CS 267: Automated Verification

Lectures 15-16: Quantitative Symbolic Analysis

Instructor: Tevfik Bultan

Outline

Information leakage and side channels

Quantifying information leakage

Side channel detection with probabilistic symbolic execution

Model counting

Attack synthesis

Information leaks via side channels

TIME

Monday, Aug. 13, 1990

And Bomb The Anchovies

By Paul Gray

Delivery people at various Domino's pizza outlets in and around Washington claim that they have learned to anticipate big news baking at the White House or the Pentagon by the upsurge in takeout orders. Phones usually start ringing some 72 hours before an official announcement. "We know," says one pizza runner. "Absolutely. Pentagon orders doubled up the night before the Panama attack; same thing happened before the Grenada invasion." Last Wednesday, he adds, "we got a lot of orders, starting around midnight. We figured something was up." This time the big news arrived quickly: Iraq's surprise invasion of Kuwait.

Information leaks via side channels

TIME

Monday, Aug. 13, 1990

And Bomb The Anchovies

By Paul Gray

Delivery people at various Domino's pizza outlets in and around Washington claim that they have learned to anticipate big news baking at the White House or the Pentagon by the upsurge in takeout orders. Phones usually start ringing some 72 hours before an official announcement. "We know," says one pizza runner. "Absolutely. Pentagon orders doubled up the night before the Panama attack; same thing happened before the Grenada invasion." Last Wednesday, he adds, "we got a lot of orders, starting around midnight. We figured something was up." This time the big news arrived quickly: Iraq's surprise invasion of Kuwait.

Information leaks via side channels

TIME

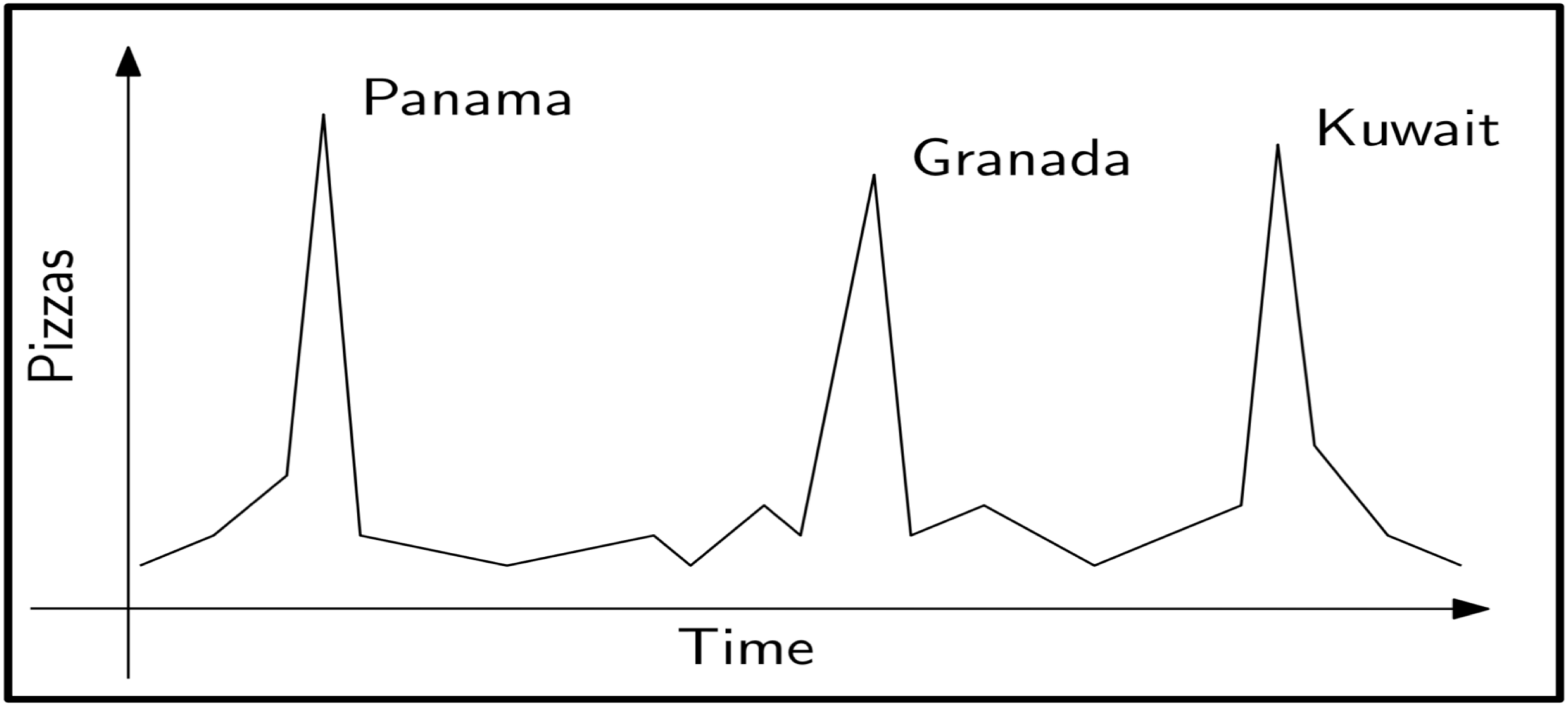
Monday, Aug. 13, 1990

And Bomb The Anchovies

By Paul Gray

Delivery people at various Domino's pizza outlets in and around Washington claim that they have learned to anticipate big news baking at the White House or the Pentagon by the upsurge in takeout orders. Phones usually start ringing some 72 hours before an official announcement. "We know," says one pizza runner. "Absolutely. Pentagon orders doubled up the night before the Panama attack; same thing happened before the Grenada invasion." Last Wednesday, he adds, "we got a lot of orders, starting around midnight. We figured something was up." This time the big news arrived quickly: Iraq's surprise invasion of Kuwait.

Information leaks via side channels



Side-channels in computing

Meltdown

Reading kernel memory from user space

Spectre

Exploiting speculative execution

Segment oracle side channel vulnerability

```
int memcmp(s1, s2, n)
    CONST VOID *s1; CONST VOID *s2; size t n;
   {
      unsigned char u1, u2;
      for ( ; n- ; s1++, s2++) {
         u1 = * (unsigned char *) s1;
         u2 = * (unsigned char *) s2;
         if ( u1 != u2) { return (u1-u2); }
      }
      return 0;
   }
```

Xbox OS, HMAC signatures compared with memcmp.
Leads to side-channel vulnerability and exploit!
Prefix attack: attacker reveals the secret input segment by segment

Segment oracle side-channel vulnerability

Timing attack in Google Keyczar library

Filed under: Crypto, Hacking, Network, Protocols, python, Security — Nate Lawson @ 11:30 pm

I recently found a security flaw in the Google Keyczar crypto library. The impact was that an attacker could forge signatures for data that was "signed" with the SHA-1

Firstly, I'm really glad to see more high-level libraries being developed so that programmers don't have to work directly with algorithms. Keyczar is definitely a step in t
responding quickly to address this issue after I notified him (Python fix and Java fix).

[security] Widespread Timing Vulnerabilities in OpenID im

Taylor Nelson taylor at rootlabs.com
Tue Jul 13 20:32:50 UTC 2010

- Next message: [security] Widespread Timing Vulnerabilities in OpenID implementations
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Every OpenID implementation I have checked this far has contained timing
dependent compares in the HMAC verification, allowing a remote attacker
to forge valid tokens.

In JOpenId:
There is a timing vulnerability in thegetAuthentication  function in
trunk/JOpenId/src/org/expressme/openid/OpenIdManager.java
```

Information leakage

- To model information leakage, classify inputs and outputs as *Secret* and *Public*
- **Confidentiality:** Information about Secret input values should not be leaked to Public output values
- In the literature security levels are typically referred as *High* (Secret) and *Low* (Public)

Non-interference

- Having no information leakage is characterized as non-interference

Non-interference: High (Secret) input values should have no influence on Low (Public) output values

Non-interference is not practical for many cases

In many cases some leakage is unavoidable:

- Any password checker leaks some information about the password
- Another example: Consider an electronic voting application
  - the result of the vote is public and it does leak information about the votes
  - but individual votes should be private
- For many practical cases non-interference is simply not possible and some information leakage from High values to Low values is unavoidable

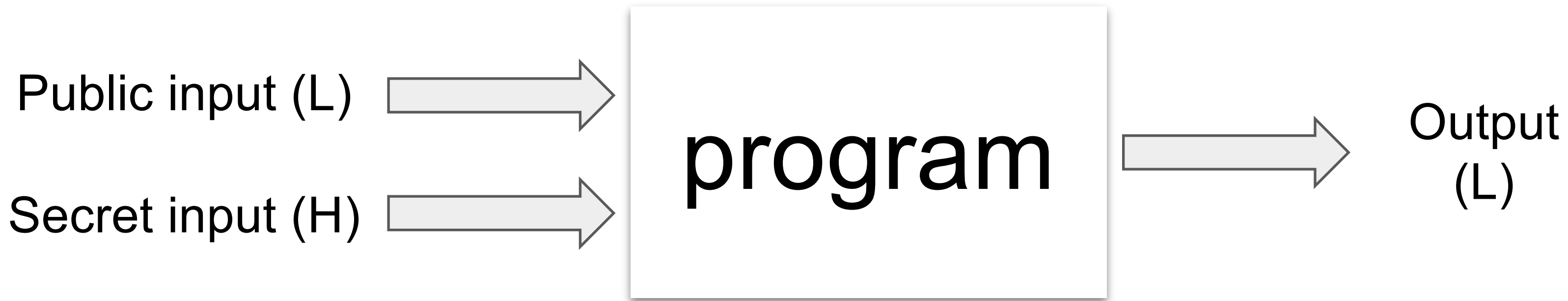
Quantifying information leakage

- If leakage is unavoidable, then the question becomes:
    - “How much information is leaked?”
- For example
    - How much information about a password can be obtained by the attacker who can enter different password guesses to the program?
- If the amount leaked is very small, the program might be considered secure even though there is some information leakage

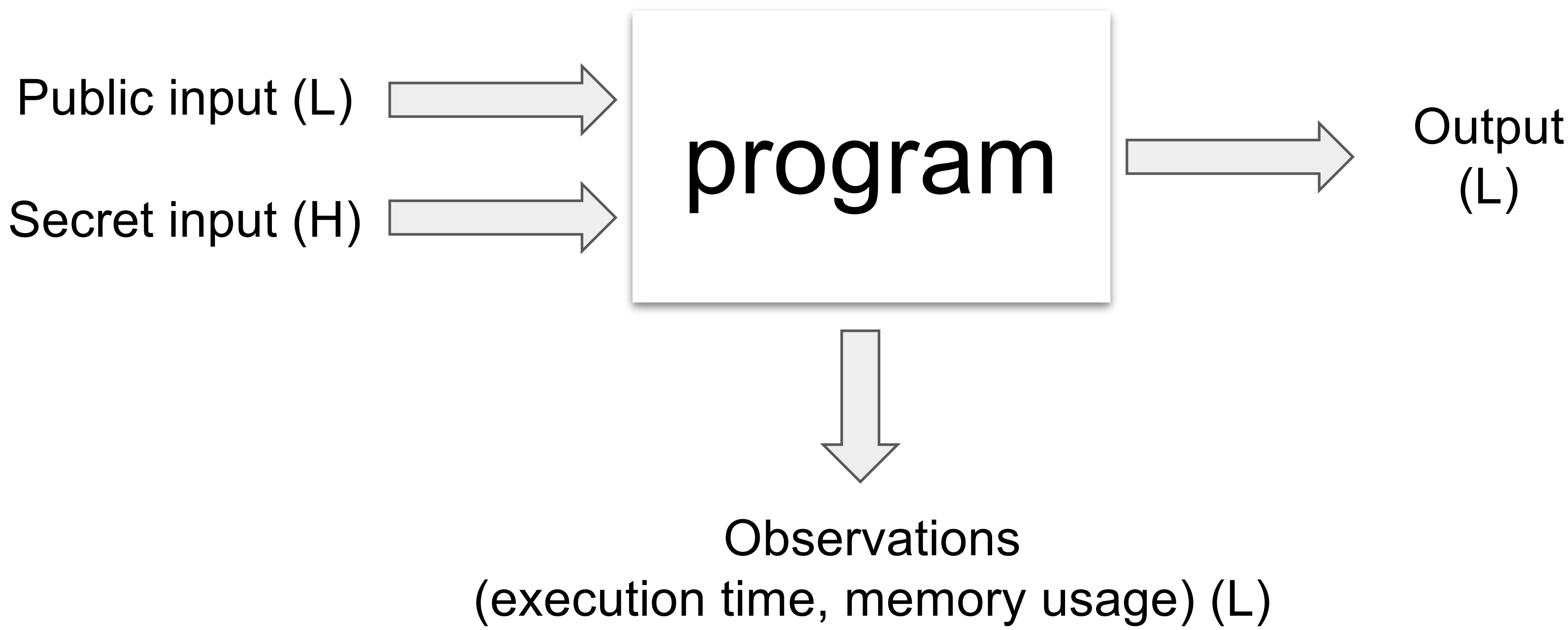
Quantitative information flow

- The goal of ***quantitative information flow*** techniques is to quantify the amount of information leaked from a given program
- Quantitative information flow techniques can be used to detect the amount of information leaked from side channels

Side Channels



Side Channels



Outline

Information leakage and side channels

Quantifying information leakage

Side channel detection with probabilistic symbolic execution

Model counting

Attack synthesis

How do we quantify information?

- ***Shannon Entropy***
  - a measure of uncertainty about a random variable X
  - expected value (***average***) of ***information gain*** (i.e., the expected amount of surprise) by observing the value of the random variable ***expressed in terms of bits***

- Or

  expected value of (average) number of bits required to transmit X optimally

Entropy example:

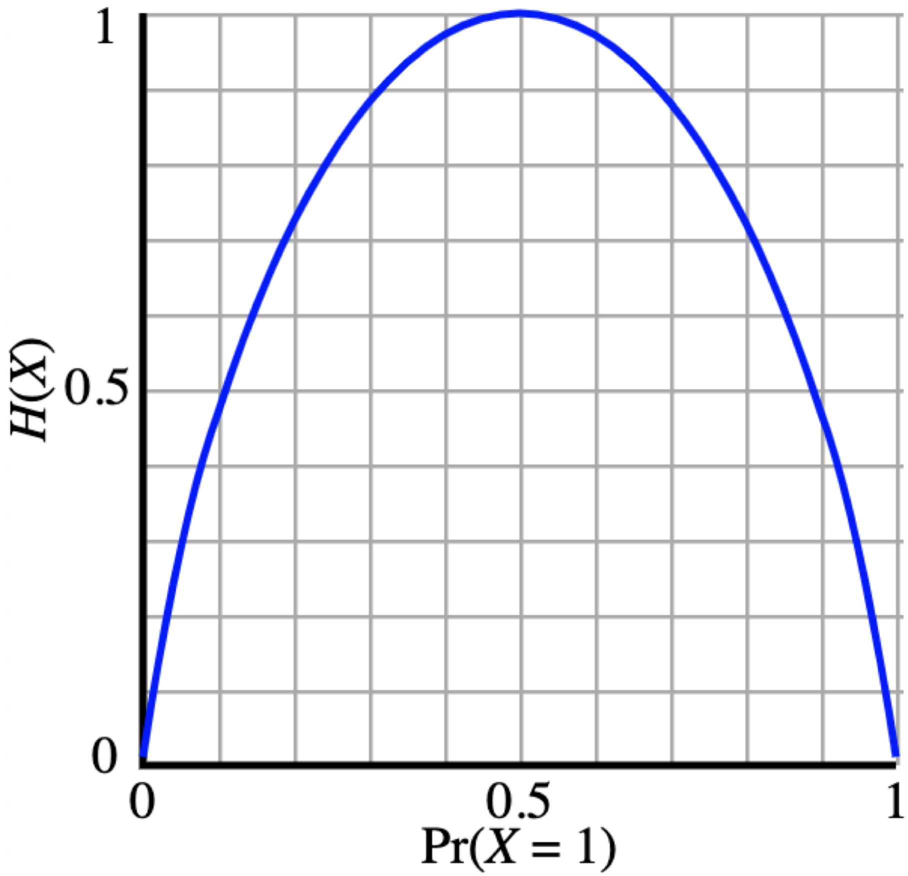
Example:

- Seattle weather, always raining: $p_{rain} = 1$
- Entropy: $H = 0$

- Costa Rica weather, coin flip: $p_{rain} = 0.5$, $p_{sun} = 0.5$
- Entropy: $H = 1$

- Santa Barbara weather, almost always beautiful:
  $p_{rain} = 0.1$, $p_{sun} = 0.9$
- Entropy: $H = 0.496$

Binary Entropy



How do we quantify information?

- Random variable: X
- Domain of the random variable: $\mathcal{X}$
- Probability that the random variable takes the value $x \in \mathcal{X}$

$$P[X = x]$$

- Shannon Entropy: $H(X)$

$$H(X) = \sum_{x \in \mathcal{X}} P[X = x] \log_2(1/P[X = x])$$
$$H(X) = E[\log_2(1/P[X = x])]$$

- Shannon entropy is the expected value of: $\log_2(1/P[X = x])$

How do we quantify information leakage?

- Now that we know how to quantify information, how can we quantify information leakage?
- First, let's give a simple program model:

  S is the secret input to the program. We will model it as a random variable.

  O is the public output of the program. We will model it also as a random variable

  f is a function from values of S to values of O we use to model a deterministic program

Initial uncertainty

- What is the initial uncertainty for S?
  - What is the amount of information that we need to learn about the secret?

$$H(S) = \sum_{s \in \mathcal{S}} P[S = s] \log_2 (1/P[S = s])$$

- Assume that the probability distribution for the secret is uniform
  - so all values are equally likely
  - then, the amount of information that we need to learn is:

$$H(S) = \log_2 |\mathcal{S}|$$

UCSB

Partitioning the secret domain

- Given a program

$$f : \mathcal{S} \to \mathcal{O}$$

- The values we observe as the output of the program define an equivalence relation for the secret S

$$s \sim s' \text{ iff } f(s) = f(s')$$

- So, by observing output of the program, we partition the secret values to equivalence classes

Partitioning the secret domain

- The number of equivalence classes in the partition are:

$$|\mathcal{O}|$$

- If the function is a constant function, where the output is constant, then

$$|\mathcal{O}| = 1$$

  - and, there is a single equivalence class where

$$\mathcal{S}_o = \mathcal{S}$$

UCSB

Non-interference

- So, if the output function is a constant function
    - the amount of information we need to learn remains the same

$$H(S) = \log_2 |\mathcal{S}|$$

    - means there is no information leakage

- This correspond to non-interference!
    - If the output/observable remains constant for all values of the secret then there is no information leakage!

Partitioning the secret domain

- Now, let us assume that the output values partition the secret domain to two equivalence classes with equal number of elements
  - I.e., there are two output values, half of the secret values map to one and the other half map to the other

- What is the remaining entropy?

Another example

```
f(S) { print S & 0xF; }
```

- Assume that S is a 32-bit unsigned integer
- 0xF is the hexadecimal constant corresponding to decimal 15, and & denotes bitwise “and” operation
  - So, the above code prints the last 4 bits of the secret
- The output partitions the secret domain to 16 equivalence classes, each of which has 2^{28} values in it
  - So, the remaining entropy is 28 bits

How do we quantify information leakage?

- Now that we know how to quantify information, how can we quantify information leakage
- Here is what we would expect:

  initial uncertainty = information leaked + remaining uncertainty

- Equivalently

  information leaked = initial uncertainty - remaining uncertainty

How do we quantify the remaining uncertainty?

- Remaining uncertainty can be characterized as the conditional entropy
- Conditional entropy: What is the uncertainty about S given O?

$$H(S|O) = \sum_{o \in \mathcal{O}} P[O = o] H(S|O = o)$$

$$H(S|O = o) = \sum_{s \in \mathcal{S}} P[S = s|O = o] \log_2(1/P[S = s|O = o])$$

UCSB

Conditional Entropy uses Conditional Probability

$$H(S|O=o) = \sum_{s \in \mathcal{S}} P[S=s|O=o] \log_2(1/P[S=s|O=o])$$

$$P[S=s|O=o] = P[S=s, O=o]/P[O=o]$$

UCSB

Mutual information

- Mutual information I(S;O) is the amount of information shared between S and O
- It is defined as:

$$I(S;O) = H(S) - H(S|O)$$

- Mutual information is symmetric:

$$I(S;O) = I(O;S)$$

UCSB

How do we quantify information leakage?

- So, the intuitive property

  information leaked = initial uncertainty - remaining uncertainty

- is formalized as

$$I(S;O) = H(S) - H(S|O)$$

UCSB

Examples

$$I(S;O) = H(S) - H(S|O)$$

`f(S) { print 10; }`	0	=	32	-	32
`f(S) { print S + 10; }`	32	=	32	-	0
`f(S) { print S & 0xF; }`	4	=	32	-	28

UCSB

What about side channels?

```
f(S) { sleep(S); }

f(S) { if (S % 2 == 0) sleep(1); else sleep (2); }
```

- These programs do not return any output or print any information.
  - So, they do not leak information from the main channel of the program.
- However, they do have side channel leakage
  - They leak information from the execution time

What about side channels?

$$I(S;O) = H(S) - H(S|O)$$

```
f(S) { sleep(S); }
```

32 = 32 - 0

```
f(S) { if (S % 2 == 0)
         sleep(1);
      else
         sleep (2); }
```

1 = 32 - 31

Deterministic programs

- If we assume that the program is deterministic with only input S and only output O
  - then the value of O is determined only by the input S
  - which means $H(O|S) = 0$

Then, we have:

$$I(S;O) = I(O;S) = H(O) - H(O|S) = H(O)$$

- So, for deterministic programs with input S and output O, the information leaked is equivalent to the uncertainty of O

Outline

Information leakage and side channels

Quantifying information leakage

Side channel detection with probabilistic symbolic execution

Model counting

Attack synthesis

A 4-digit PIN Checker

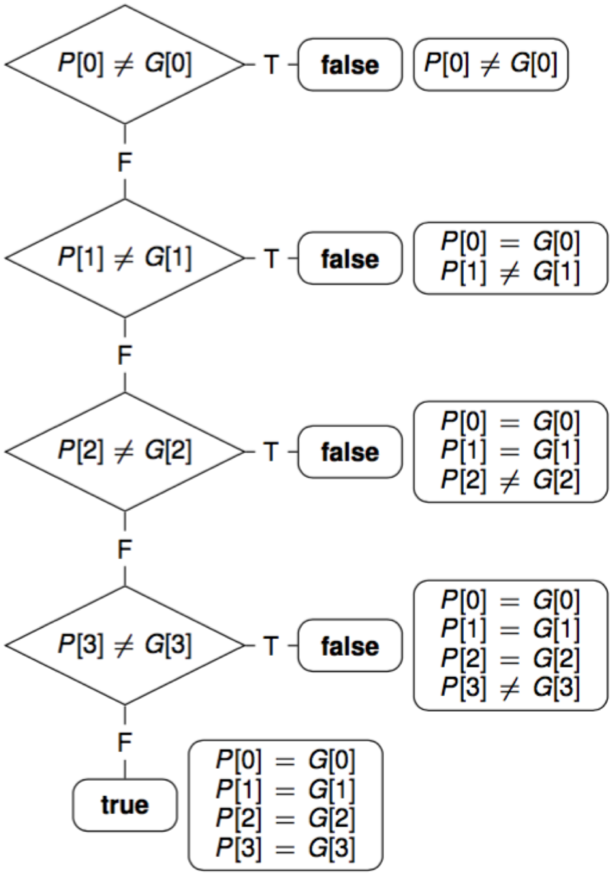
```
bool checkPIN(guess[])
for(i = 0; i < 4; i++)
 if(guess[i] != PIN[i])
  return false
return true
```

P: PIN, *G*: guess

Symbolic Execution of PIN Checker

```
bool checkPIN(guess[])
for(i = 0; i < 4; i++)
 if(guess[i] != PIN[i])
  return false
return true
```

P: PIN, *G*: guess



Probabilistic symbolic execution

Can we determine the probability of executing a program path?

- Let PC_i denote the path constraint for a program path
- Let $|PC_i|$ denote the number of possible solutions for PC_i
- Let $|D|$ denote the size of the input domain
- Assume uniform distribution over the input domain

- Then the probability of executing that program path is:

$$p(PC_i) = |PC_i| / |D|$$

Probabilistic symbolic execution of PIN checker

- Assume binary 4 digit PIN, P and G each have 4 bits

  $|D| = 2^8 = 256$

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
$\|PC_i\|$					
p_i					

$p(PC_i) = |PC_i| / |D|$

Probabilistic symbolic execution of PIN checker

- Assume binary 4 digit PIN, P and G each have 4 bits

$|D| = 2^8 = 256$

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
$\|PC_i\|$	128				
p_i	1/2				

$p(PC_i) = |PC_i| / |D|$

Probabilistic symbolic execution of PIN checker

- Assume binary 4 digit PIN, P and G each have 4 bits

  $|D| = 2^8 = 256$

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
$\|PC_i\|$	128	64			
p_i	1/2	1/4			

$p(PC_i) = |PC_i| / |D|$

Probabilistic Symbolic Execution of PIN Checker

- Assume binary 4 digit PIN, P and G each have 4 bits

$|D| = 2^8 = 256$

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
$\lvert PC_i \rvert$	128	64	32	16	16
p_i	1/2	1/4	1/8	1/16	1/16

Probability that an adversary can guess a prefix of length i in one guess is given by $\boldsymbol{p_i}$

Extending symbolic execution

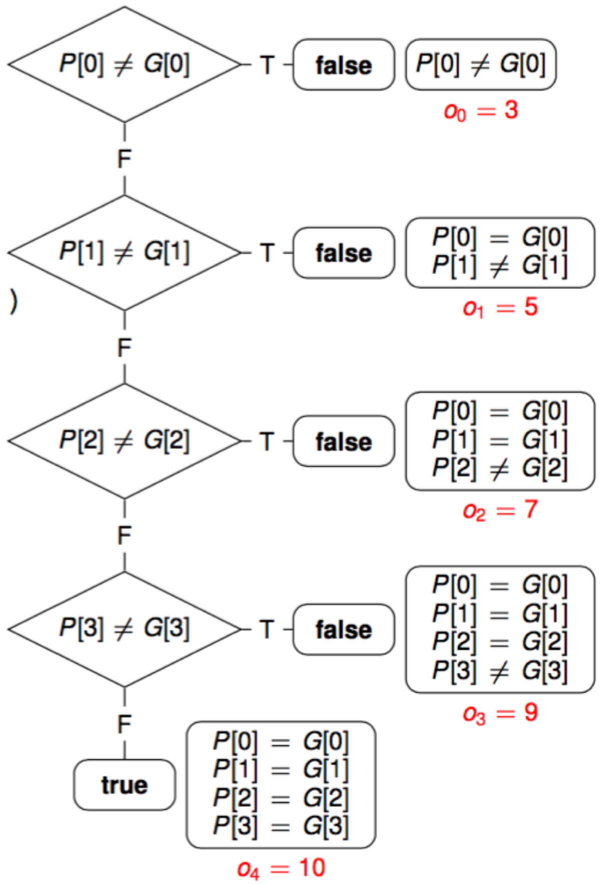
- We need to extend symbolic execution to keep track of observables
- Implement listeners to *collect time/memory costs* for all explored (complete) paths
    - Costs corresponding to the “observables”

Symbolic execution with observable tracking

```
bool checkPIN(guess[])
for(i = 0; i < 4; i++)
 if(guess[i] != PIN[i])
  return false
return true
```

P: PIN, *G*: guess

$o_i =$ lines of code



Timing side channel:

- Estimate the execution time using the number of instructions executed
- Estimate can be improved with profiling

We call this the ``observable''

- For a space side channel the observable could be amount of memory allocated or size of a file

Probabilistic symbolic execution of PIN checker

- Assume binary 4 digit PIN, P and G each have 4 bits

  $|D| = 2^8 = 256$

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
return	false	false	false	false	true
$\|PC_i\|$	128	64	32	16	16
p_i	1/2	1/4	1/8	1/16	1/16
o_i	3	5	7	9	10

Information leakage

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
return	false	false	false	false	true
$\|PC_i\|$	128	64	32	16	16
p_i	1/2	1/4	1/8	1/16	1/16
o_i	3	5	7	9	10

$$H = \sum p_i \log \frac{1}{p_i} = 1.8750$$

- ***H***: Information leakage or the expected amount of information gain by the adversary

A secure PIN checker

```
public verifyPassword (guess[])
  matched = true
  for (int i = 0; i < 4; i++)
    if (guess[i] != PIN[i])
          matched = false
    else
          matched = matched
  return matched
```

- Only two observables (just the main channel, no side channel): $\boldsymbol{o_0}$: does not match, $\boldsymbol{o_1}$: full match

- $\boldsymbol{p}(\boldsymbol{o_0}) = 15/16$, $\boldsymbol{p}(\boldsymbol{o_1}) = 1/16$
- $\boldsymbol{H_{secure}} = 0.33729$

Secure vs. vulnerable PIN checker

- Given a PIN of length L where each PIN digit has K values
- Secure PIN checker
  - K^L guesses in the worst case
  - Example: 16 digit password where each digit is ASCII

128^{16} tries in the worst case, which would take a lot of time!

Secure vs. vulnerable PIN checker

- Vulnerable PIN checker
  - A **prefix attack** that determines each digit one by one starting with the leftmost digit
  - Example: 16 digit password where each digit is ASCII

128×16 tries in the worst case, which would not take too much time

Outline

Information leakage and side channels

Quantifying information leakage

Side channel detection with probabilistic symbolic execution

Model counting

Attack synthesis

Model Counting

- Model counting: Counting the number of satisfying solutions for a given formula

- Many variations of the problem:
  - Boolean logic
  - Integers
  - Strings
  - SMT

Model counting with BDDs

- As we discussed before, we can construct a BDD from a given Boolean logic formula
- BDD is a directed acyclic graph, and it encodes all the satisfying solutions for the Boolean logic formula
  - Each path from the root node of the BDD to the “True” leaf node represents a unique satisfying solution to the Boolean logic formula
- Once you construct a BDD, you can count the number of models by counting paths of the BDD
  - Count the paths that reach from the root to the “True” leaf node

UCSB

Model counting with BDDs

- You need to take into account the variables that are not represented in the BDD
  - they are removed as redundant tests but we need to keep track of them to count
- Count the number of paths that reach True
  - keep track of missing (redundant) variables on a path, and add 2^k to the count for each path that has k missing variables
- Can compute the count in linear time by traversing the nodes from leaves towards the root node

UCSB

Model Counting with DPLL

- As we discussed DPLL is a decision procedure for satisfiability of Boolean formulas in conjunctive normal form (CNF-SAT).

- DPLL can be modified to do model counting

- Let us first give a recursive version of the DPLL algofrithm

DPLL

function DPLL (F: CNF formula): *(returns true iff formula is satisfiable)*

1. if F is empty; return true *(satisfiable)*
2. if F contains an empty clause; return false
3. if there exists a pure literal l in F *(l is s pure literal iff ¬l is not in F)*

   return DPLL(F ∧ l)
4. if F contains a unit clause {l} *(unit propagation)*

   $F_1 = \{C - \{\neg l\}\} \mid C \in F,\ l \notin C\}$

   return DPLL(F_1)
5. Choose a variable x of F *(decide, tries both decisions recursively)*

   return DPLL(F ∧ x) ∨ DPLL(F ∧ ¬x)

UCSB

Model Counting with DPLL

function CDPLL (F: CNF formula, n integer): *(returns number of satisfying solutions)*

1. if F is empty; return 2^n
2. if F contains an empty clause; return 0
3. if F contains a unit clause {l} *(unit propagation)*

   $F_1 = \{C - \{\neg l\}\} \mid C \in F \;\; l \notin C\}$

   return CDPLL(F_1, n-1)

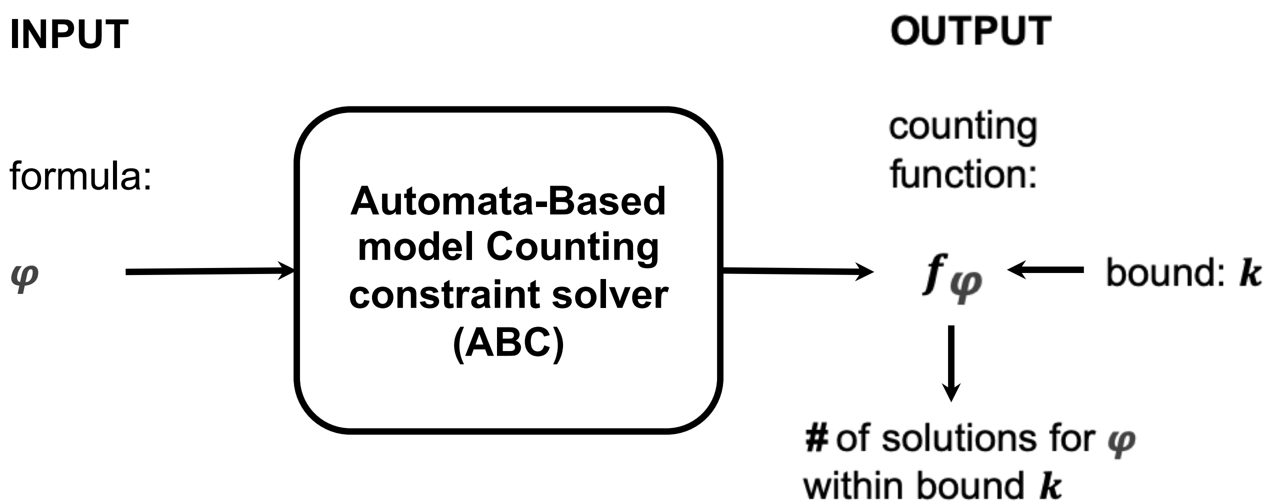
4. Choose a variable x of F *(decide, tries both decisions recursively)*

   $F_1 = \{C - \{\neg x\}\} \mid C \in F \;\; x \notin C\}$

   $F_2 = \{C - \{x\}\} \mid C \in F \;\; \neg x \notin C\}$

return CDPLL(F_1, n - 1) + CDPLL(F_2, n-1)

ABC: Model counting constraint solver



ABC in a nutshell

Automata-based constraint solving

Why?

ABC in a nutshell

Automata-based constraint solving

Basic idea:

Constructing an automaton for the set of solutions of a constraint reduces model counting problem to path counting!

Automata-based constraint solving

Generate automaton that accepts satisfying solutions for the constraint

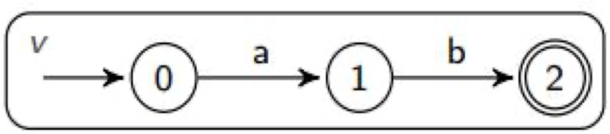
ABC can handle both
string and **integer** constraints

- Constraints over only string variables (e.g., v = “abcd”)
- Constraints over only integer variables (e.g., i = 2×j)
- Constraints over both string and integer variables (e.g., length(v) = i)

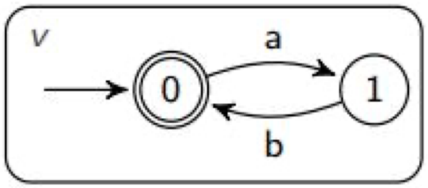
Automata-based constraint solving: expr, ¬

Basic string constraints are directly mapped to automata

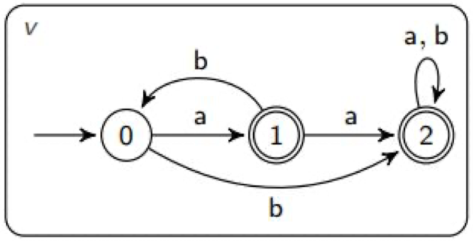
`v = "ab"`



`match(v, (ab)*)`



`¬match(v, (ab)*)`

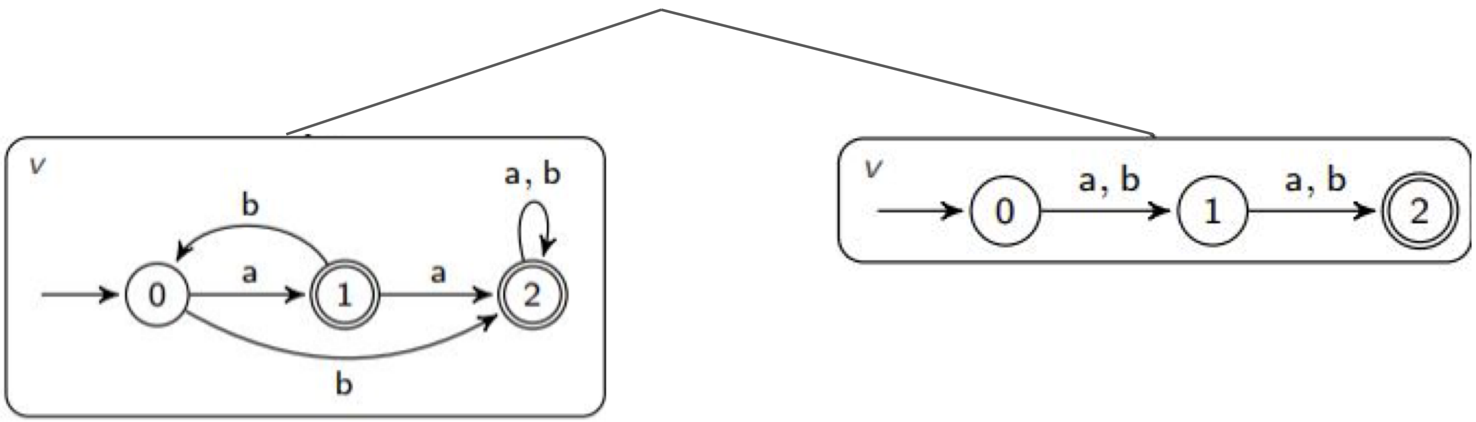


automata
complement

Automata-based constraint solving: expr, ¬, ∧, ∨

More complex constraints are solved by creating automata for subformulae then combining their results

```
¬match(v, (ab)*) ∧ length(v) = 2
```

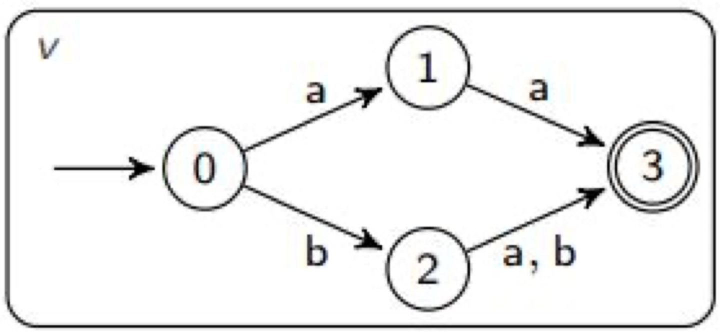


automata
product

Automata-based constraint solving: expr, ¬, ∧, ∨

More complex constraints are solved by creating automata for subformulae then combining their results

```
¬match(v, (ab)*) ∧ length(v) = 2
```

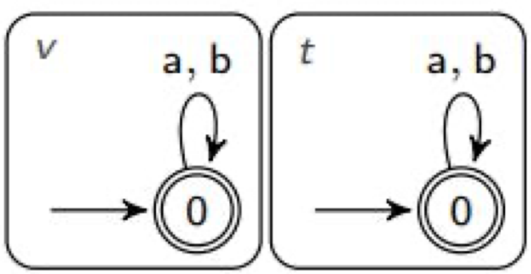


automata
product

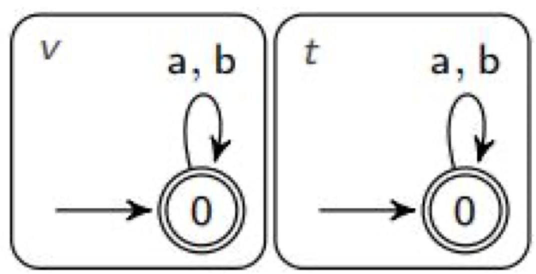
Automata-based constraint solving: relational

For multi-variable constraints, generate an automaton for each variable

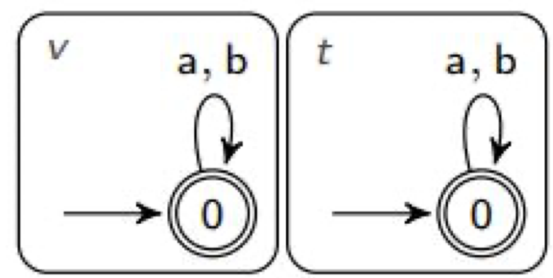
`v = t`



`v ≠ t`



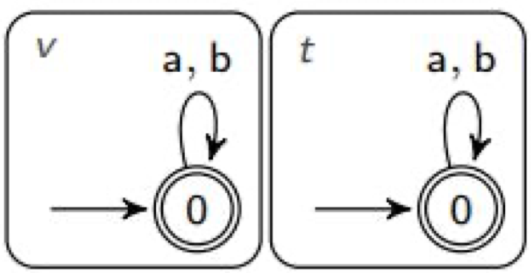
`v = t ∧ v ≠ t`



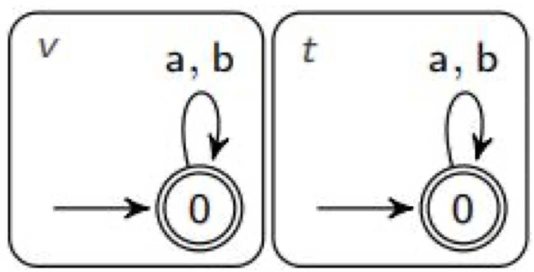
Automata-based constraint solving: relational

For multi-variable constraints, generate an automaton for each variable

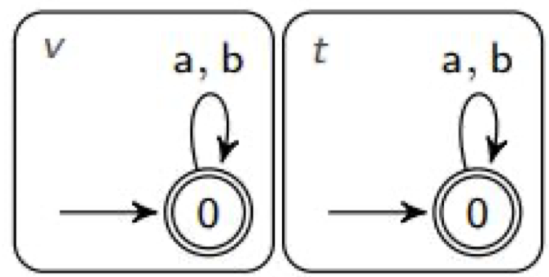
$v = t$



$v \neq t$



$v = t \land v \neq t$



Satisfiable!

Automata-based constraint solving: relational

Single track automata cannot precisely capture relational constraints

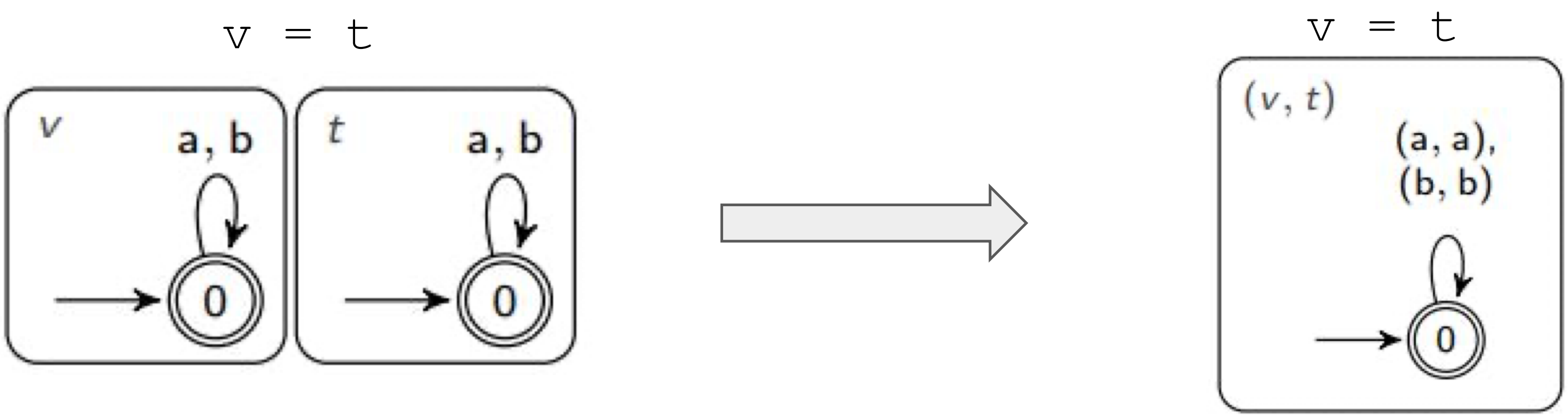
Generated automata significantly over-approximate # of satisfying solutions

Use multi-track automata

Multi-track automata

Multi-track automaton = DFA accepting tuples of strings

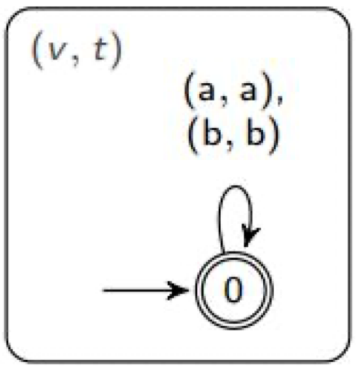
Each track represents the values of a single variable



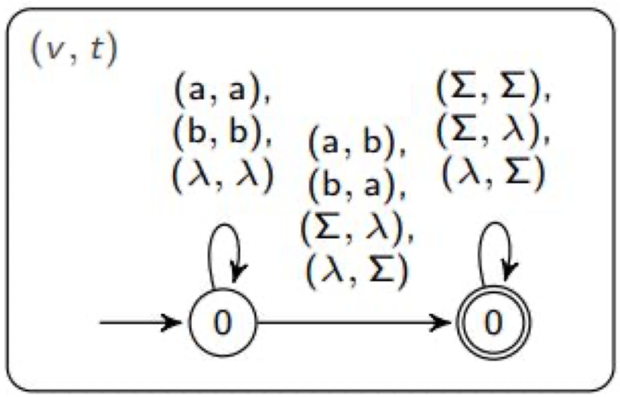
Preserves relations among variables!

Multi-track automata

`v = t`



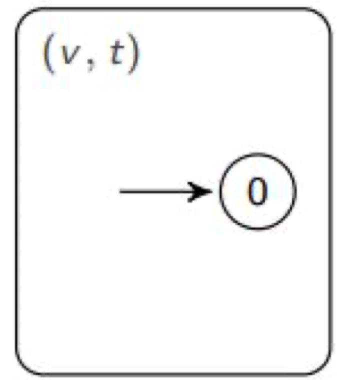
`v ≠ t`



Padding symbol λ ∉ Σ used to align tracks of different length (appears at the end)

automata product

`v = t ∧ v ≠ t`



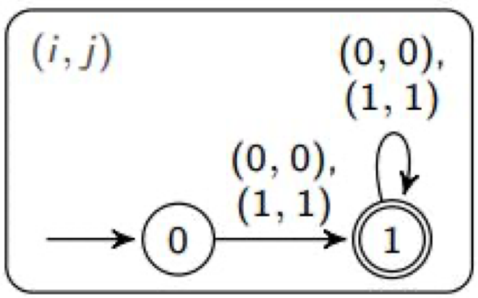
Correctly encodes the constraint

Multi-track automata

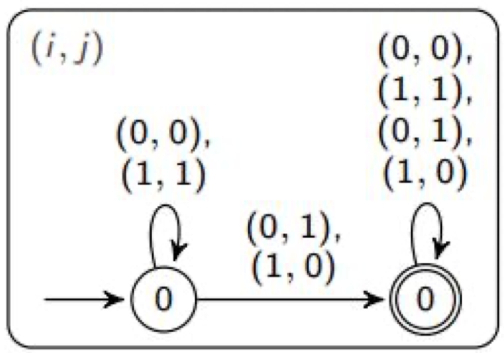
Multi-track automata can also represent Presburger arithmetic constraints

- Each track represents a single numeric variable
- Encoded as binary integers in 2's complement form

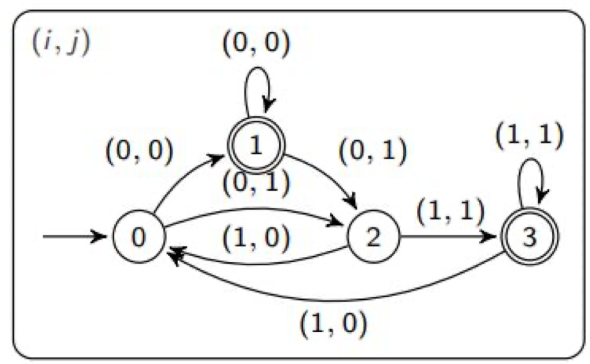
`i = j`



`i ≠ j`

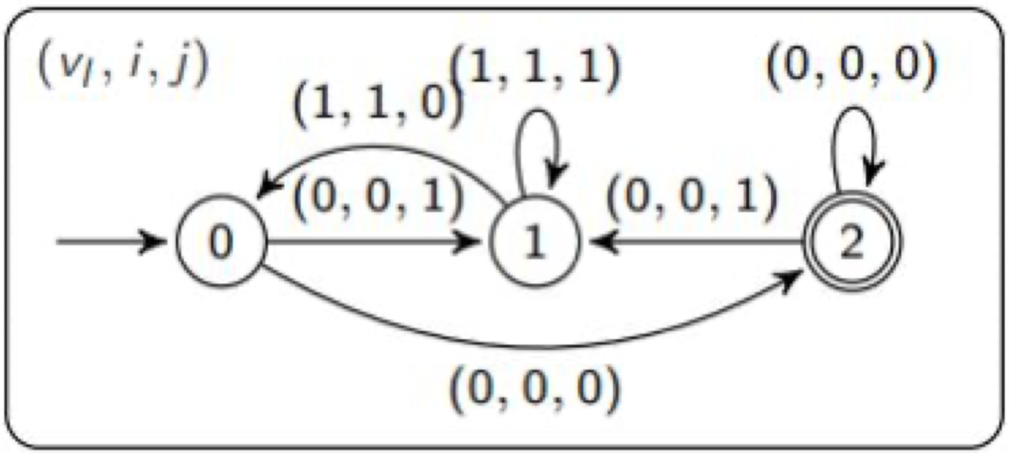


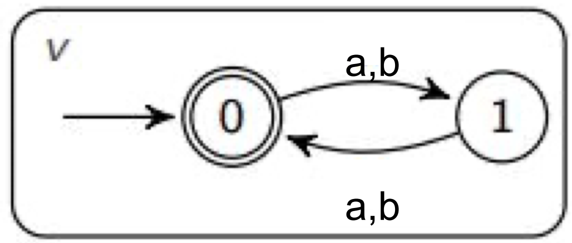
`i = 2×j`



Constraint Solving: Example

`i = 2×j ∧ length(v) = i ∧ match(v, (a | b)*)`

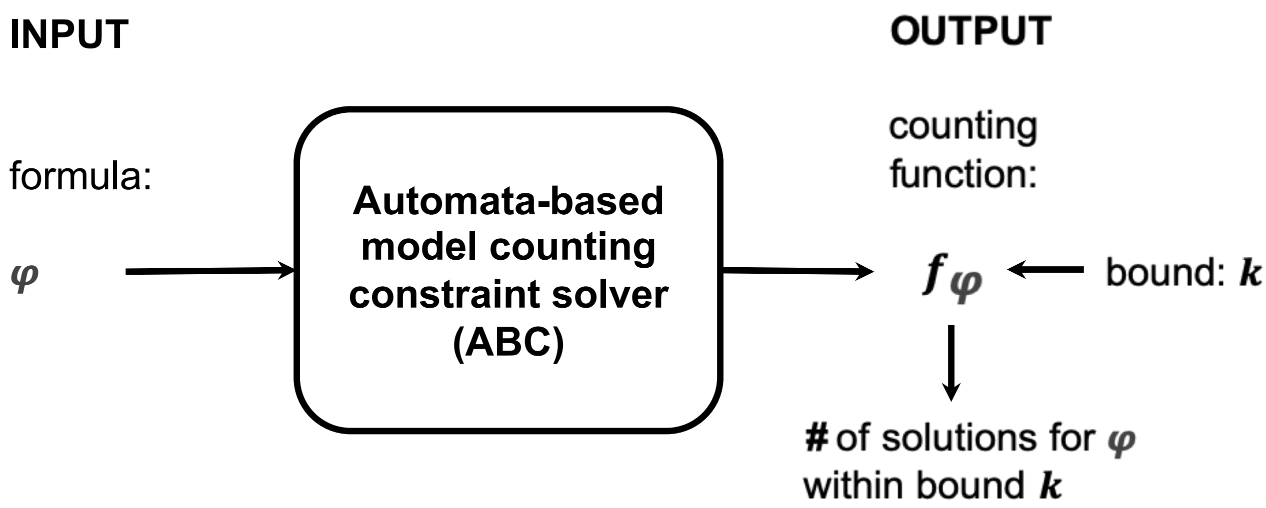




automaton for numeric variables
(v_l auxiliary variable encoding length of v)

automaton for string variables

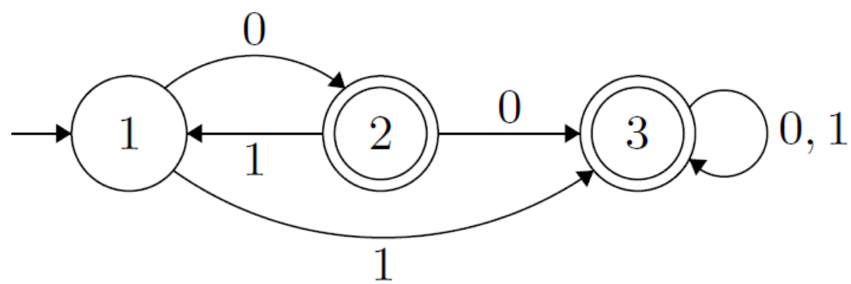
ABC: Model counting constraint solver



Automata-based model counting

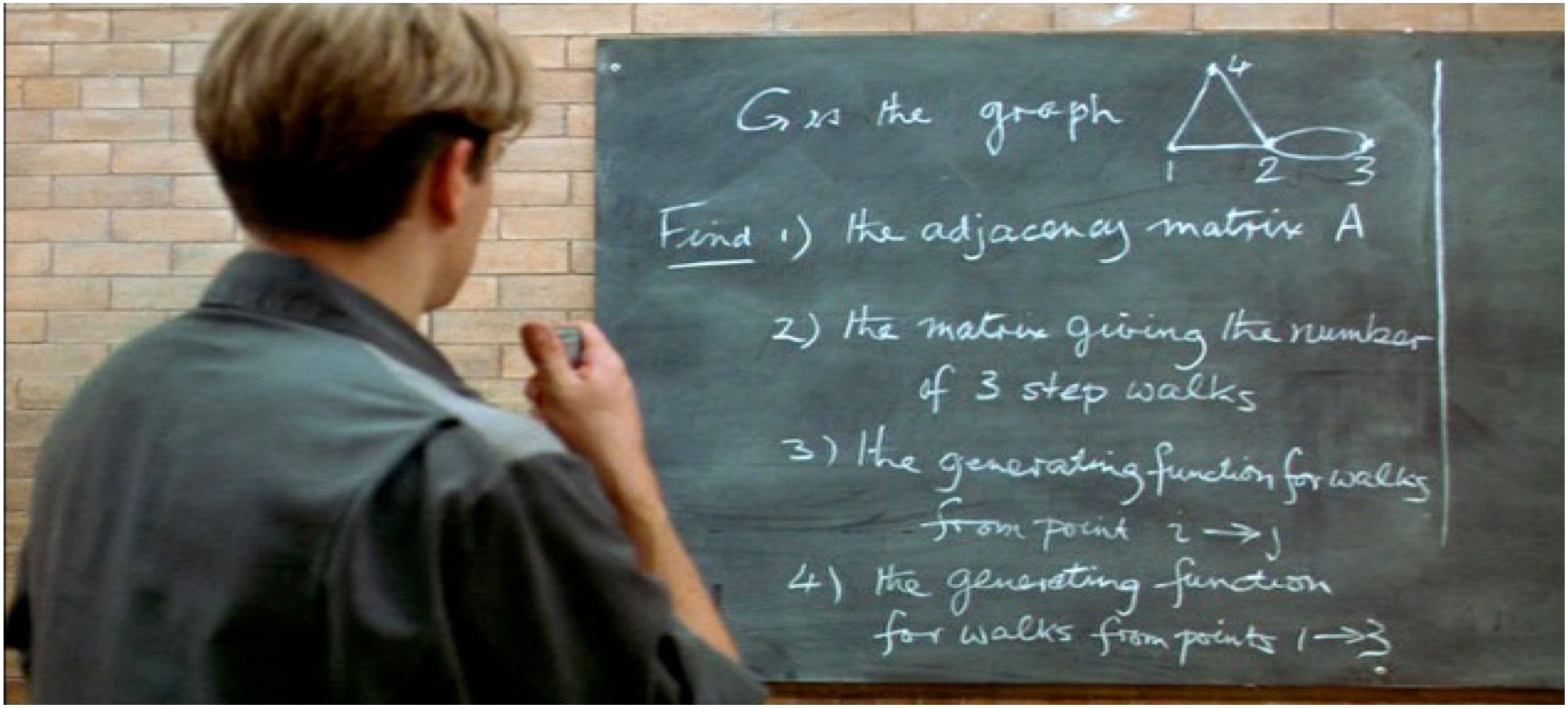
- Converting constraints to automata reduces the model counting problem to path counting problem in graphs

$C \equiv \neg(x \in (01)^*)$



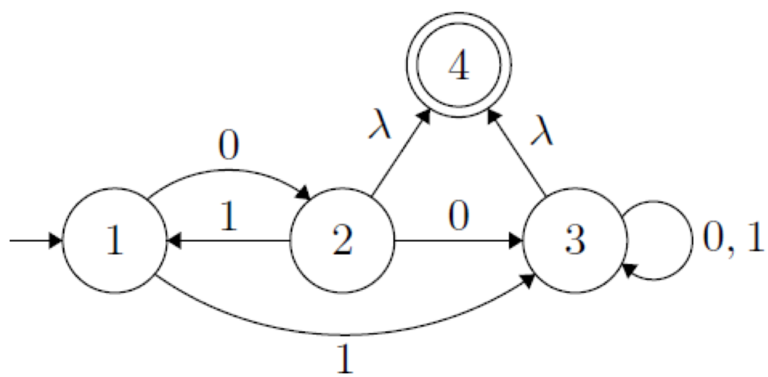
- We want to generate a function $f(k)$: Given length bound k, it will count the number of paths with length k.
  - $f(0) = 0,\ \{\}$
  - $f(1) = 2, \{0,1\}$
  - $f(2) = 3, \{00,10,11\}$

Can you count the paths Will Hunting?



Path Counting via Matrix Exponentiation

$$C = \neg(x \in (01)^*)$$



T: adjacency matrix for the automaton

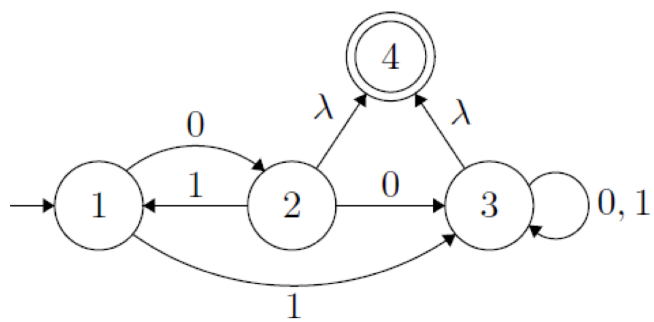
(i,j): number of edges from i to j

$$T = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, T^2 = \begin{bmatrix} 1 & 0 & 3 & 2 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}, T^3 = \begin{bmatrix} 0 & 1 & 7 & 3 \\ 1 & 0 & 7 & 4 \\ 0 & 0 & 8 & 4 \\ 0 & 0 & 0 & 0 \end{bmatrix}, T^4 = \begin{bmatrix} 0 & 1 & 15 & 8 \\ 1 & 0 & 15 & 7 \\ 0 & 0 & 16 & 8 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$f(0) = 0$ $f(1) = 2$ $f(2) = 3$ $f(3) = 8$

Counting Paths via Generating Functions

- We can compute a generating function, $g(z)$, for a DFA using the adjacency matrix



$$T = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$g(z) = (-1)^n \frac{\det(I - zT : n+1, 1)}{z \times \det(I - zT)} = \frac{2z - z^2}{1 - 2z - z^2 + 2z^3}$$

Counting Paths via Generating Functions

$$g(z) = \frac{2z - z^2}{1 - 2z - z^2 + 2z^3}$$

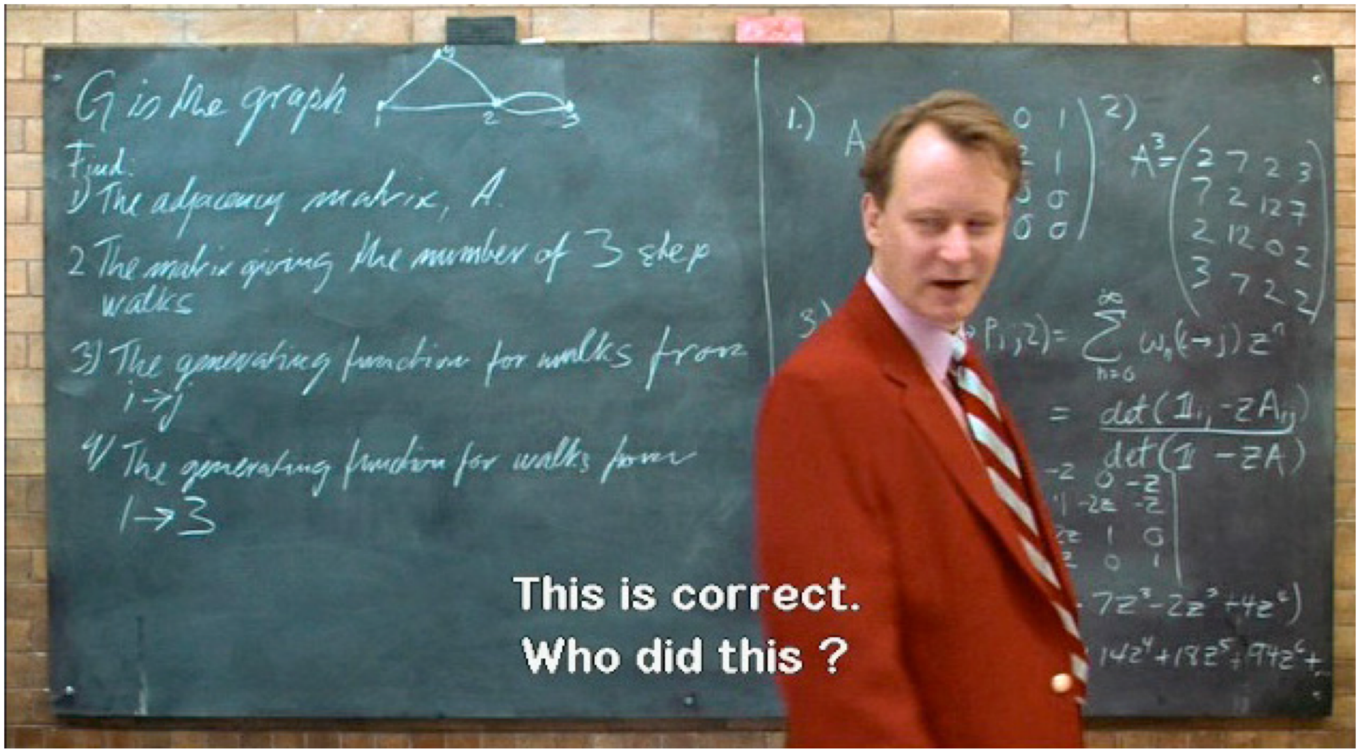
- Each $f(i)$ can be computed by Taylor expansion of $g(z)$

$$g(z) = \frac{g(0)}{0!}z^0 + \frac{g^{(1)}(0)}{1!}z^1 + \frac{g^{(2)}(0)}{2!}z^2 + \cdots + \frac{g^{(n)}(0)}{n!}z^n + \cdots$$

$$g(z) = 0z^0 + 2z^1 + 3z^2 + 8z^3 + 15z^4 + \cdots$$

$$g(z) = f(0)z^0 + f(1)z^1 + f(2)z^2 + f(3)z^3 + f(4)z^4 + \cdots$$

Good job Will Hunting!



ABC DEMO

http://drum.cs.ucsb.edu

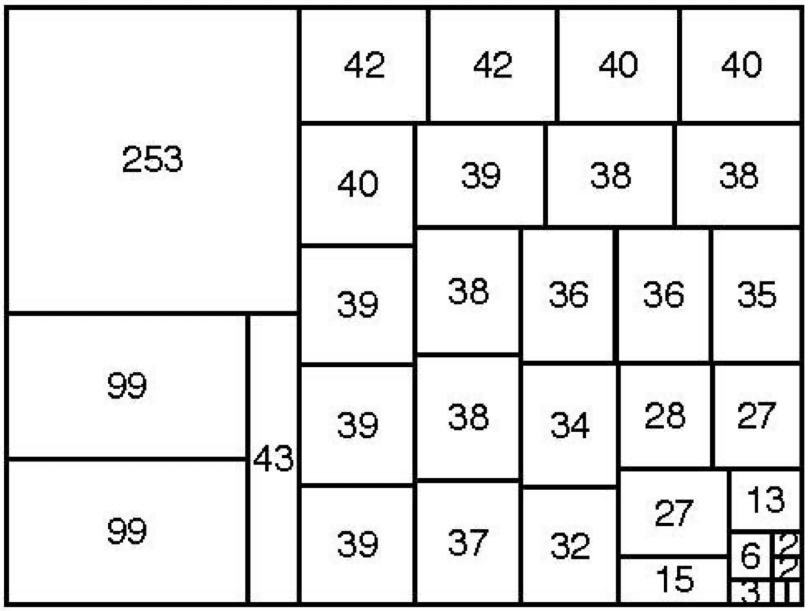
Automata-based model counting extensions

- In order to scale the automata-based model counting, it is necessary to cache the prior results

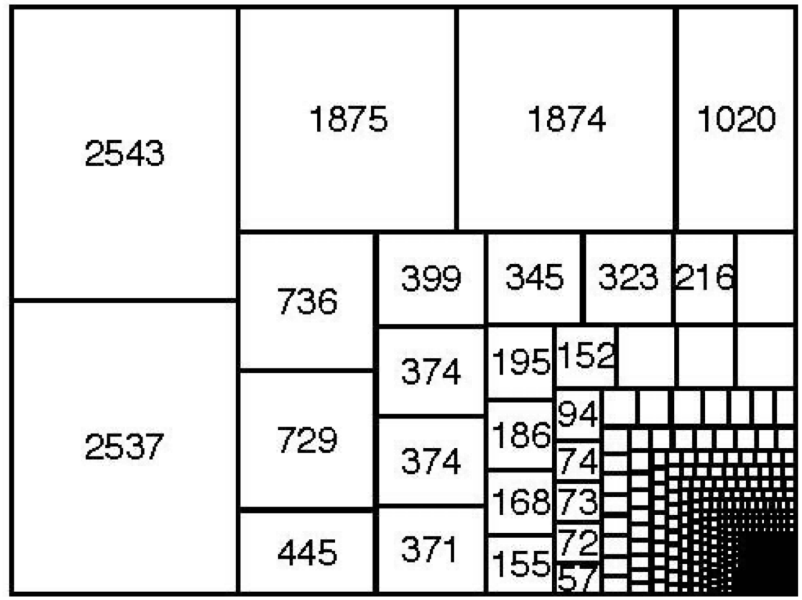
- Many constraints generated from programs are equivalent
  - By normalizing constraints we can identify many equivalent constraints

- 87X improvement for the Kaluza big data set

Kaluza Dataset:
1,342 big constraints and 17,554 small constraints



1,342 big constraints are reduced to 34 equivalent constraints after normalization



17,554 small constraints are reduced to 360 equivalent constraints after normalization

Outline

Information leakage and side channels

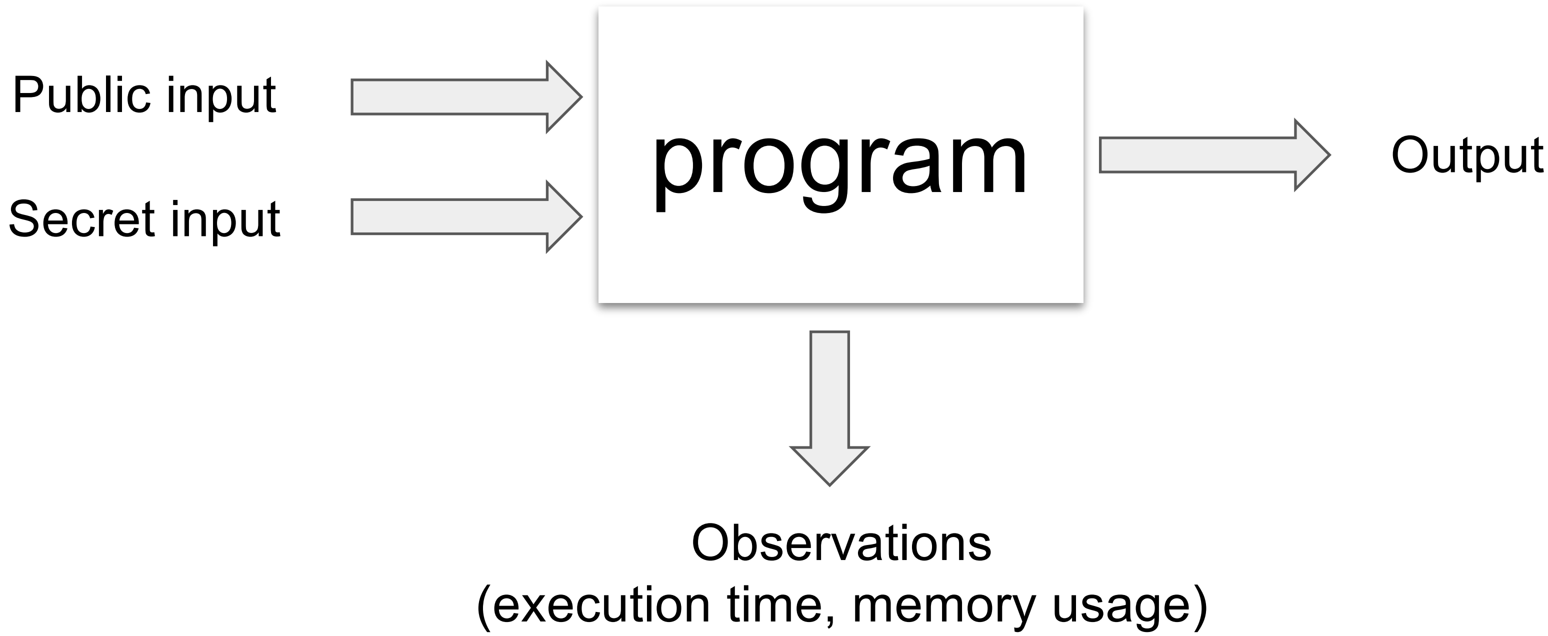
Quantifying information leakage

Side channel detection with probabilistic symbolic execution

Model counting

Attack synthesis

Can we automate attack synthesis?



- ***Which public input values would allow us to learn the secret as fast as possible?***

A Simple Function

```
public int comparison(int i) {

    if(s <= i)
        do something simple; // 1 milisecond
    else
        do something complex; // 2
    miliseconds

    return 0;
}
```

A Simple Function

```
public int comparison(int i) {

    if(s <= i)
        do something simple; // 1 milisecond
    else
        do something complex; // 2 miliseconds

    return 0;
}
```

$$O = 1 \Rightarrow s \leq i$$

$$O = 2 \Rightarrow s > i$$

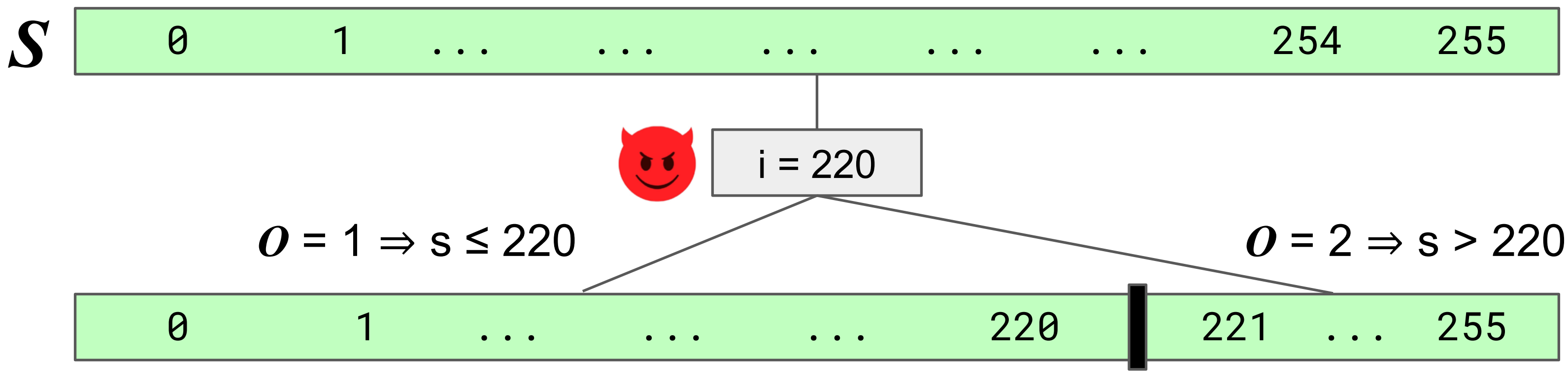
$$\boldsymbol{o} = 1 \Rightarrow s \leq i$$

$$\boldsymbol{o} = 2 \Rightarrow s > i$$

S	0	1	...	...	...	...	...	254	255

$O = 1 \Rightarrow s \leq i$

$O = 2 \Rightarrow s > i$



Attacker’s input and observation **partitions** domain of ***S***

How should the attacker
choose the inputs
to reveal the secret
as fast as possible?

$$o = 1 \Rightarrow s \leq i$$

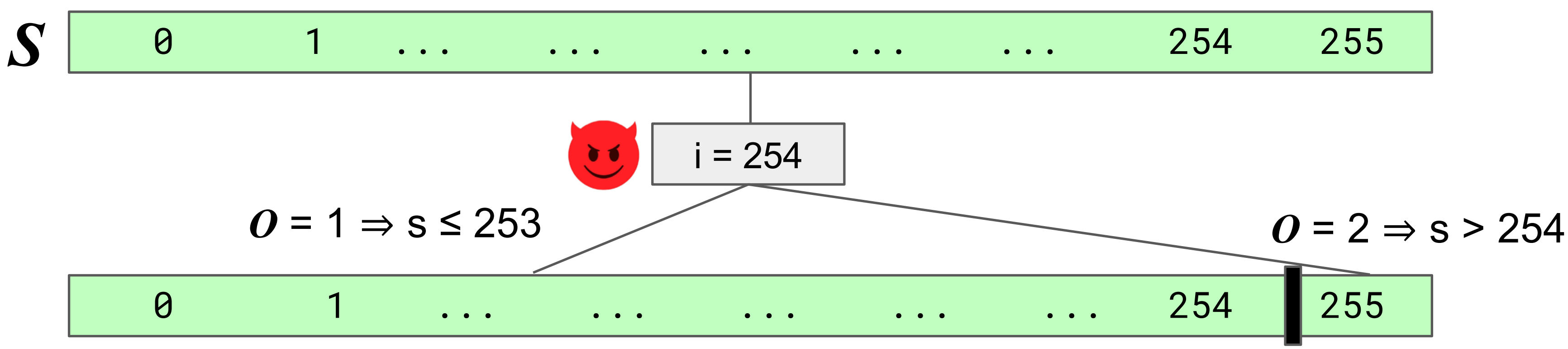
$$o = 2 \Rightarrow s > i$$

S

0	1	...	...	...	...	...	254	255

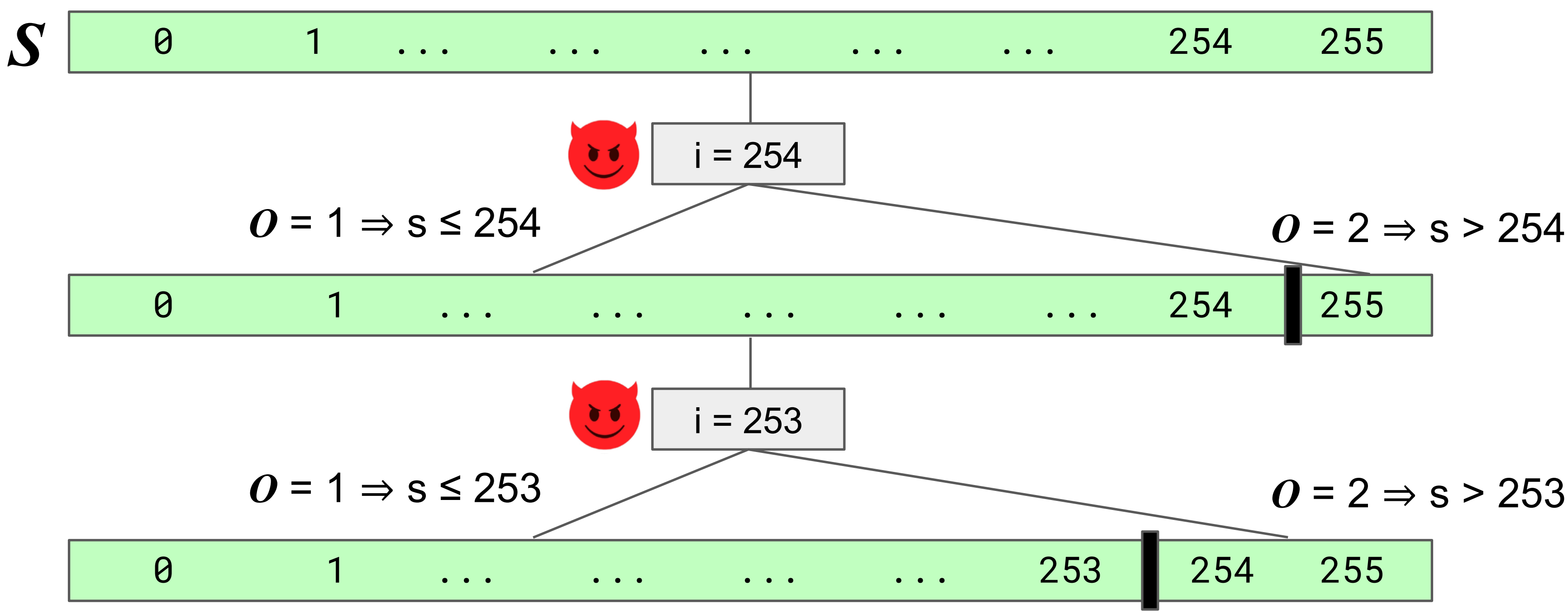
$\mathcal{O} = 1 \Rightarrow s \leq i$

$\mathcal{O} = 2 \Rightarrow s > i$



$O = 1 \Rightarrow s \leq i$

$O = 2 \Rightarrow s > i$



Imbalanced partitions

⇓

Worst case :

number of inputs = domain size = 2^8 = 256

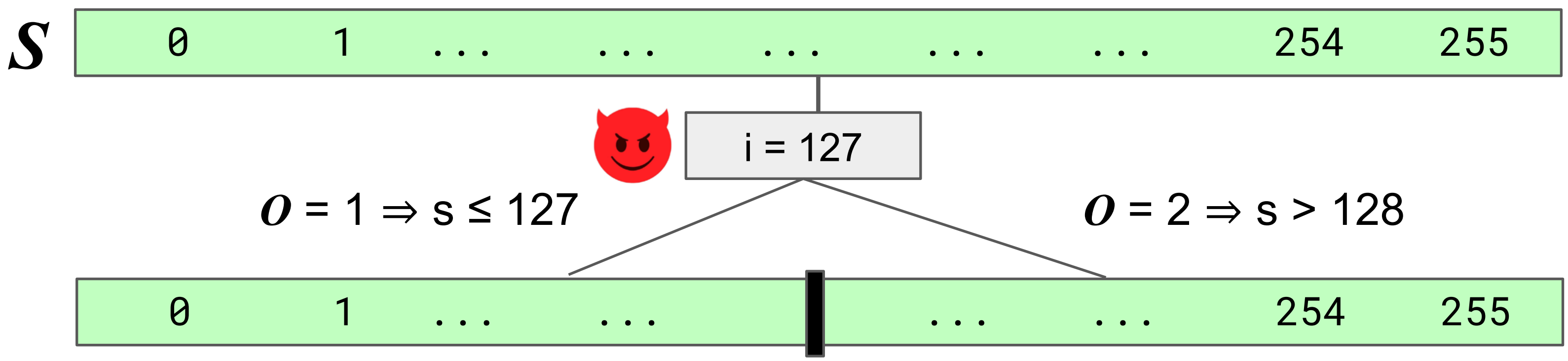
$$o = 1 \Rightarrow s \le i$$

$$o = 2 \Rightarrow s > i$$

S	0	1	...	...	...	...	...	254	255

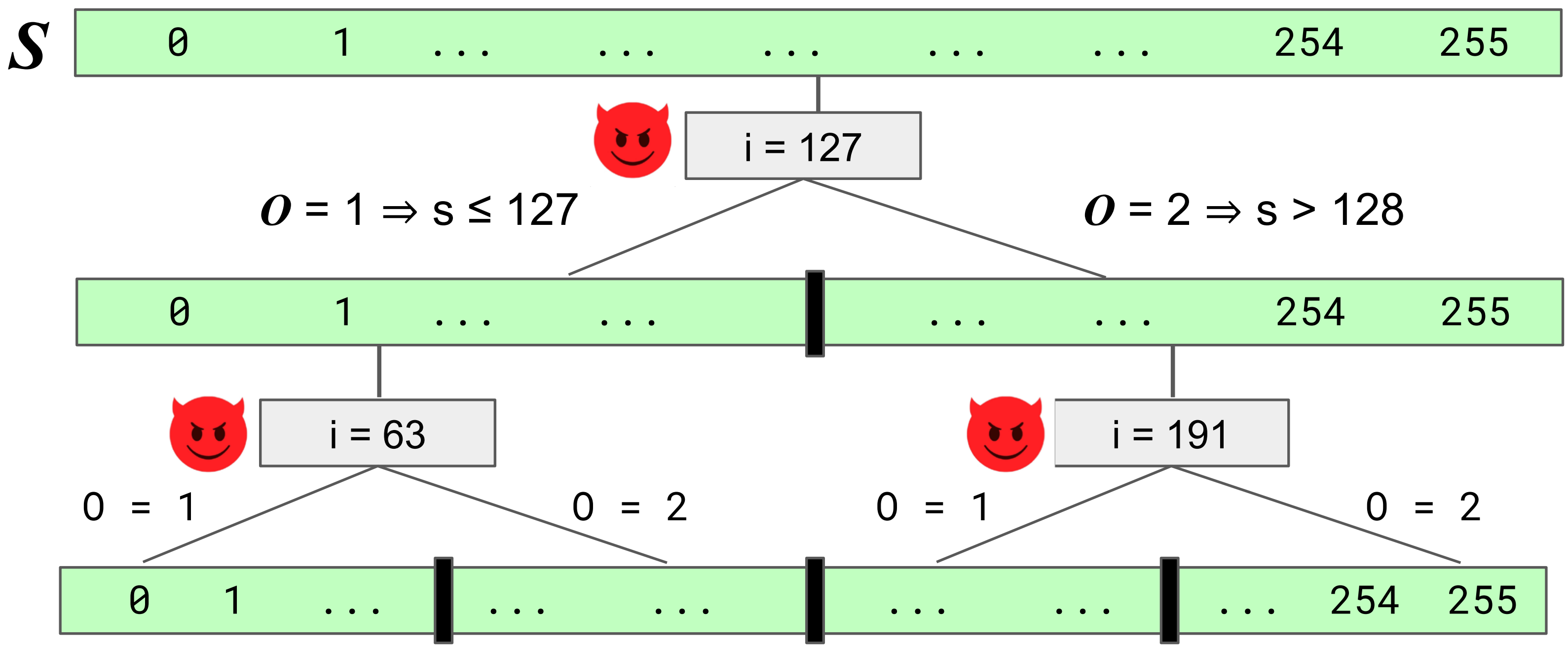
$\mathcal{O} = 1 \Rightarrow s \leq i$

$\mathcal{O} = 2 \Rightarrow s > i$



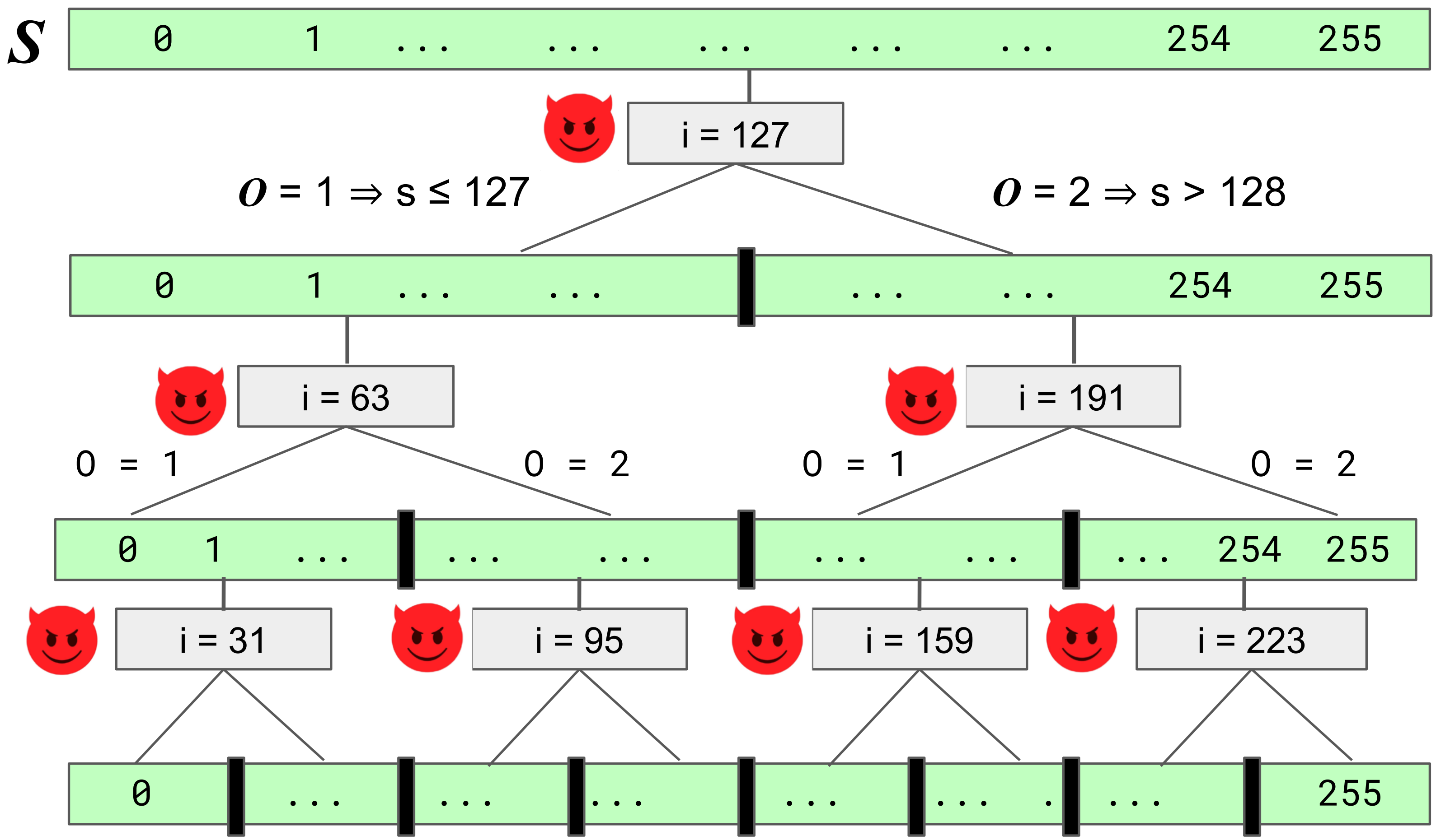
$O = 1 \Rightarrow s \leq i$

$O = 2 \Rightarrow s > i$



$o = 1 \Rightarrow s \leq i$

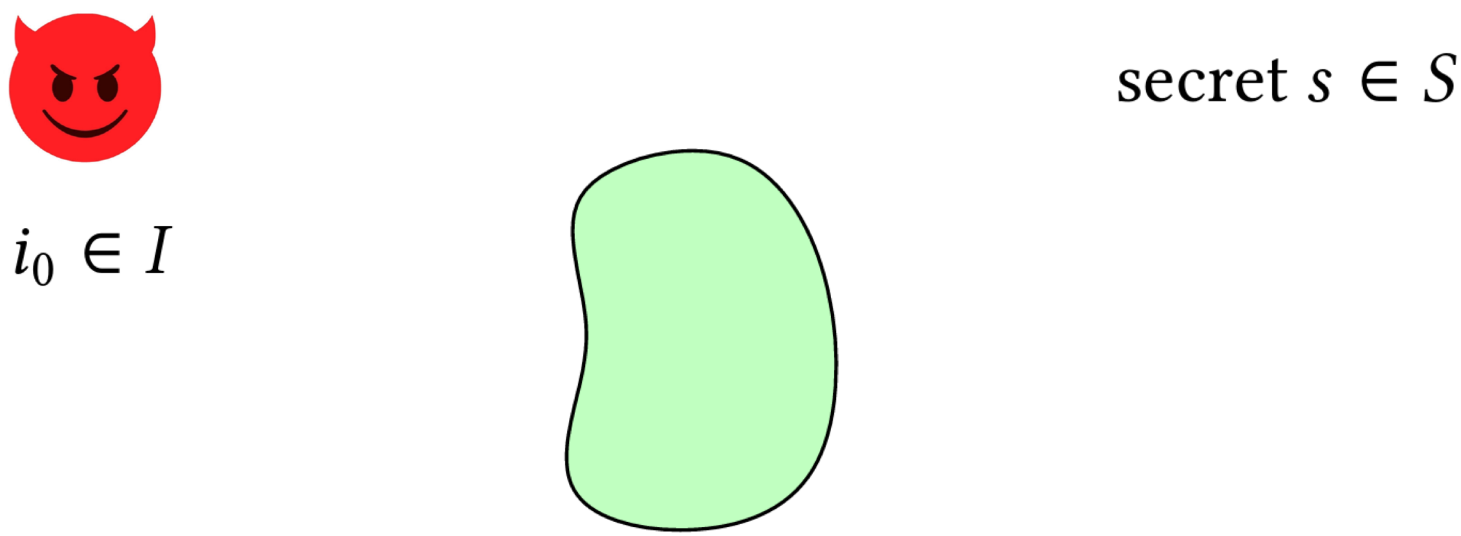
$o = 2 \Rightarrow s > i$



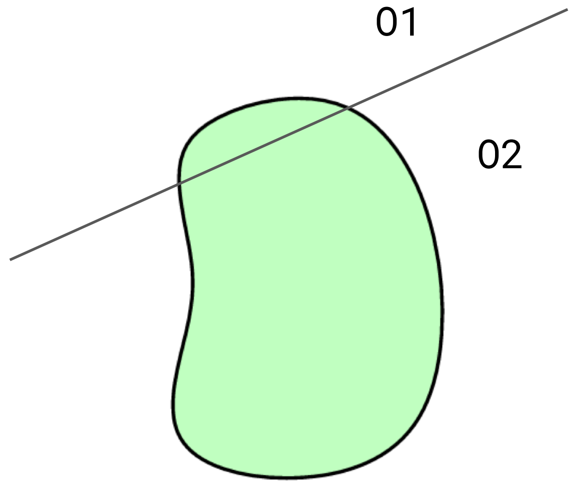
Balanced partitions

Worst case :

number of inputs = $\log_2(256) = 8$


$i_0 \in I$
secret $s \in S$

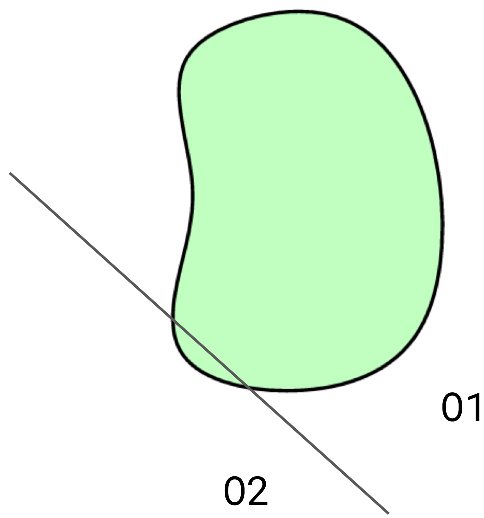
$i_0 \in I$



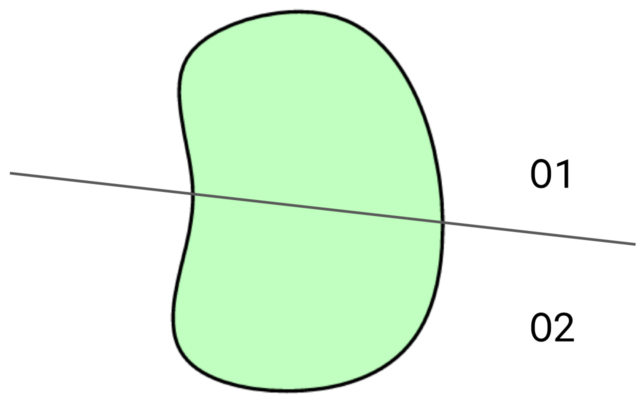
secret $s \in S$

$i_0 \in I$

secret $s \in S$



$i_0 \in I$

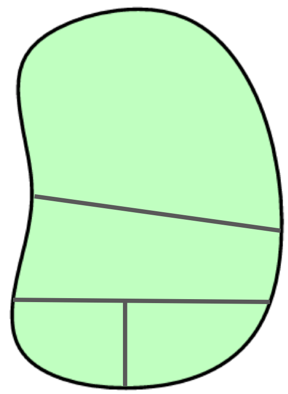


secret $s \in S$

$i_0 \in I$

$i_1 \in I$

$i_2 \in I$



secret $s \in S$

Objective Function

Balanced partitions

⇩

Maximizes information gain

Objective Function

$$\boldsymbol{o} = 1 \Rightarrow s \leq i$$
$$\boldsymbol{o} = 2 \Rightarrow s > i$$

Maximize information gain ⇒ Binary Search

Objective Function

$$\boldsymbol{o} = 1 \Rightarrow s \leq i$$
$$\boldsymbol{o} = 2 \Rightarrow s > i$$

Maximize information gain ⇒ Binary Search

Programs in general

Maximize information gain ⇒ Optimal Search

Objective Function

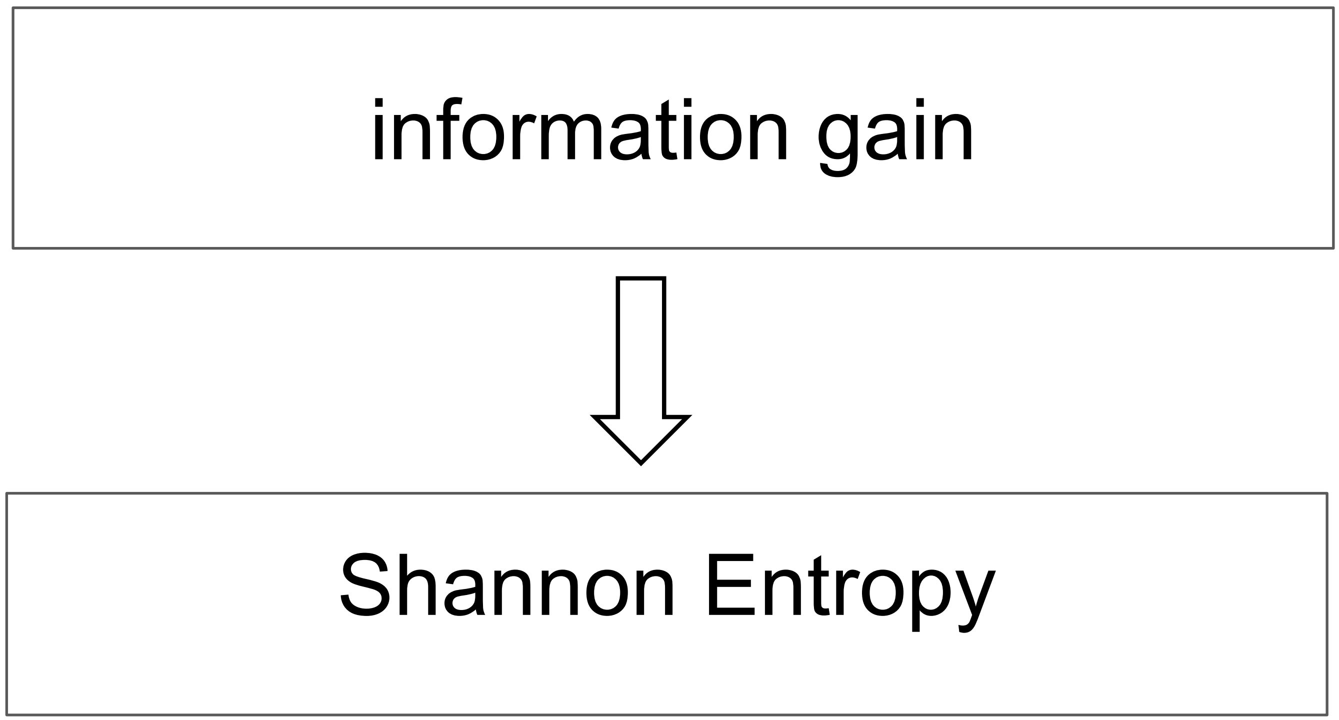
$$\mathcal{O} = 1 \Rightarrow s \le i$$
$$\mathcal{O} = 2 \Rightarrow s > i$$

Maximize information gain ⇒ Binary Search

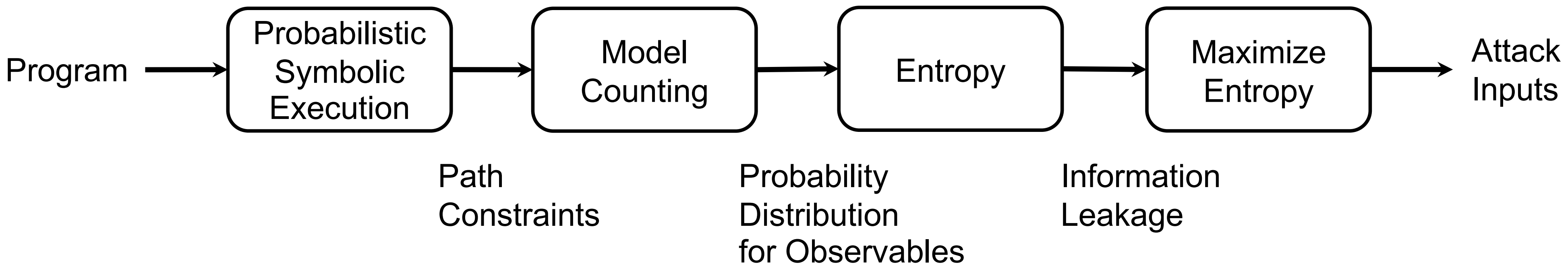
Programs in general

Maximize information gain ⇒ Optimal Attack

Objective Function



Attack synthesis summary



- The attacks that are synthesized are ***adaptive attacks***
  - Each attack step depends on the results of previous steps
- How to find the input value that maximizes the entropy?
  - Use meta-heuristics such as simulated annealing or genetic algorithm

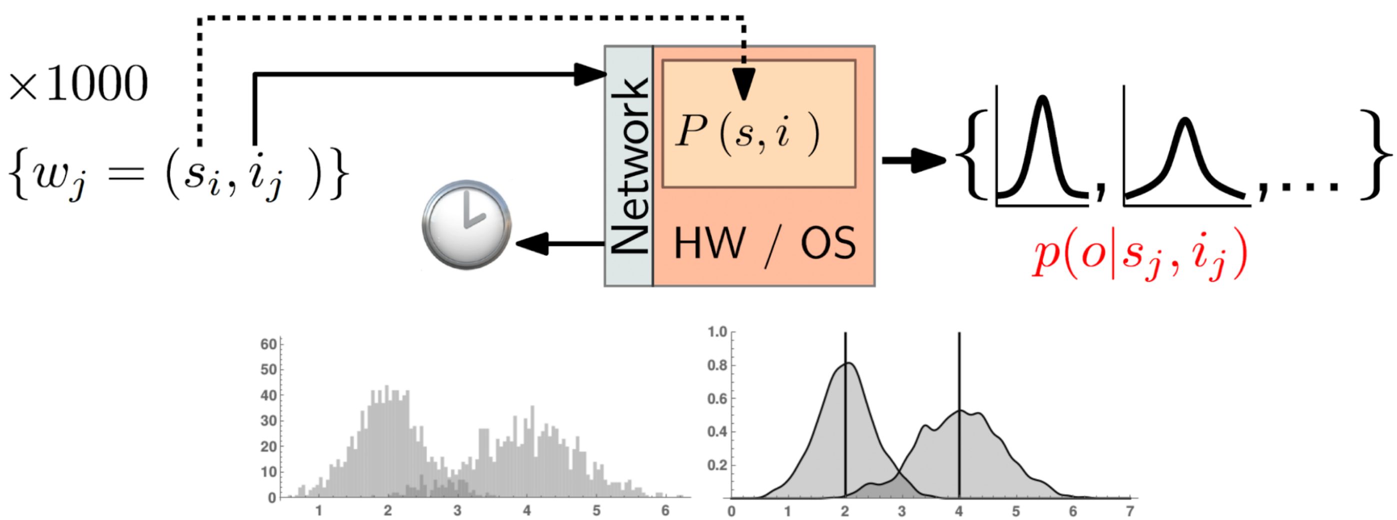
Attack synthesis extensions: Online attack synthesis

- Generating the full attack tree is expensive
- A full attack tree provides all public input sequences for all possible secret values
  - Full attack tree can be computed offline
  - Exponential blow up with attack depth

- Use online attack synthesis
  - Compute the attack on the fly for a single secret

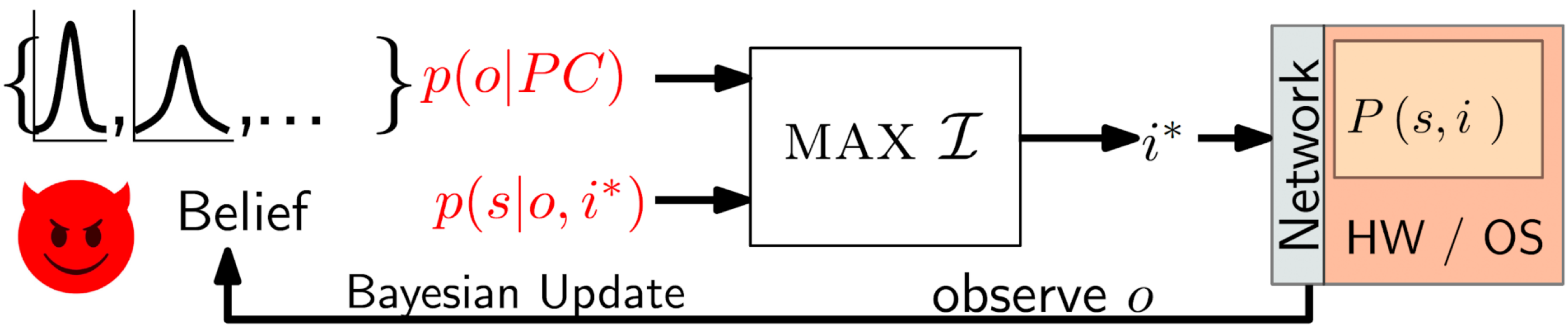
Attack synthesis extensions: Noise modeling

- Use profiling to model the noise
  - Use a witness (a satisfying solution) for each path constraint to profile the observable distribution
  - Generate a noise distribution using smooth kernel density estimation



Attack synthesis extensions: Online attack synthesis

- During attack synthesis, use a probability distribution to model the current belief about the secret
- Use Bayesian inference to update the probability distribution for the secret based on the observations and the noise model



Automatically generated prefix attack against a vulnerable password checker

Phase 0			Phase 1		Phase 2					Phase 3					Phase 4
prefix $= \varepsilon$			prefix = c		prefix = ci					prefix = ciq					prefix = ciqa
ε	fzgk	maau	**c**nte	**c**ved	**ci**ub	**ci**ij	**ci**mq	**ci**tz	**ciq**z	**ciq**i	**ciq**z	**ciq**z	**ciq**u	**ciq**z	**ciqa**
daaz	zgap	vzsc	**c**tdo	**ci**il	**ci**az	**ci**ok	**ci**da	**ci**jw	**ci**hs	**ciq**c	**ciq**z	**ciq**e	**ciq**r	**ciq**r	**ciqa**
uaak	bnza	qyas	**c**vfo	**c**eyu	**ci**gz	**ci**su	**ci**sp	**ci**ne	**ciq**k	**ciq**k	**ciq**d	**ciq**d	**ciq**r	**ciq**z	**ciq**g
ecjq	zmna	asvr	**c**sja	**ci**vf	**ci**fl	**ci**ld	**ci**cz	**ci**le	**ci**eb	**ciq**z	**ciq**q	**ciq**o	**ciq**i	**ciqa**	**ciqa**
tzar	zmna	**c**mxq	**c**wcs		**ci**kt	**ci**pa	**ci**bn	**ci**rx	**ciqa**	**ciq**s	**ciq**z	**ciq**x	**ciq**v		

Secret is "**ciqa**"
Matching characters are shown in **bold**

A case study from DARPA STAC Program: LawDB

- A web service with a law enforcement database that contains
  - Restricted (secret) & unrestricted (public) employee IDs

- Supports SEARCH & INSERT queries
  - Restricted IDs are not visible during SEARCH and INSERT queries

- **Question**: Is there a side channel in time that a third party can determine the value of a single restricted ID in the database?

Code Inspection

- Using code inspection we identified that the SEARCH and INSERT operations are implemented in:

```
class UDPServerHandler
method channelRead0
switch case 1: INSERT
switch case 8: SEARCH
```

Symbolic Path Finder Driver

```
public class Driver {
       public static void main(String[] args){
          BTree tree = new BTree(10);
          CheckRestrictedID checker = new CheckRestrictedID();
          // create two concrete unrestricted ids
          int id1 = 64, id2 = 85;
          tree.add(id1, null, false);
          tree.add(id2, null, false);
          // create one symbolic restricted id
          int h = Debug.makeSymbolicInteger("h");
          Debug.assume(h!=id1 && h!=id2);
          tree.add(h, null, false);
          checker.add(h);
          UDPServerHandler handler = new
UDPServerHandler(tree,checker);
          int key = Debug.makeSymbolicInteger("key");
          handler.channelRead0(8,key);  // send a search query with
       }                                // with search range 50 to 100
}
```

SPF Output

```
>>>>> There are 5 path conditions and 5 observables
cost: 9059
(assert (<= h 100))
(assert (> h 85))
(assert (> h 64))
(assert (not (= h 85)))
(assert (not (= h 64)))
Count = 15
-----------------------
cost: 8713
(assert (<= h 85))
(assert (> h 64))
(assert (not (= h 85)))
(assert (not (= h 64)))
Count = 20
-----------------------
cost: 7916
(assert (> h 100))
(assert (> h 85))
(assert (> h 64))
(assert (not (= h 85)))
(assert (not (= h 64)))
Count = 923
-----------------------
cost: 8701
(assert (>= h 50))
(assert (<= h 64))
(assert (not (= h 85)))
(assert (not (= h 64)))
Count = 14
-----------------------
cost: 7951
(assert (< h 50))
(assert (<= h 64))
(assert (not (= h 85)))
(assert (not (= h 64)))
Count = 50
-----------------------
************************************************************
PC equivalence class model counting results.
************************************************************
Cost: 9059     Count:      15  Probability:  0.014677
Cost: 8713     Count:      20  Probability:  0.019569
Cost: 7916     Count:     923  Probability:  0.903131
Cost: 8701     Count:      14  Probability:  0.013699
Cost: 7951     Count:      50  Probability:  0.048924

Domain Size: 1022
Single Run Leakage: 0.6309758112933285
```

Observation & Proposed Attack

- SEARCH operation:

takes longer when the secret is within the search range (9059, 8713, 8701 byte code instructions)

as opposed to the case when the secret is out of the search range (7916, 7951 byte code instructions)

- ***Proposed attack***: Measure the time it takes for the search operation to figure out if there is a secret within the search range

Proposed Attack

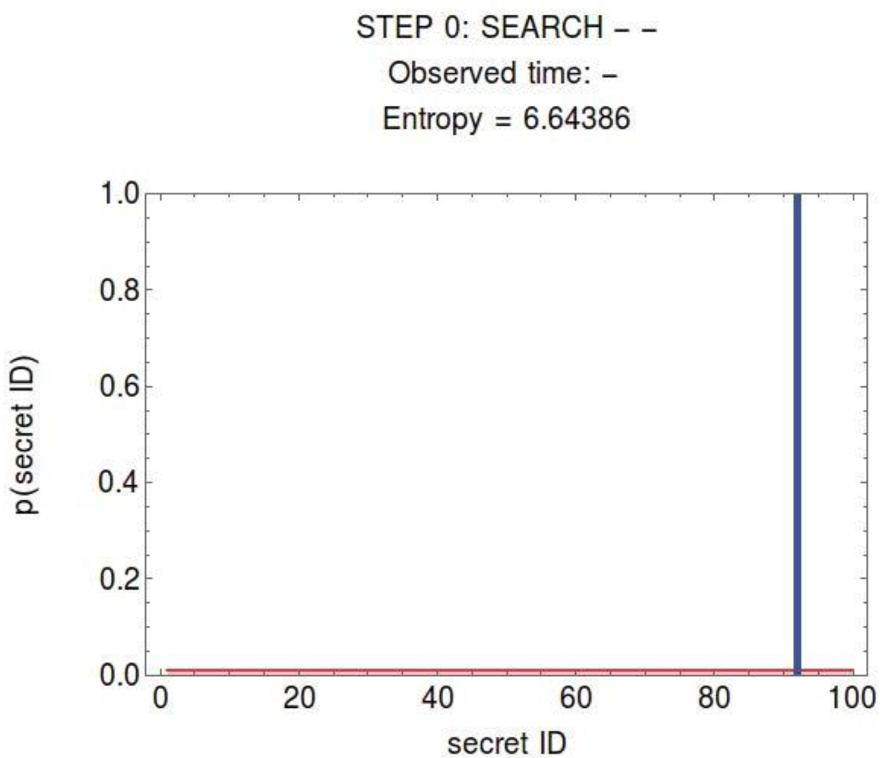
- Binary search on the ranges of the IDs
- Send two search queries at a time and compare their execution time
- Refine the search range based on the result

Attack

```
Running [0, 40000000] at 0.
Comparing 467821 vs 612252...
Running [20000000, 40000000] at 2.
Comparing 400377 vs 333665...
Running [20000000, 30000000] at 4.
Comparing 200603 vs 237025...
Running [25000000, 30000000] at 6.
Comparing 163564 vs 115072...
Running [25000000, 27500000] at 8.
Comparing 95736 vs 37388...
Running [25000000, 26250000] at 10.
Comparing 85305 vs 30118...
Running [25000000, 25625000] at 12.
Comparing 22765 vs 72958...
Running [25312500, 25625000] at 14.
Comparing 2147483647 vs 19353...
Running [25312500, 25468750] at 16.
Comparing 517 vs 2147483647...
Running [25390625, 25468750] at 18.
Comparing 317 vs 2147483647...
Running [25429687, 25468750] at 20.
Comparing 2147483647 vs 302...
Running [25429687, 25449218] at 22.
Comparing 2147483647 vs 287...
Running [25429687, 25439452] at 24.
Comparing 336 vs 2147483647...
Running [25434569, 25439452] at 26.
Comparing 300 vs 2147483647...
Running [25437010, 25439452] at 28.
Comparing 2147483647 vs 265...
Running [25437010, 25438231] at 30.
Comparing 2147483647 vs 328...
Running [25437010, 25437620] at 32.
Comparing 280 vs 2147483647...
Running [25437315, 25437620] at 34.
Comparing 293 vs 2147483647...
Running [25437467, 25437620] at 36.
Comparing 2147483647 vs 281...
Running [25437467, 25437543] at 38.
Comparing 2147483647 vs 613...
Running [25437467, 25437505] at 40.
Comparing 2147483647 vs 258...
Running [25437467, 25437486] at 42.
Comparing 2147483647 vs 291...
Running [25437467, 25437476] at 44.
Comparing 362 vs 2147483647...
Running [25437471, 25437476] at 46.
Comparing 311 vs 2147483647...
Running [25437473, 25437476] at 48.
Comparing 2147483647 vs 2147483647...
Checking oracle for: 25437474... true
Checking oracle for: 25437475... false
```

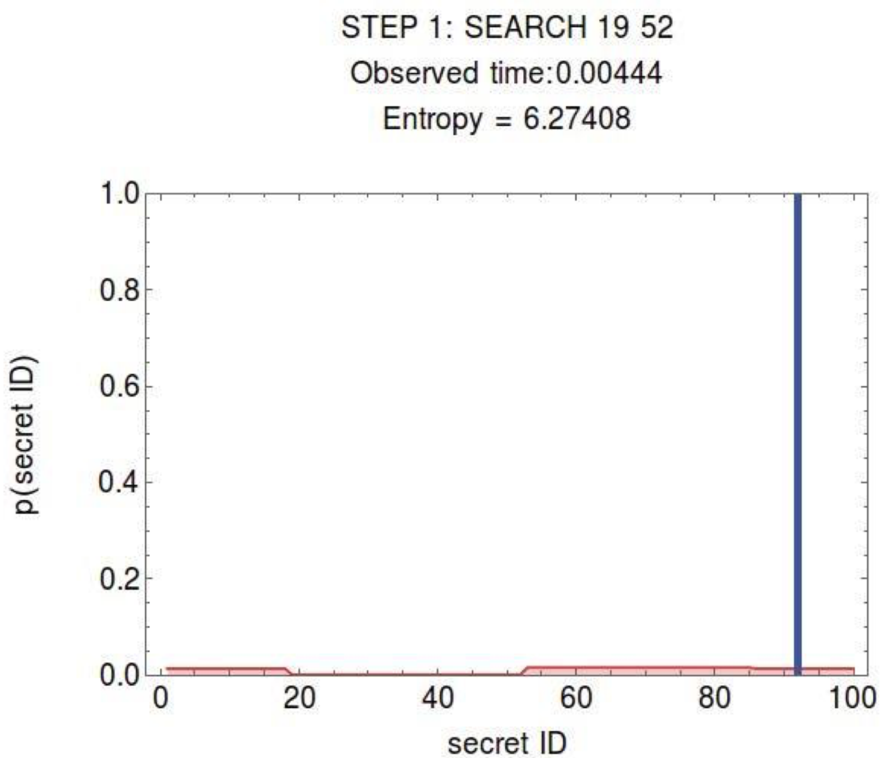
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



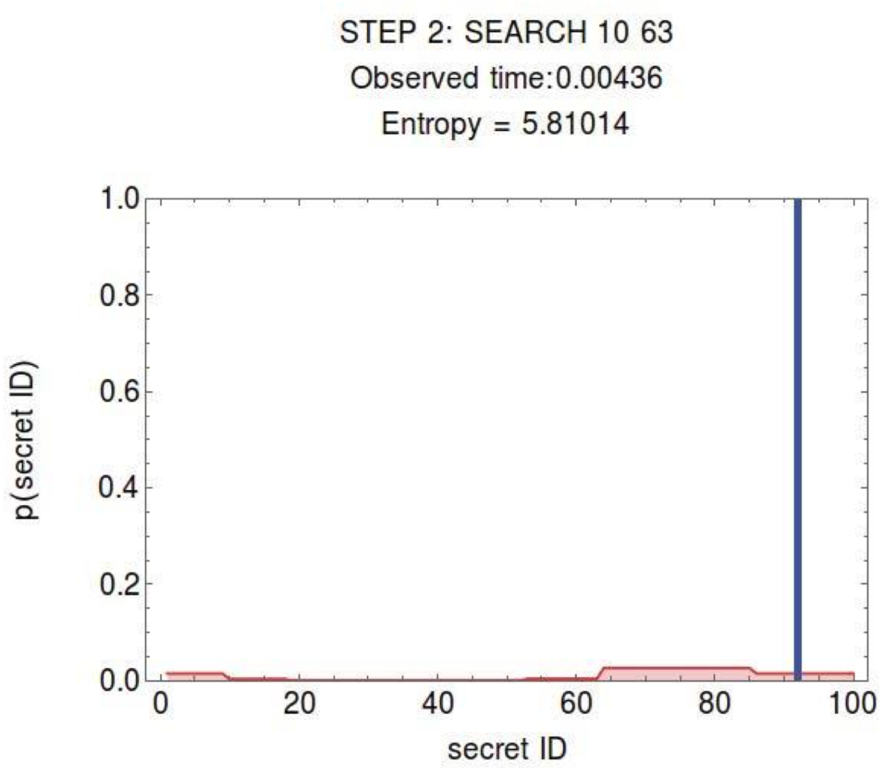
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



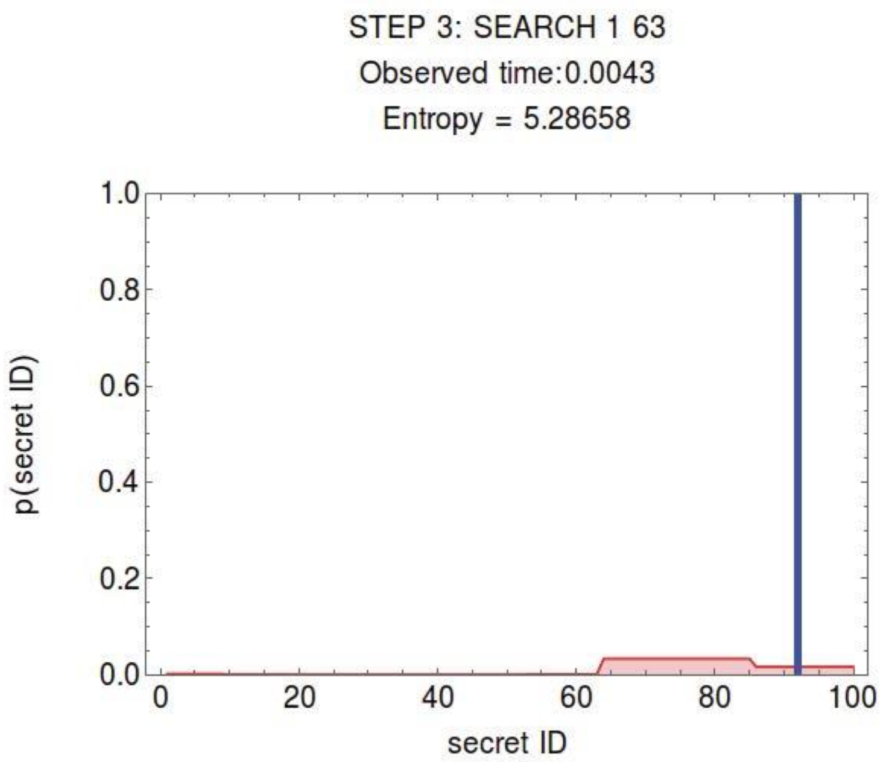
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



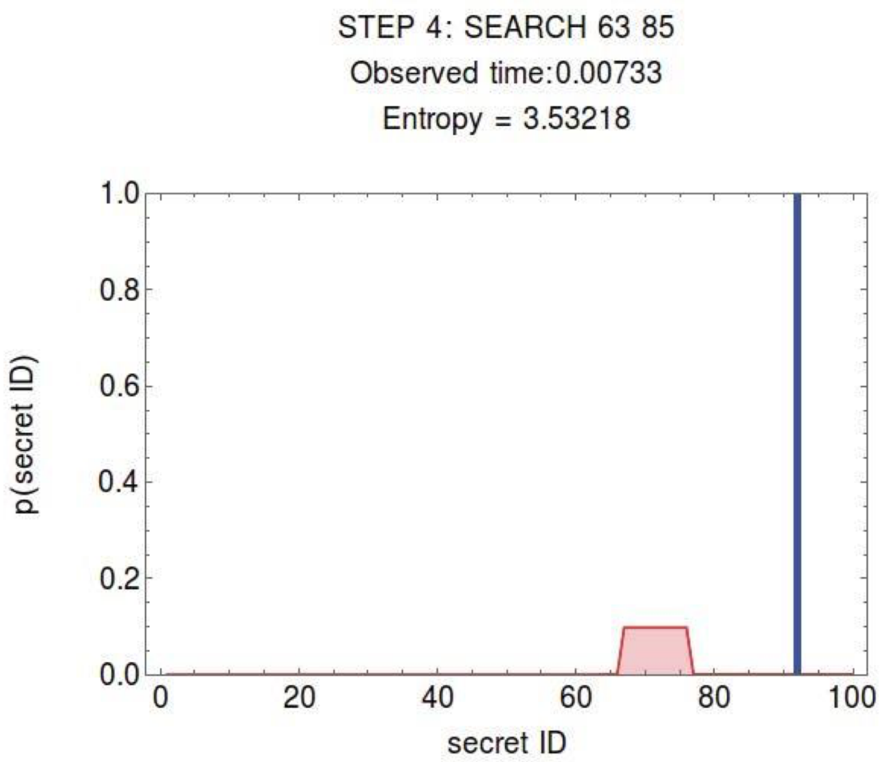
Automatically generated attack against LawDB

$$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$$



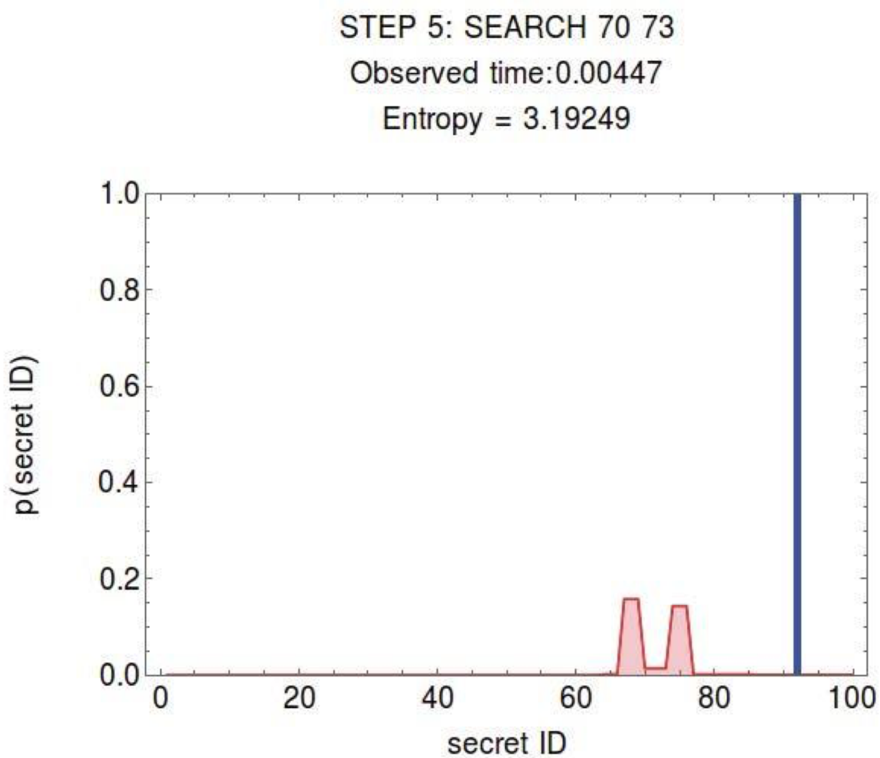
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



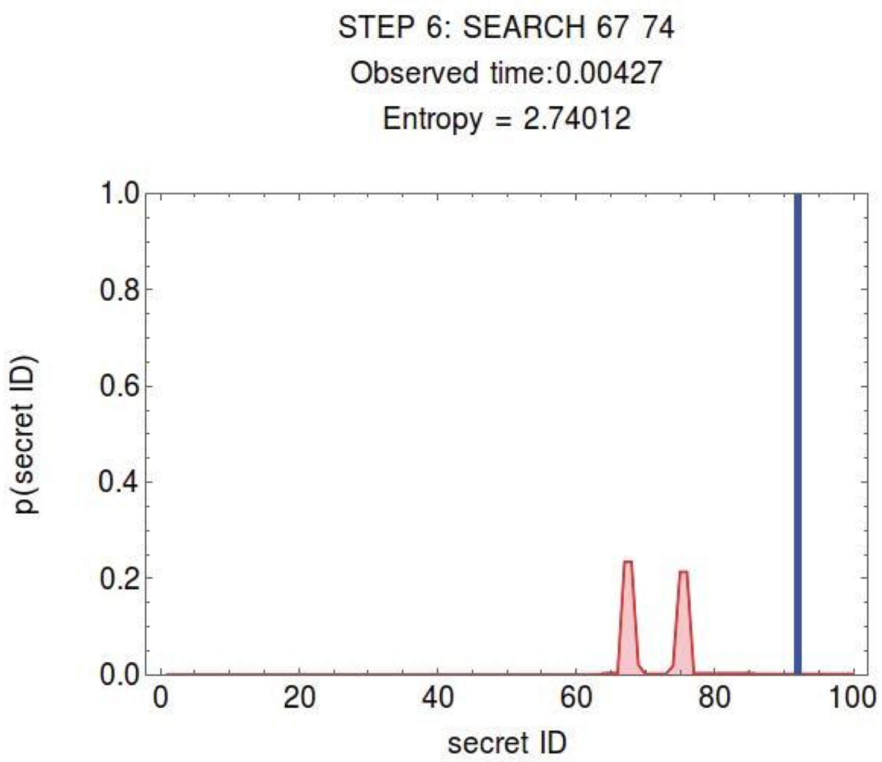
Automatically generated attack against LawDB

$$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$$



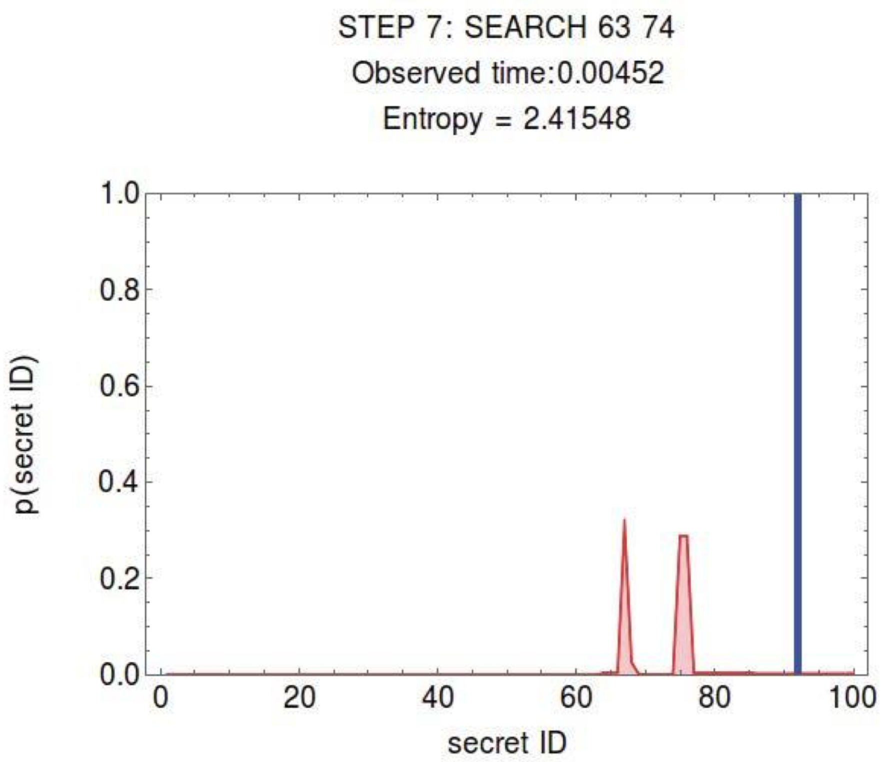
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



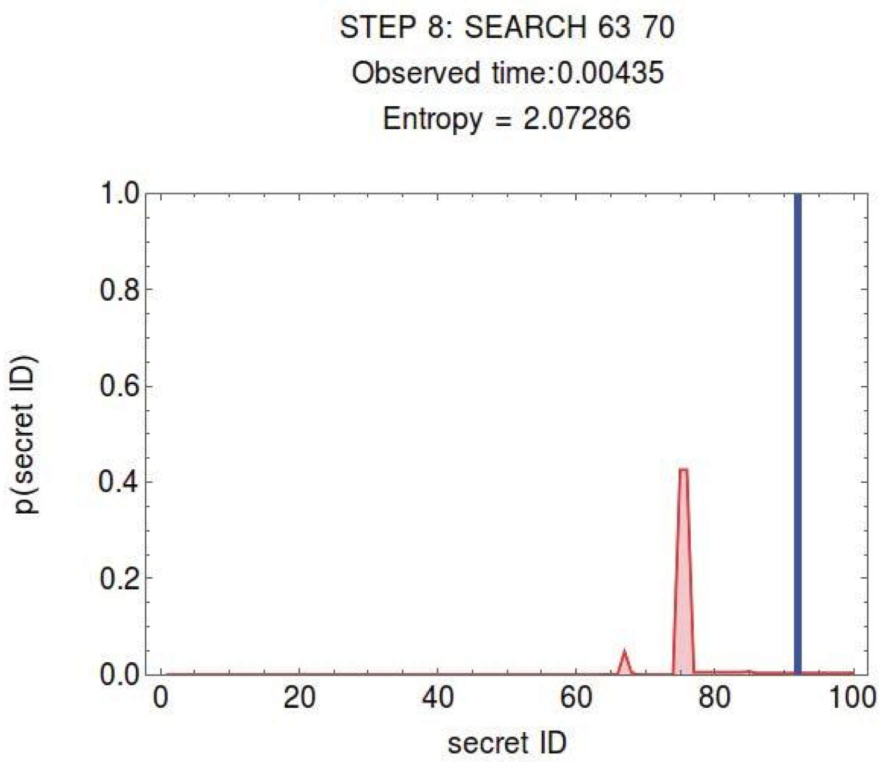
Automatically generated attack against LawDB

$$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$$



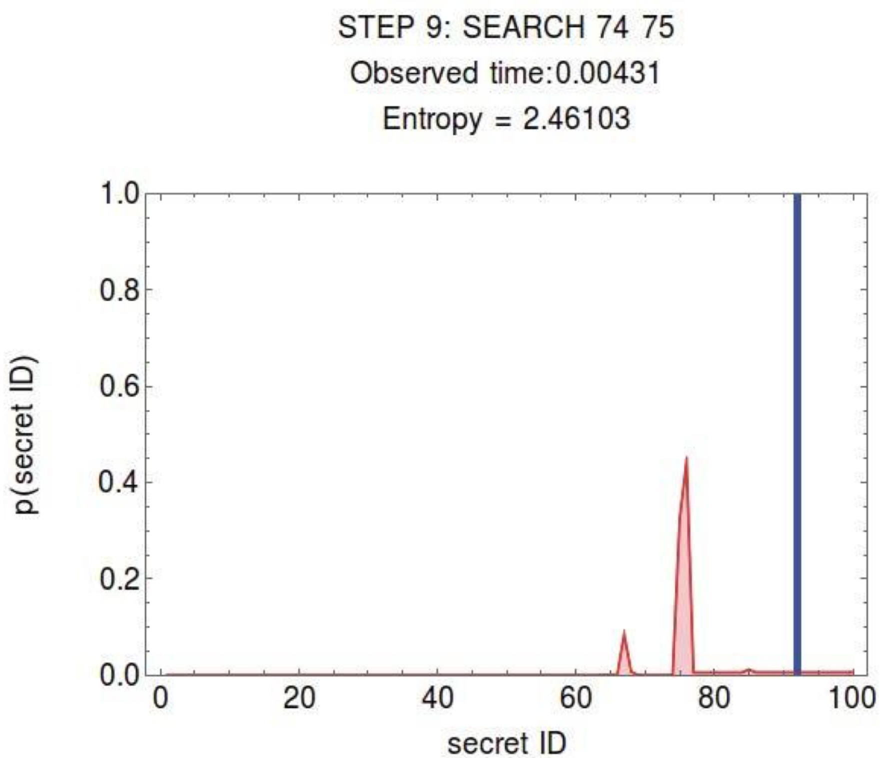
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



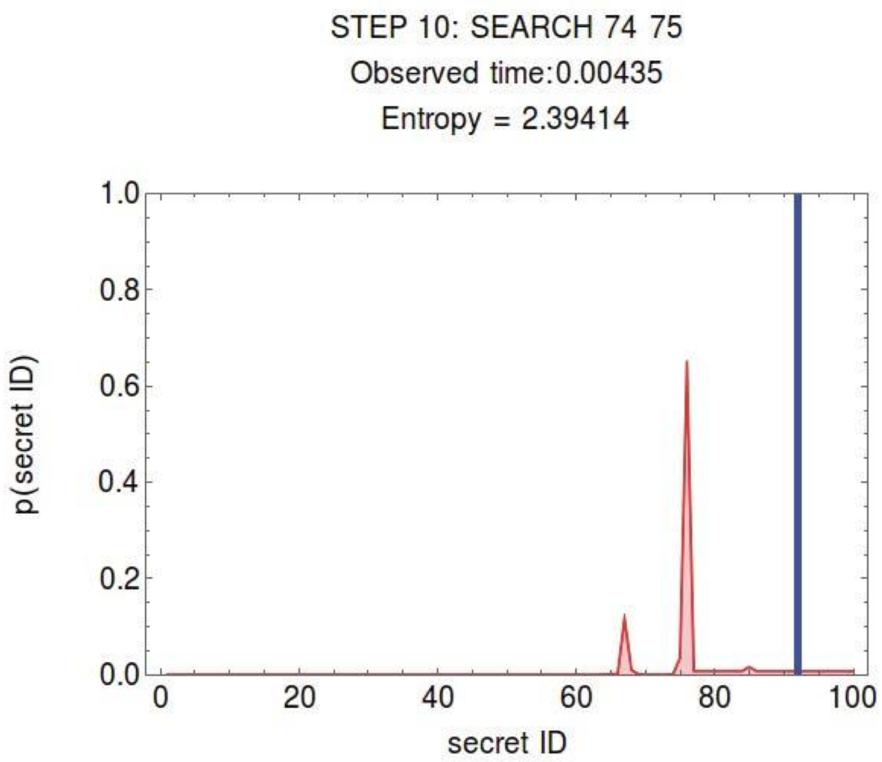
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



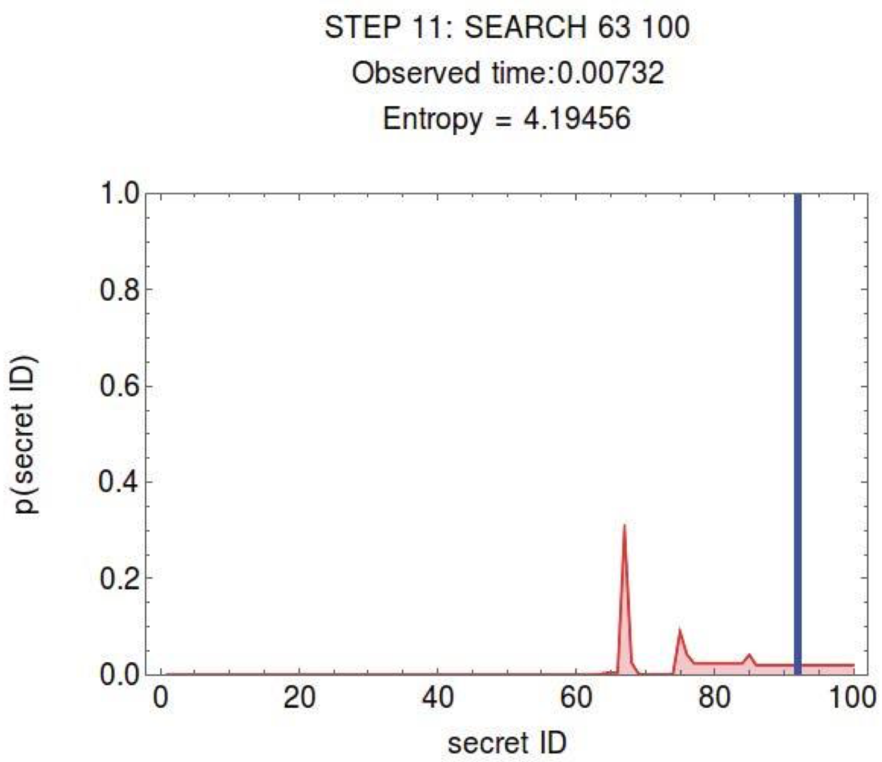
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



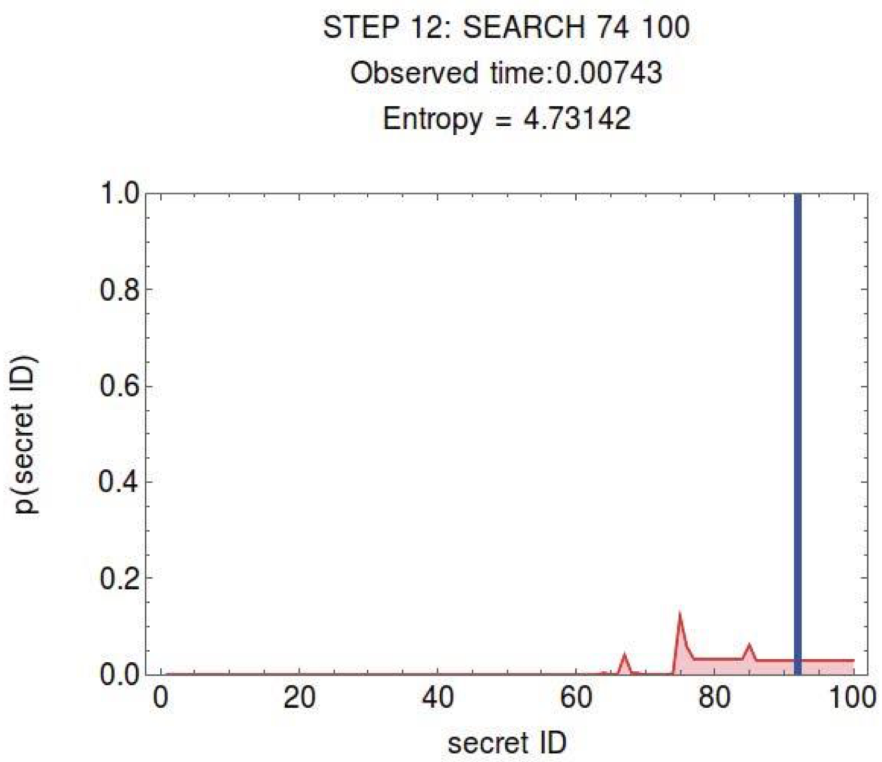
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



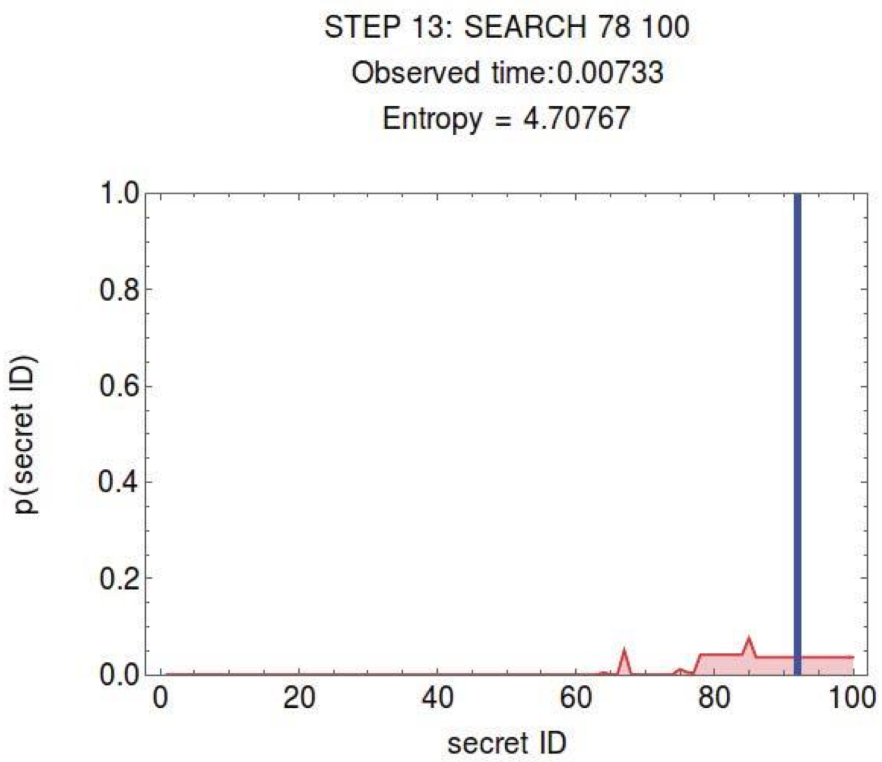
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



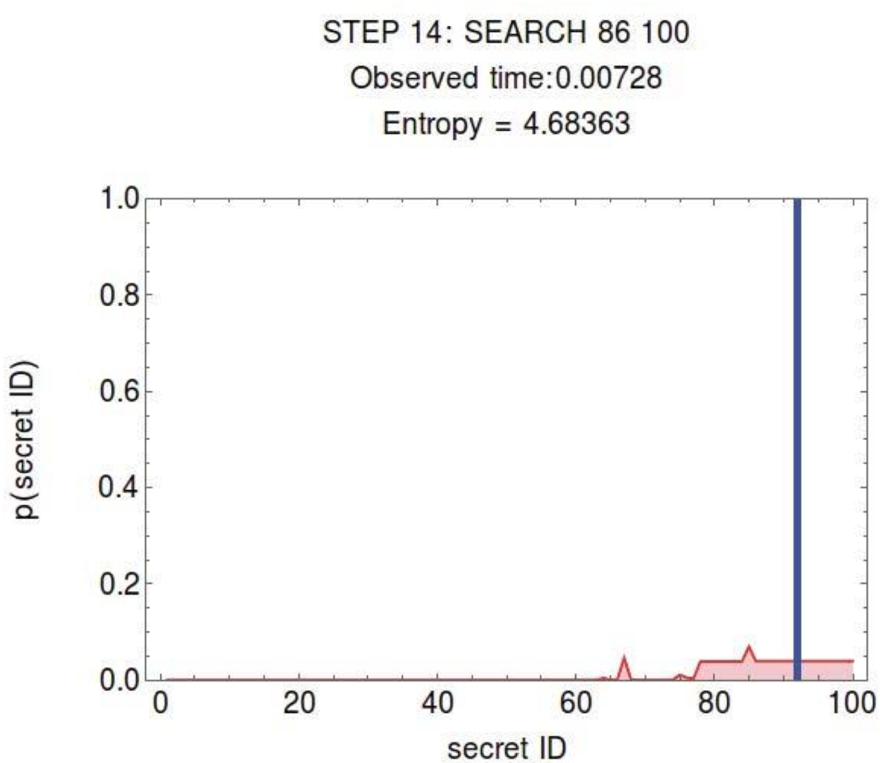
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



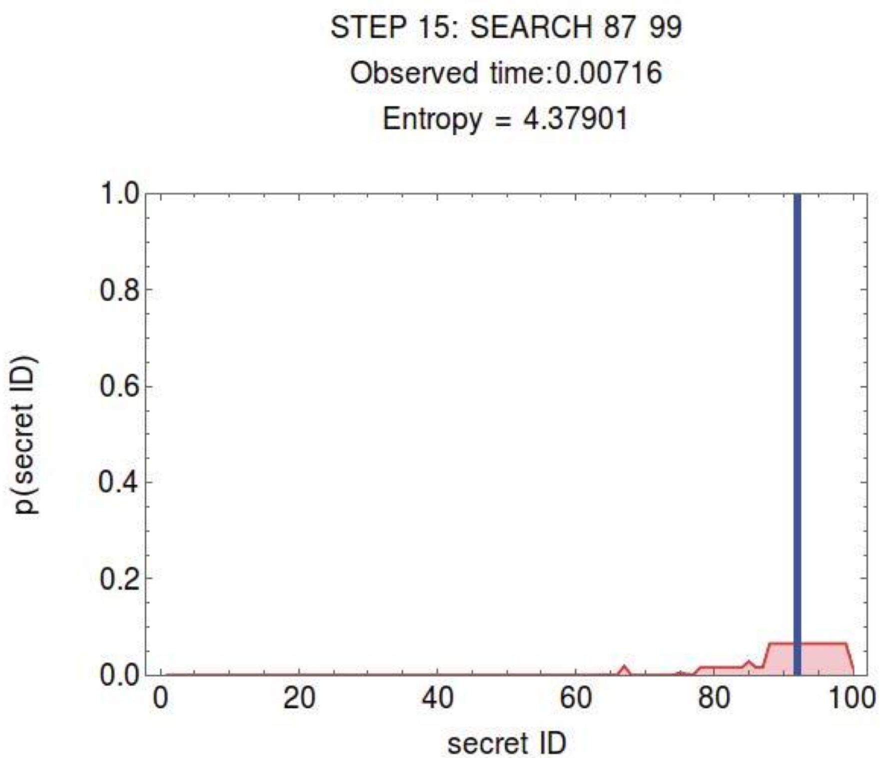
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



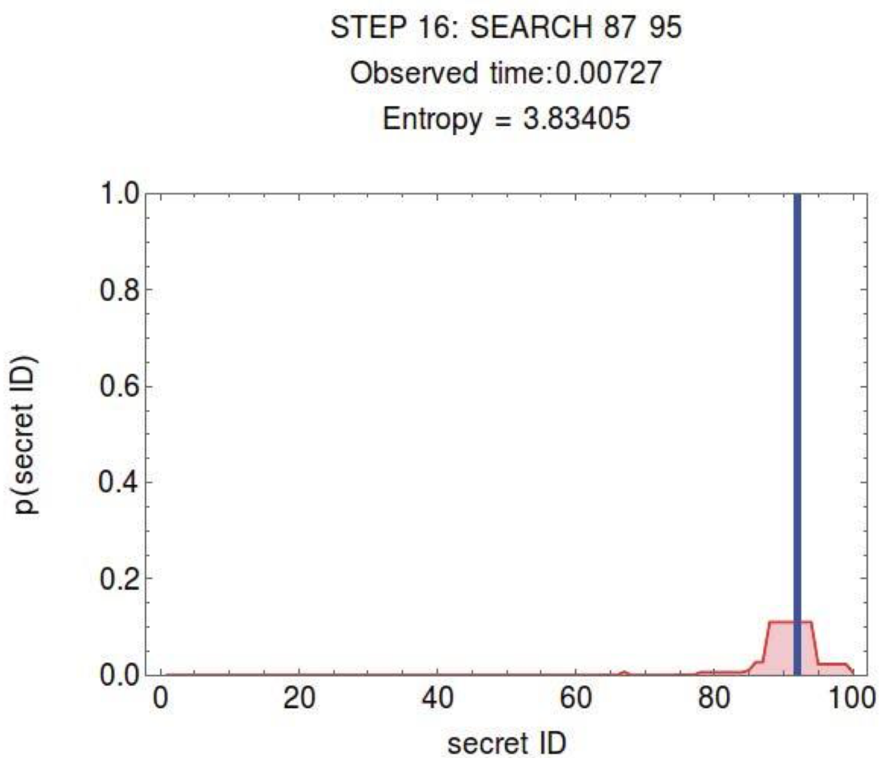
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



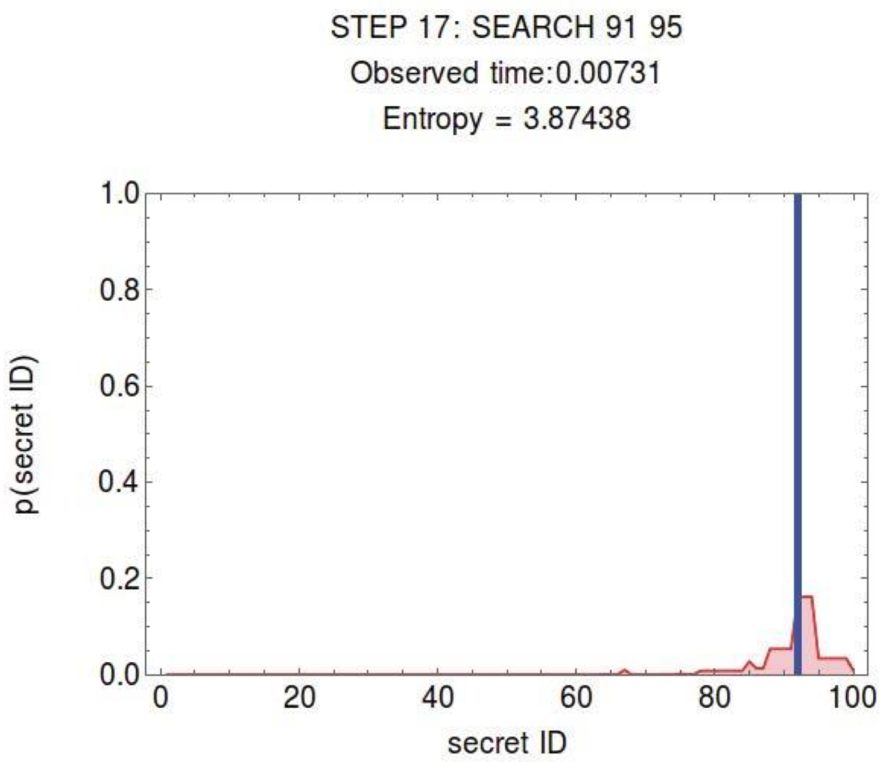
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



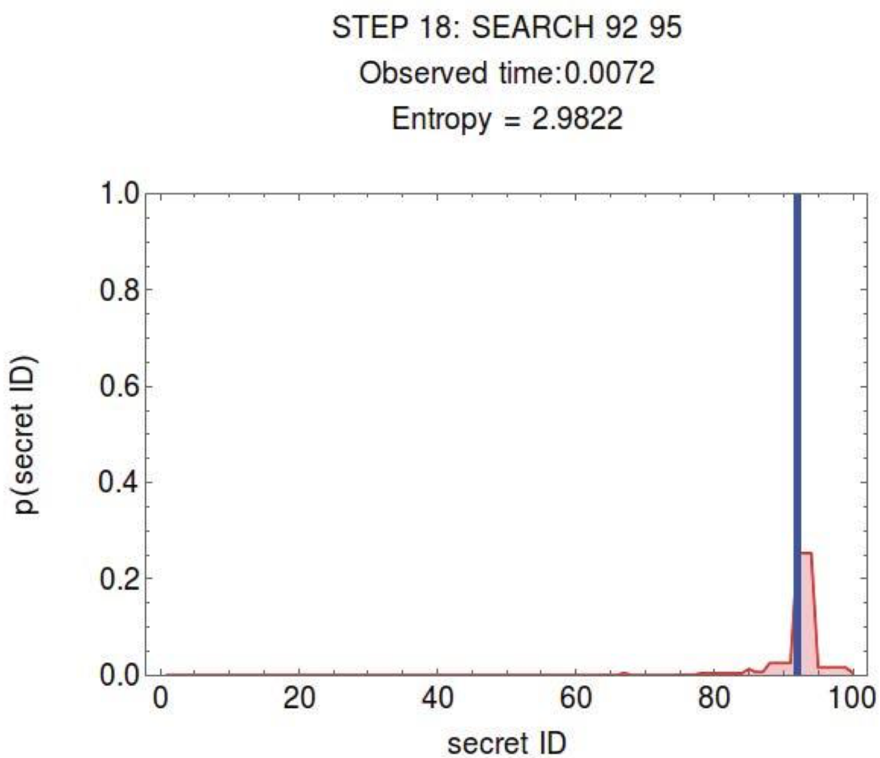
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



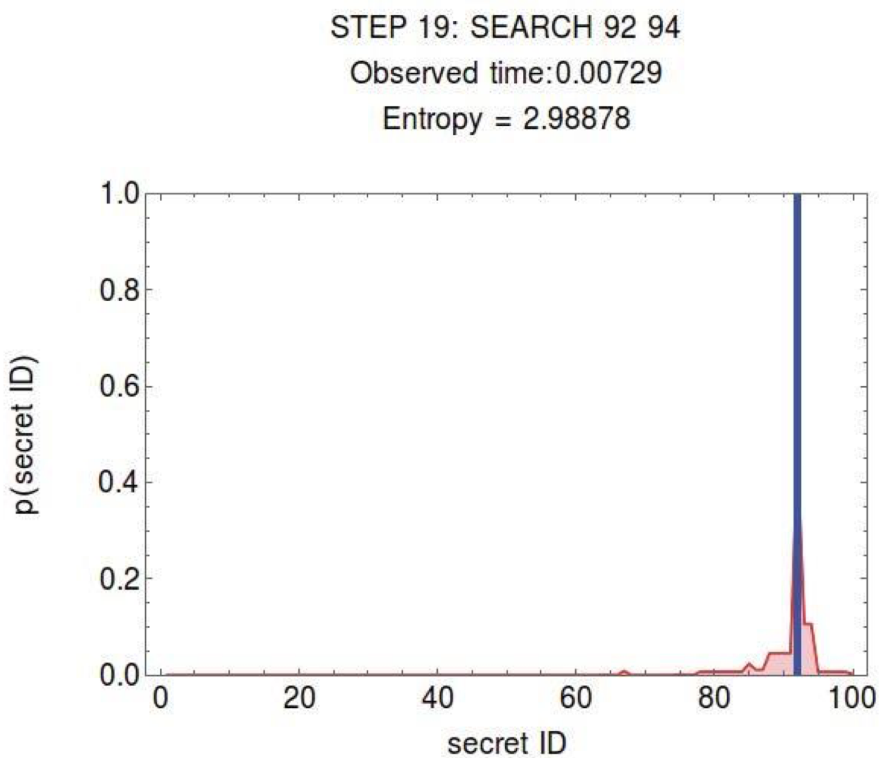
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



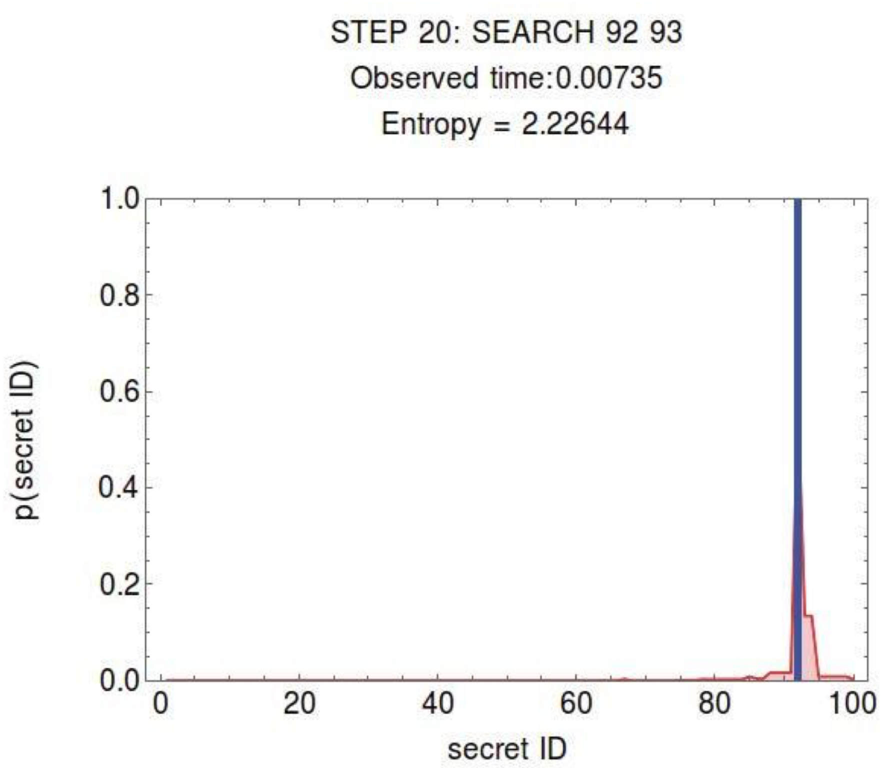
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



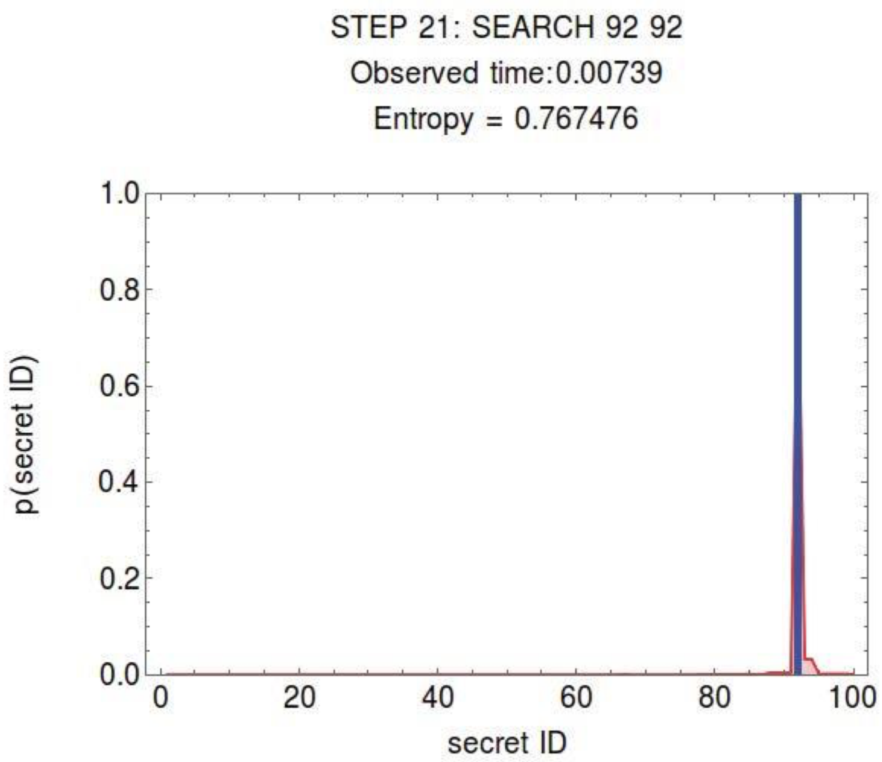
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



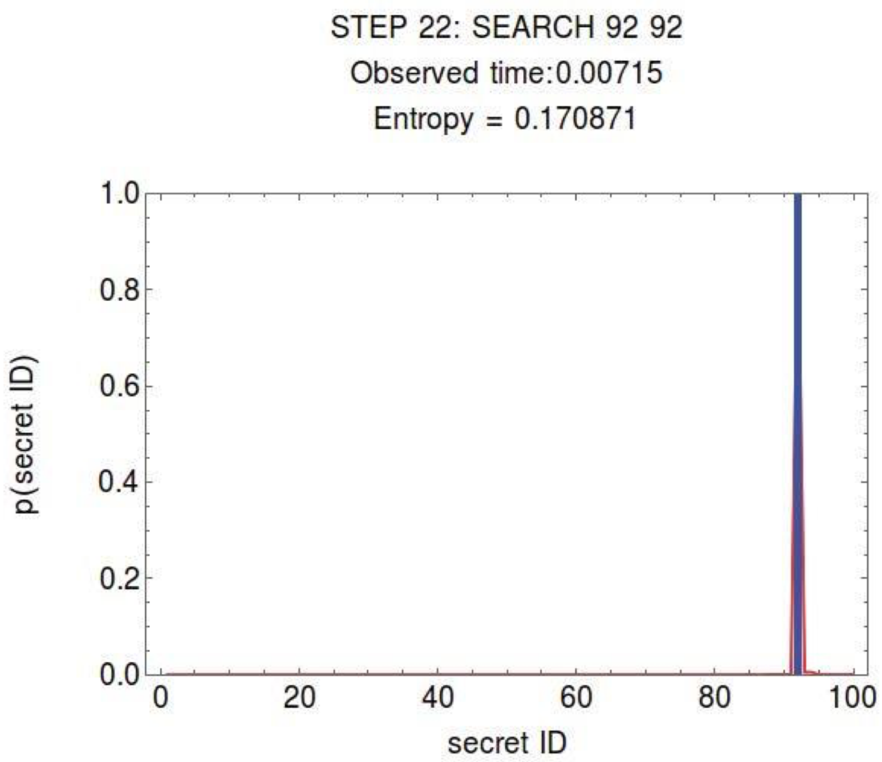
Automatically generated attack against LawDB

$$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$$



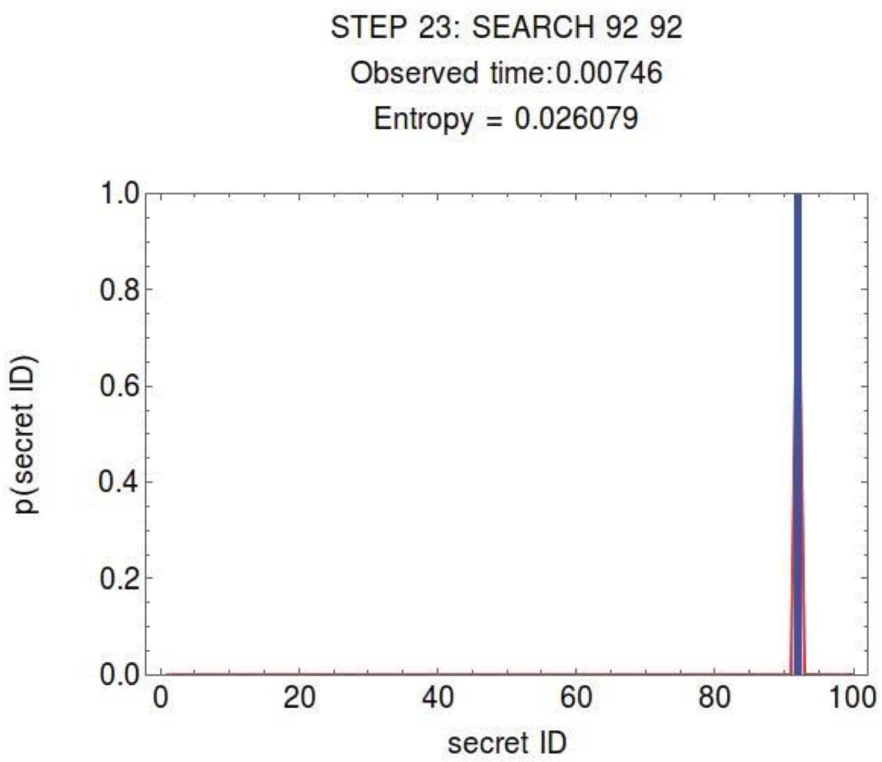
Automatically generated attack against LawDB

$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$



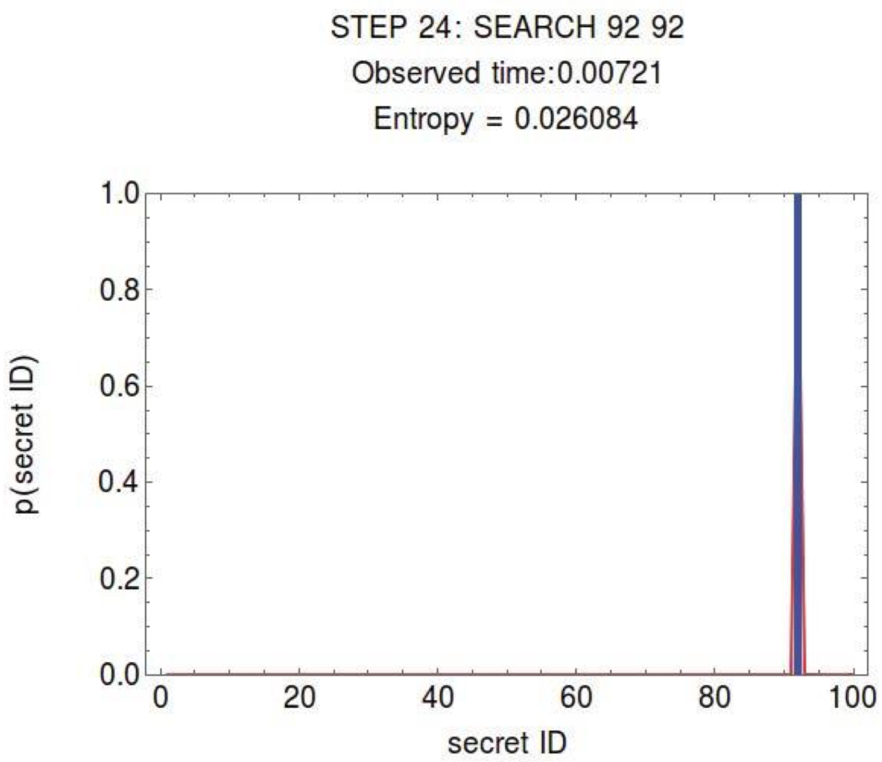
Automatically generated attack against LawDB

$$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$$



Automatically generated attack against LawDB

$$1 \leq ID \leq 100 \quad ID_1 = 64 \quad ID_2 = 85 \quad ID_{res} = 92$$



VLab papers on side channel analysis & model counting

- Tegan Brennan, Seemanta Saha, and Tevfik Bultan: ***JVM Fuzzing for JIT-Induced Side-Channel Detection.*** ICSE 2020
- Tegan Brennan, Nicolás Rosner, and Tevfik Bultan: ***JIT Leaks: Inducing Timing Side Channels through Just-In-Time Compilation***. IEEE S&P 2020
- Ismet Burak Kadron, Nicolás Rosner, Tevfik Bultan: ***Feedback-driven side-channel analysis for networked applications.*** ISSTA 2020: 260-271
- Nicolás Rosner, Ismet Burak Kadron, Lucas Bang, Tevfik Bultan: ***Profit: Detecting and Quantifying Side Channels in Networked Applications***. NDSS 2019
- Seemanta Saha, William Eiers, Ismet Burak Kadron, Lucas Bang, Tevfik Bultan: ***Incremental Adaptive Attack Synthesis***. JPF Workshop 2019
- Seemanta Saha, Ismet Burak Kadron, William Eiers, Lucas Bang, Tevfik Bultan: ***Attack Synthesis for Strings using Meta-Heuristics***. JPF Workshop 2018
- Nestan Tsiskaridze, Lucas Bang, Joseph McMahan, Tevfik Bultan, Timothy Sherwood: ***Information Leakage in Arbiter Protocols. ATVA 2018***: 404-421
- Lucas Bang, Nicolás Rosner, Tevfik Bultan: ***Online Synthesis of Adaptive Side-Channel Attacks Based On Noisy Observations***. EuroS&P 2018: 307-322
- Tegan Brennan, Seemanta Saha, Tevfik Bultan, Corina S. Pasareanu: ***Symbolic path cost analysis for side-channel detection.*** ISSTA 2018: 27-37

VLab papers on side channel analysis & model counting

- William Eiers, Seemanta Saha, Tegan Brennan, Tevfik Bultan: ***Subformula Caching for Model Counting and Quantitative Program Analysis.*** ASE 2019: 453-464

- Abdulbaki Aydin, William Eiers, Lucas Bang, Tegan Brennan, Miroslav Gavrilov, Tevfik Bultan, Fang Yu: ***Parameterized model counting for string and numeric constraints***. ESEC/SIGSOFT FSE 2018: 400-410
- Tevfik Bultan, Fang Yu, Muath Alkhalaf, Abdulbaki Aydin: ***String Analysis for Software Verification and Securit***y. Springer 2017, ISBN 978-3-319-68668-4, pp. 1-166
- Quoc-Sang Phan, Lucas Bang, Corina S. Pasareanu, Pasquale Malacaria, Tevfik Bultan: ***Synthesis of Adaptive Side-Channel Attacks***. CSF 2017: 328-342
- Tegan Brennan, Nestan Tsiskaridze, Nicolás Rosner, Abdulbaki Aydin, Tevfik Bultan: ***Constraint normalization and parameterized caching for quantitative program analysis***. ESEC/SIGSOFT FSE 2017: 535-546
- Lucas Bang, Abdulbaki Aydin, Quoc-Sang Phan, Corina S. Pasareanu, Tevfik Bultan: ***String analysis for side channels with segmented oracles***. SIGSOFT FSE 2016: 193-204
- Abdulbaki Aydin, Lucas Bang, Tevfik Bultan: ***Automata-Based Model Counting for String Constraints***. CAV (1) 2015: 255-272

BIBLIOGRAPHY: Quantitative Information Flow

- Geoffrey Smith. On the Foundations of Quantitative Information Flow. FOSSACS 2009: 288-302
- Geoffrey Smith. Quantifying Information Flow Using Min-Entropy. QEST 2011: 159-167
- Pasquale Malacaria. Assessing security threats of looping constructs. POPL 2007: 225-235
- David Clark, Sebastian Hunt, Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. Journal of Computer Security 15(3): 321-371 (2007) Alternate link
- Michael Backes, Boris Köpf, Andrey Rybalchenko. Automatic Discovery and Quantification of Information Leaks. IEEE Symposium on Security and Privacy 2009: 141-153
- Jonathan Heusser, Pasquale Malacaria. Quantifying information leaks in software. ACSAC 2010: 261-269
- Quantitative Security Analysis for Programs with Low Input and Noisy Output. Tri Minh Ngo, Marieke Huisman.
- Quantitative information flow under generic leakage functions and adaptive adversaries M. Boreale, Francesca Pampaloni.
- Measuring Information Leakage Using Generalized Gain Functions Mario S. Alvim, Kostas Chatzikokolakis, Catuscia Palamidessi, Geoffrey Smith.
- Quoc-Sang Phan, Pasquale Malacaria, Oksana Tkachuk, Corina S. Pasareanu. Symbolic quantitative information flow. ACM SIGSOFT Software Engineering Notes 37(6): 1-5 (2012)
- Quoc-Sang Phan, Pasquale Malacaria, Corina S. Pasareanu, Marcelo d'Amorim. Quantifying information leaks using reliability analysis. SPIN 2014: 105-108
- Stephen McCamant, Michael D. Ernst. Quantitative information flow as network flow capacity. PLDI 2008: 193-205
- Stephen McCamant, Michael D. Ernst. Quantitative information flow tracking for C and related languages. MIT-CSAIL-TR-2006-076
- On the relation between Differential Privacy and Quantitative Information Flow. Mario S. Alvim, Miguel E. Andres.
- Ian Sweet, José Manuel Calderón Trilla, Chad Scherrer, Michael Hicks, Stephen Magill. What's the Over/Under? Probabilistic Bounds on Information Leakage. POST 2018: 3-27
- Piotr Mardziel, Mário S. Alvim, Michael W. Hicks, Michael R. Clarkson. Quantifying Information Flow for Dynamic Secrets. IEEE Symposium on Security and Privacy 2014: 540-555
- Giovanni Cherubin, Konstantinos Chatzikokolakis, Catuscia Palamidessi: F-BLEAU: Fast Black-box Leakage Estimation. CoRR abs/1902.01350 (2019)

BIBLIOGRAPHY: Side-Channel Analysis

- Timing Analysis of Keystrokes and Timing Attacks on SSH. Dawn Xiaodong Song, David Wagner, Xuqing Tian.
- An information-theoretic model for adaptive side-channel attacks. Boris Köpf, David Basin.
- Automatically deriving information-theoretic bounds for adaptive side-channel attacks. Boris Köpf, David Basin.
- Shuo Chen, Rui Wang, XiaoFeng Wang, Kehuan Zhang. Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. IEEE Symposium on Security and Privacy 2010: 191-206
- Goran Doychev, Dominik Feld, Boris Köpf, Laurent Mauborgne, Jan Reineke. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. USENIX Security 2013: 431-446
- Multi-run Side-Channel Analysis Using Symbolic Execution and Max-SMT. Corina S. Pasareanu, Quoc-Sang Phan, Pasquale Malacaria.
- SMT-Based Verification of Software Countermeasures against Side-Channel Attacks. Hassan Eldib, Chao Wang, Patrick Schaumont.
- Lucas Bang, Abdulbaki Aydin, Quoc-Sang Phan, Corina S. Pasareanu, Tevfik Bultan. String analysis for side channels with segmented oracles. SIGSOFT FSE 2016: 193-204
- Pasquale Malacaria, M. H. R. Khouzani, Corina S. Pasareanu, Quoc-Sang Phan, Kasper Søe Luckow. Symbolic Side-Channel Analysis for Probabilistic Programs. CSF 2018: 313-327
- Tom Chothia, Yusuke Kawamoto, Chris Novakovic. LeakWatch: Estimating Information Leakage from Java Programs. ESORICS (2) 2014: 219-236
- Tom Chothia, Yusuke Kawamoto, Chris Novakovic. A Tool for Estimating Information Leakage. CAV 2013: 690-695
- Tom Chothia, Yusuke Kawamoto, Chris Novakovic, David Parker. Probabilistic Point-to-Point Information Leakage. CSF 2013: 193-205 2012
- Timos Antonopoulos, Paul Gazzillo, Michael Hicks, Eric Koskinen, Tachio Terauchi, and Shiyi Wei. Decomposition instead of self-composition for proving the absence of timing channels. In ACM SIGPLAN Notices, volume 52, pages 362–375. ACM, 2017.
- Jia Chen, Yu Feng, and Isil Dillig. Precise detection of side-channel vulnerabilities using quantitative cartesian hoare logic. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 875–890. ACM, 2017.
- Tegan Brennan, Seemanta Saha, Tevfik Bultan, and Corina S Pasareanu. Symbolic Path cost analysis for side-channel detection. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pages 27–37. ACM, 2018.

BIBLIOGRAPHY: Side-Channel Analysis, Continued

- Shirin Nilizadeh, Yannic Noller, and Corina S Pasareanu. Diffuzz: Differential fuzzing for side-channel analysis. arXiv preprint arXiv:1811.07005, 2018
- Shengjian Guo, Meng Wu, Chao Wang: Adversarial symbolic execution for detecting concurrency-related cache timing leaks. ESEC/SIGSOFT FSE 2018: 377-388
- Shengjian Guo, Yueqi Chen, Peng Li, Yueqiang Cheng, Huibo Wang, Meng Wu, Zhiqiang Zuo: SpecuSym: Speculative Symbolic Execution for Cache Timing Leak Detection. CoRR abs/1911.00507 (2019)
- Guanhua Wang, Sudipta Chattopadhyay, Arnab Kumar Biswas, Tulika Mitra, Abhik Roychoudhury: KLEESPECTRE: Detecting Information Leakage through Speculative Cache Attacks via Symbolic Execution. CoRR abs/1909.00647 (2019)
- Sudipta Chattopadhyay, Abhik Roychoudhury: Symbolic Verification of Cache Side-Channel Freedom. IEEE Trans. on CAD of Integrated Circuits and Systems 37(11): 2812-2823 (2018)

BIBLIOGRAPHY: Model Counting

- A Model Counter For Constraints Over Unbounded Strings. Loi Luu, Shweta Shinde, Prateek Saxena.
- Abdulbaki Aydin, Lucas Bang, Tevfik Bultan. Automata-Based Model Counting for String Constraints. CAV (1) 2015: 255-272
- Abdulbaki Aydin, William Eiers, Lucas Bang, Tegan Brennan, Miroslav Gavrilov, Tevfik Bultan, Fang Yu. Parameterized model counting for string and numeric constraints. ESEC/SIGSOFT FSE 2018: 400-410
- The good old Davis-Putnam procedure helps counting models. Elazar Birnbaum, Eliezer L. Lozinskii.
- Satisfiability modulo counting: a new approach for analyzing privacy properties. Matthew Fredrikson, Somesh Jha.
- Symbolic Polytopes for Quantitative Interpolation and Verification. Klaus v. Gleissenthall1, Boris Kopf, and Andrey Rybalchenko.
- An Automata-Theoretic Algorithm for Counting Solutions to Presburger Formulas. Erin Parker, Siddhartha Chatterjee.
- Abstract model counting: a novel approach for quantification of information leaks. Quoc-Sang Phan, Pasquale Malacaria.
- A Polynomial Time Algorithm for Counting Integral Points in Polyhedra When the Dimension Is Fixed. Alexander I. Barvinok.
- Effective lattice point counting in rational convex polytopes. Jesús A. De Loerab, Raymond Hemmeckeb, Jeremiah Tauzera, Ruriko Yoshidab.
- Distribution-Aware Sampling and Weighted Model Counting for SAT. Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, Moshe Y. Vardi.
- From Weighted to Unweighted Model Counting. Supratik Chakraborty, Dror Fried, Kuldeep S. Meel, Moshe Y. Vardi.
- Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls Supratik Chakraborty, Kuldeep S. Meel, Moshe Y. Vardi.
- Approximate Probabilistic Inference via Word-Level Counting. Supratik Chakraborty, Kuldeep S. Meel, Rakesh Mistry, Moshe Y. Vardi.
- Mateus Borges, Quoc-Sang Phan, Antonio Filieri, Corina S. Pasareanu. Model-Counting Approaches for Nonlinear Numerical Constraints. NFM 2017: 131-138
- Antonio Filieri, Marcelo F. Frias, Corina S. Pasareanu, Willem Visser. Model Counting for Complex Data Structures. SPIN 2015: 222-241
- Minh-Thai Trinh, Duc-Hiep Chu, Joxan Jaffar. Model Counting for Recursively-Defined Strings. CAV (2) 2017: 399-418
- Dmitry Chistikov, Rayna Dimitrova, Rupak Majumdar: Approximate Counting in SMT and Value Estimation for Probabilistic Programs. TACAS 2015: 320-334
- Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model Counting

BIBLIOGRAPHY: Probabilistic Symbolic Execution

- Jaco Geldenhuys, Matthew B. Dwyer, Willem Visser. Probabilistic symbolic execution. ISSTA 2012: 166-176
- Corina S. Pasareanu, Willem Visser, David H. Bushnell, Jaco Geldenhuys, Peter C. Mehlitz, Neha Rungta. Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis. Autom. Softw. Eng. 20(3): 391-425 (2013) Alternate link
- Antonio Filieri, Corina S. Pasareanu, Willem Visser. Reliability analysis in symbolic pathfinder. ICSE 2013: 622-631
- Mateus Borges, Antonio Filieri, Marcelo d'Amorim, Corina S. Pasareanu, Willem Visser. Compositional solution space quantification for probabilistic software analysis. PLDI 2014: 15

BIBLIOGRAPHY: Attack Synthesis

- Quoc-Sang Phan, Lucas Bang, Corina S. Pasareanu, Pasquale Malacaria, Tevfik Bultan. Synthesis of Adaptive Side-Channel Attacks. CSF 2017: 328-342
- Lucas Bang, Nicolás Rosner, Tevfik Bultan. Online Synthesis of Adaptive Side-Channel Attacks Based On Noisy Observations. EuroS&P 2018: 307-322
- Seemanta Saha, Ismet Burak Kadron, William Eiers, Lucas Bang, Tevfik Bultan. Attack Synthesis for Strings using Meta-Heuristics. ACM SIGSOFT Software Engineering Notes 43(4): 56 (2018)