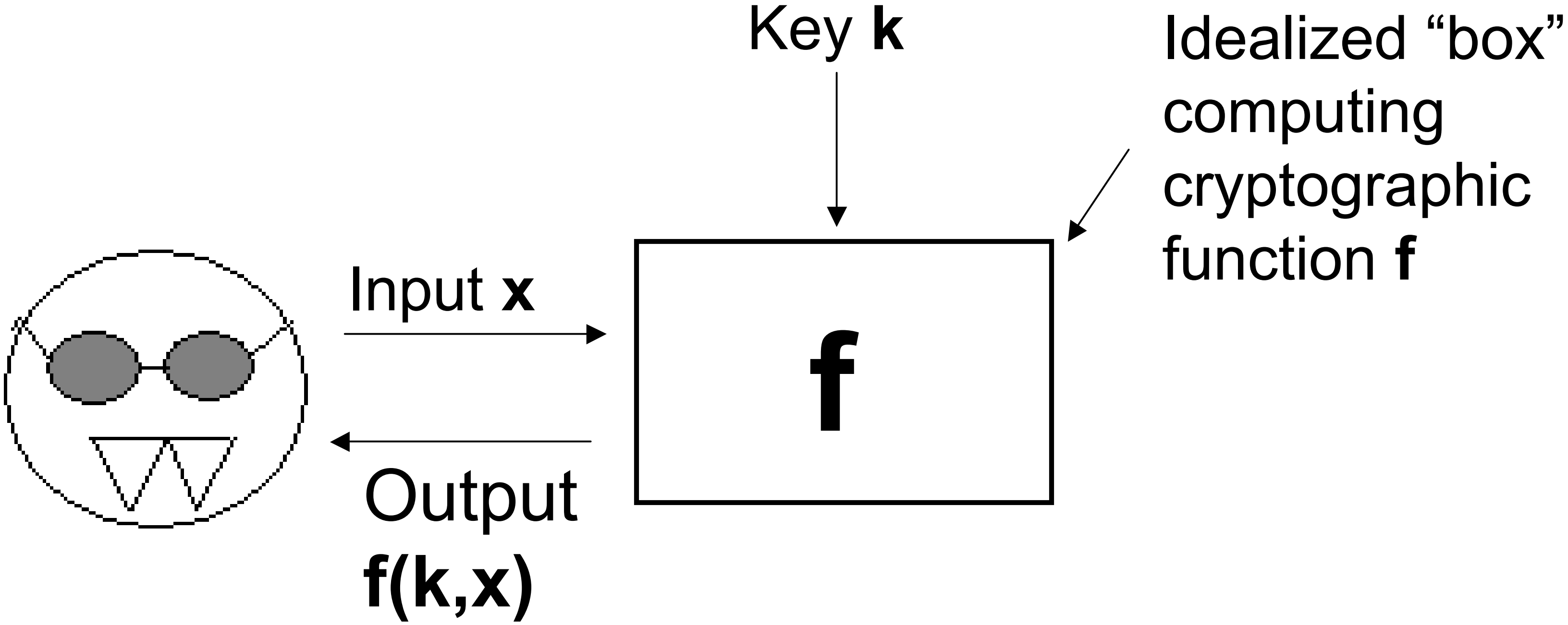
# The Program Counter Security Model: Automatic Detection and Removal of Control-Flow Side Channel Attacks

**David Molnar**, Matt Piotrowski,
David Schultz, and David Wagner
UC-Berkeley and MIT

# Regular Cryptographic Attacks



Key **k**

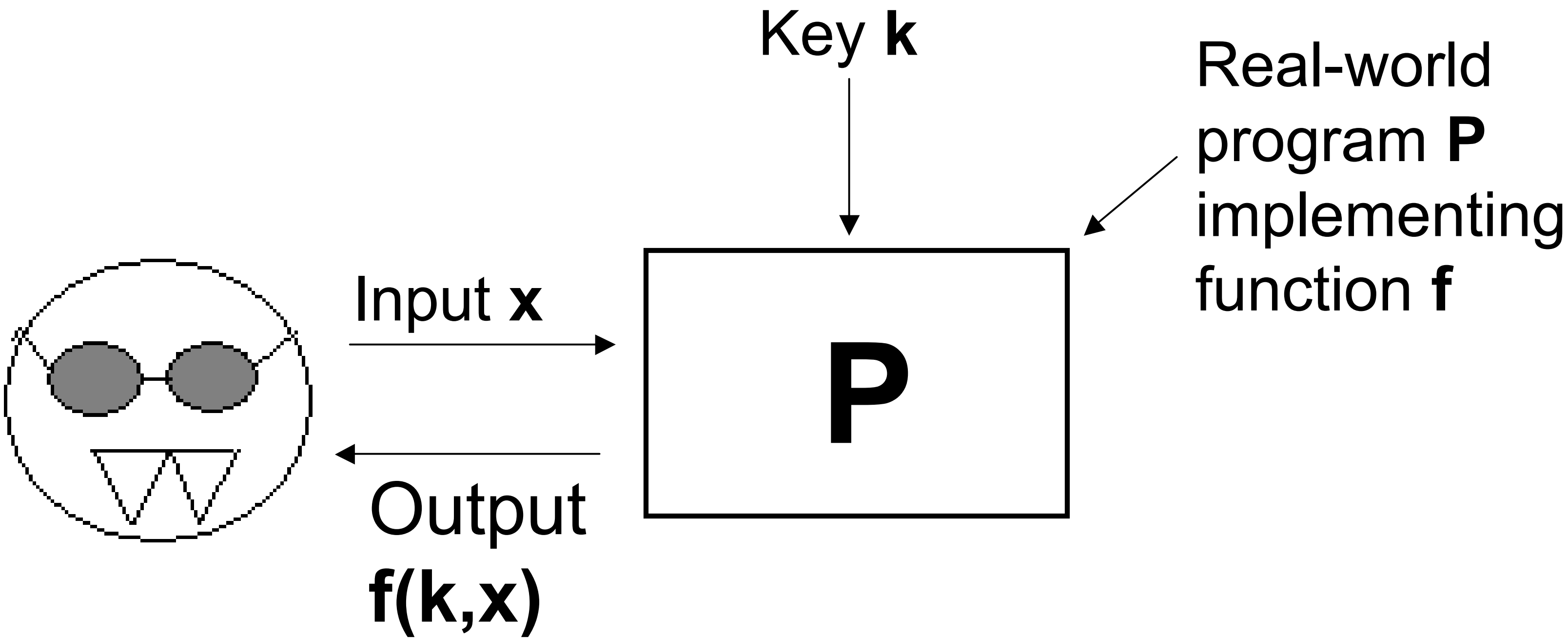
Idealized “box” computing cryptographic function **f**

Input **x**

**f**

Output **f(k,x)**

# Side Channel Attacks



Key **k**

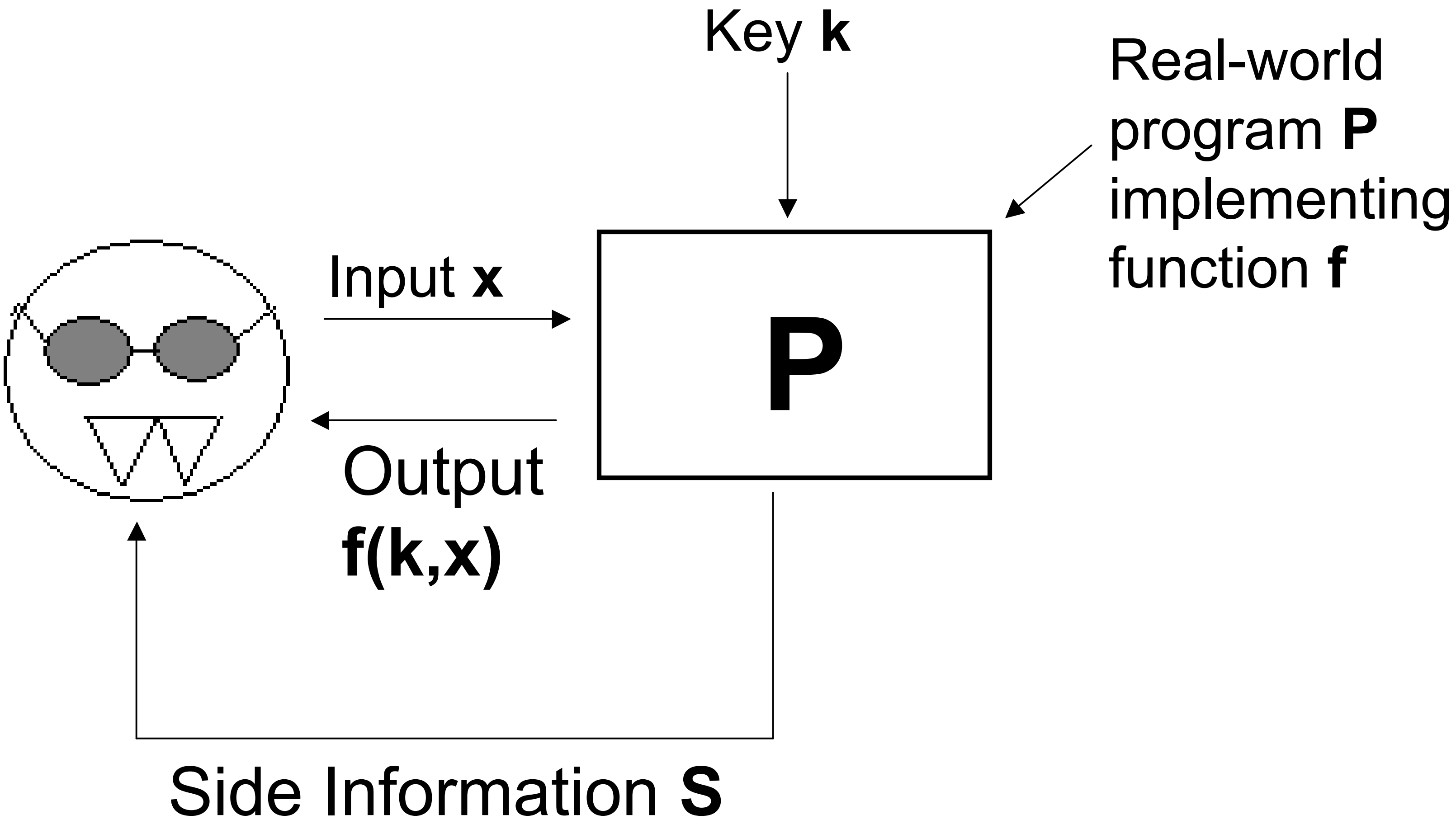
Real-world program **P** implementing function **f**

Input **x**

**P**

Output **f(k,x)**

# Side Channel Attacks



Key **k**

Real-world program **P** implementing function **f**

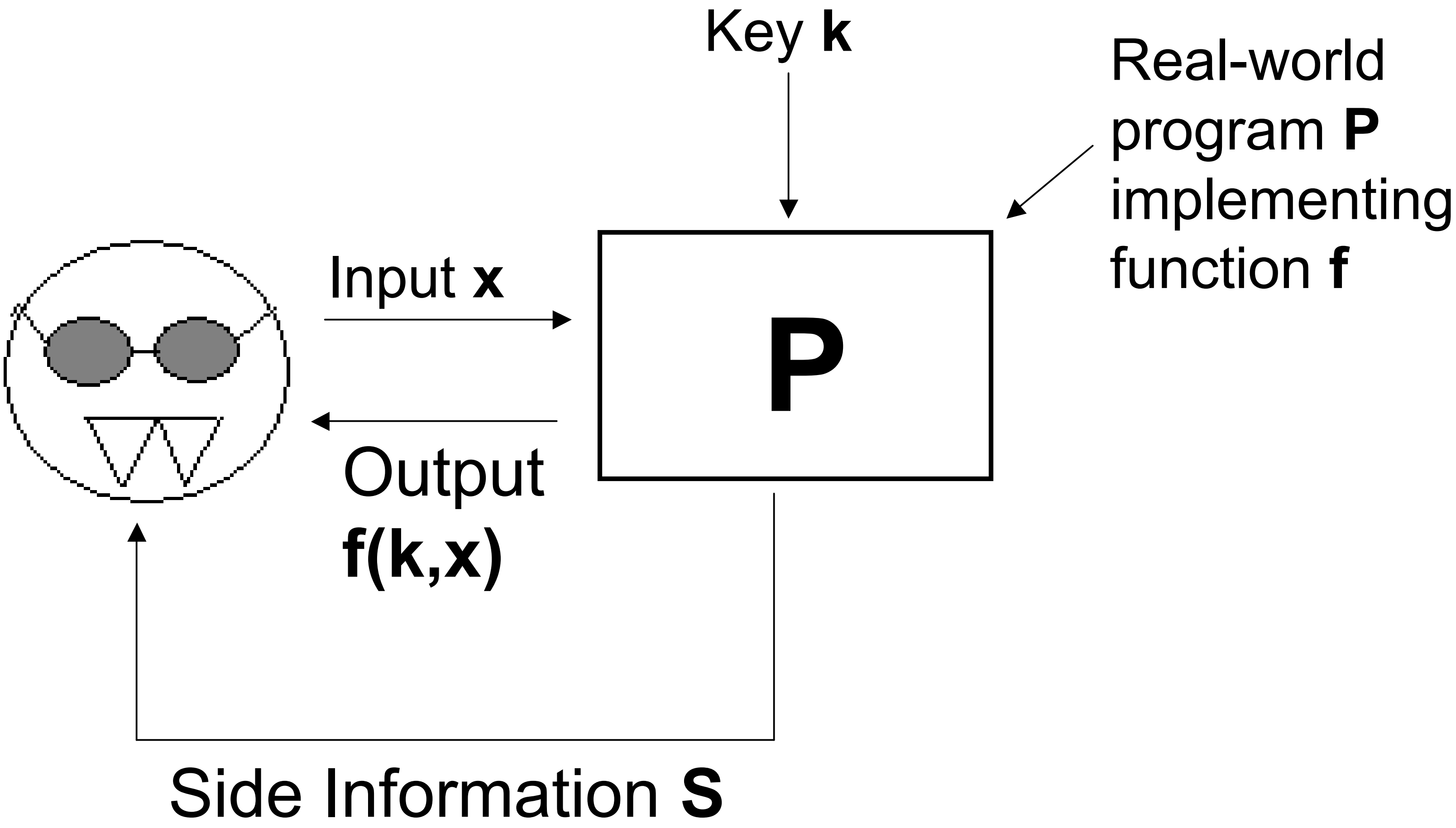
Input **x**

**P**

Output **f(k,x)**

Side Information **S**

# Side Channel Attacks



Key **k**

Real-world program **P** implementing function **f**

Input **x**

**P**

Output **f(k,x)**

Side Information **S**

**Control-Flow Side Channel**: **S** depends on control flow of **P**

# What We Do

- **Define** “control-flow side information”
- **Detect** potential control-flow attacks
- **Transform** C code to remove attacks
- **Check** compiled C code free of attacks

# **Define** Program Counter Model

- Adversary sees **transcript** of all values of program counter (PC) in run of **P**(k,x)
- States “contract” with hardware
  - Only PC transcript leaked on run of program
  - Could be none of today’s HW meets contract
- Define security with simulation argument
  - Program is **PC-secure** if exists simulator that can “fake” PC transcripts without secret key k
  - Informally, adversary “learns nothing”

# **Detect** potential attacks

- Use **gcov** to see code coverage for **P**(k,x)
- Run P with many different keys **k**, same **x**
- Different code coverage → potential attack
- Example: PGP implementation of IDEA

Over fixed **x**, 10,000 different keys **k**

```
p = a * b;
if (p) {
      b = low16(p);
      a = p >> 16;                  mean    std     mean    max
      return (b - a) + (b < a);    27      0.03    26      27
      } else if (a) {
             return 1 - a;         7       0.02    7       8
      } else {
             return 1 - b;         0       0.02    0       1
      }
}
```

# Transform

- C-to-C source transform

```
If (n % 2) {
        r = r * b;
        n = n - 1;
} else {
        b = b * b;
        n = n/2;
}
```

→

```
m = -(n % 2);
  r = (m & (r * b)) | (~m & r);
  n = (m & (n-1)) | (~m & n);
m = ~m;
  b = (m & (b * b)) | (~m & b);
  n = (m & (n/2)) | (~m & n);
```

- Transformed code provably PC-secure
  - For subset of C including most crypto code
- ~5x slowdown, ~2x stack space

# Check

- Will C compiler preserve PC-security?
- We built static checker for x86 assembly
- Check information flow between key, PC
- Caught unsafe compilation of “!” by gcc
    - Even with –O0 flag
- Found Intel compiler output PC-secure assembly even with optimizations

**Recap:**

1) Formal security model for control-flow side channels
2) Automatic detection of potential control-flow attacks
3) C-to-C transform to remove attacks
4) Static x86 assembly checker verifies compiled code
5) Result: remove large class of side channel attacks (not all)

# Questions?

**dmolnar@eecs.berkeley.edu**

www.cs.berkeley.edu/~dmolnar/pcmodel-wip.ppt