Raccoon: Closing Side-Channels through Obfuscated Execution

by Ashay Rane, Calvin Lin, Mohit Tiwari

Presentation by Arjun Khurana and Timothy Wong

Outline

- Background
- Raccoon Design
- Evaluation
- Conclusion
- Questions
- Debate

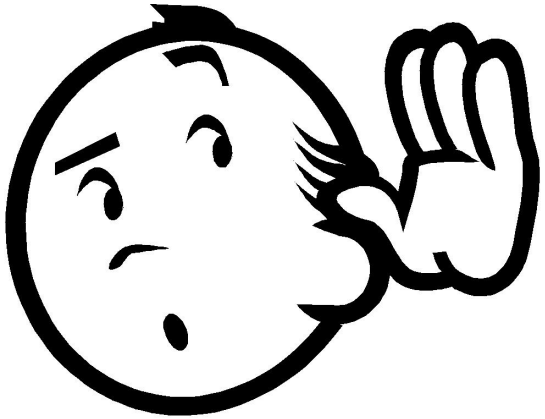
Security Is Everywhere

Side-Channel Attacks

Any attack based on information gained from the physical implementation

- Timing
- Power Consumption
- Cache Usage
- Sound
- Data Size
- Electromagnetic Leaks

Which of these are ***digital*** side-channels?



Square and Multiply

```
if (secret_bit == 1) {
      z = (m*z*z) mod n;
}
else {
      z = (z*z) mod n;
}
```

Threat Model

- Hardware Assumptions:
  - Adversary can monitor and tamper with digital signals on processor's I/O pins
  - The processor is a sealed chip

- Software Assumptions:
  - Adversary can run malicious applications on victim's OS
  - Malicious applications can probe run-time statistics
  - Input program is error-free
  - Adversary has access to transformed binary code

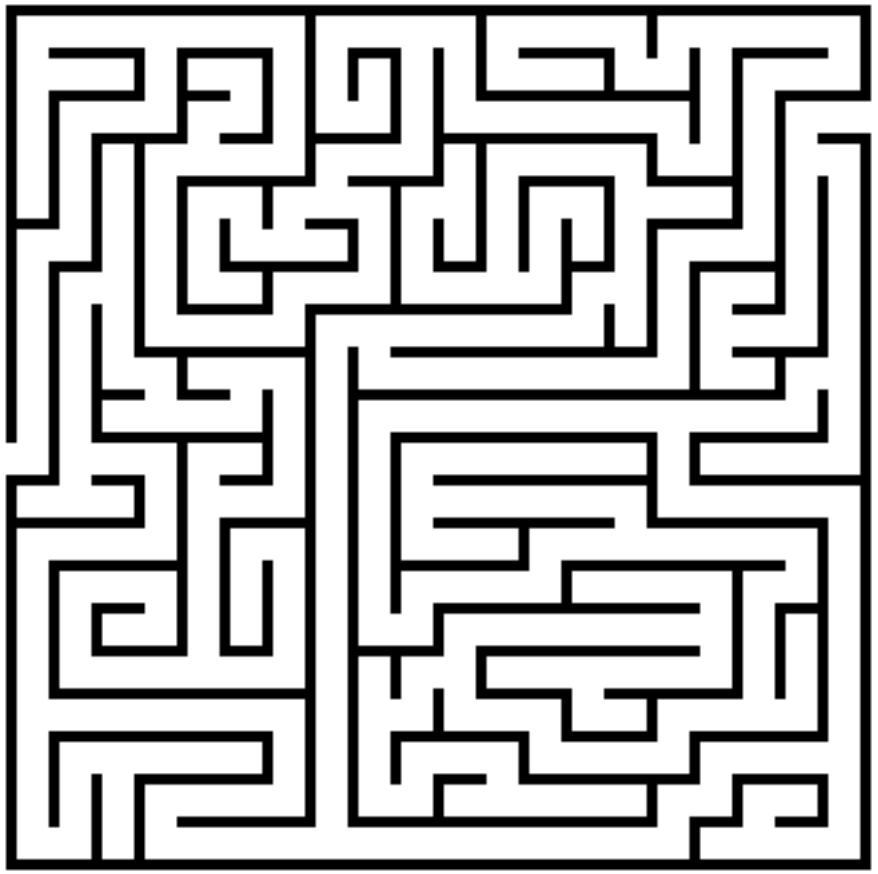
System Guarantees

- Adversary cannot differentiate between real path and decoy path
- Same final program output as original program
- Obfuscation does not introduce new info leaks
- Respects the original program's control and data dependences

Outline

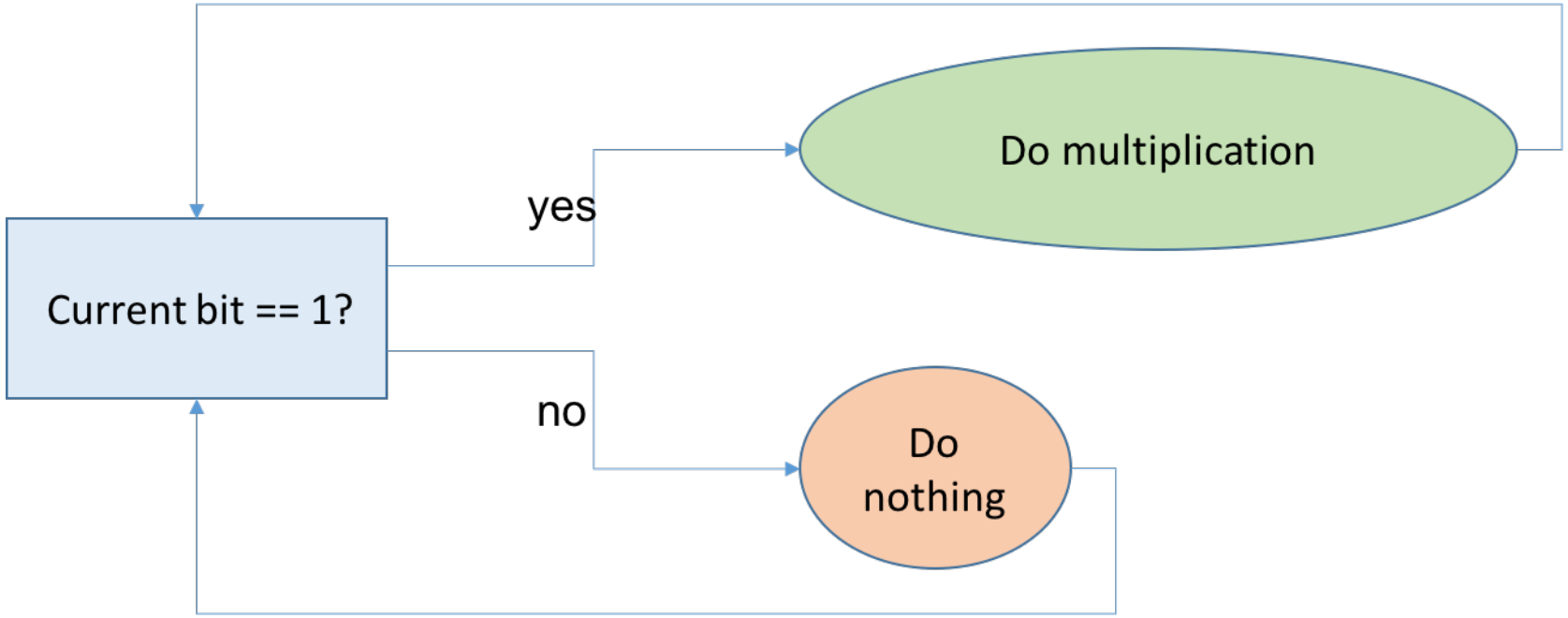
- Background
- **Raccoon**
- Evaluation
- Conclusion
- Questions
- Debate

Why Raccoon?



Key Properties

1. Both real and decoy paths execute actual program instructions
   - e.g. square and multiply function in our mini-project...



Key Properties

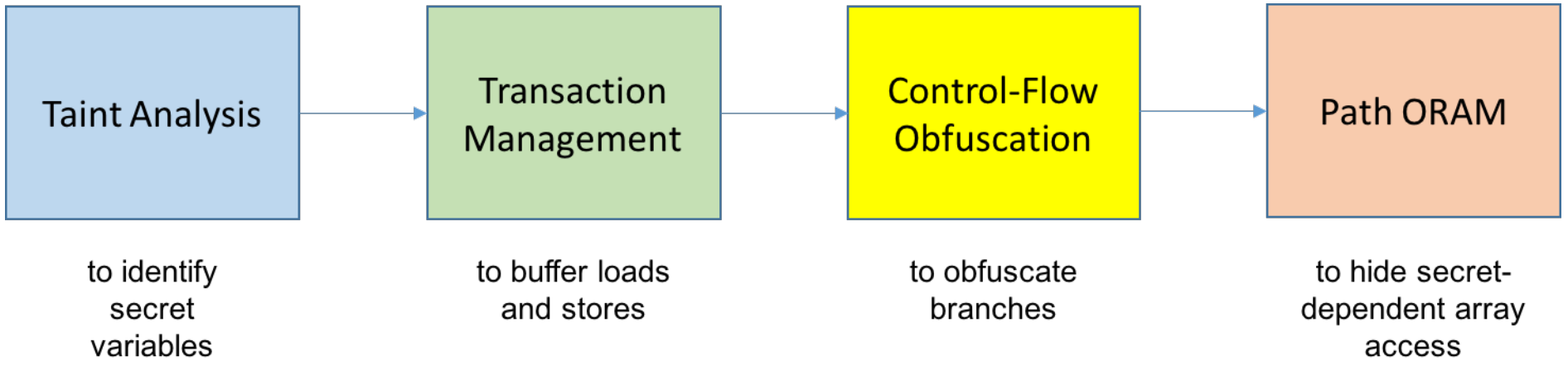
1. Both real and decoy paths execute actual program instructions
   - e.g. square and multiply function in our mini-project...
2. Both real and decoy paths are allowed to update memory
   - e.g. This code…

```
p = &a;
if (secret == true)
{
      …
}
else {
      …
      p = &b;
      *p = 10;
}
```

real path

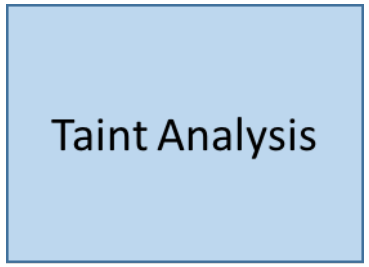
decoy path

Components


Taint Analysis
Transaction Management
Control-Flow Obfuscation
Path ORAM
to identify secret variables
to buffer loads and stores
to obfuscate branches
to hide secret-dependent array access

Taint Analysis

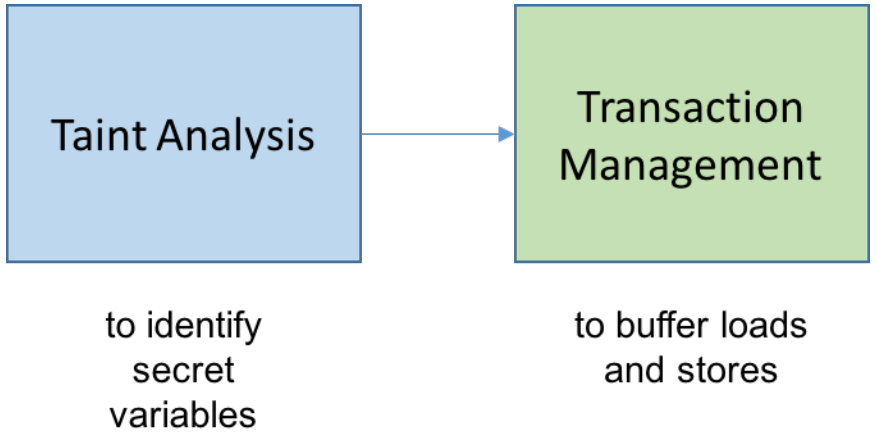
- User annotates secret variables using `__attribute__` construct
- Raccoon identifies branches and data accesses that require obfuscation
- Result: a list of memory and branch instructions



to identify secret variables

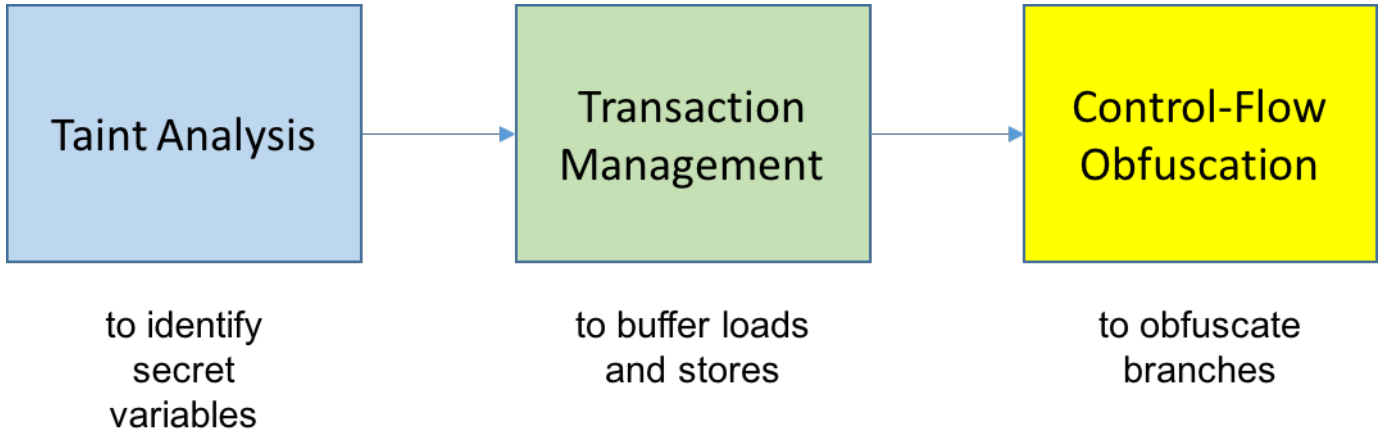
Transaction Management

- Uses both transactional buffer and non-transactional memory
- **Real path**: write value to DRAM using oblivious store operation
- **Decoy path**: use oblivious store operation to read existing value and write back



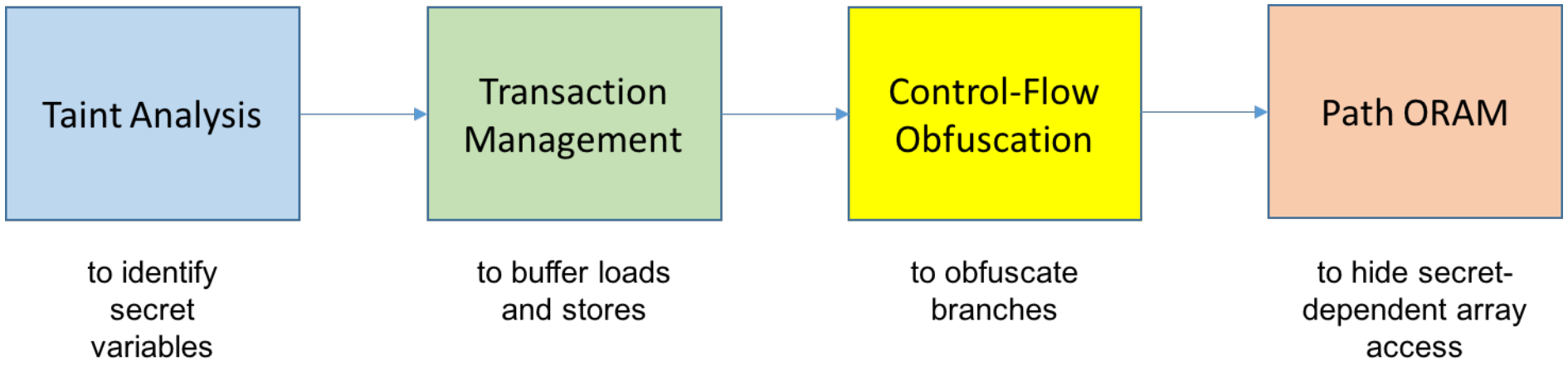
Control-Flow Obfuscation

- 3 key facilities:
  - obfuscate() – forces sequential execution of both paths
  - epilog() – transfer control-flow between if-statements
  - oblivious store operation – ensure decoy path does not mess up memory



Path ORAM

- Cannot directly index into arrays
- Streams over arrays; selectively read/update using oblivious store
- Can be implemented recursive or non-recursive



A Precaution…

- Limiting Termination Channel Leaks
  - e.g. This statement…

```
if (y != 0)
{
      z = x / y;
}
```

A Precaution…

- Limiting Termination Channel Leaks
  - e.g. This statement…

```
if (y != 0)
{
      z = x / y;
}
```

If y=0, and Raccoon executes decoy path…

DIVISION-BY-ZERO ERROR!!!

Since Raccoon assumes original input program is error-free…

=> **occurrence of crash reveals the decoy path!**

A Precaution…

- Limiting Termination Channel Leaks
  - e.g. This statement…

```
if (y != 0)
{
      z = x / y;
}
```

If y=0, and Raccoon executes decoy path…

DIVISION-BY-ZERO ERROR!!!

Since Raccoon assumes original input program is error-free…

=> **occurrence of crash reveals the decoy path!**

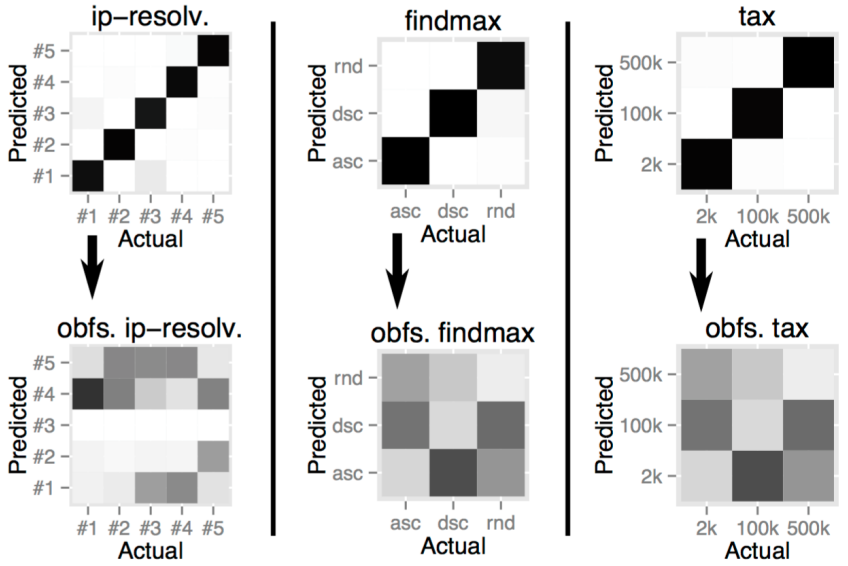
Raccoon prevents the program from terminating abnormally due to exceptions
=> for integer division, obliviously replaces divisor with non-zero value

Outline

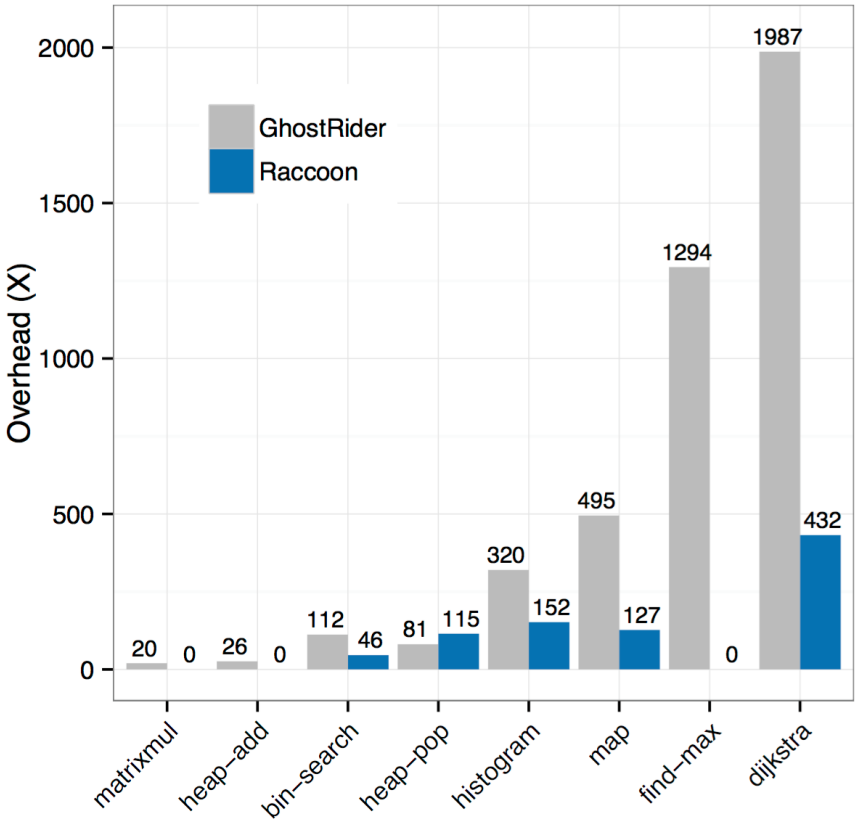
- Background
- Raccoon
- **Evaluation**
- Conclusion
- Questions
- Debate

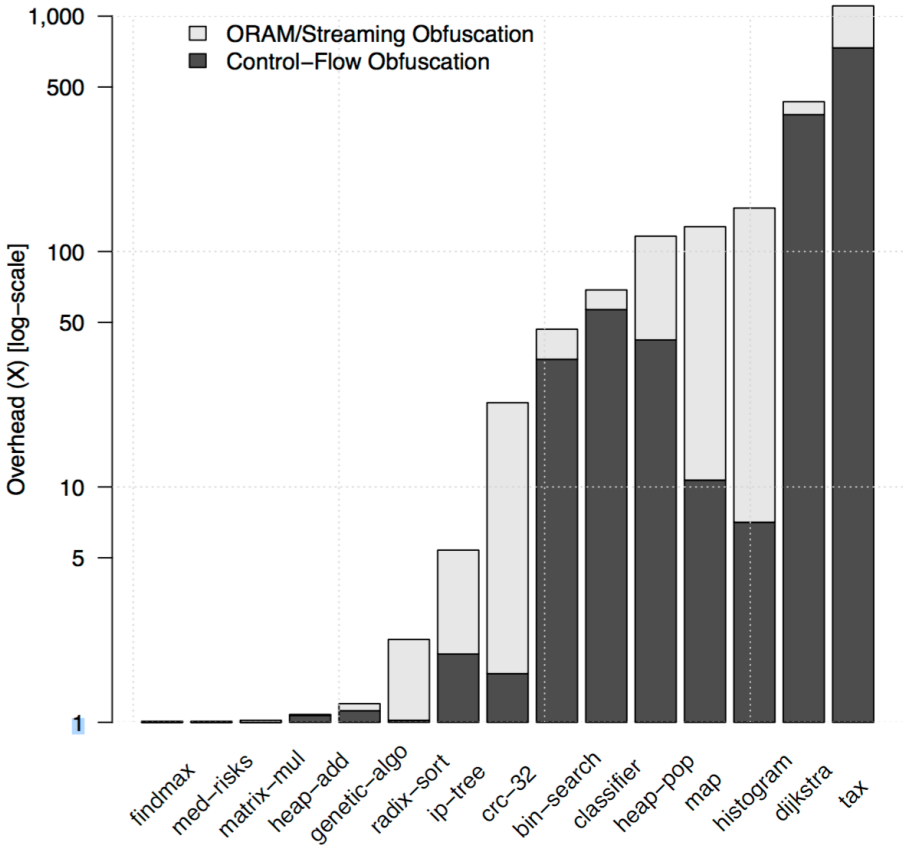
Security Evaluation

- Correctness of obfuscated code
- Security of obfuscation code
- Defense against Side-Channel Attacks



Performance





Outline

- Background
- Raccoon Design
- Evaluation
- **Conclusion**
- Questions
- Debate

Conclusion

- Raccoon is a more well-rounded solution for side-channel attacks
- Raccoon adjusts program on instruction level
- Raccoon disguises the secret value by using decoy paths

Thank you for listening!

Questions?

Debate!!!

Considering security vs. performance overhead, is it worth it to implement Raccoon on current hardware?

Is streaming through the entire array using ORAM the best method to access data in Raccoon?