \in H} p(l_j | h_i) \leq \log \sum_{l_j \in L} 1$
- [Alvim/Chatzikokolakis/Palamidessi/Smith, CSF 2012, Thm. 5.3]:  
For any channel  $C$ ,  $C$ 's min-capacity is at least as great as its Shannon capacity.

# Approach 1: Information Theory and Reachability Analysis

## Information Theory



### Quantifying the Leakage of a Program

- view program as a channel
- let  $O$  be the set of possible outputs (side-channel observations)
- worst-case leakage can be measured by channel capacity
- multiple notions of capacity exist, e.g. Shannon-capacity and min-capacity
- both capacities can be upper bounded by  $\log |O|$

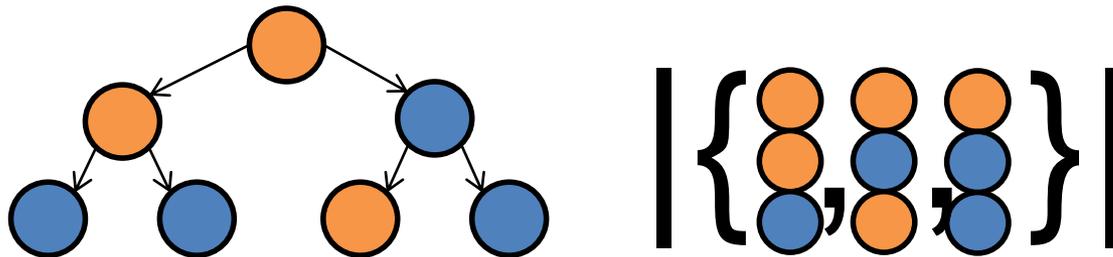
# Approach 1: Information Theory and Reachability Analysis

## Reachability Analysis

---

### Computing Leakage Bounds

- compute  $\log |O|$
- $O$  are the possible side-channel observations
- How to compute  $O$ ?



### Overapproximating Observations

- logarithm is monotonic
- for  $O' \supseteq O$ ,  $\log |O| \leq \log |O'|$
- can overapproximate reachable observations
- statically analyze program code

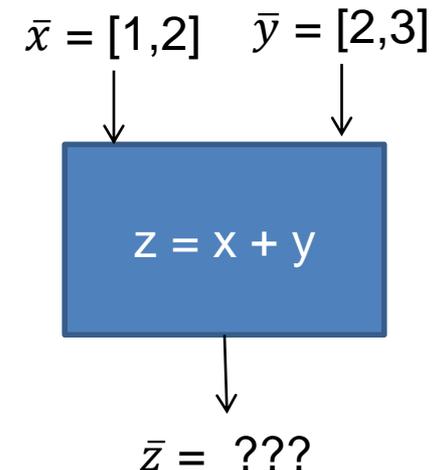
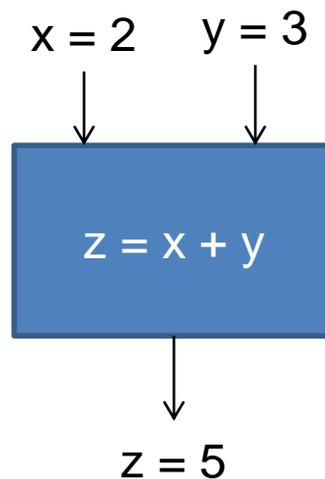
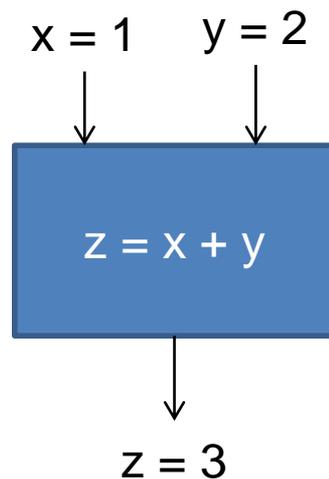
# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation

---

### Overapproximating Observations

- abstract from details of execution, e.g., from exact input
- interpret program abstractly (executions on abstract states)



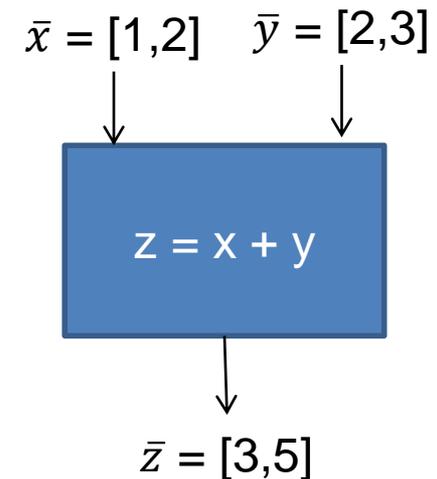
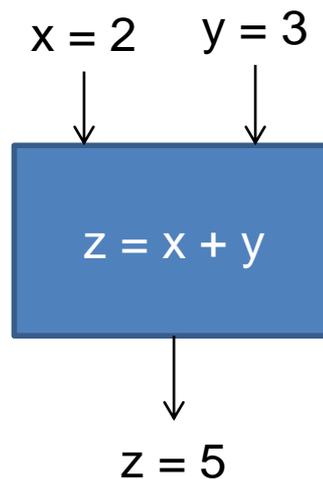
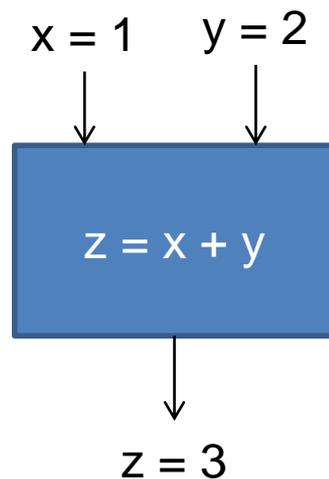
# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation

---

### Overapproximating Observations

- abstract from details of execution, e.g., from exact input
- interpret program abstractly (executions on abstract states)



# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation

---

concrete state  $q$

**concrete domain  $D$**

# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation

---

concrete state  $q$   $\xrightarrow{\text{concrete semantics } s: D \rightarrow D}$

**concrete domain  $D$**

# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation

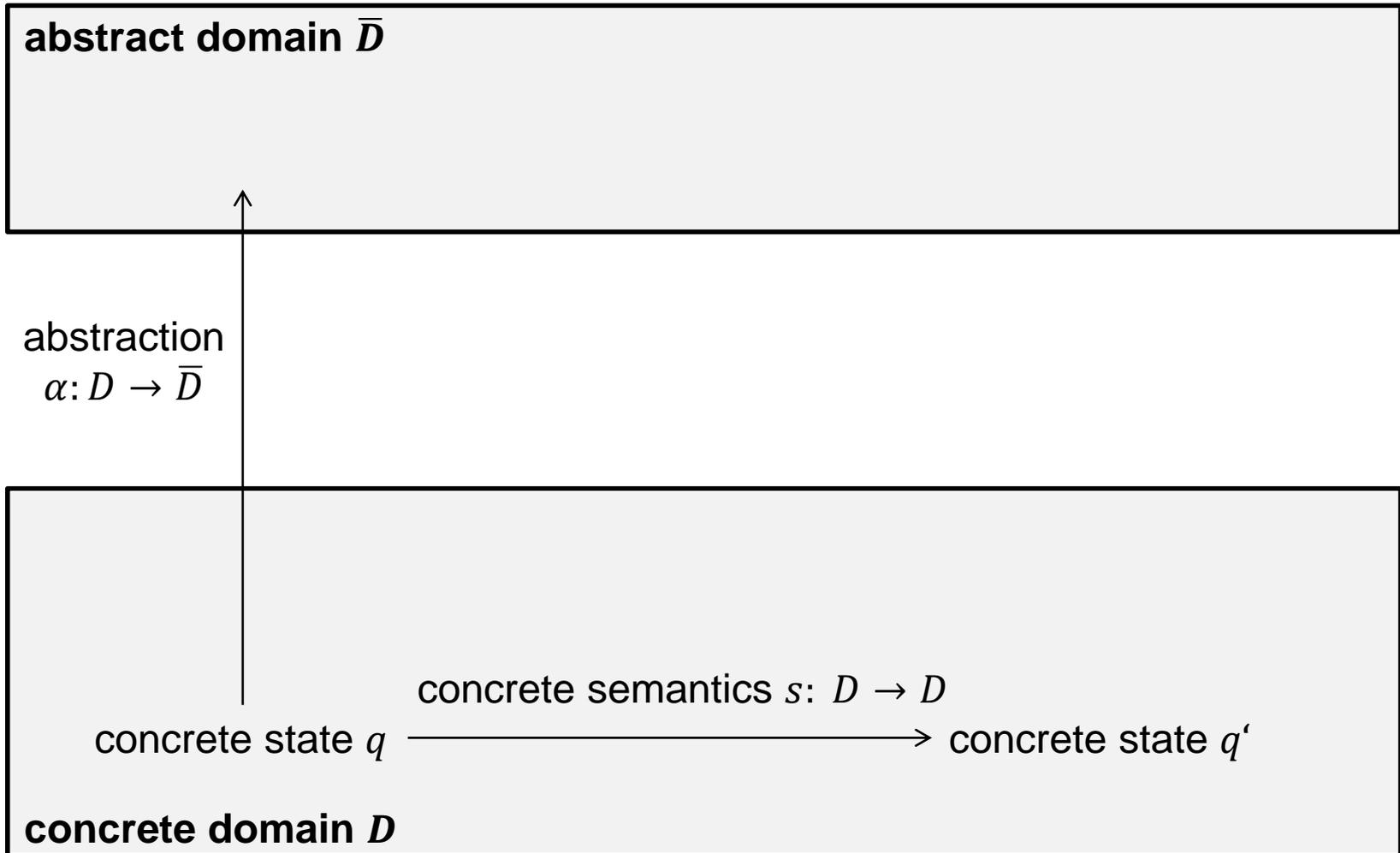
---

concrete state  $q$   $\xrightarrow{\text{concrete semantics } s: D \rightarrow D}$  concrete state  $q'$

**concrete domain  $D$**

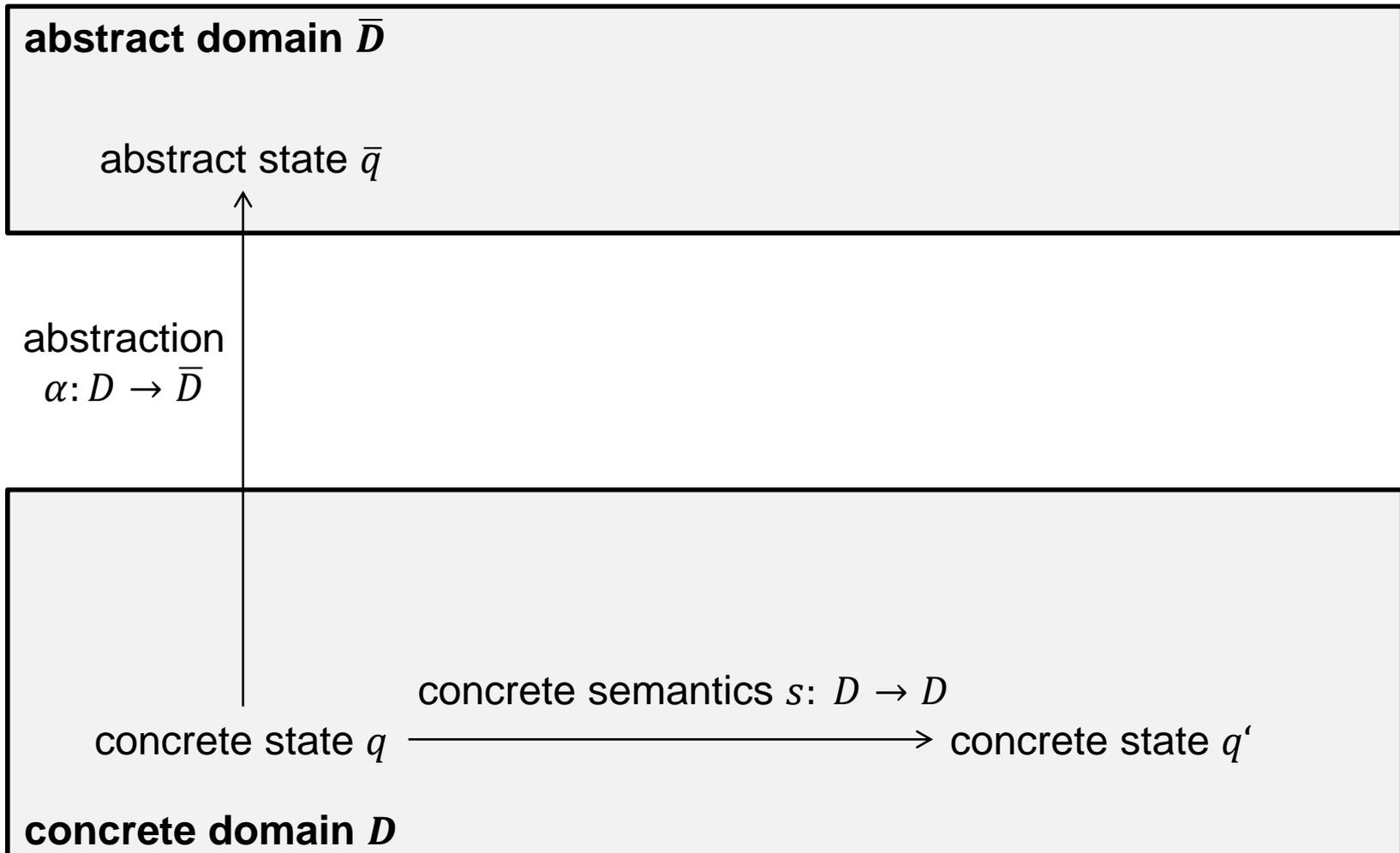
# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation



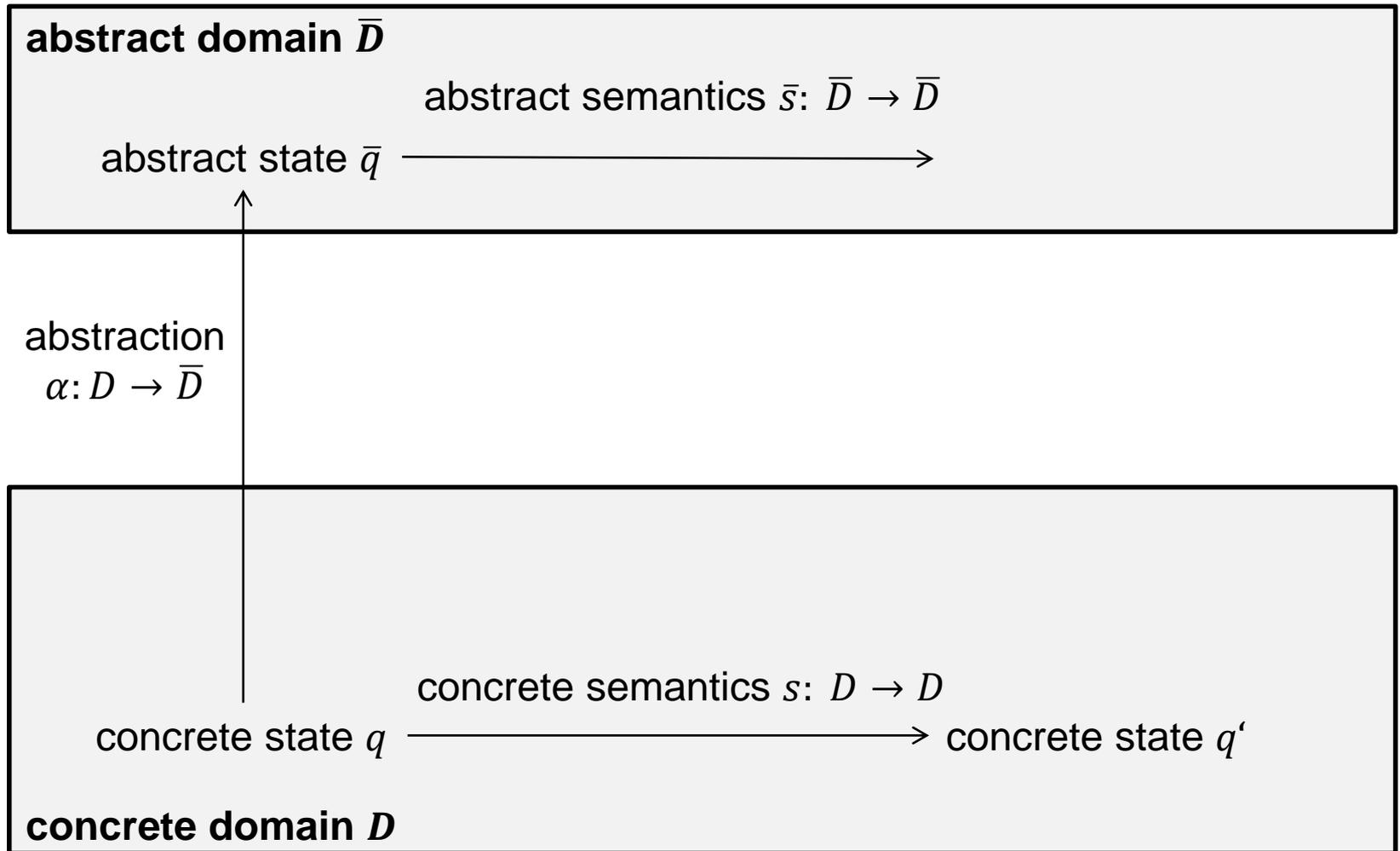
# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation



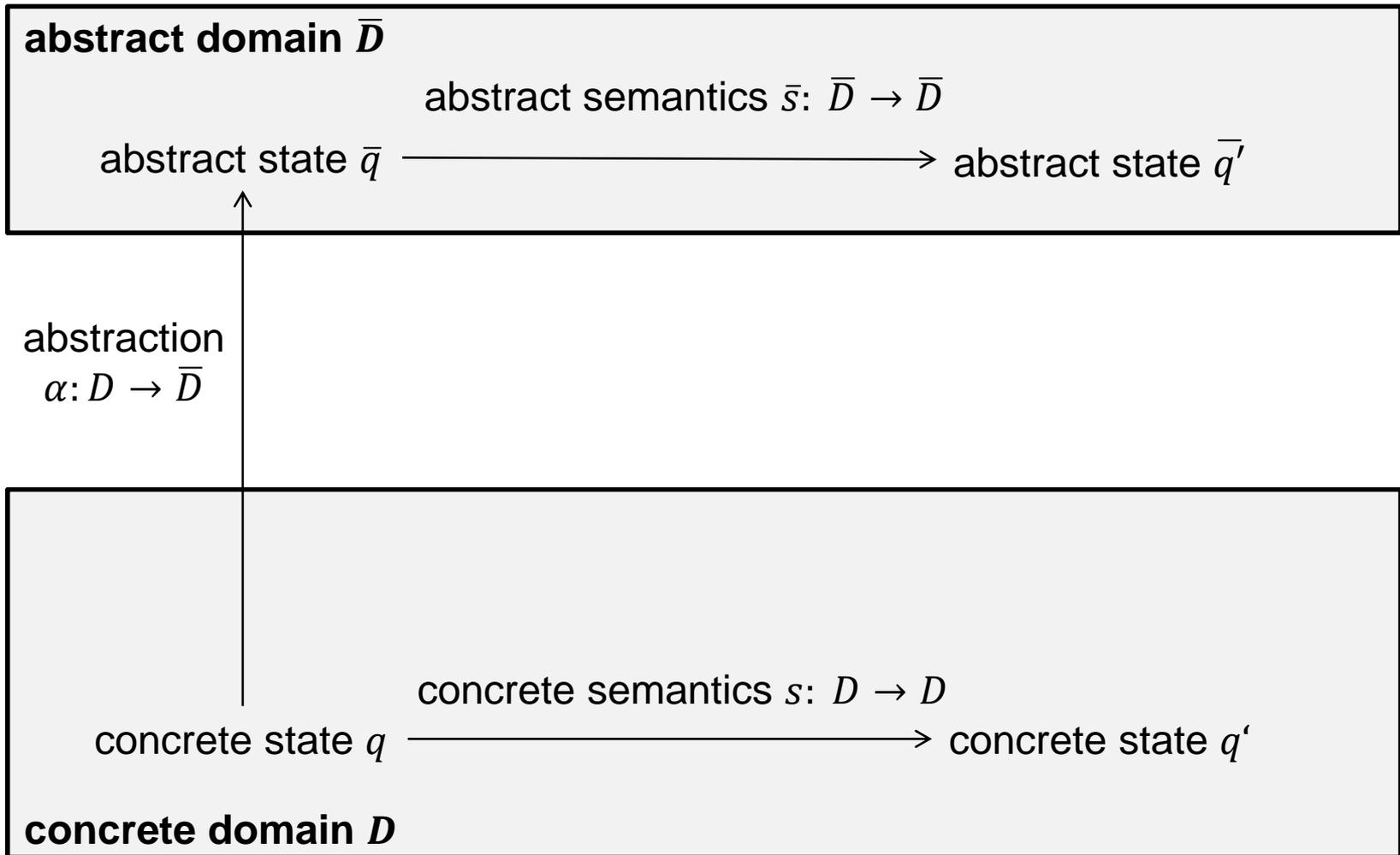
# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation



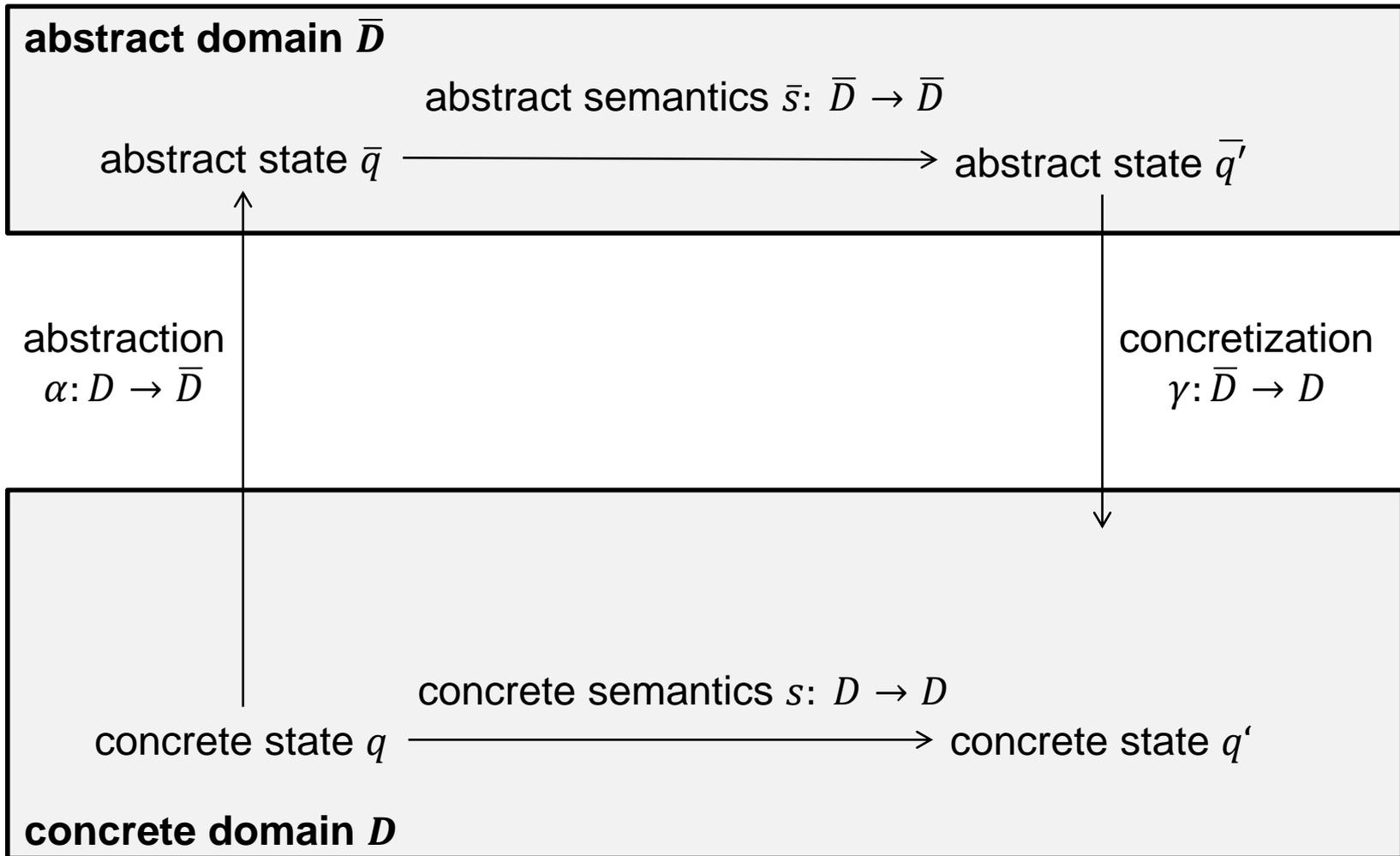
# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation



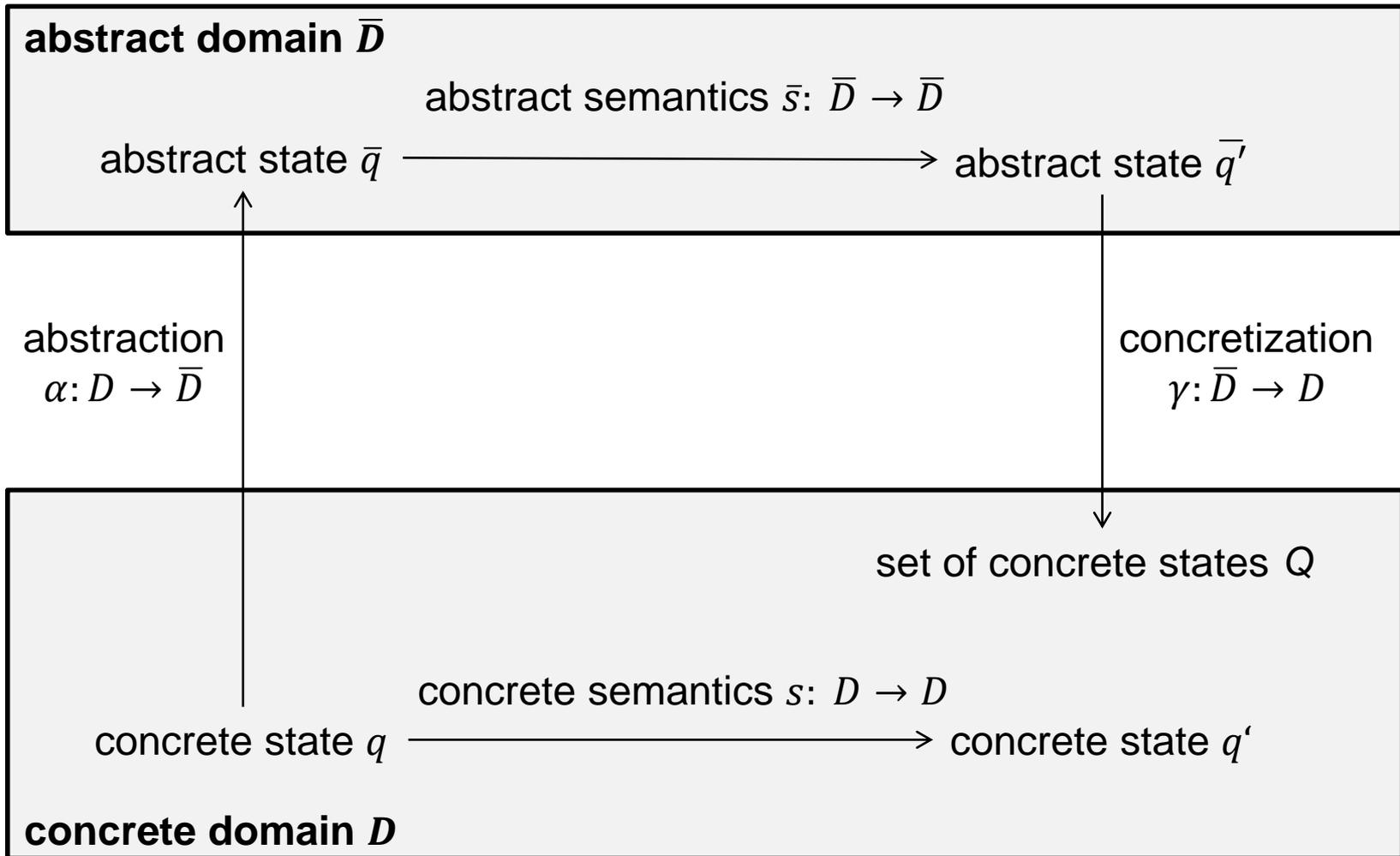
# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation



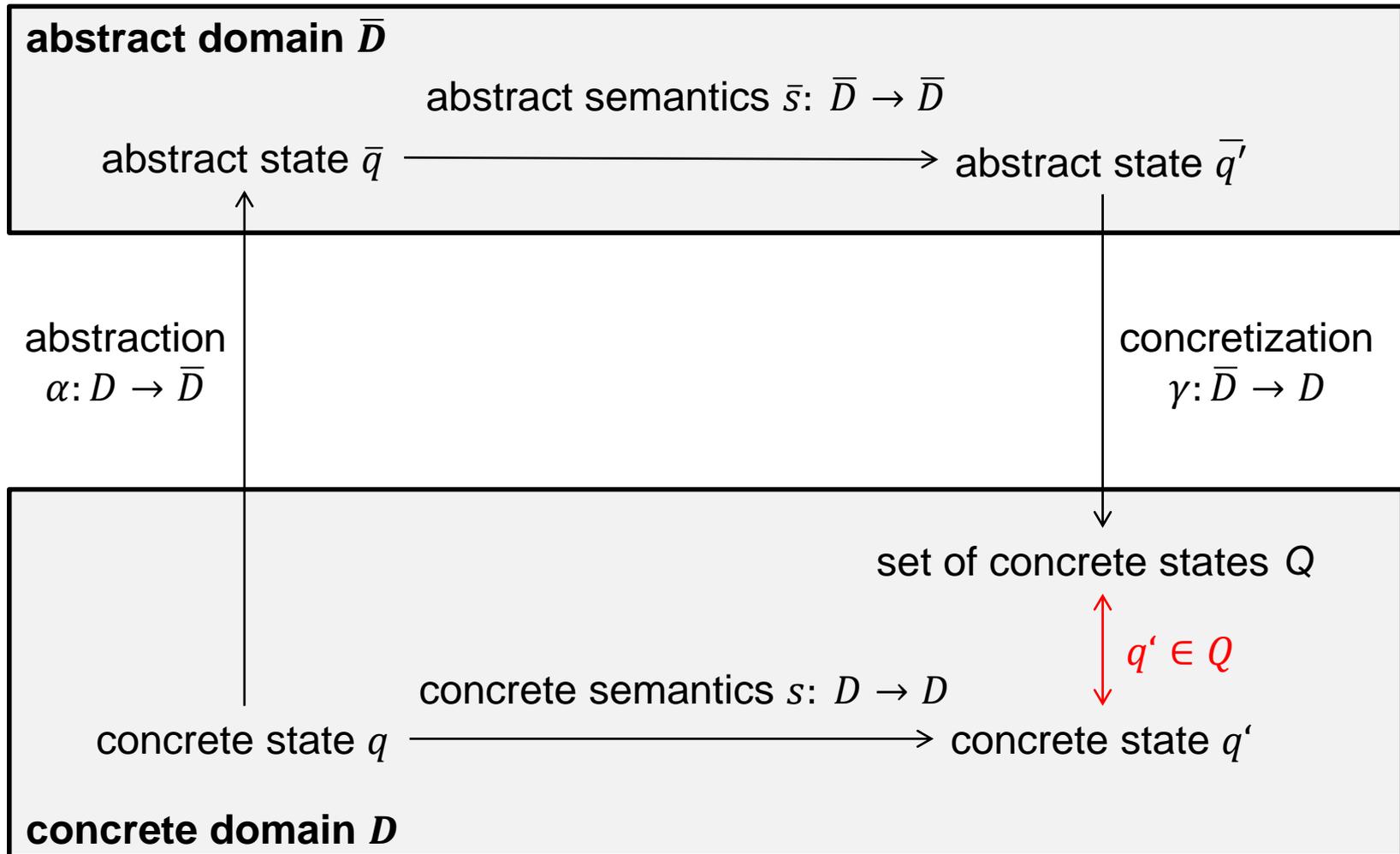
# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation



# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation



# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation - Soundness

---

### When is an abstract domain helpful?

- if it is efficient to use and results are sufficiently precise
  - choose abstract domain accordingly
- if it does not lose information about possible states
  - choose abstraction and concretization functions such that
    - $\forall Q \in 2^D. \gamma(\alpha^{set}(Q)) \supseteq Q$
    - $\forall \bar{q} \in \bar{D}. \bar{q} = \alpha^{set}(\gamma(\bar{q}))$
    - $\alpha^{set}, \gamma$  are order-preserving

### When is abstract semantics helpful?

- if it does not lose information about possible successors
- i.e., if it overapproximates the concrete semantics
  - choose abstract semantics and concretization function such that
    - $\forall \bar{q} \in \bar{D}. \gamma(\bar{s}(\bar{q})) \supseteq s^{set}(\gamma(\bar{q}))$

# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation

---

### Reachability Analysis with Abstract Interpretation

- given concrete domain and semantics of programming language
- choose an abstract domain, e.g., sets, intervals
  - with smallest and largest element, e.g.,  $[\ ]$  and  $[-\infty, \infty]$
  - with operators for comparing and merging (“join”) abstract states
- define abstraction function
- define abstract semantics
- obtain states reachable by abstract semantics
- define concretization function
- concretize reachable abstract states
  
- count concretized states and take the logarithm of the result

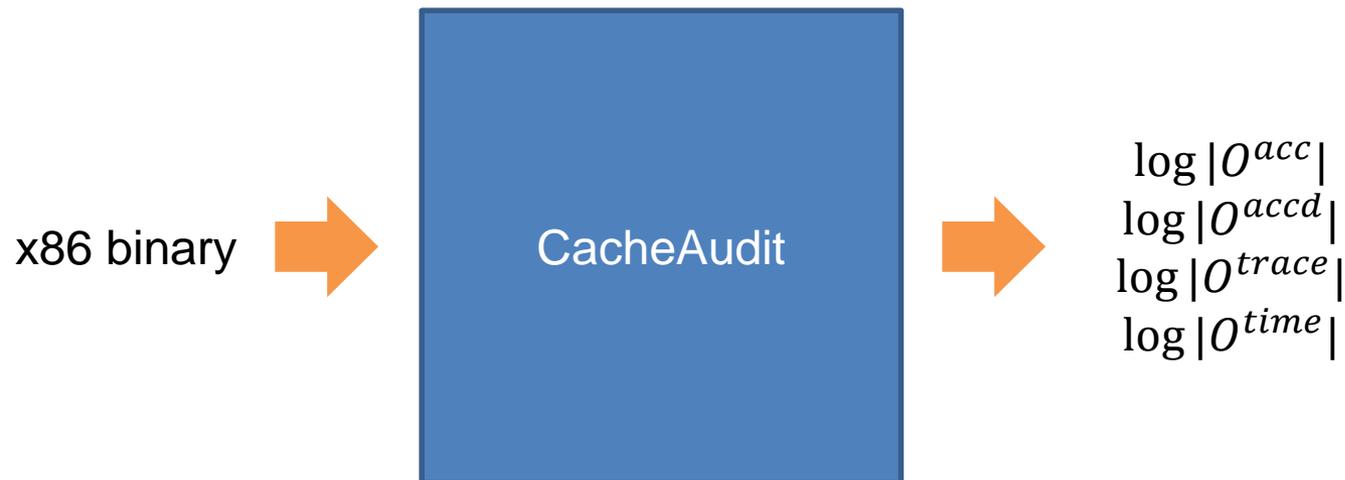
# Approach 1: Information Theory and Reachability Analysis

## Tool Support

---

### CacheAudit

- implements the approach using abstract interpretation
- computes upper bounds on cache-side-channel leakage



# Approach 1: Information Theory and Reachability Analysis

## Abstract Interpretation

---

### More Information on Abstract Interpretation

- Cousot, Patrick, and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints." Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. ACM, 1977.

### More Information on CacheAudit

- Doychev, Goran, Köpf, Boris, Mauborgne, Laurent, & Reineke, Jan. „Cacheaudit: A tool for the static analysis of cache side channels“. ACM Transactions on Information and System Security (TISSEC), 18(1), 4, 2015.

# Approach 1: Information Theory and Reachability Analysis

## Summary

---

### Information Theory

- capture leakage based on mutual information
- using Shannon entropy or min-entropy
- leakage is bounded by  $\log|O|$  for possible observations  $O$

### Reachability Analysis

- count the number of reachable observations
- overapproximate the reachable observations
- e.g., using abstract interpretation

# Application of Approach 1: Cache Side Channels in AES Implementations

Heiko Mantel, Alexandra Weber and Boris Köpf.

A Systematic Study of Cache Side Channels across AES  
Implementations.

In Proceedings of the 9th International Symposium on Engineering  
Secure Software and Systems (ESSoS), pages 213-230, 2017.

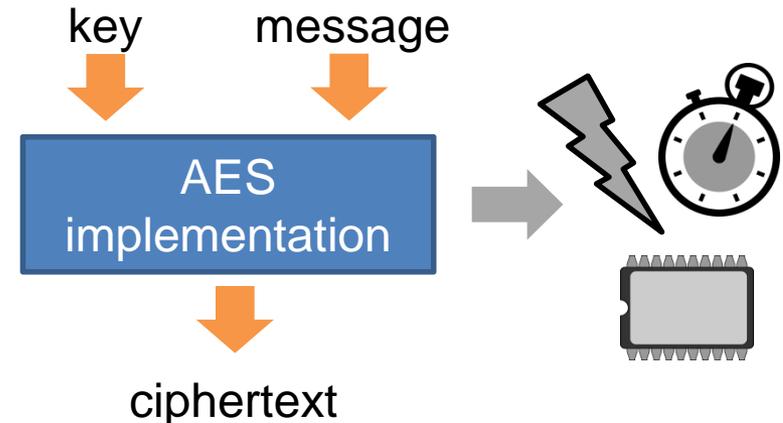
# Application 1: Cache Side Channels in AES Implementations

## Threats to AES Implementations

---

### Crypto Algorithm

- AES-128
- symmetric encryption of message with secret key
- considered secure



### Side-Channel Attacks on AES

[Bernstein2005], [Aciicmez/Koc2006], [Osvik/Shamir/Tromer2006],  
[Tromer/Osvik/Shamir2010], [Gullasch/Bangerter/Krenn2011],  
[Irazoqui/Inci/Eisenbarth/Sunar2014], [Yarom/Falkner2014],  
[Irazoqui/Eisenbarth/Sunar2015],[Lipp/Gruss/Spreitzer/Maurice/Mangard2016]

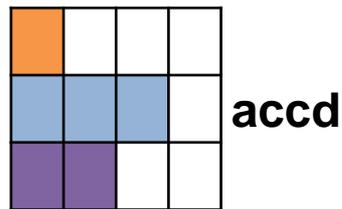
# Application 1: Cache Side Channels in AES Implementations

## Cache Side Channels

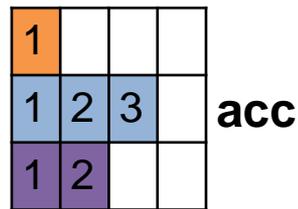
### Cache Side Channels



### Attacker Models



generalizes  
EVICT+TIME and  
PRIME+PROBE



motivated by  
FLUSH+  
RELOAD



models sample in  
time-based  
attacks



models sample in  
trace-based  
attacks

# Application 1: Cache Side Channels in AES Implementations

## AES Implementations

Lookup-table-based AES implementations

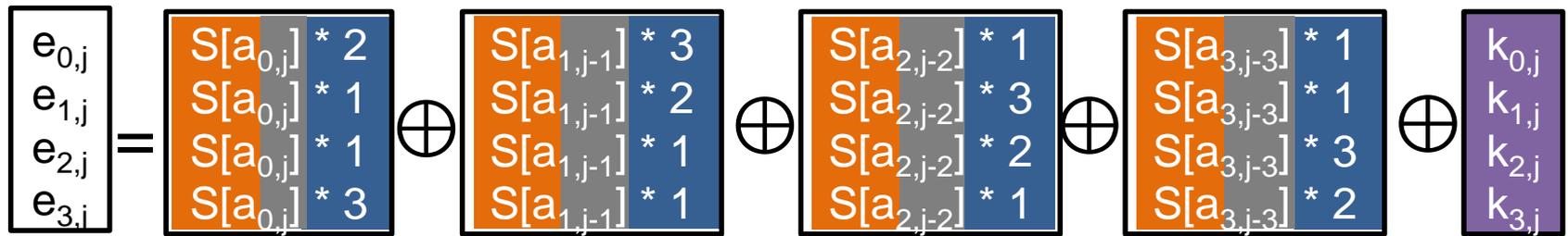
main  
rounds

SubBytes

ShiftRows

MixColumns

AddRoundKey

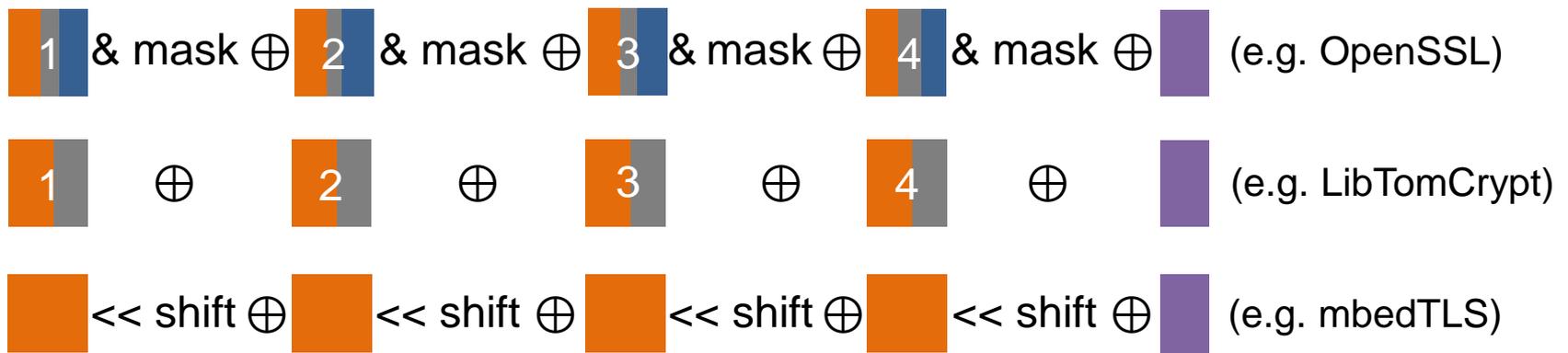


last  
round

SubBytes

ShiftRows

AddRoundKey



# Application 1: Cache Side Channels in AES Implementations

## A Systematic Study

---

### AES Implementations

- LibTomCrypt 1.17 
- OpenSSL 1.0.1t 
- mbedTLS 2.2.1, Nettle 3.2 



Which lookup-table variant leads to the lowest leakage bounds?

Is one implementation superior for all cache configurations?

What is the effect of a reduced attack surface?

### Dimensions

- attacker model
- cache configuration (size, replacement strategy)
- countermeasure

# Application 1: Cache Side Channels in AES Implementations

## Leakage Bounds in Our Study

---

### Tool: Extended CacheAudit

- extension by support for x86 instructions in AES implementations
- extension by more precise abstraction of processor status flags



### Leakage Bounds

- based on information theory
- computed with abstract interpretation

CacheAudit Extension available on GitHub and under [www.mais.informatik.tu-darmstadt.de/cacheaudit-essos17.html](http://www.mais.informatik.tu-darmstadt.de/cacheaudit-essos17.html)

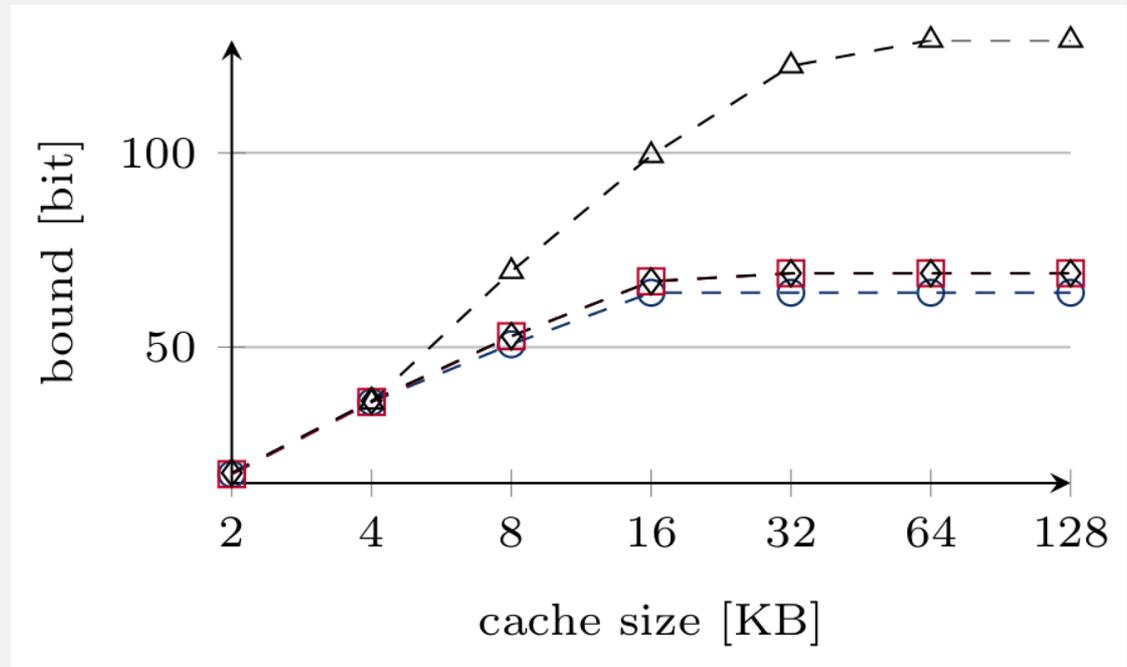
# Application 1: Cache Side Channels in AES Implementations

## Results in Our Study

Nettle  
OpenSSL mbedTLS  
LibTomCrypt

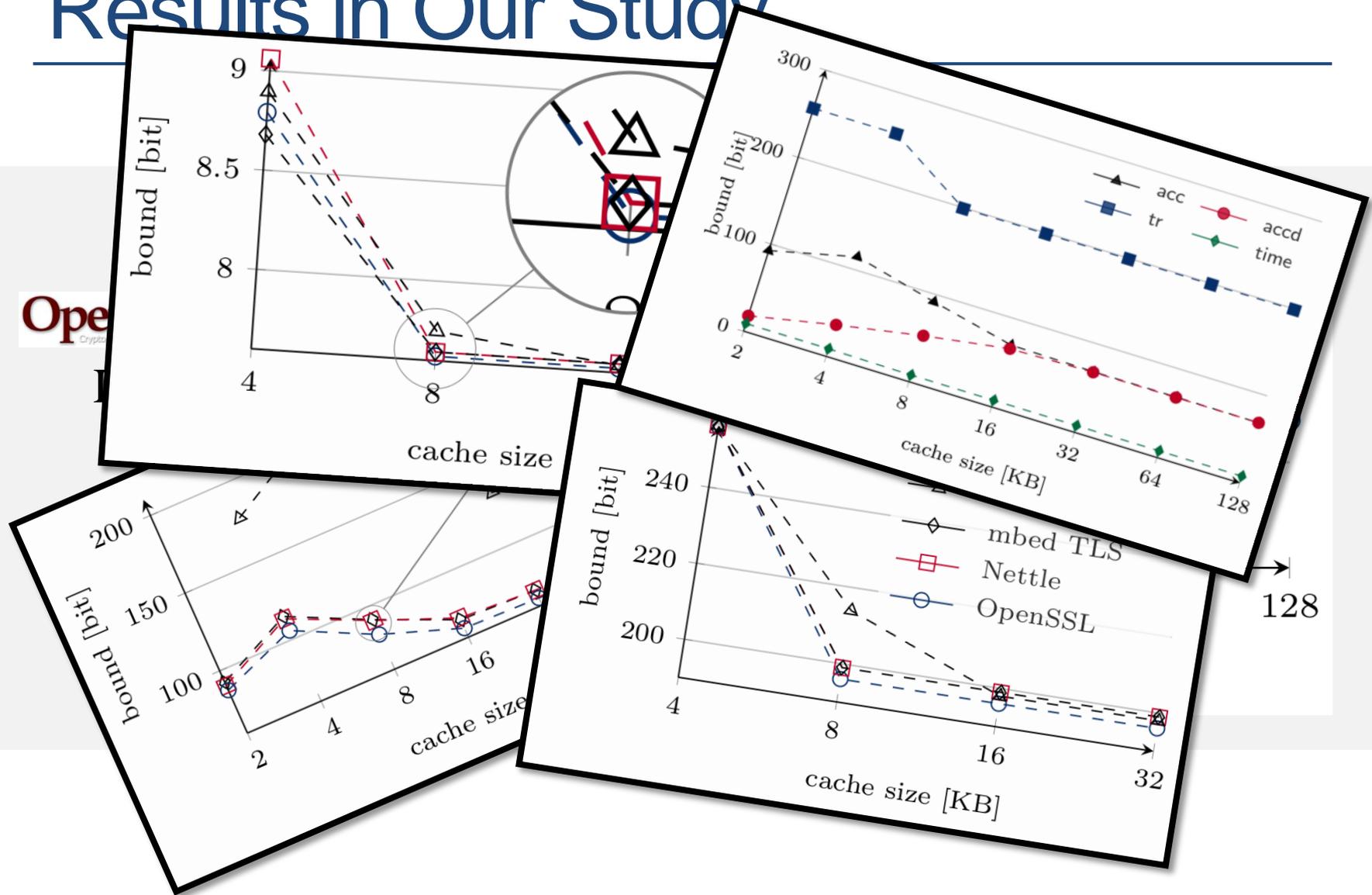


extended  
Cache  
Audit



# Application 1: Cache Side Channels in AES Implementations

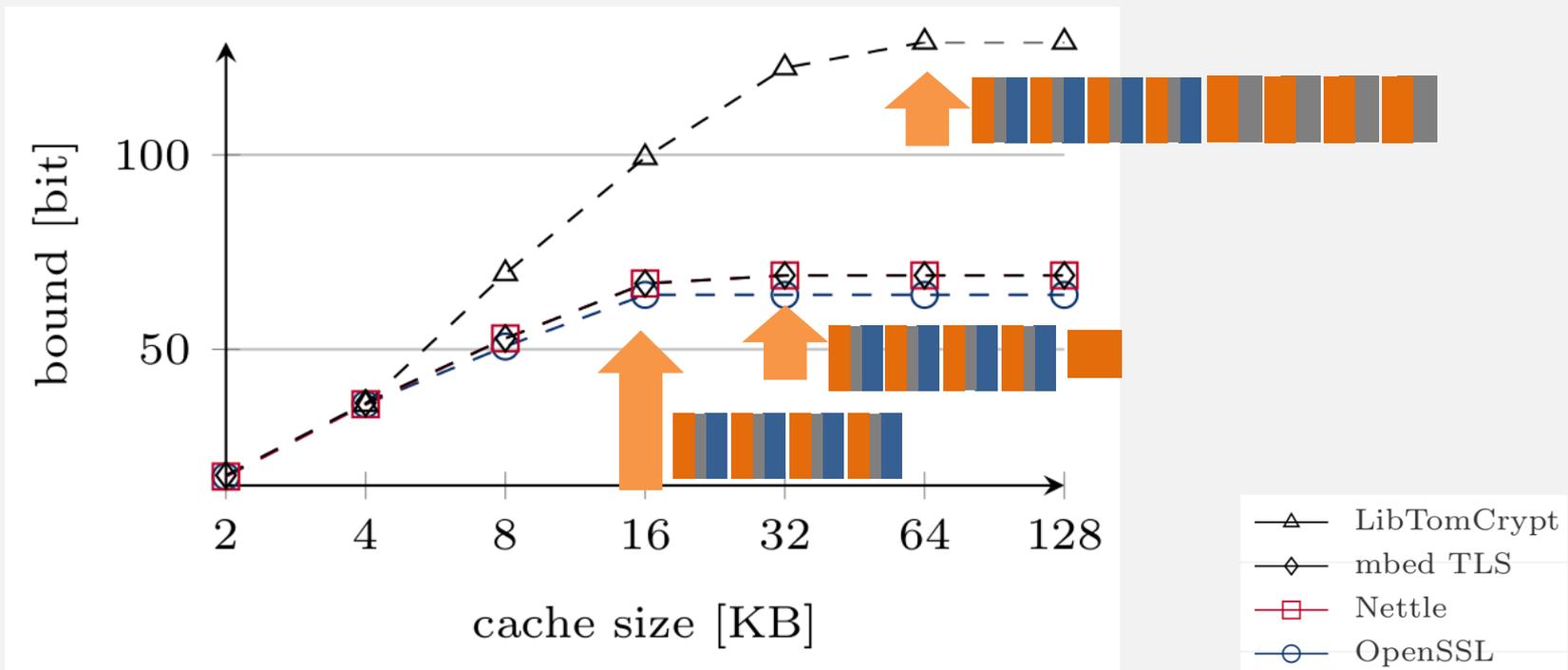
## Results in Our Study



# Application 1: Cache Side Channels in AES Implementations

## Leakage Bounds **accd**: Stabilization

**accd** can observe fill degree of all cache sets after execution

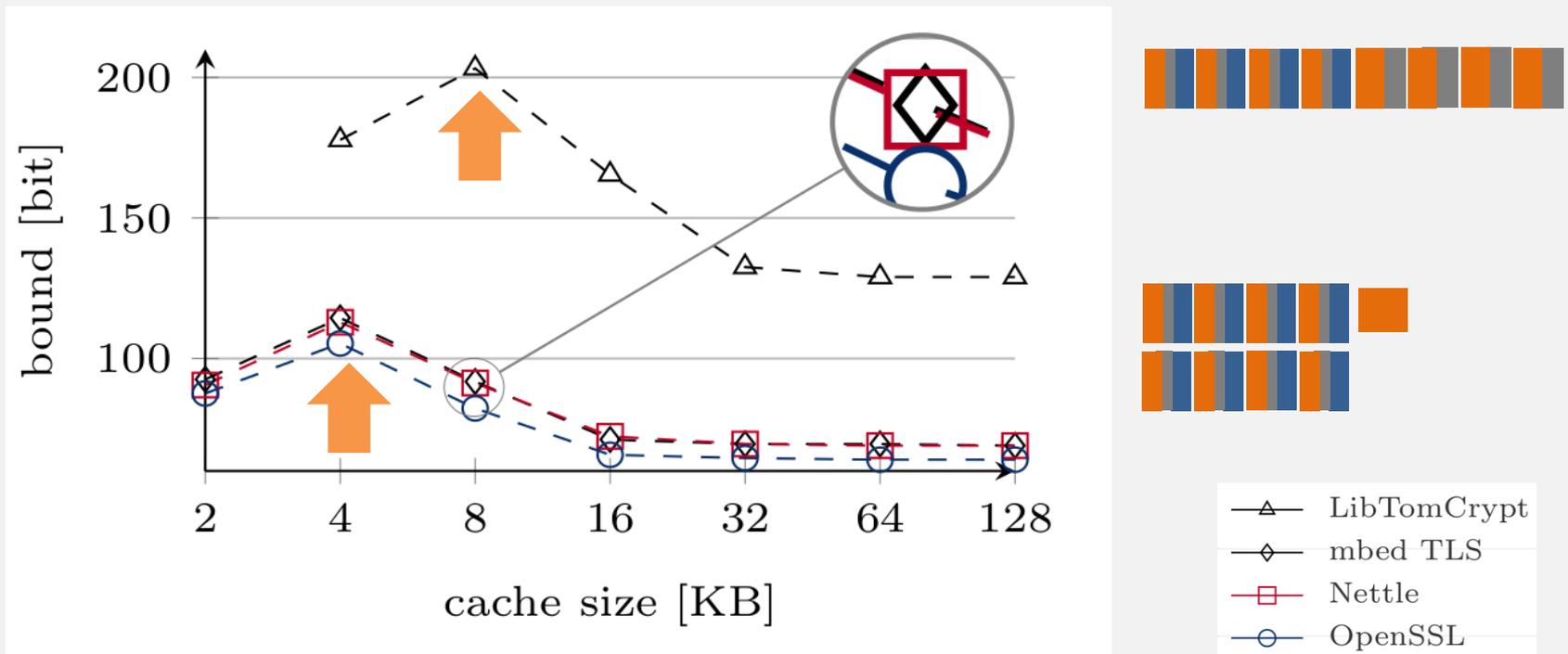


- stabilization with increasing cache size
- when mapping from memory blocks to cache sets (4-way) injective
- robustness to larger caches in future hardware

# Application 1: Cache Side Channels in AES Implementations

## Leakage Bounds acc: Peak

**acc** can observe the order of blocks in all cache sets after execution

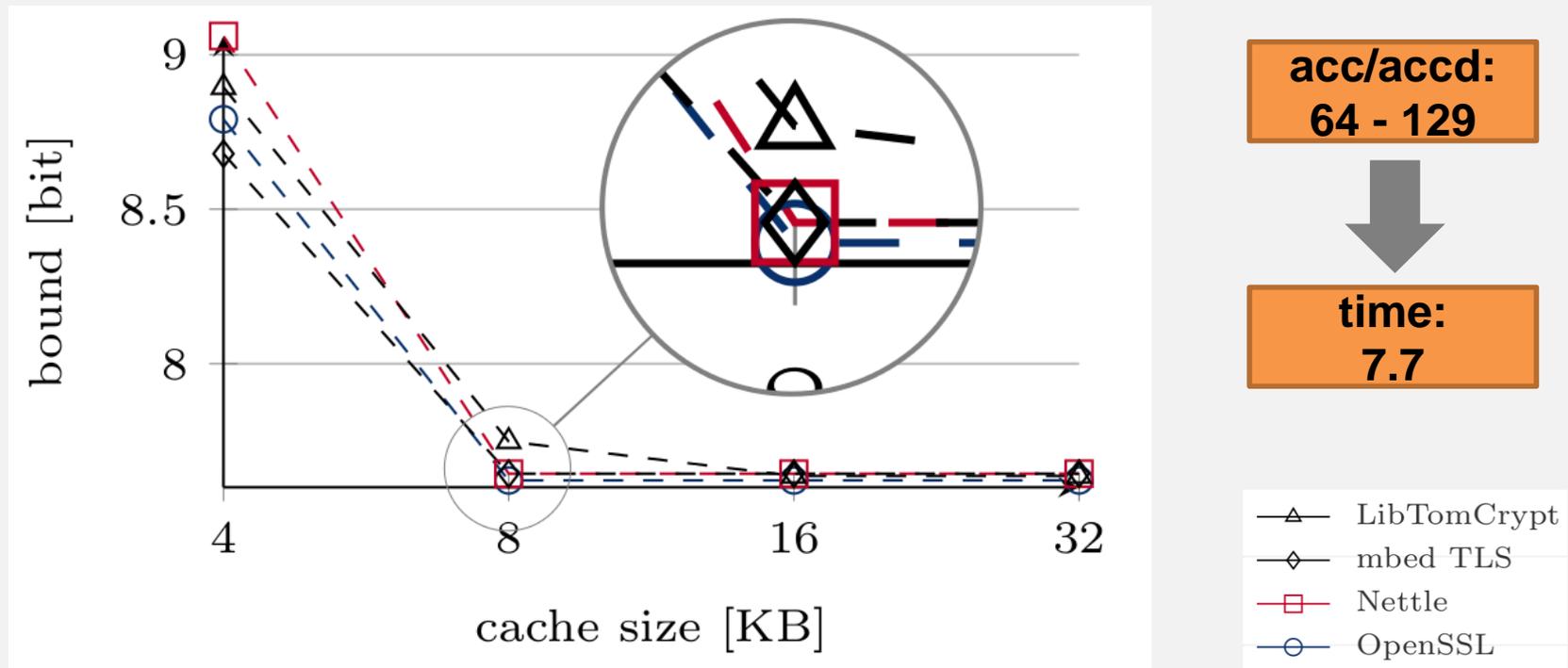


- exact fit in cache → order reveals more
- the more tables, the later the peak

# Application 1: Cache Side Channels in AES Implementations

## Leakage Bounds time: Height

**time** can observe the time taken by cache hits and misses in execution with fixed time per hit/miss



- bounds are roughly 3% of bits in secret key and message
- reduction of attack surface from acc/accd to time greatly improves the leakage bounds

# Application 1: Cache Side Channels in AES Implementations

## Summary

- results across all considered dimensions in the paper
- results shown in presentation are shaded in blue

### Extension of CacheAudit

Type	New Instructions
Arithmetic	2D (Sub), 18 (Sbb), 19 (Sbb), 11 (Adc), F7/6 (Div), 3C (Cmp)
Logic	08 (Or), 30 (Xor), 84 (Test), A9 (Test), F6/0 (Test)
Bitstring	0FA4 (Shld), 0FA5 (Shld), 0FAC (Shrd), 0FAD (Shrd)
Stack	07 (Pop)
Jump	7C, 0F8C, 7D, 0F8D, 70, 0F80, 71, 0F81, 78, 0F88, 79, 0F89, 7E, 0F8E, 7F, 0F8F (all Jcc)
Move	0F48 (Cmovs)

Table 2: Extended language coverage in CacheAudit

### Leakage across implementations

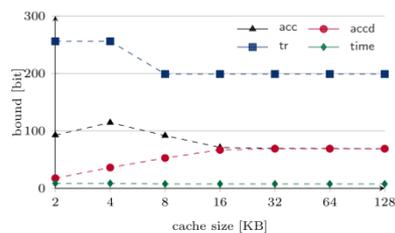
	LibTomCrypt	mbed TLS	Nettle	OpenSSL
accd	129bit	69bit	69bit	64bit
acc	129bit	69bit	69bit	64bit

Table 4: acc/accd leakage bounds for 128KB cache

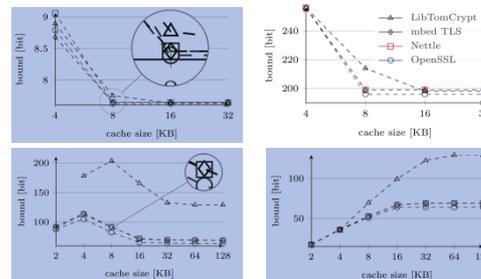
	LibTomCrypt	mbed TLS	Nettle	OpenSSL
time	7.7bit	7.7bit	7.7bit	7.7bit
tr	198bit	199bit	199bit	196bit

Table 5: time/trace leakage bounds for 128KB cache

### Effect of attack surface



### Effect of cache configuration

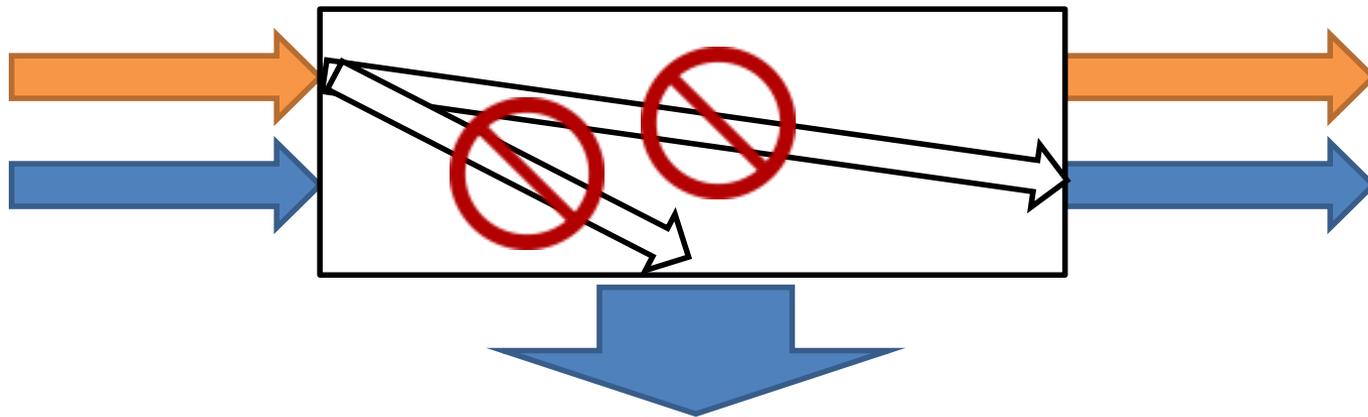


### Effect of hardening

Cache Size [KB]	4	8	16	32	64	128
LibTomCrypt			✓	✓	✓	✓
Nettle		✓	✓	✓	✓	✓
OpenSSL		✓	✓	✓	✓	✓
mbed TLS		✓	✓	✓	✓	✓

Table 6: Preloading effectiveness for acc/accd/tr/time

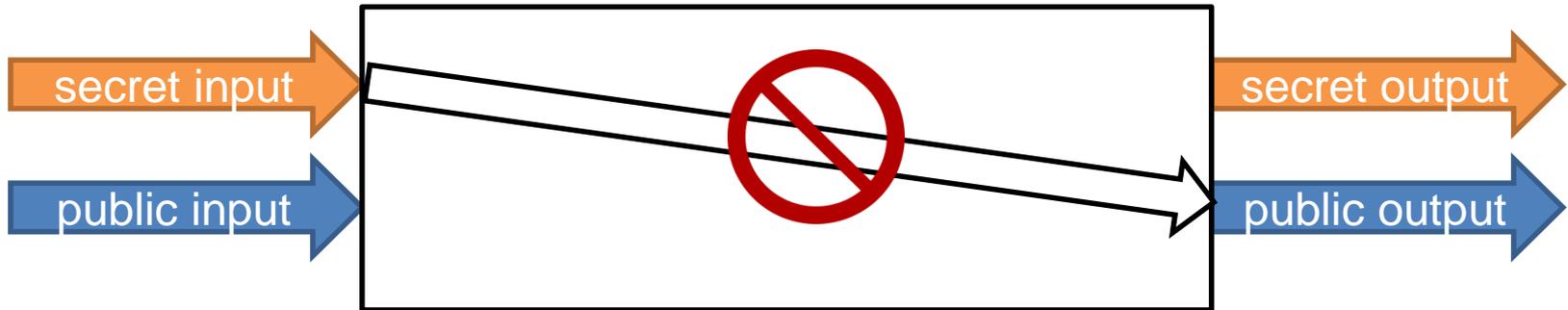
## Approach 2: Security Type System



## Approach 2: Security Type Systems

# Information Flow Security

---



### Direct Information Flow

```
p_var := s_var
```

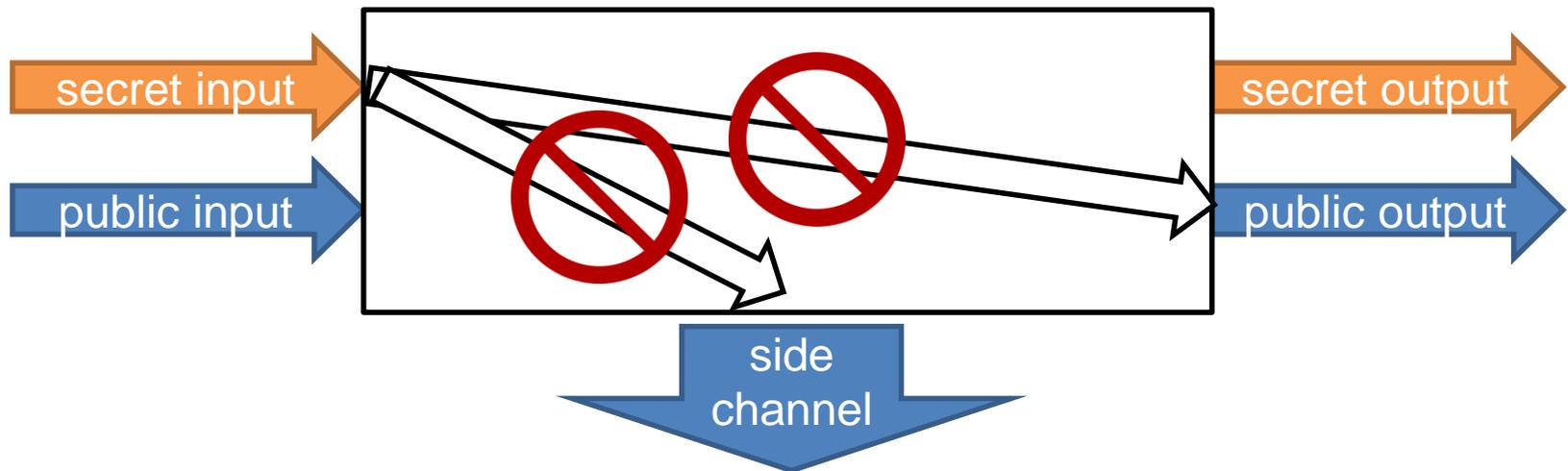
### Indirect Information Flow

```
if s_var == 1 then  
  p_var := 1  
else  
  p_var := 0
```

## Approach 2: Security Type Systems

# Information Flow Security

---



### Information Flow to Timing Side Channel

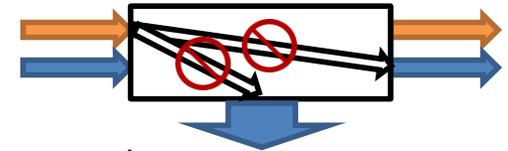
```
if s_var == 1 then
  sleep(1000)
else
  skip
```

## Approach 2: Security Type Systems

# Attacker Model

### Attacker

- passive, but knows program code
- can observe public information (e.g., side channel output)
- cannot observe secret information



### Indistinguishable Execution States

- domain assignment (da): label information containers
- “low” if they only contain public information
- “high” if they can contain secret information
- states indistinguishable under da if they differ only in “low” containers

$$\begin{array}{l}
 x = 1 \approx x = 1 \\
 y = 0 \quad \begin{array}{|c|} \hline \text{low} \\ \hline \end{array} \quad y = 0 \\
 z = 1 \quad \begin{array}{|c|} \hline \text{high} \\ \hline \end{array} \quad z = 0 \\
 \quad \begin{array}{|c|} \hline \text{low} \\ \hline \end{array}
 \end{array}
 \quad \times$$

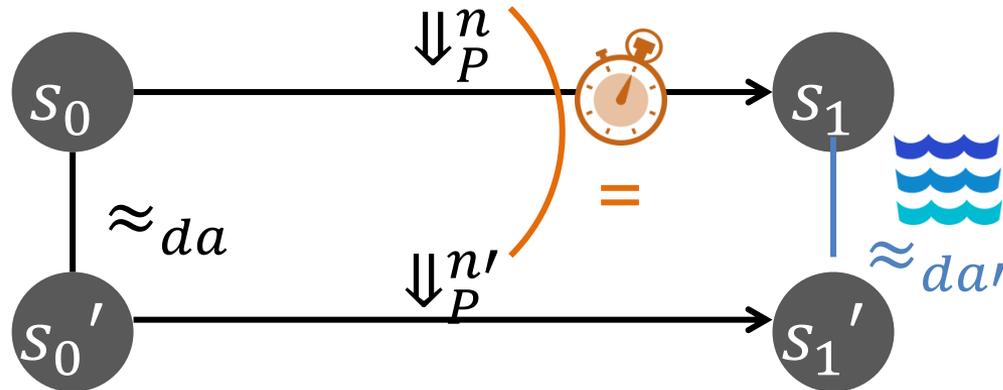
$$\begin{array}{l}
 x = 1 \approx x = 1 \\
 y = 0 \quad \begin{array}{|c|} \hline \text{low} \\ \hline \end{array} \quad y = 1 \\
 z = 1 \quad \begin{array}{|c|} \hline \text{high} \\ \hline \end{array} \quad z = 1 \\
 \quad \begin{array}{|c|} \hline \text{low} \\ \hline \end{array}
 \end{array}
 \quad \checkmark$$

$$\begin{array}{l}
 x = 1 \approx x = 1 \\
 y = 0 \quad \begin{array}{|c|} \hline \text{high} \\ \hline \end{array} \quad y = 1 \\
 z = 1 \quad \begin{array}{|c|} \hline \text{low} \\ \hline \end{array} \quad z = 1 \\
 \quad \begin{array}{|c|} \hline \text{low} \\ \hline \end{array}
 \end{array}
 \quad \times$$

## Approach 2: Security Type Systems

# Timing-Sensitive Noninterference

A program  $P$  satisfies TSNI with initial and finishing domain assignments  $da$  and  $da'$  if and only if



### Noninterference

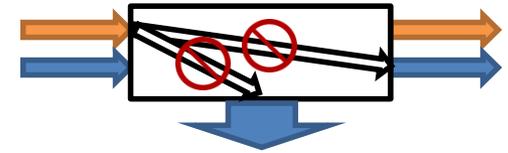
- TSNI is a variant of a common information-flow property
- noninterference originally defined in Joseph A. Goguen, José Meseguer. "Security Policies and Security Models." Proceedings of the 3rd IEEE Symposium on Security and Privacy pp. 11-20, 1982

## Approach 2: Security Type Systems

# Security Type System

### Type System

- assign types to information containers:  
e.g., integer, boolean
- set of rules to check that types are consistent



### Security Type System

- assign security types (domain assignment):  
e.g., high, low
- set of rules to check absence of undesired information flow
- security environment to check for indirect flows
  - lower bound on type of surrounding branching conditions

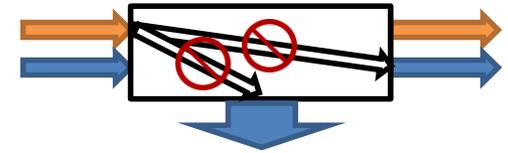
$$\begin{array}{c}
 P(ep) = adc \ x \ y \\
 erg = da(x) \sqcup da(y) \sqcup se(ep) \\
 da' = da[x \mapsto erg] \\
 \hline
 P, se, ep: da \vdash da'
 \end{array}$$

## Approach 2: Security Type Systems

# Security Type System: Soundness

### Soundness wrt. Security Property

- all programs that are typable with the type system satisfy TSN1
- security type system has no false positives
- can be proven, e.g., by unwinding



■ Step Consistency: every step outside secret branching preserves indistinguishability

▴ Locally Respects: every step inside secret branching preserves indistinguishability

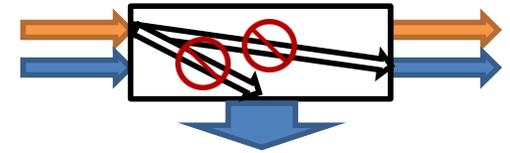
## Approach 2: Security Type Systems

# Security Type System: Soundness

---

### Basis for Soundness

- a faithful semantics of executions
- underlying the security property
- should capture all relevant aspects of program execution
- difficult for TSNI: effect of caches, etc. on timing hard to predict statically
  
- example: program transformations (cross-copying, unification, ...)
  - provably establish timing-sensitive security property
  - but effectiveness in practice varies [Mantel/Starostin2015]



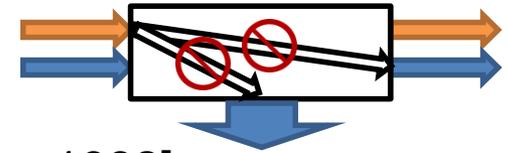
## Approach 2: Security Type Systems

# Summary

---

### Info-Flow Analysis with Type Systems

- absence of undesired information flow
- usually variant of noninterference [Goguen/Meseguer1982]
  - secrets do not influence attacker observations
- time-based flow: hard to capture optimizations faithfully
  - affects security in practice, e.g., [Mantel/Starostin2015]
- checked with security type system
  - sound = no false negatives



### More Information on Information-Flow Security

Andrei Sabelfeld, and Andrew C. Myers.

Language-based information-flow security.

IEEE Journal on selected areas in communications 21(1)

pages 5-19, 2003.

# Application of Approach 2: Timing Side Channels in IoT Implementations

Florian Dewald, Heiko Mantel, and Alexandra Weber.  
AVR Processors as a Platform for Language-Based Security.  
In Proceedings of the 22<sup>nd</sup> European Symposium on Research in  
Computer Security (ESORICS), pages 213-230, 2017.

# Application 2: Timing Side Channels in IoT Implementations

## Application Domain

---

### Attacks on devices in the Internet of Things (IoT)

- IoT: smart, interconnected devices in our homes
- compromised devices can affect or daily life
- recently, smart light bulbs were compromised
  - firmware signed with stolen AES key
  - [Ronen/O'Flynn/Shamir/Weingarten2017]

### AVR processors

- the vulnerable light bulbs contain an AVR processor
- AVR processors are popular in IoT devices
- variants exist for aerospace and automotive

## Application 2: Timing Side Channels in IoT Implementations

# Program Analysis Against Side Channels

---

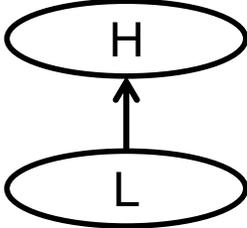
In this paper

- a security type system for AVR assembly
  - flow-sensitive and timing-sensitive
- soundness proof for the type system
- based on a formal operational semantics for AVR assembly
  - semantics faithfully captures execution times
- Side-Channel Finder<sup>AVR</sup>
  - tool to check programs against timing side channels and other leakage
- case study on crypto implementations from library  $\mu\text{NaCl}$

## Application 2: Timing Side Channels in IoT Implementations

# Security Type System I

---

- security types are **H** (secret) and **L** (public)
  - information may flow from L to H, but not vice versa
  - least upper bound  $s \sqcup t$  is high as soon as  $s$  or  $t$  is H
- 
- program **P** maps program points to instructions
  - security types are assigned by domain assignments
    - **sda**: stack, **mda**: memory, **rda**: registers, **srda**: status register (flags)
  - security environment **se** tracks security type of information on which control flow depends

$$\begin{array}{c}
 P(i) = \text{mov } x \ y \\
 rda' = rda[x \mapsto rda(y) \sqcup se(i)] \\
 \hline
 P, se, i: (sda, mda, rda, srda) \vdash (sda, mda, rda', srda)
 \end{array}$$

## Application 2: Timing Side Channels in IoT Implementations

# Security Type System II

---

$\text{breq epa}$  branches on the zero-flag

$\text{loop}_P(i)$  models whether program point  $i$  is head of a loop

$\text{region}_P^r(i)$  contains the program points in the  $r$ -branch of  $i$

$\text{lift}(sda, H)$  lifts the security types for the stack to  $H$

$\text{branchtime}_P^{\text{then}}, \text{branchtime}_P^{\text{else}}$  captures execution time for branches

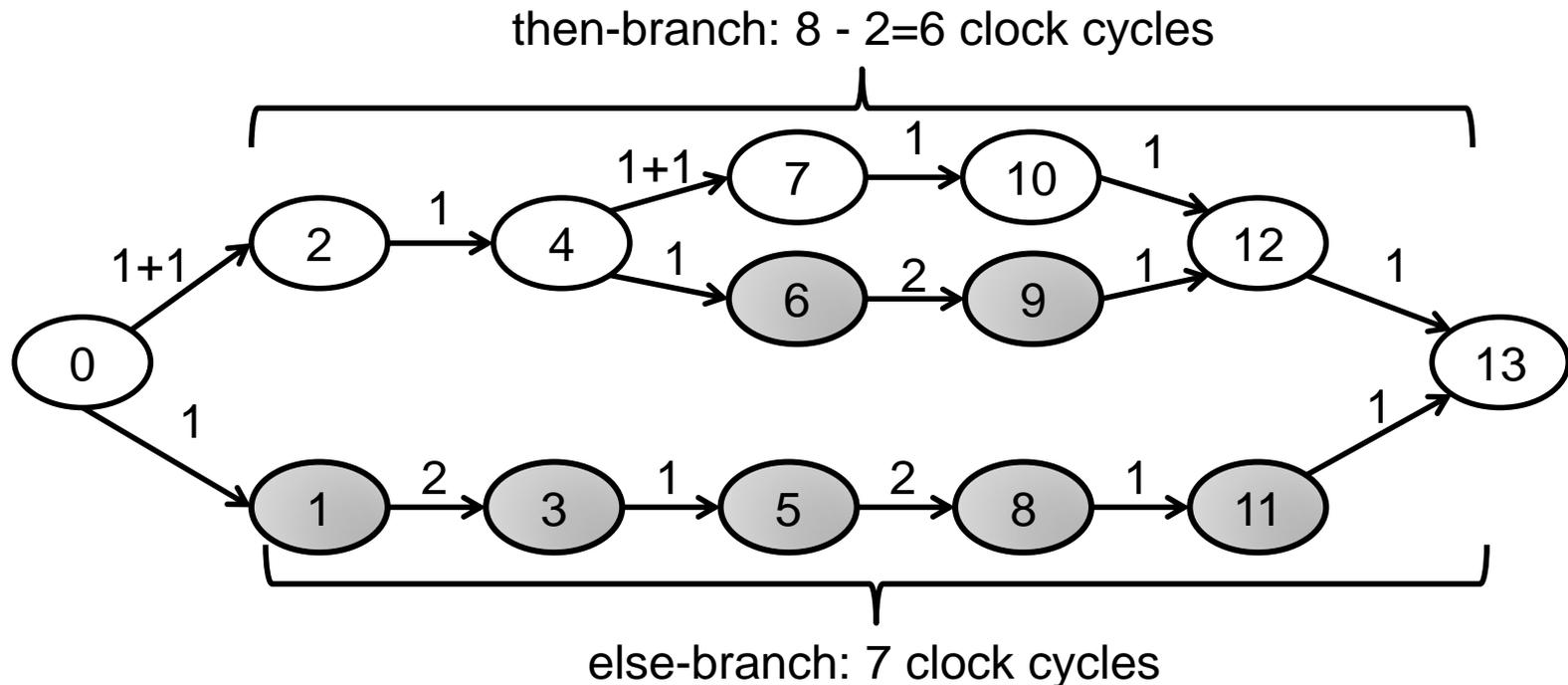
$$\begin{array}{c}
 P(i) = \text{breq epa} \\
 \neg \text{loop}_P(i) \quad se(i) \sqcup srda(Z) = H \quad se(i) = H \\
 \forall j \in \text{region}_P^{\text{then}}(i) \cup \text{region}_P^{\text{else}}(i): se(j) = H \quad sda' = \text{lift}(sda, H) \\
 \text{branchtime}_P^{\text{then}}(i) + 1 = \text{branchtime}_P^{\text{else}}(i) \\
 \hline
 P, se, i: (sda, mda, rda, srda) \vdash (sda', mda, rda, srda)
 \end{array}$$

## Application 2: Timing Side Channels in IoT Implementations

# Security Type System III

### Branchtime

- $t(P(j))$  = time taken by instruction at  $j$
- $branchtime_P^r(i) := \sum_{\substack{j \in region_P^r(i) \\ j \neq i}} (t(P(j)) - branchtime_P^{then}(j))$



## Application 2: Timing Side Channels in IoT Implementations

# Security Type System IV

---

### Typable Program

- Program  $P$  is typable wrt. initial domain assignment  $da$  and finishing domain assignment  $da'$  if
  - there are intermediate domain assignments for all program points
  - for each program point and its domain assignment, a typing rule is applicable
  - the domain assignments at the successor of each program point are at least as restrictive as required by the applicable typing rule

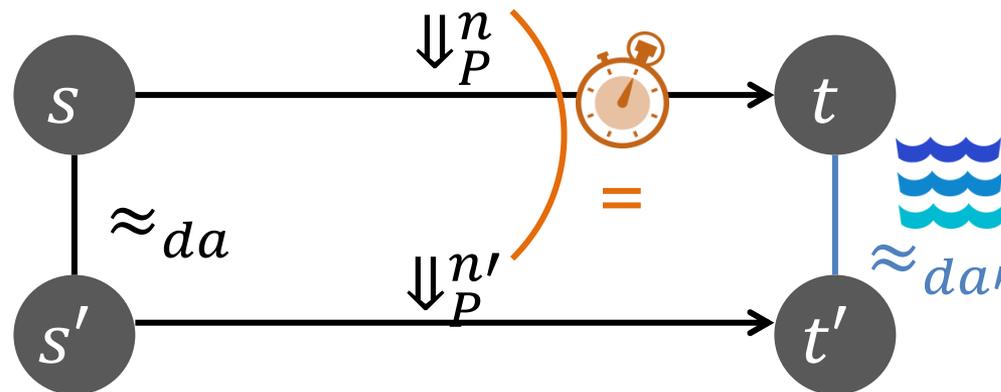
$$P, se, i_0: da \Vdash da'$$

## Application 2: Timing Side Channels in IoT Implementations

# Soundness I

- $\Downarrow_P^n$  is judgment for terminating program execution

A program  $P$  satisfies **TSNI** starting from program point  $i_0$  with initial and finishing domain assignments  $da$  and  $da'$  if and only if



$$\forall s, t, s', t' : \forall n, n' :$$

$$i_s = i_{s'} = i_0 \wedge s \approx_{da} s' \wedge s \Downarrow_P^n t \wedge s' \Downarrow_P^{n'} t' \\ \Rightarrow t \approx_{da'} t' \wedge n = n'$$

# Application 2: Timing Side Channels in IoT Implementations

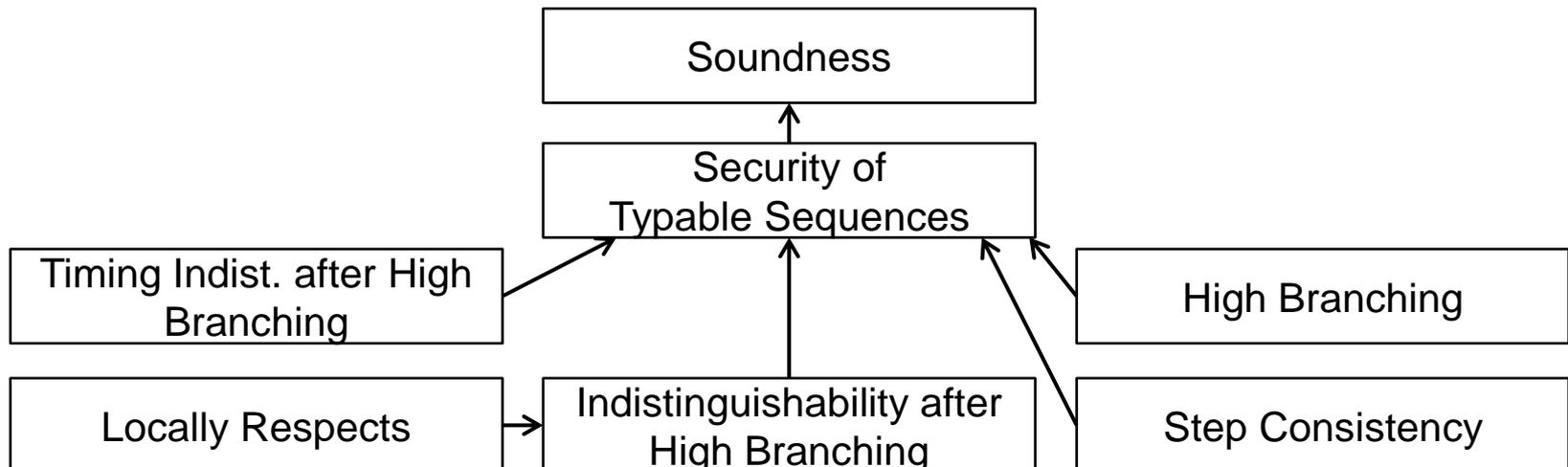
## Soundness II

### Soundness Theorem

If  $P, se, i_0: da \Vdash da'$ , then  $P$  satisfies TSNi starting from  $i_0$  with the initial and finishing domain assignments  $da$  and  $da'$ .

### Proof Sketch

- proven using an unwinding technique (local respect, step consistency)
- additional lemma: secret-dependent branches are constant-time



# Application 2: Timing Side Channels in IoT Implementations

## Formal Semantics I

---

### Semantics of AVR Assembly

- soundness required formal semantics of AVR assembly
- semantics is specified informally in AVR reference manual [Atmel AVR 8-bit Instruction Set: Instruction Set Manual 0856K-AVR-05/2016]
- we formalized it as an operational semantics
- language-coverage of our semantics suffices to analyze crypto implementations from an open-source library

# Application 2: Timing Side Channels in IoT Implementations

## Formal Semantics II

---

### Faithfulness

- time consumption is usually hard to model faithfully in semantics
  - effect of optimizations like caches is hard to predict statically
  - affects security in practice, e.g., [Mantel/Starostin2015]
- our semantics capture execution time faithfully
- we exploit characteristics of 8-bit AVR processors
  - execution times are statically predictable

## Application 2: Timing Side Channels in IoT Implementations

# Formal Semantics III

---

### Execution State

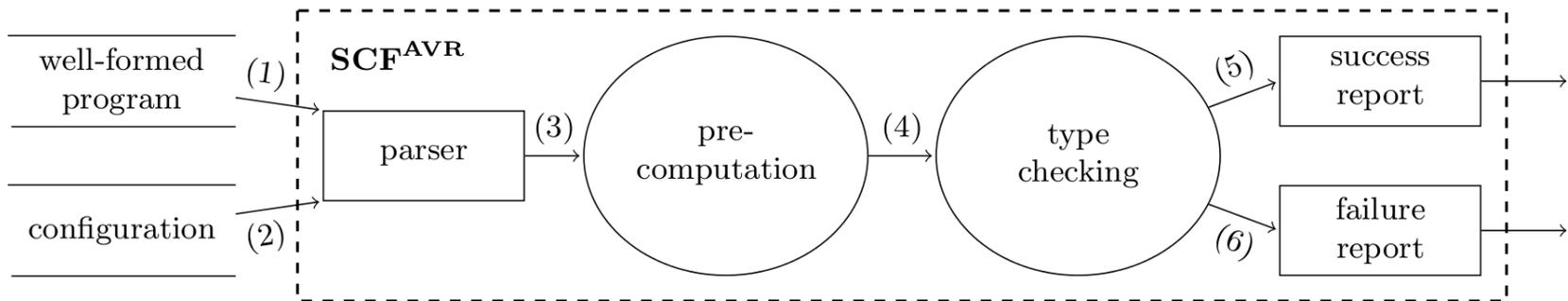
- $sr$  maps status flags to values (1 or 0)
- $m$  maps memory addresses to values
- $r$  maps registers to values
- $st$  maps stack to values
- program point = program counter + call stack

### Example

$$\frac{P(i) = \text{breq } epa \quad sr(Z) = 1 \quad i = (ep, fs) \quad j = (epa, fs)}{(sr, m, r, st, i) \xrightarrow{t(P(i))+1}_P (sr, m, r, st, j)}$$

# Application 2: Timing Side Channels in IoT Implementations

## Side-Channel Finder<sup>AVR</sup>



- takes a well-formed AVR program
  - syntactically correct, single return instruction, in supported sublanguage
- takes a configuration
  - specification which information is considered secret
- checks for absence of timing side channels and other leakage
- implemented in roughly 1250 lines of Python code

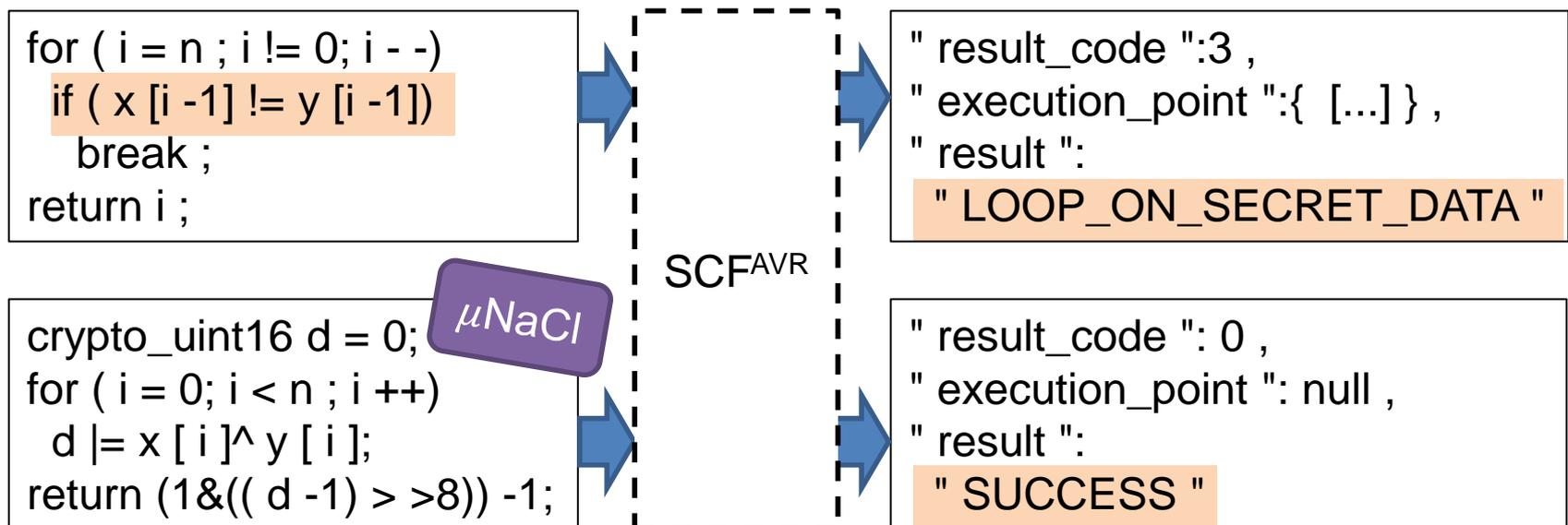
# Application 2: Timing Side Channels in IoT Implementations

## Case Study on $\mu$ NaCl

### Case Study

- $\mu$ NaCl is a crypto library for AVR microcontrollers
- we certified TSNI for cryptographic implementations from  $\mu$ NaCl
  - Salsa20, XSalsa20, Poly1305, String Comparison

### Example



# Application 2: Timing Side Channels in IoT Implementations

## Summary

**Dec**  

$$\frac{P(ep) = (\text{dec}, (Rd)) \quad r' = r[Rd \mapsto r(Rd) - 1] \quad sr' = sr[Z \mapsto_s r'(Rd) = 0]}{(sr, m, r, st, ep) \xrightarrow{P(ep)}_P (sr', m, r', st, ep +_{app} 1)} \text{ (dec)}$$

**Description** The dec instruction takes a registers and decreases by one. The result is stored inside the Rd register.  
**Status Flags** None, except Z.  
**Faithfulness** The decreasing operation is captured by-  
**Effect**  $[Rd \mapsto r(Rd) - 1]$ . It directly captures

**Eor**  

$$P(ep) = (\text{eor}, P)$$

registers and performs an XOR operation. The result is stored in Rd.  
**Description** The eor instruction takes registers and performs an XOR operation. The result is stored in Rd.  
**Status Flags** None, except Z.  
**Faithfulness** The XOR operation is captured by updating the register  $Rd \mapsto r(Rd) \oplus r(Rr)$ . Here,  $\oplus$  is the exclusive-or operation.  
**Effect**  $[Rd \mapsto r(Rd) \oplus r(Rr)]$ . It directly captures the XOR operation.

**inc**  

$$\frac{P(ep) = (\text{inc}, (Rd)) \quad r' = r[Rd \mapsto r(Rd) + 1] \quad sr' = sr[Z \mapsto_s r'(Rd) = 0]}{(sr, m, r, st, ep) \xrightarrow{P(ep)}_P (sr', m, r', st, ep +_{app} 1)} \text{ (inc)}$$

**Description** The inc instruction takes a registers and increases its content by one. The result is stored inside the Rd register.  
**Status Flags** None, except Z.  
**Faithfulness** The increasing operation is captured by performing the update  $[Rd \mapsto r(Rd) + 1]$ . It directly captures the increment operation.  
**Effect**  $[Rd \mapsto r(Rd) + 1]$ . It directly captures the increment operation.

**Faithful timing-sensitive semantics for AVR assembly**

**Type system detecting direct, indirect and time-based flows**

**Figure 3: Typing rules for branching instructions.**

$$\frac{\exists instr \in \{\text{breq}, \text{brne}\}: P(ep) = (\text{instr}, (eps)) \quad sr(ep) \sqcup srda(Z) = C}{P(\dots, ep : (sda, nd, rda, arda) \vdash (sda', nd, rda, arda)) \vdash (t-brZ-h)}$$

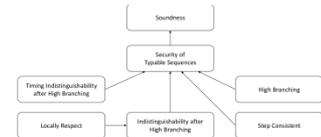
$$\frac{\exists instr \in \{\text{breq}, \text{brne}\}: P(ep) = (\text{instr}, (eps)) \quad sr(ep) \sqcup srda(Z) = H \quad sr(ep) = H \quad \text{loop}_p(ep) \quad sr(ep) \sqcup srda(Z) = H \quad sr(ep) = H \quad \text{branchtime}_p^{(m)}(ep) + br = \text{branchtime}_p^{(m)}(ep)}{P(\dots, ep : (sda, nd, rda, arda) \vdash (sda', nd, rda, arda)) \vdash (t-brZ-h)}$$

$$\frac{\exists instr \in \{\text{brc}, \text{brca}\}: P(ep) = (\text{instr}, (eps)) \quad sr(ep) \sqcup srda(C) = C}{P(\dots, ep : (sda, nd, rda, arda) \vdash (sda', nd, rda, arda)) \vdash (t-cpe-h)}$$

$$\frac{\exists instr \in \{\text{brc}, \text{brca}\}: P(ep) = (\text{instr}, (eps)) \quad sr(ep) \sqcup srda(C) = C \quad \text{loop}_p(ep) \quad sr(ep) \sqcup srda(C) = C}{P(\dots, ep : (sda, nd, rda, arda) \vdash (sda', nd, rda, arda)) \vdash (t-cpe-h)}$$

**Figure 4: Typing rules for instructions with one register as argument.**

$$\frac{\exists instr \in \{\text{dec}, \text{inc}\}: P(ep) = (\text{instr}, (Rd, k)) \quad rda' = rda[Rd \mapsto rda(Rd) \pm sr(ep)] \quad arda' = arda[Z \mapsto rda(Rd) \sqcup sr(ep)]}{P(\dots, ep : (sda, nd, rda, arda) \vdash (sda', nd, rda, arda)) \vdash (t-1a-z)}$$

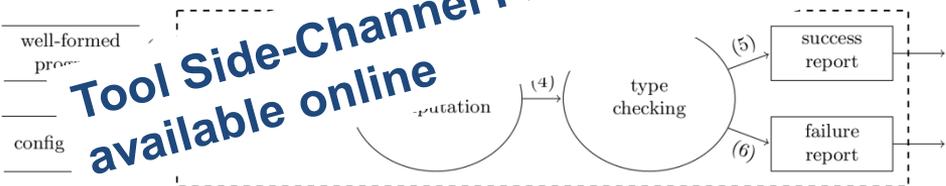
$$\frac{\exists instr \in \{\text{lar}, \text{neg}, \text{ror}\}: P(ep) = (\text{instr}, (Rd, k)) \quad rda' = rda[Rd \mapsto rda(Rd)] \sqcup sr(ep) \quad arda' = arda[Z \mapsto rda(Rd) \sqcup sr(ep)]}{P(\dots, ep : (sda, nd, rda, arda) \vdash (sda', nd, rda, arda)) \vdash (t-1a-az)}$$


**Case  $P(\text{select}(s_1)) = (\text{instr}, (Rd, k))$  for  $\text{instr} \in \{\text{lar}, \text{neg}, \text{ror}\}$**  All possible instructions have in common, that they modify register Rd and status registers C and Z, according to contents of register Rd. Thus, one gets directly that  $m_1 = m_2, m'_1 = m'_2, s_1 = s_2$  and  $s'_1 = s'_2$ . So it remains to show that  $rda_1(Rd) = C \Rightarrow s_2(Rd) = r'_2(Rd), arda_1(C) = sr_1(C)$  and  $arda_1(Z) = C \Rightarrow sr_2(Z) = sr'_2(Z)$ . By rule (t-1a-az)

**Soundness proof for the type system**

**Case  $P(\text{select}(s_1)) = (\text{lar}, (Rd, k))$**  Instruction lar loads the given immediate value k into register Rd. No status registers are modified. Thus, one gets directly that  $s_1 = s_2, s'_1 = s'_2, m_1 = m_2, m'_1 = m'_2, s_1 = s_2$  and  $s'_1 = s'_2$ .

**Tool Side-Channel FinderAVR available online**



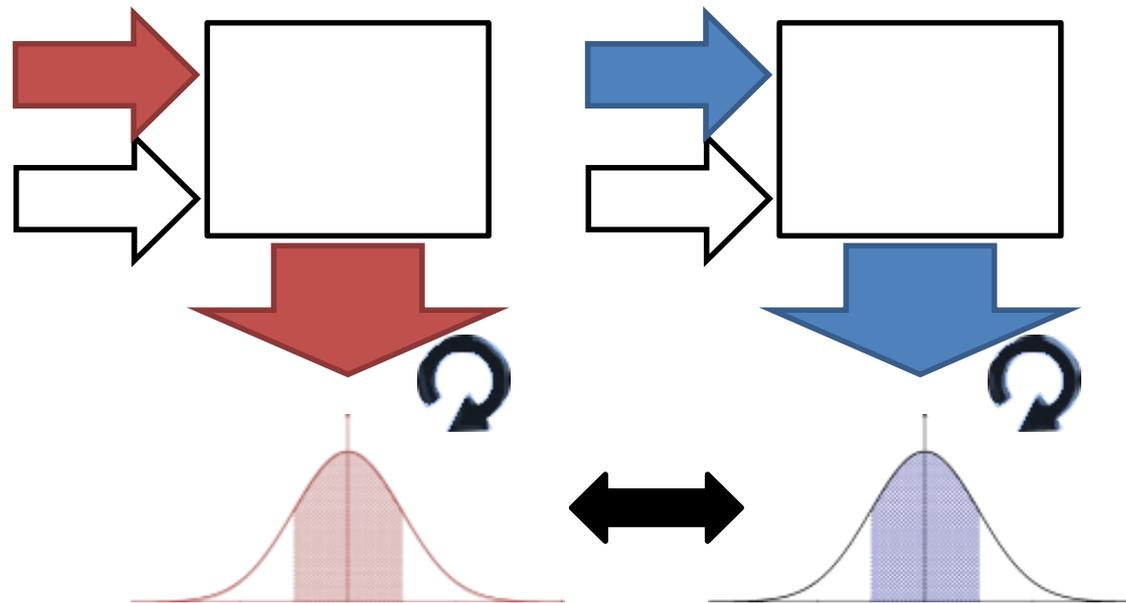
**Evaluation on Salsa20 and Poly1305 from  $\mu\text{NaCl}$  library**

```

23
24 1e:
25 20: c1,
26 22: df 93
27 24: cd b7
28 26: de b7
29 28: c8 55
30 2a: d1 09
31 2c: 0f b6
32 2e: f8 94
33 30: de bf
34 32: 0f be
35 34: cd bf
36 36: 1c 01
37 38: 7b 01
38 3a: 2a 01
39 3c: 38 01

```

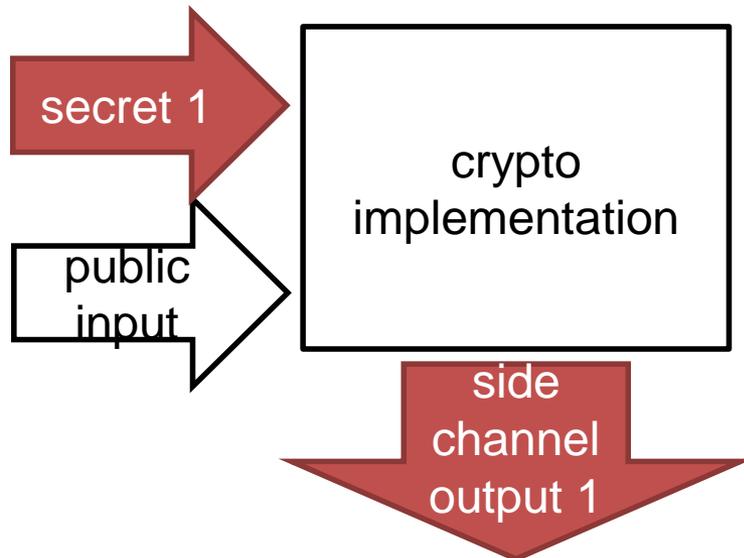
# Approach 3: Experimental Evaluation



## Approach 3: Experimental Evaluation

# Distinguishing Experiments

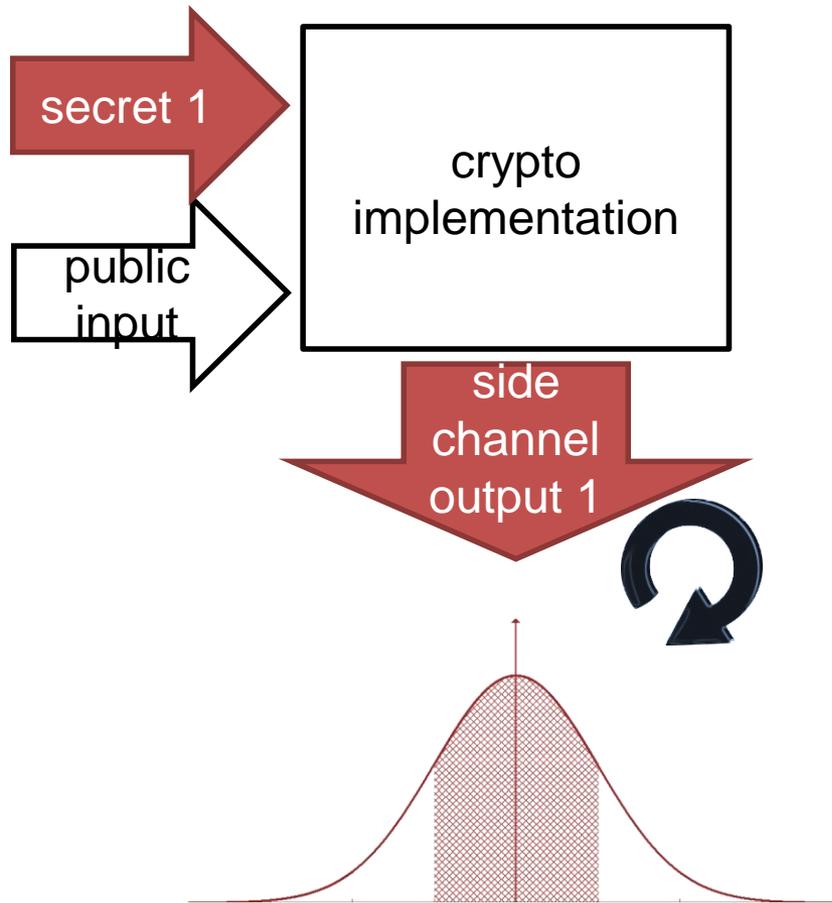
---



## Approach 3: Experimental Evaluation

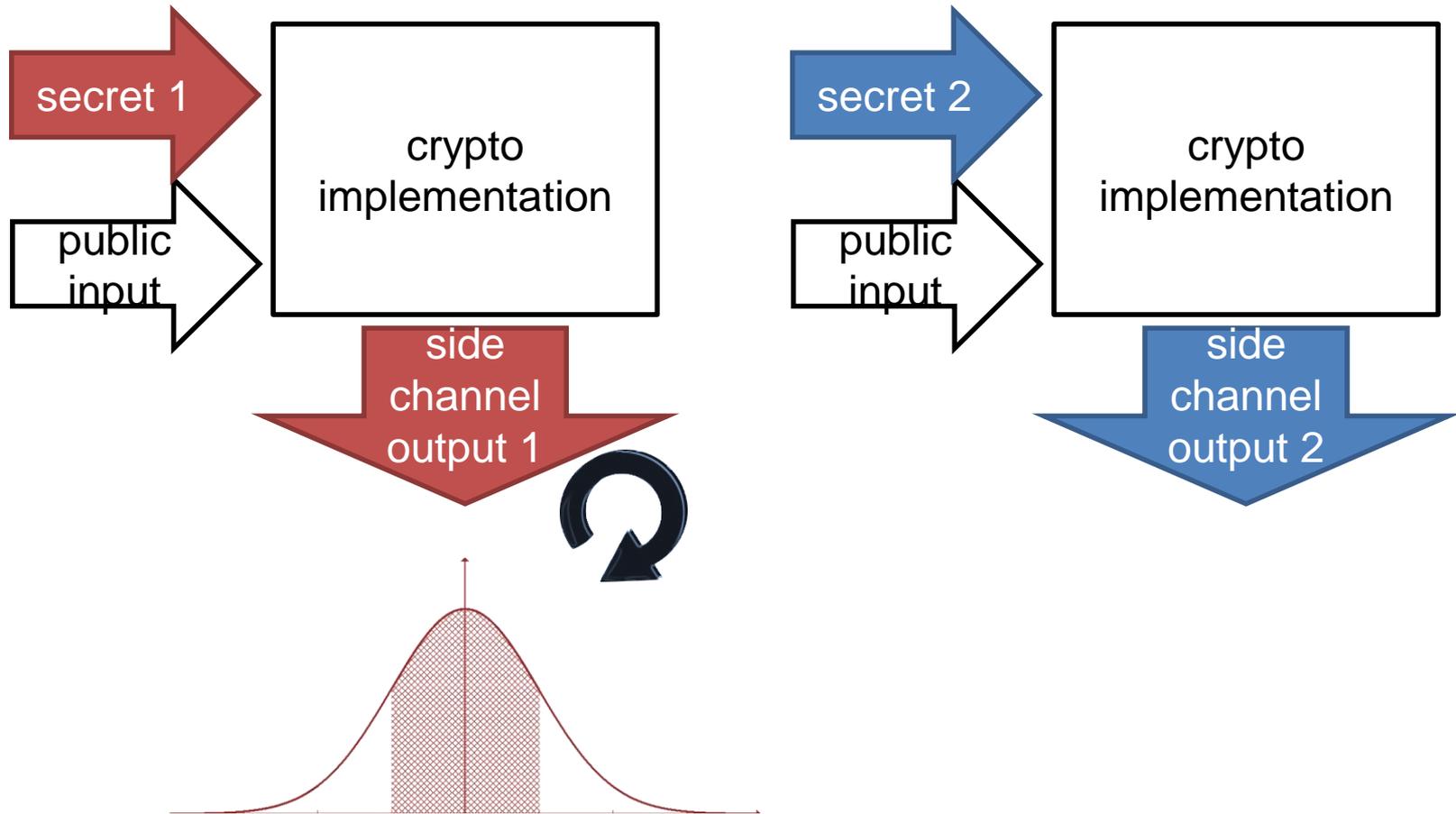
# Distinguishing Experiments

---



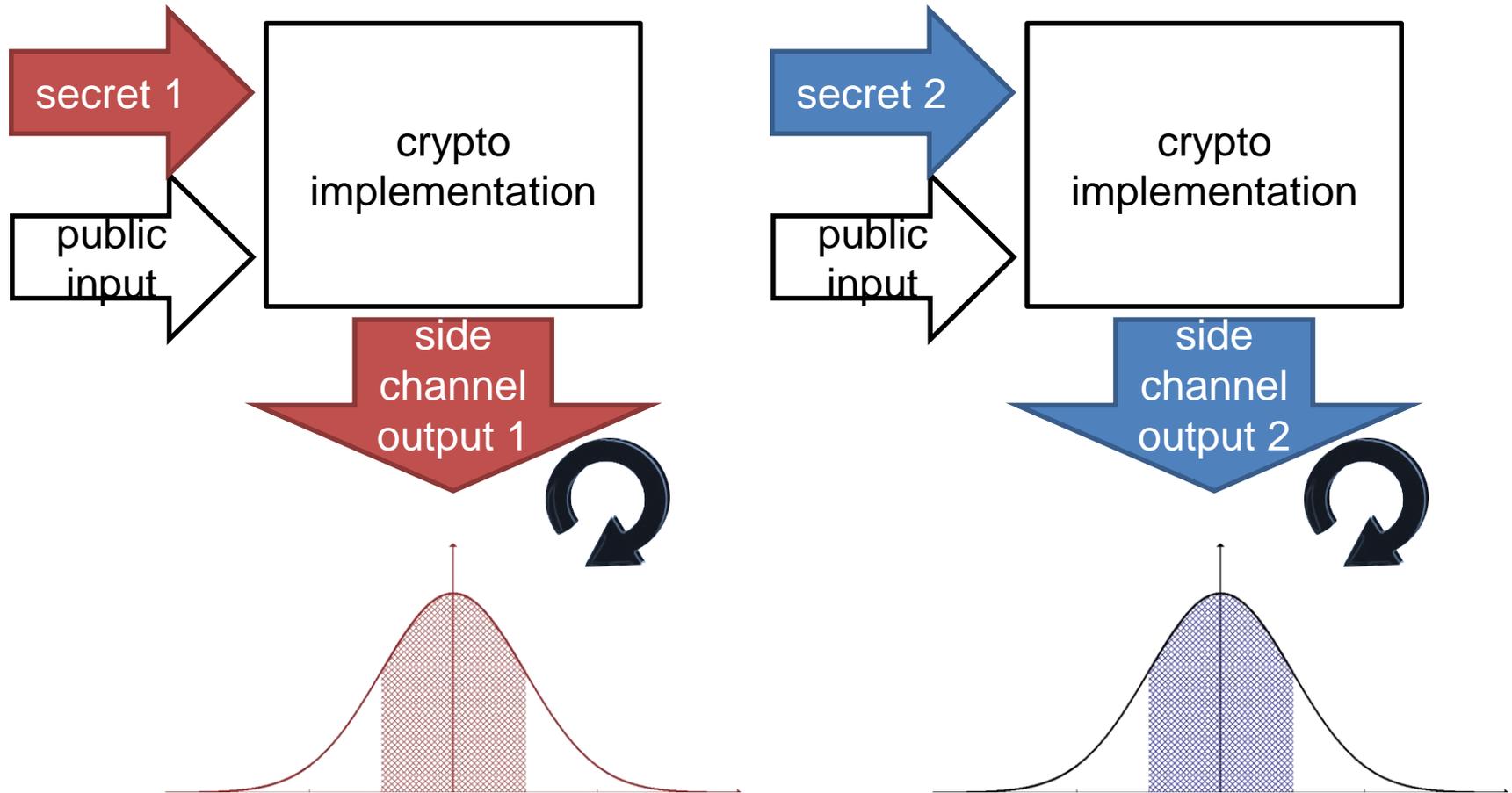
## Approach 3: Experimental Evaluation

# Distinguishing Experiments



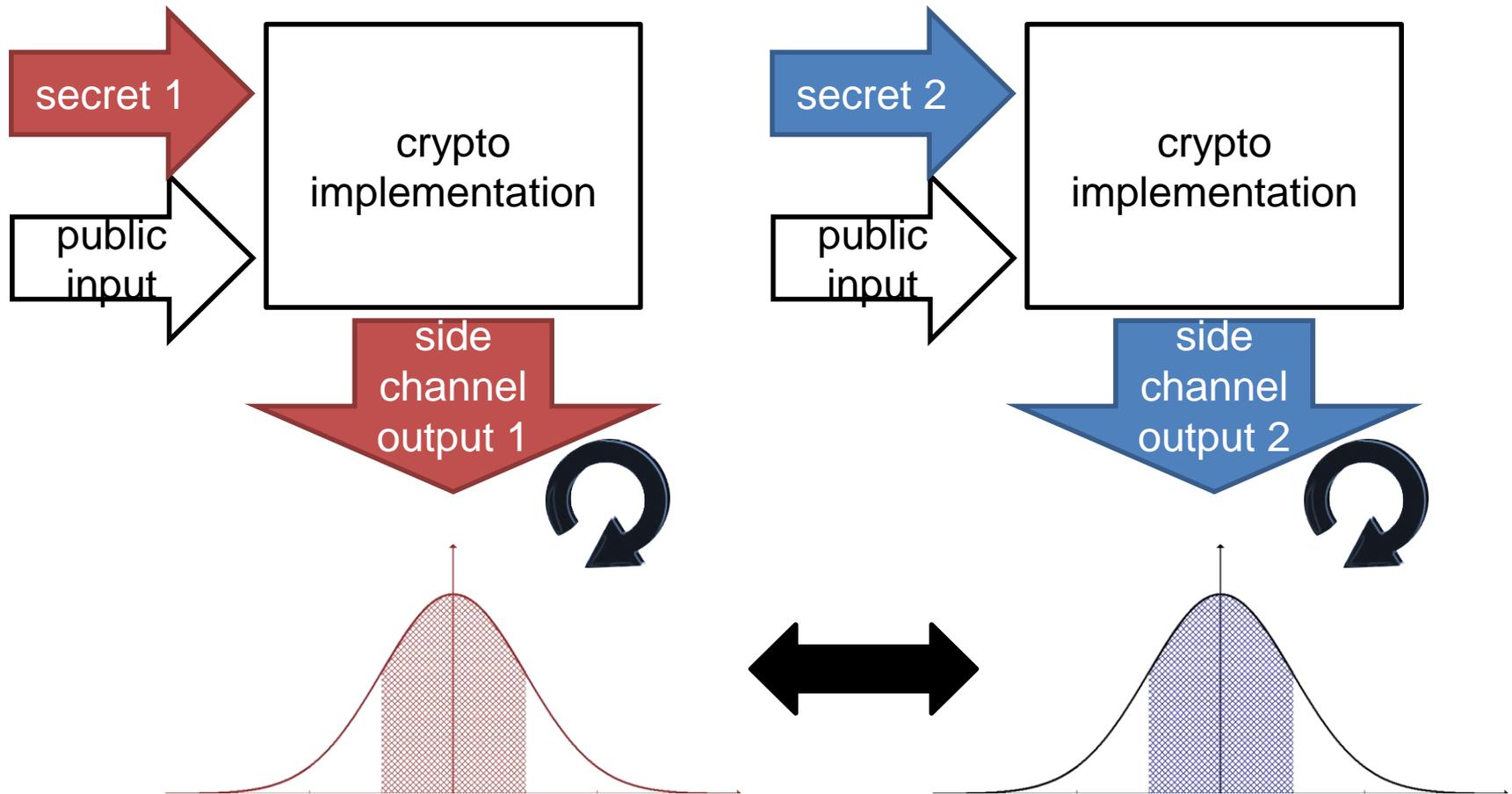
## Approach 3: Experimental Evaluation

# Distinguishing Experiments



## Approach 3: Experimental Evaluation

# Distinguishing Experiments



## Approach 3: Experimental Evaluation

# Reminder: Square-and-Multiply

## Square-and-Multiply Implementation of Modular Exponentiation

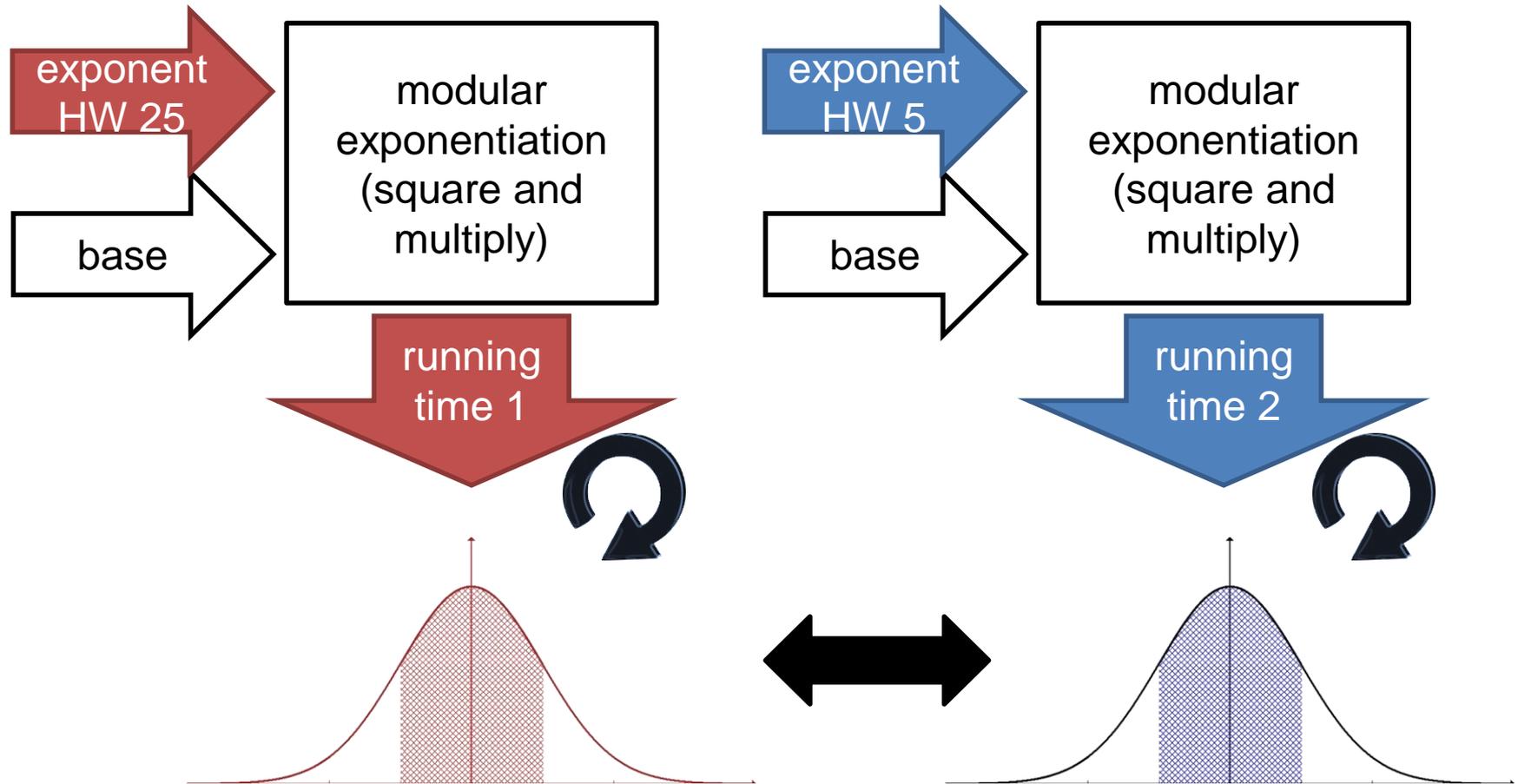
```
square-and-multiply(x, exponent)
  res = 1
  for (i=0; i<|exponent|; i++){
    res=res*res mod N;
    if (exponent[i]=1){
      res=res*x mod N
    }
  }
```



- $res=res*x \bmod N$  is only executed for 1-bits in exponent
- running time depends on Hamming weight of exponent

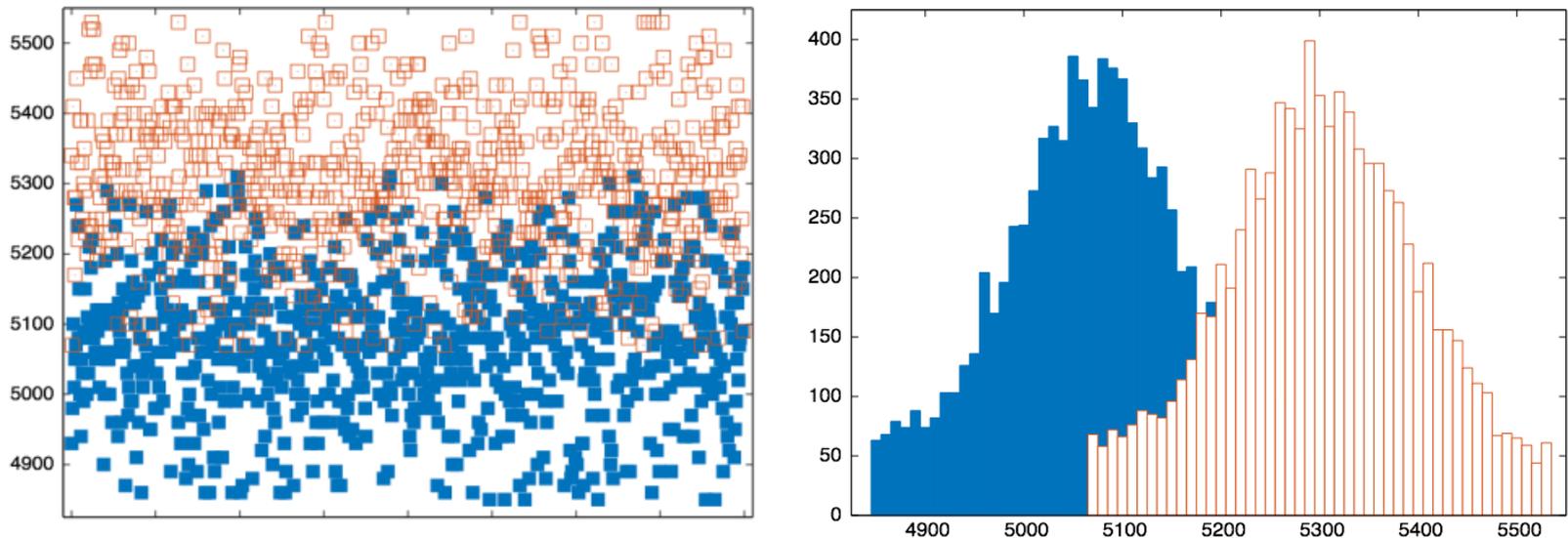
## Approach 3: Experimental Evaluation

# Experiment for Square-and-Multiply



## Approach 3: Experimental Evaluation

# Results for Square-and-Multiply



modular exponentiation (1000 running time samples)  
for Hamming weight 5 and Hamming weight 25

# Application of Approach 3: Evaluation of Side-Channel Mitigations

Heiko Mantel and Artem Starostin.

Transforming Out Timing Leaks, More or Less.

In Proceedings of the 20th European Symposium on Research in  
Computer Security (ESORICS) - Part I, pages 447-467, 2015.

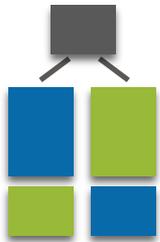
# Application 3: Evaluation of Side-Channel Mitigations

## Mitigation of Timing Side Channels

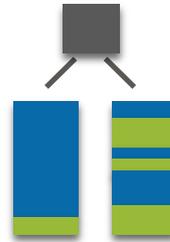
---

### Program Transformations against Timing Channels

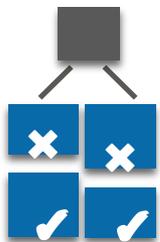
- focus: timing side channels caused by conditional branching
- a spectrum of program transformations has been proposed as mitigations
- previously investigated analytically wrt. abstract models of computation



cross-copying  
[Agat. POPL, 2000]



unification  
[Köpf and Mantel. FAST, 2005]



transactional branching  
[Barthe, Rezk and Warnier.  
QAPL, 2005]



conditional assignment  
[Molnar, Piotrowski, Schultz  
and Wagner. ICISC, 2006]

# Application 3: Evaluation of Side-Channel Mitigations

## Goals of the Evaluation

---

### Research Questions

- improve the understanding of side-channel mitigation by program transformations
- clarify the relationship between four well-known transformations, in practice
  - wrt. the security that they add to a program
  - wrt. the performance overhead that they introduce



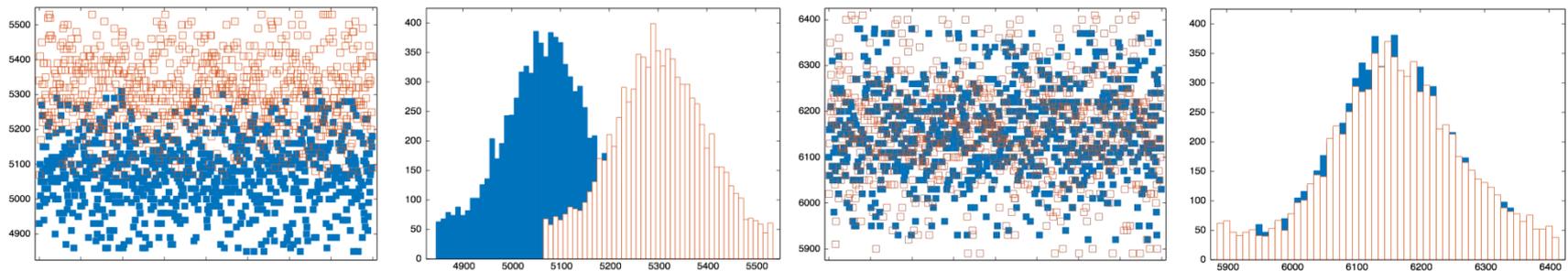
How much security is added by each transformation  
at which costs in practice?

# Application 3: Evaluation of Side-Channel Mitigations

## Design of Experiments

### Security Evaluation

- estimation of timing side-channel capacity in distinguishing experiments
- computation of side-channel capacity reduction by transformations, in %



modExp, baseline

for Hamming weight 5 and Hamming weight 25

modExp, cross-copied

for Hamming weight 5 and Hamming weight 25

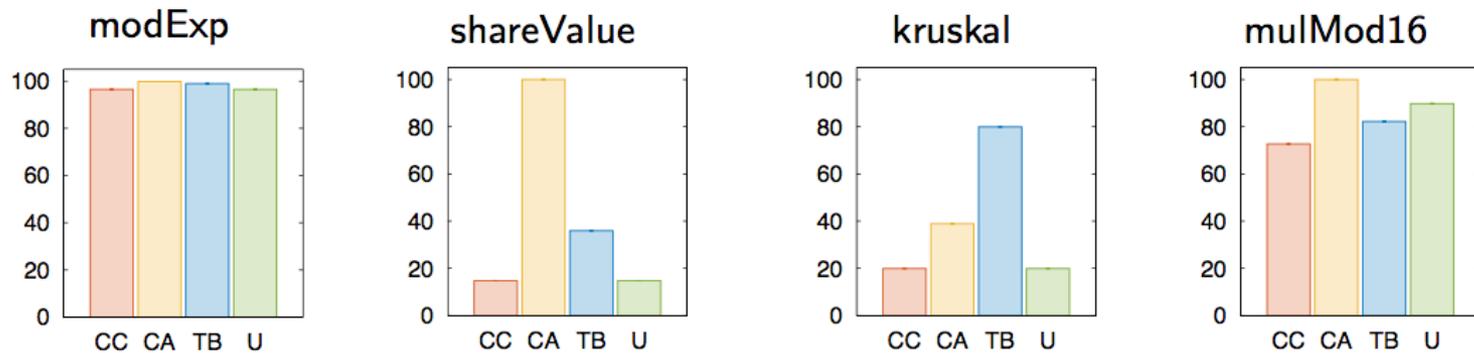
### Performance Evaluation

- estimation of mean running timing by random sampling
- computation of performance overhead induced by transformations, in %

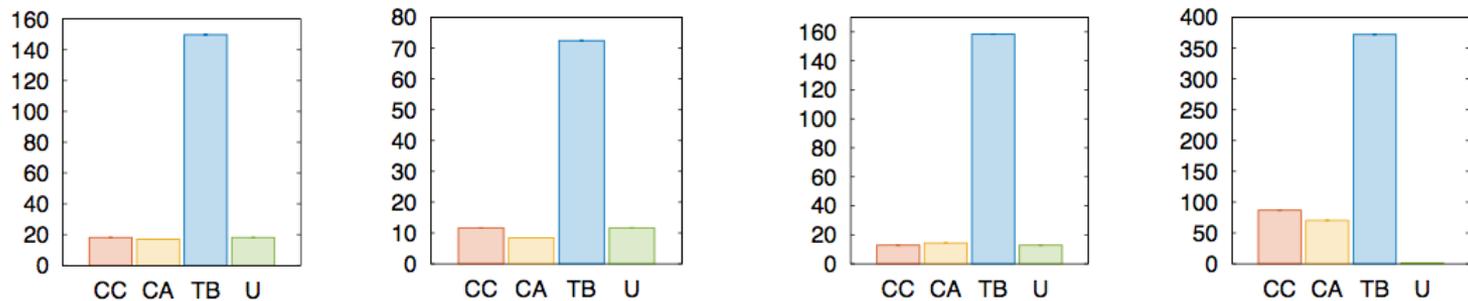
# Application 3: Evaluation of Side-Channel Mitigations

## Experimental Results

### Reduction of the Timing-Side-Channel Capacity, in %



### Performance Overhead Induced by Transformations, in %

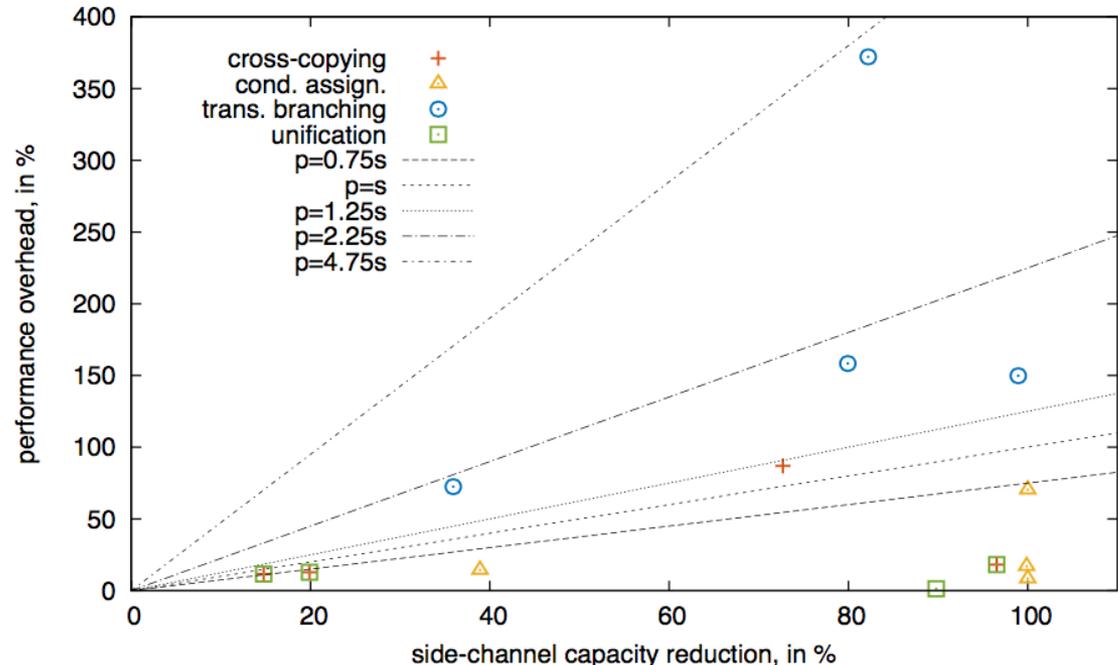


# Application 3: Evaluation of Side-Channel Mitigations

## Trade-off: Performance vs. Security

“We are willing to pay  $a\%$  in performance overhead for 1 percent of side-channel capacity reduction”, or

$$p = a s.$$

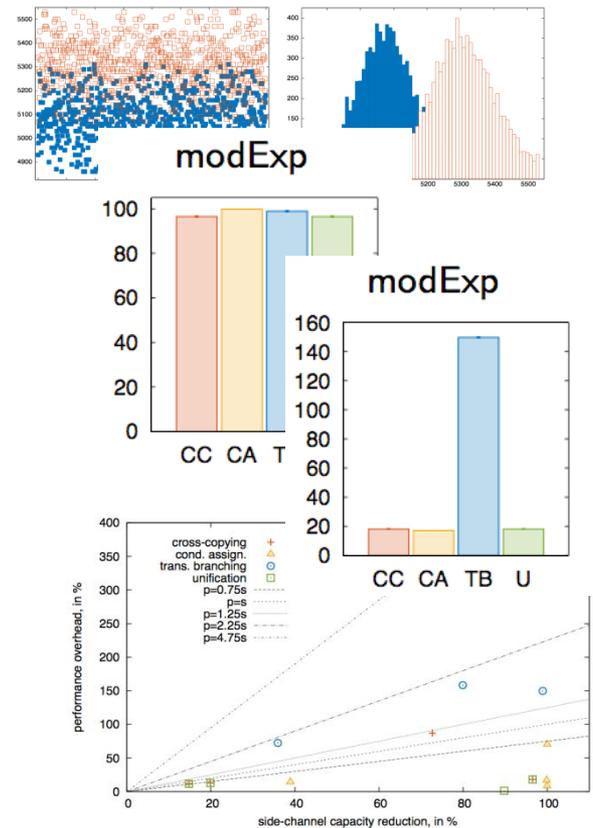


Since all data points for conditional assignment (CA) lie below the beam for 0.75s, the results suggest that CA satisfies the performance-security requirement for  $a = 0.75$  – best among the four transformations.

# Application 3: Evaluation of Side-Channel Mitigations

## Summary

- the first systematic empirical evaluation of source-to-source transformations for removing timing side channels
- the quantification of the positive and negative consequences of four well-known program transformations based on experiments
- the clarification of the trade-off between performance overhead and security in this context
- guidance on choosing suitable program transformations



# Approaches to Reliable Side-Channel Security

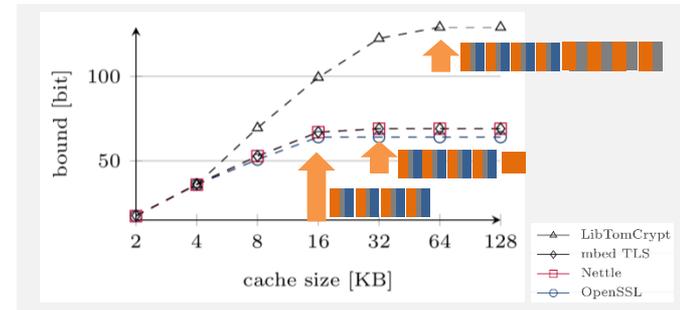
## Summary

# Approaches to Reliable Side-Channel Security

## Summary

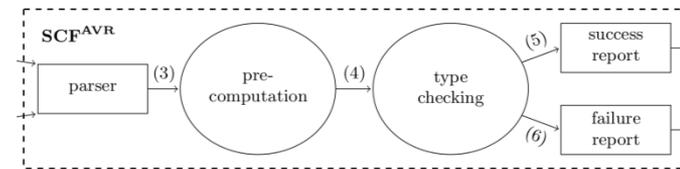
### Information Theory and Reachability Analysis

- Example Application: Cache Side Channels in AES Implementations



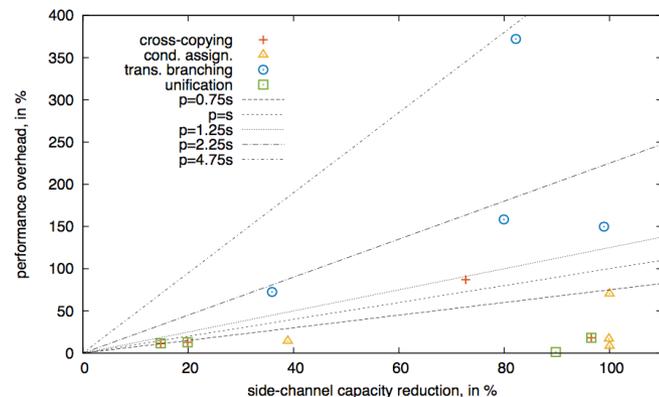
### Security Type Systems

- Example Application: Timing Side Channels in IoT Implementations



### Experimental Evaluation

- Example Application: Evaluation of Side-Channel Mitigations



Thank you for you attention!