Securing Hardware Platforms Against Malicious Circuits Through Static Analysis

Matthew Hicks - University of Illinois
Samuel T. King - University of Illinois
Milo M. K. Martin - University of Pennsylvania
Jonathan M. Smith - University of Pennsylvania

Building Secure Systems

- We make assumptions when designing secure systems
- Break secure system, break assumptions
  - E.g., look for crypto keys in memory
- People assume hardware is correct

- What can we do if we can't make this assumption?

Why Hardware?

- Key hardware properties
  - Complex
  - Expensive
  - Static
  - Base of the system

The Threat

Dead Circuit Identification (DCI)

- Goal: Help designers by highlighting all potentially malicious circuits automatically

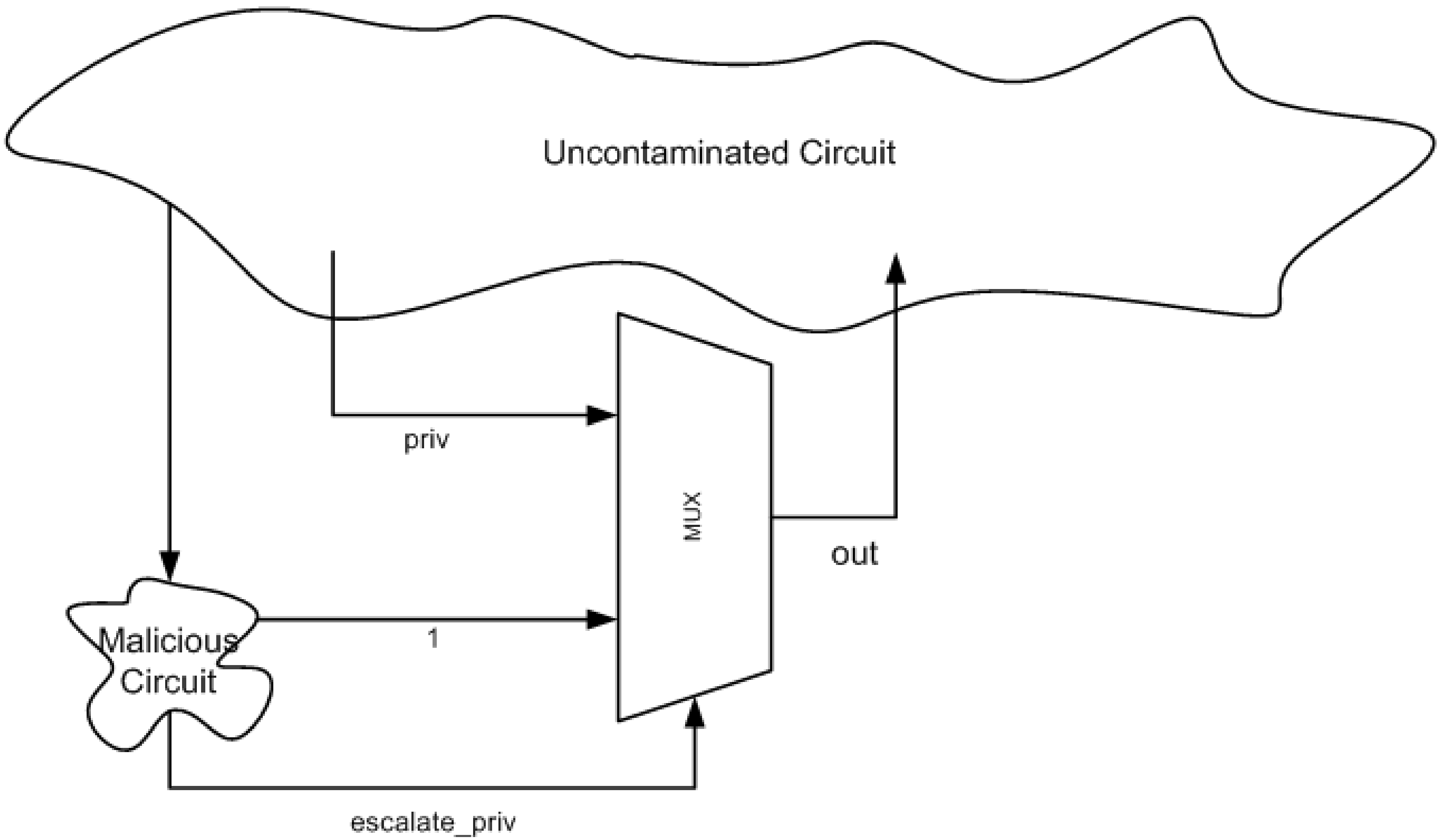
Dead Circuit Identification (DCI)

- Goal: Help designers by highlighting all potentially malicious circuits automatically
- Intuition: Attacker is motivated to avoid impacting functionality during testing
  - Otherwise they would be caught

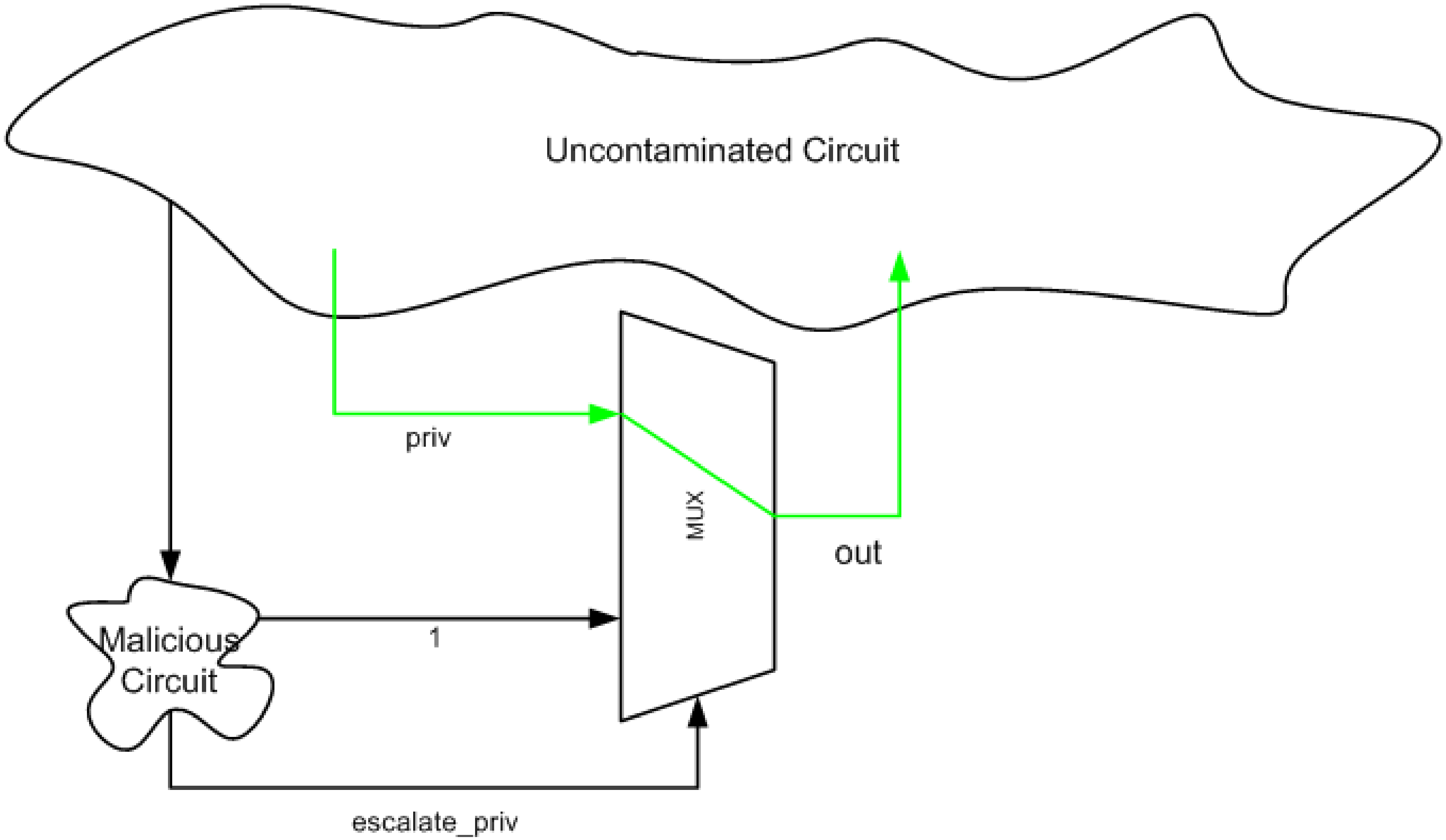
Dead Circuit Identification (DCI)

- Goal: Help designers by highlighting all potentially malicious circuits automatically
- Intuition: Attacker is motivated to avoid impacting functionality during testing
  - Otherwise they would be caught
- Detect all circuits where the output value is identical to the input value for all tests
  - Internal circuits don't impact functionality

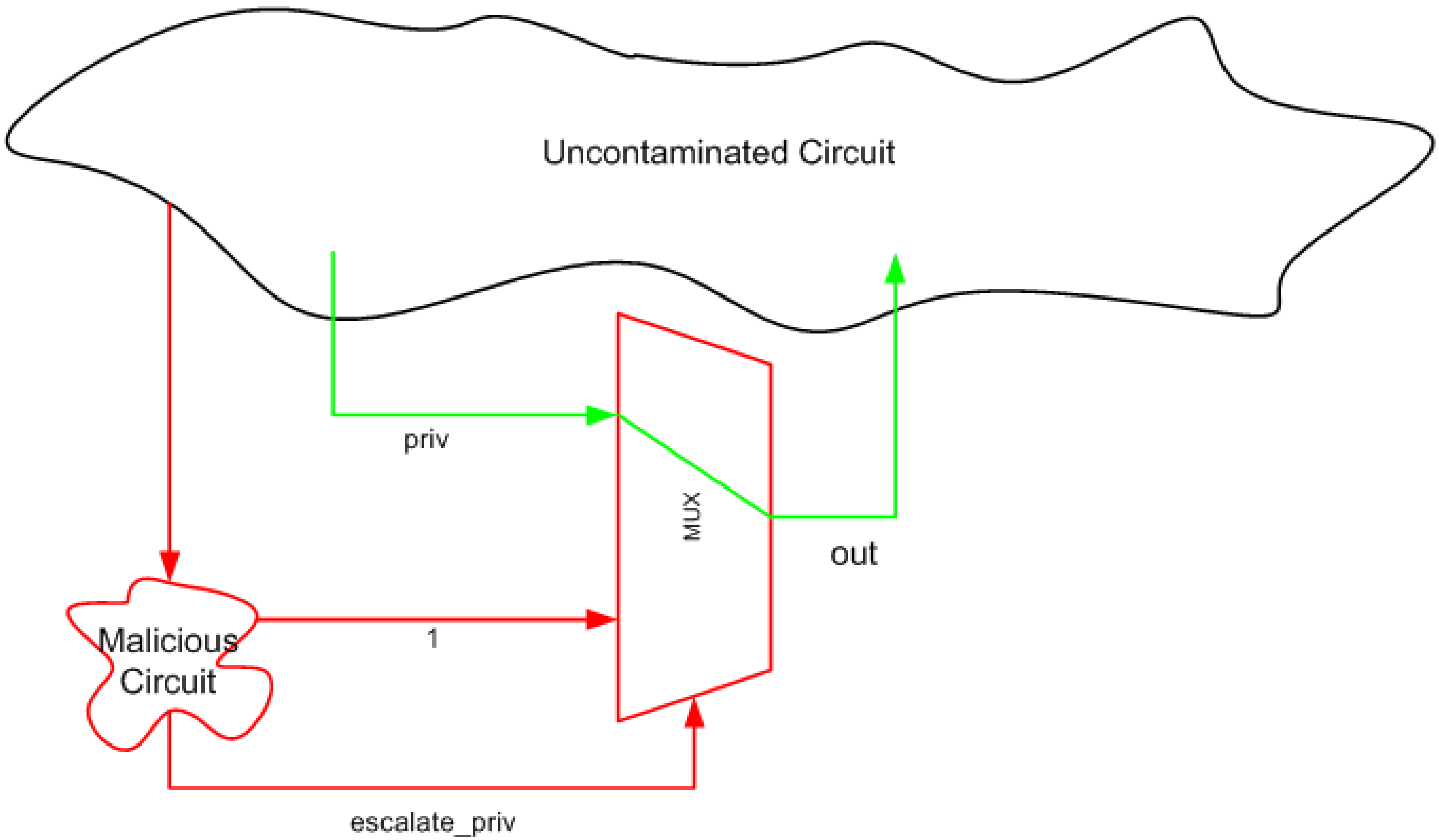
Simple Example



Simple Example



Simple Example



What the numbers say... so far

Foothold	Num. nodes in data-flow graph		False negatives	False positives
	Non-foothold	Foothold		
Supervisor transition	532	3	0 (0%)	147 (28%)
Shadow mode	532	4	0 (0%)	144 (27%)

Questions

It Can Happen

```
IF ( r.d.inst ( conv_integer ( r.d.set ) ) = X"80082000" ) THEN
   hackStateM1 <= '1';
ELSE
   hackStateM1 <= '0';
END IF;

IF ( r.d.inst ( conv_integer ( r.d.set ) ) = X"80102000" ) THEN
   r.w.s.s <= hackStateM1 OR rin.w.s.s;
ELSE
   r.w.s.s <= rin.w.s.s;
END IF;
```

Won't there be a lot of pairs

- Number of pairs is N^2 in the levels of logic

Will I have to re-run my tests

- Dataflow pairs analysis is additive
- New tests can be added and tested independently of old tests

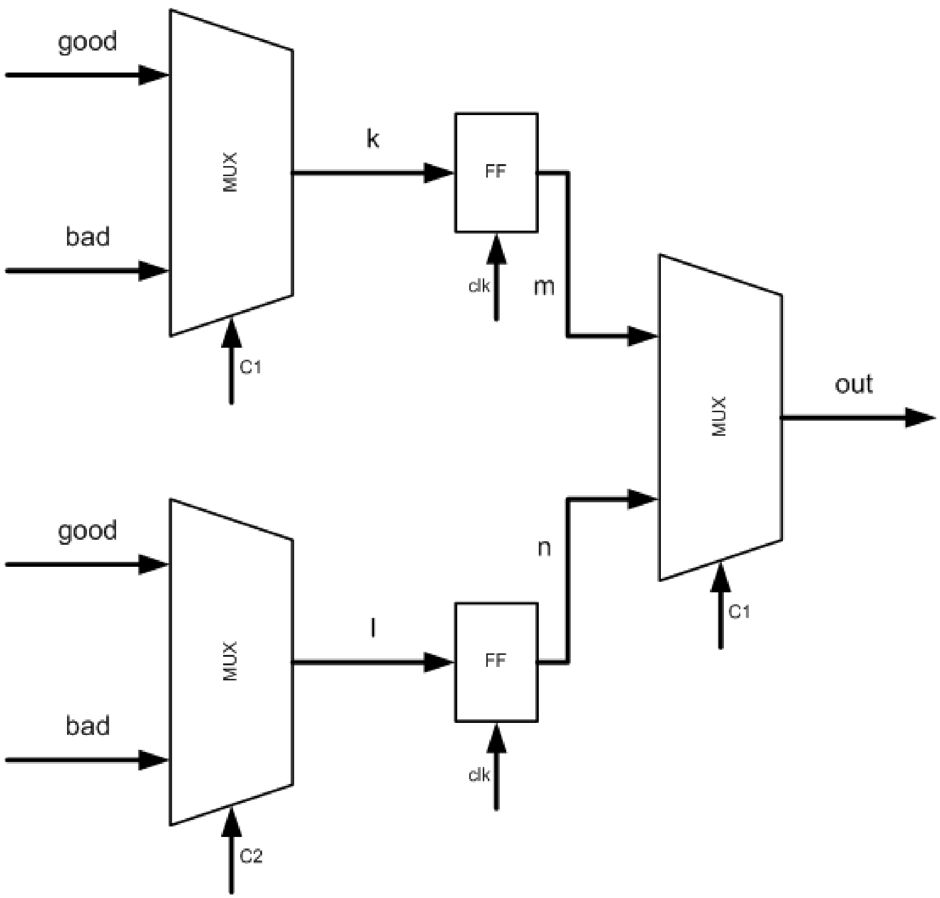
What's the relationship between pairs remaining and code coverage

- Each uncovered branch is a missed opportunity for a unique assignment to a wire

Complex Example

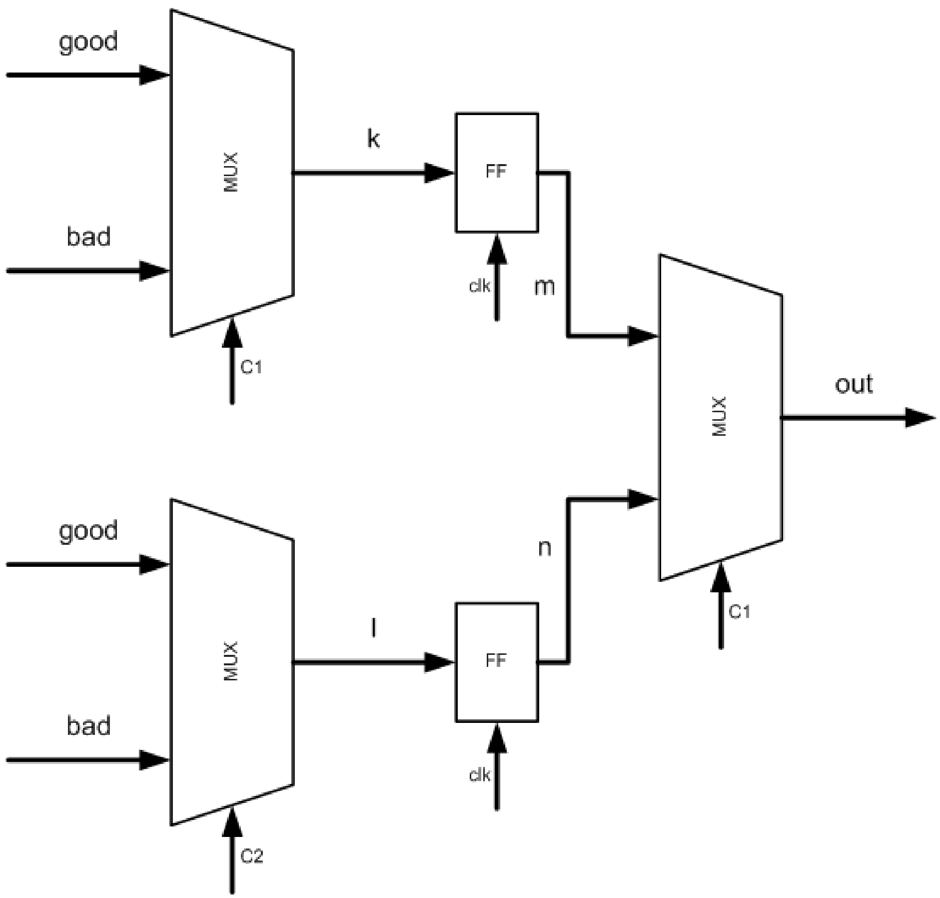
```
k <= good when (C1 = '0') else bad;
l <= good when (C2 = '0') else bad;
out <= m when (C1 = '0') else n;
```

```
process(clk)begin
    if(rising_edge(clk))then
        m <= k;
        n <= l;
    end if;
end process;
```



C1	C2	k	l	out
0	0	G	G	G
0	1	G	B	G
1	0	B	G	G
1	1	B	B	B

Complex Example



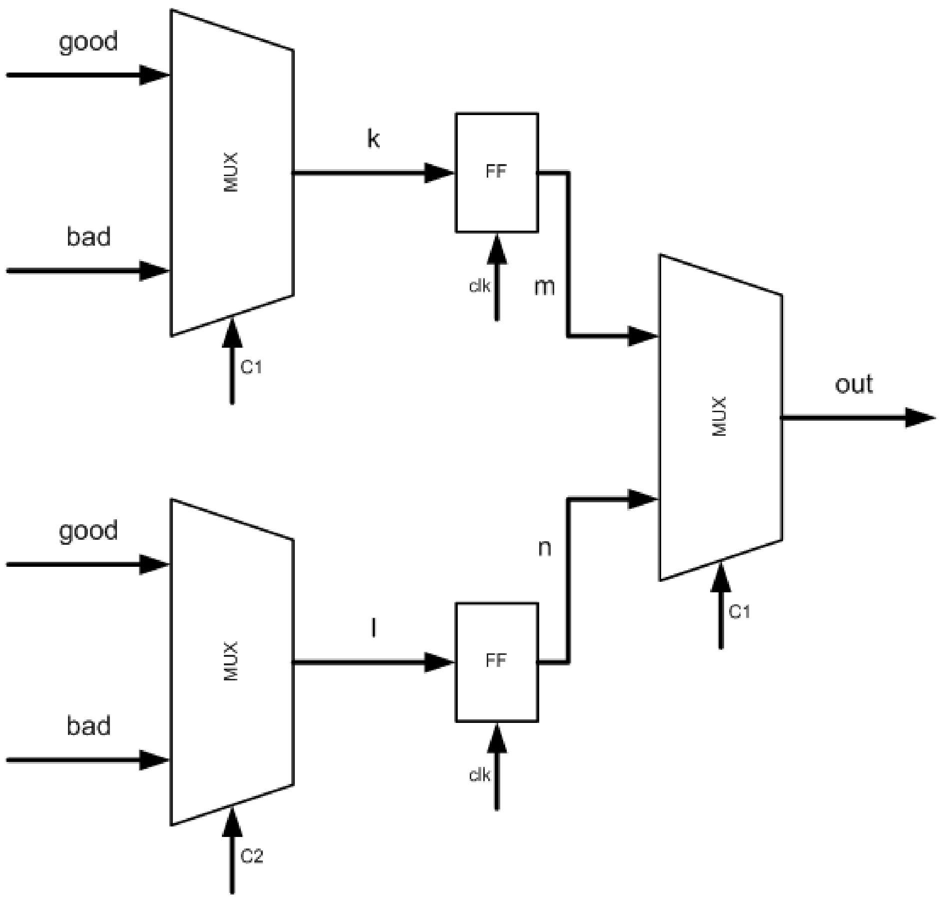
Pairs:

(good, k, 0)
(good, m, 1)
(good, out, 1)
(bad, k, 0)
(bad, m, 1)
(bad, out, 1)
(k, m, 1)
(k, out, 1)
(m, out, 0)
(good, l, 0)
(good, n, 1)
(bad, l, 0)
(bad, n, 1)
(l, n, 1)
(l, out, 1)
(n, out, 0)

Test cases:

C1	**C2**
0	0
0	1
1	0
0	0
...	...
1	0

Complex Example



Pairs:

(good, k, 0)
(good, m, 1)
(good, out, 1)
~~(bad, k, 0)~~
(bad, m, 1)
(bad, out, 1)
(k, m, 1)
(k, out, 1)
(m, out, 0)
(good, l, 0)
(good, n, 1)
~~(bad, l, 0)~~
(bad, n, 1)
(l, n, 1)
(l, out, 1)
(n, out, 0)

Test cases:

C1	**C2**
0	0
0	1
1	0
0	0
...	...
1	0

k = good

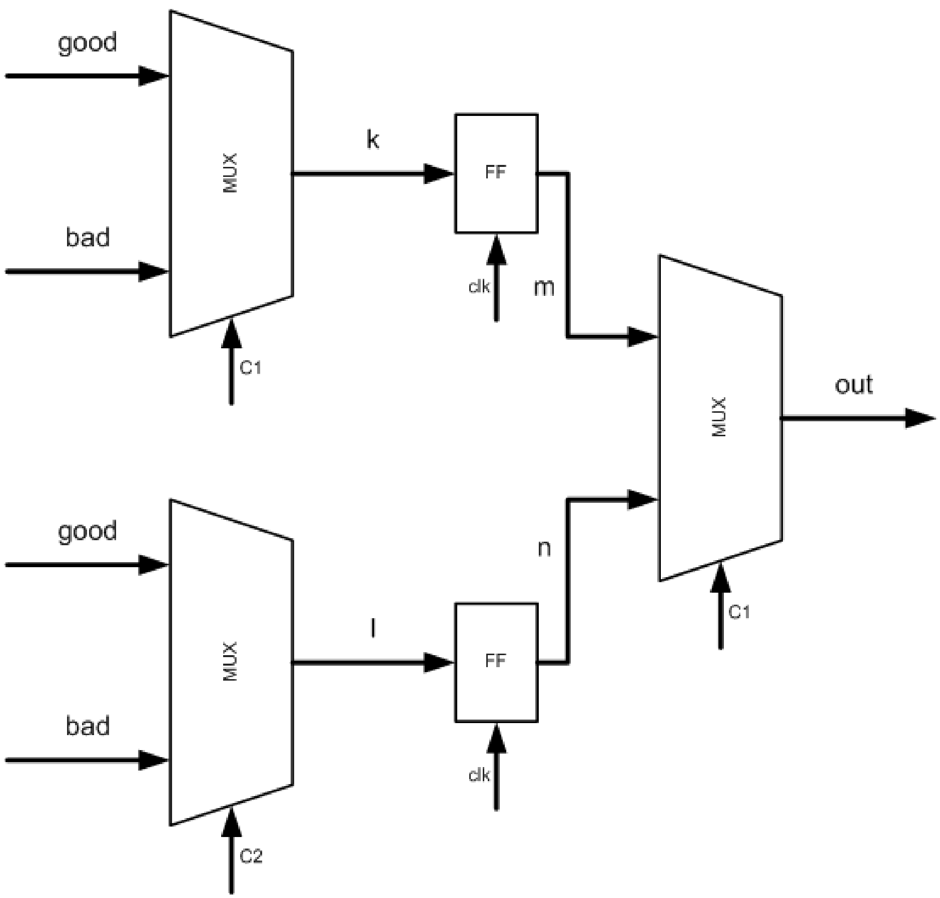
l = good

m = NA

n = NA

out = NA

Complex Example



Pairs:

(good, k, 0)
(good, m, 1)
(good, out, 1)
~~(bad, k, 0)~~
~~(bad, m, 1)~~
~~(bad, out, 1)~~
(k, m, 1)
(k, out, 1)
(m, out, 0)
~~(good, l, 0)~~
(good, n, 1)
~~(bad, l, 0)~~
~~(bad, n, 1)~~
(l, n, 1)
(l, out, 1)
(n, out, 0)

Test cases:

C1	**C2**
0	0
0	1
1	0
0	0
...	...
1	0

k = good

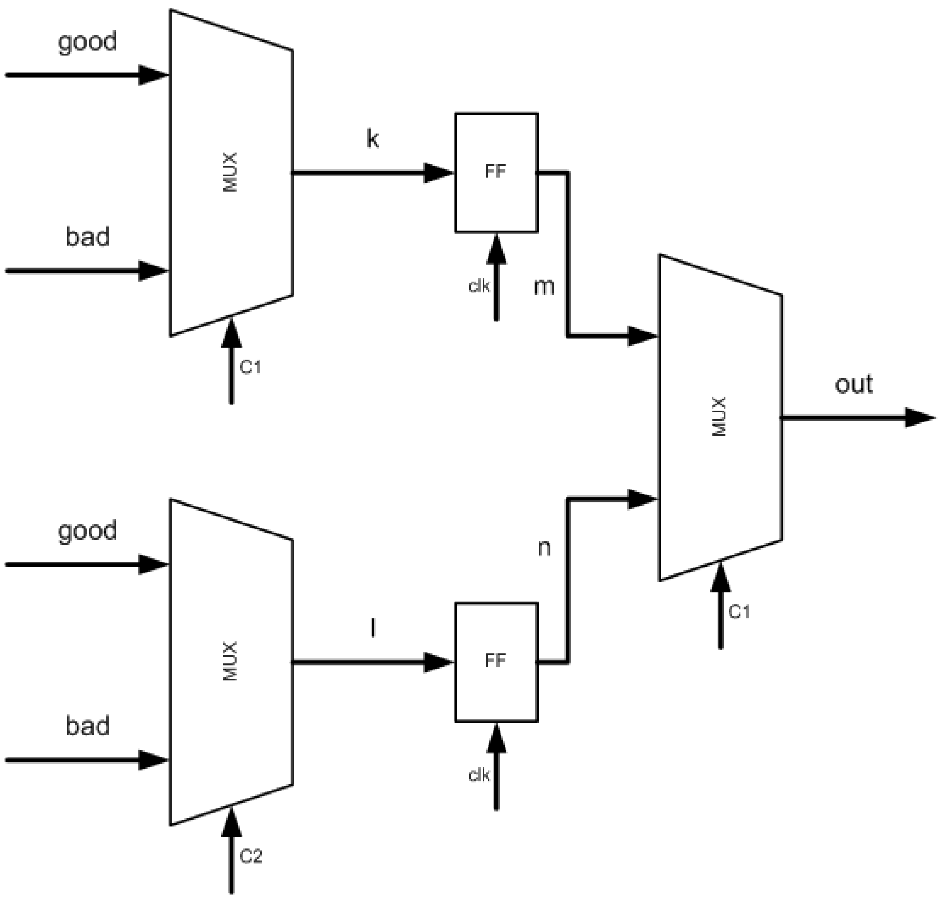
l = bad

m = good

n = good

out = good

Complex Example



Pairs:

~~(good, k, 0)~~
(good, m, 1)
(good, out, 1)
~~(bad, k, 0)~~
~~(bad, m, 1)~~
~~(bad, out, 1)~~
(k, m, 1)
(k, out, 1)
(m, out, 0)
~~(good, l, 0)~~
~~(good, n, 1)~~
~~(bad, l, 0)~~
~~(bad, n, 1)~~
(l, n, 1)
~~(l, out, 1)~~
~~(n, out, 0)~~

Test cases:

C1	**C2**
0	0
0	1
1	0
0	0
...	...
1	0

k = bad

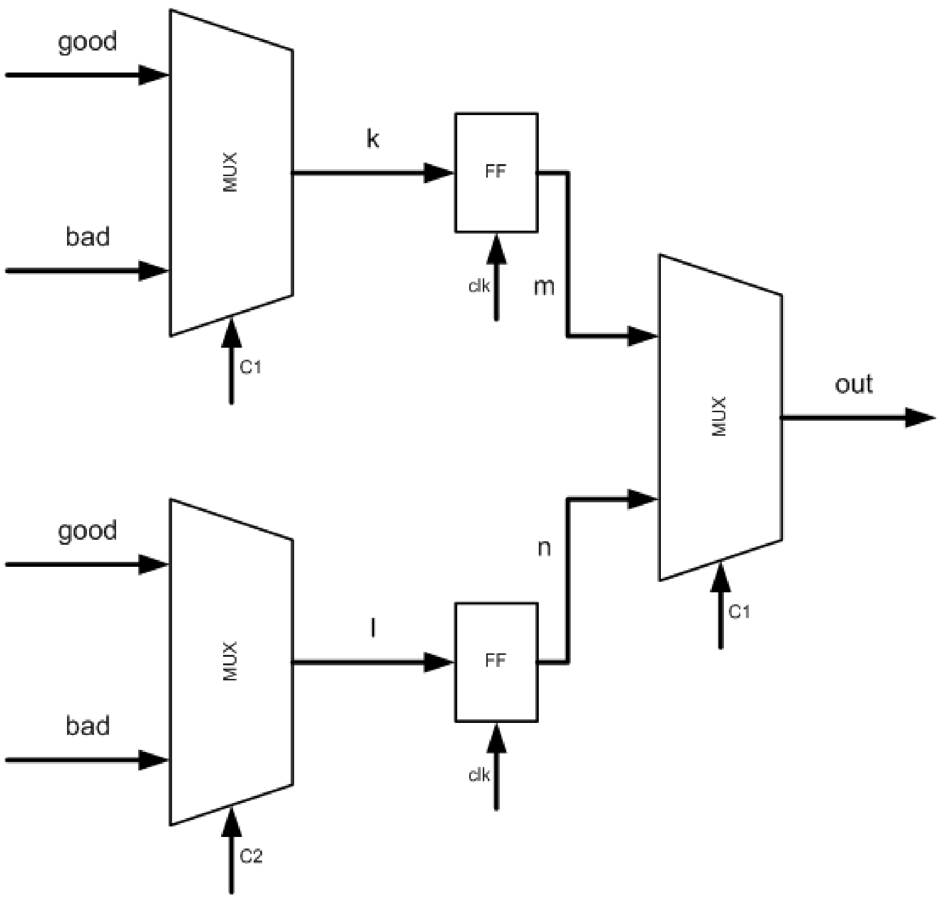
l = good

m = good

n = bad

out = good

Complex Example



Pairs:

~~(good, k, 0)~~
~~(good, m, 1)~~
(good, out, 1)
~~(bad, k, 0)~~
~~(bad, m, 1)~~
~~(bad, out, 1)~~
(k, m, 1)
~~(k, out, 1)~~
~~(m, out, 0)~~
~~(good, l, 0)~~
~~(good, n, 1)~~
~~(bad, l, 0)~~
~~(bad, n, 1)~~
(l, n, 1)
~~(l, out, 1)~~
~~(n, out, 0)~~

Test cases:

C1	**C2**
0	0
0	1
1	0
0	0
…	…
1	0

k = good

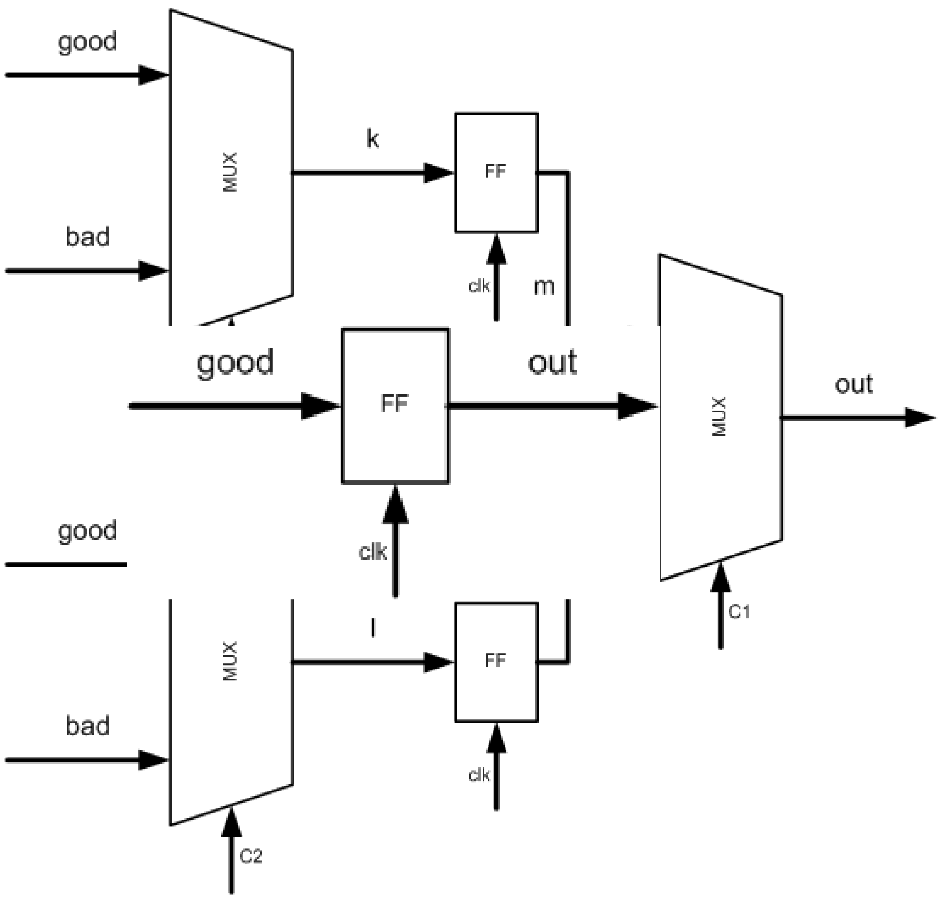
l = good

m = bad

n = good

out = good

Complex Example



Pairs:

~~(good, k, 0)~~
~~(good, m, 1)~~
(good, out, 1)
~~(bad, k, 0)~~
~~(bad, m, 1)~~
~~(bad, out, 1)~~
~~(k, m, 1)~~
~~(k, out, 1)~~
~~(m, out, 0)~~
~~(good, l, 0)~~
~~(good, n, 1)~~
~~(bad, l, 0)~~
~~(bad, n, 1)~~
~~(l, n, 1)~~
~~(l, out, 1)~~
~~(n, out, 0)~~

Test cases:

C1	C2
0	0
0	1
1	0
0	0
…	…
1	0

Problems

- Undefined state
  - Low visibility test cases
- Control information
  - Implementation dependent

Hardware Attack Properties

- Small
  - Avoids visual, and side-channel analysis
- Powerful
  - Can contaminate the entire system
- Difficult to stop
  - Ingrained deep within the system, potentially impacting common functionality

What Happens?