How to build **efficient** HW that **verifiably** prevents illegal information flows?

# Secure HDLs

- Idea: add security annotations to hardware description language
  - **SecVerilog** = Verilog + security types [ASPLOS'15, ASPLOS'17, DAC'17]
  - **ChiselFlow** = Chisel + security types [CCS'18]. Enforces nonmalleable hardware-level downgrading.

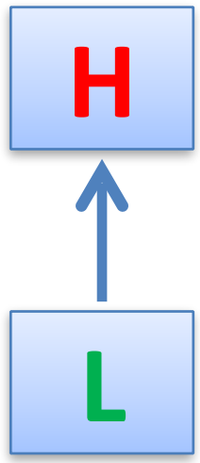
# Shared HW Leaks Information

- Data cache
  - AES [Osvik et al.'05, Bernstein'05, Gullasch et. al.'11]
  - RSA [Percival'05]
- Instruction cache [Aciiçmez'07]
- Computation unit [Z. Wang&Lee'06]
- Memory controller [Wang&Suh'12]
- On-chip network [Wang et al.'14]

# Threat Model

- Attacker sees contents of public HW state at each clock tick

  (synchronous logic)

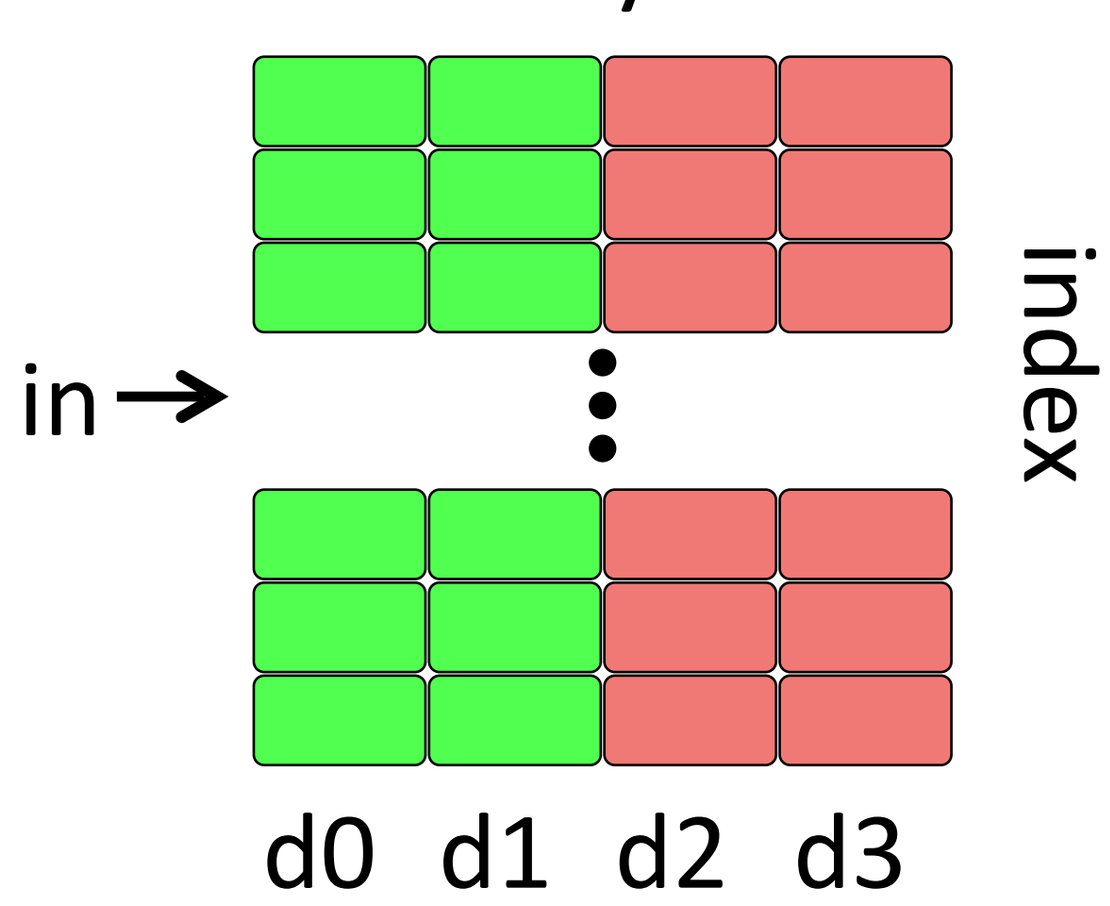


# Statically partitioned cache

A 4-way cache in Verilog

```
reg[31:0]      d0[256],d1[256];
reg[31:0]      d2[256],d3[256];
wire[7:0]      index;
wire[1:0]      way;
wire[31:0]     in;

...
case (way)
  0: begin d0[index]=in; end
  1: begin d1[index]=in; end
  2: begin d2[index]=in; end
  3: begin d3[index]=in; end
endcase
...
```



# SecVerilog

= Verilog + security labels

*Partitioned* cache

```
reg[31:0]{L}   d0[256],d1[256];
reg[31:0]{H}   d2[256],d3[256];
wire[7:0]{L}   index;
wire[1:0]{L}   way;
wire[31:0]     in;

...
case (way)
  0: begin d0[index]=in; end
  1: begin d1[index]=in; end
  2: begin d2[index]=in; end
  3: begin d3[index]=in; end
endcase
...
```

Annotations on variable declarations

- General
- Few annotations
- Verify HW design as-is

# Static labels ⇒ no resource sharing?

```
reg[31:0]{L}   d0[256],d1[256];
reg[31:0]{H}   d2[256],d3[256];
wire[7:0]{L}   index;
wire[1:0]{L}   way;
wire[31:0]     in;

...
case (way)
   0: begin d0[index]=in; end
   1: begin d1[index]=in; end
   2: begin d2[index]=in; end
   3: begin d3[index]=in; end
endcase
...
```

label?

When `way` = 0 or 1, `in` has label **L**

When `way` = 2 or 3, `in` has label **H**

# SecVerilog

- Verilog + ***dependent*** security labels

An example of partitioned cache

```
reg[31:0]{L}   d0[256],d1[256];
reg[31:0]{H}   d2[256],d3[256];
wire[7:0]{L}   index;
wire[1:0]{L}   way;
wire[31:0] {Par (way)}    in;

...
case (way)
  0: begin d0[index]=in; end
  1: begin d1[index]=in; end
  2: begin d2[index]=in; end
  3: begin d3[index]=in; end
endcase
...
```

Resource "in" shared across security labels

Using type-level function:

```
Par(0) = Par(1) = L
Par(2) = Par(3) = H
```

**Less HW needed for secure designs**

# A permissive yet sound type system

Soundness

A well-typed HW design provably enforces *low-security observational determinism*

**L** info. at each clock tick leaks no **H** info.

# Soundness challenges

- Label channels [ASPLOS'15]
- Statically preventing implicit declassification and endorsement [DAC'17]

# Label Channels

```
reg{L}      p;
reg{H}      s;
reg{LH(x)} x;
if (s) begin x = 1; end
if (x==0) begin
    p = 0;
end
```

Type-level function:

`LH(0)=L`  `LH(1)=H`

Change of label leaks information

When p = 1,
s = 0

| p | x |
|---|---|
| 1 | 0 |

| p | x |
|---|---|
| 1 | 0 |

| p | x |
|---|---|
| 0 | 0 |

When p = 1,
s = 1

| p | x |
|---|---|
| 1 | 0 |

| p | x |
|---|---|
| 1 | 1 |

| p | x |
|---|---|
| 1 | 1 |

p = s !

# No-Sensitive-Upgrade

## [Austin&Flanagan'09]

"No update to public variable in secret context"

NSU rejects secure designs

From a real processor design

```
reg{H}      hit2, hit3;
reg[1:0]{Par(way)} way;
if (hit2||hit3)
  way ⇐ hit2 ? 2 : 3;
else
  way ⇐ 2;
```

(incorrectly) rejected

Label of way is always **H** after branch

# Solution: definite assignment

No update to public variable in secret context,
***if the variable is not updated in all branches***

```
reg{H}      hit2, hit3;
reg[1:0]{Par(way)} way;
if (hit2||hit3)
 way ⇐ hit2 ? 2 : 3;

else
 way ⇐ 2;
```

(correctly) accepted

Also more permissive than flow-sensitive systems [Hunt&Sands'06, Russo&Sabelfeld'10]

# Precision of dependent labels

```
reg[31:0]{L}   d0[256],d1[256];
reg[31:0]{H}   d2[256],d3[256];
wire[7:0]{L}   index;
wire[1:0]{L}   way;
wire[31:0] {Par (way)}    in;

...
case (way)
  0: begin d0[index]=in; end
  1: begin d1[index]=in; end
  2: begin d2[index]=in; end
  3: begin d3[index]=in; end
endcase
...
```

Type-level function:

Par(0)=Par(1)=**L**

Par(2)=Par(3)=**H**

**Par(way)** ⊑ **P**?

# Predicate generation

P(c) : a predicate that holds before c executes

```
reg[31:0]{L}   d0[256],d1[256];
reg[31:0]{H}   d2[256],d3[256];
wire[7:0]{L}   index;
wire[1:0]{L}   way;
wire[31:0] {Par (way)}    in;

...
case (way)
   0: begin d0[index]=in; end
   1: begin d1[index]=in; end
   2: begin d2[index]=in; end
   3: begin d3[index]=in; end
endcase
...
```

$P(c)$: (way = 0)

Type-level function:

Par(0)=Par(1)=**P**

Par(2)=Par(3)=**S**

Par(way) ⊑ L

when way=0?

Soundness

Permissiveness

Type system

Other analyses

Variables not always updated

Predicate generation

Typing obligations discharged using Z3 SMT solver.

# Verified MIPS processor

Rich ISA: runs OpenSSL with off-the-shelf GCC

Classic 5-stage in-order pipeline

– Typical pipelining techniques

  - data hazard detection
  - stalling
  - data bypassing/forwarding

# Overhead of SecVerilog

- Verification time:

  2 seconds for complete MIPS processor

- Designer effort
  - Annotation burden:
    one label/variable declaration (mostly inferable, as shown in forthcoming work)
  - Imprecision leads to little extra logic:
    27 LoC to establish necessary invariants

# Overhead of secure processor

- Added HW resources
- Performance overhead on SW

# Overhead of verification

Believed secure but not type-checked

| | Unverified | Verified | Overhead |
|---|---|---|---|
| *Delay w/ FPU (ns)* | *4.20* | *4.20* | *0%* |
| *Delay w/o FPU (ns)* | *1.67* | *1.66* | *-0.6%* |
| *Area (μ²)* | *401420* | *402079* | *0.2%* |
| *Power (mW)* | *575.6* | *575.6* | *0%* |

**Verification overhead is small!**

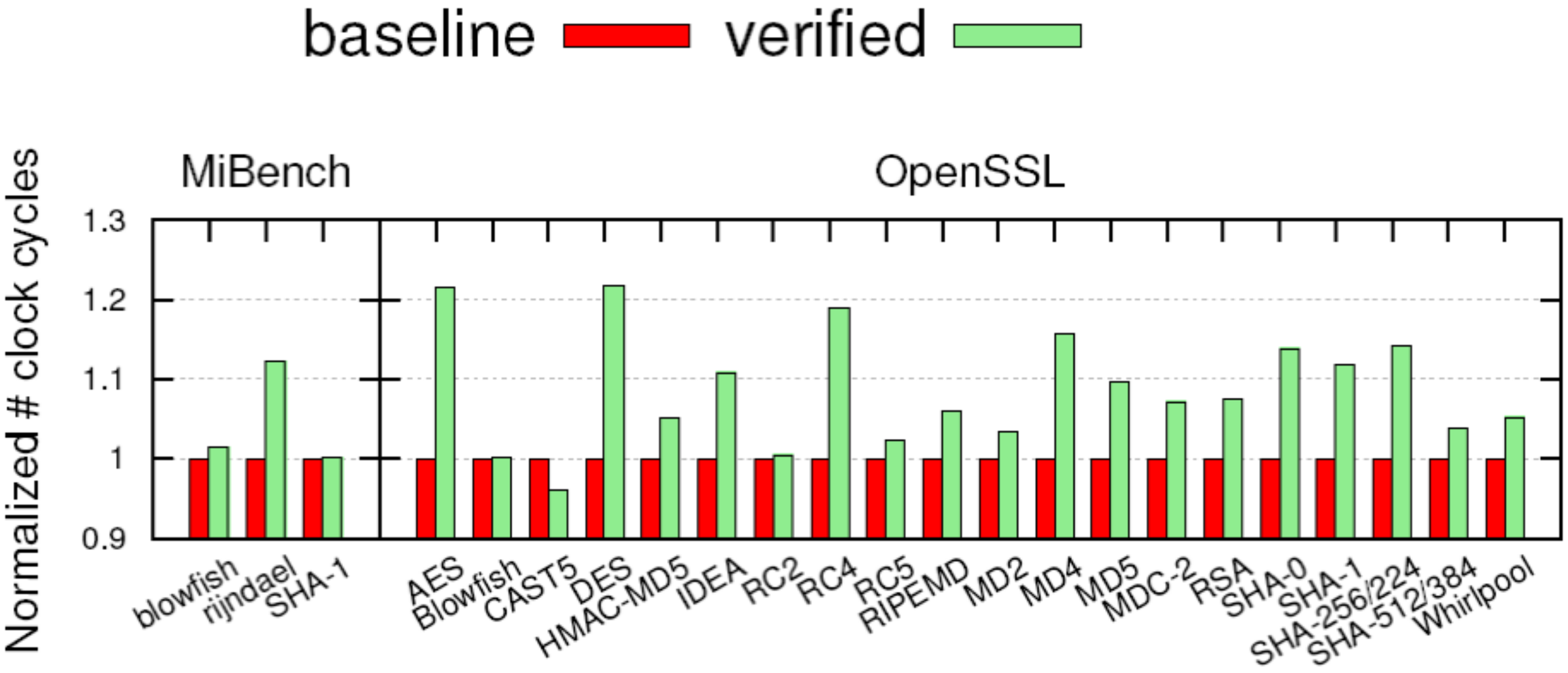
# Overhead of secure processor (HW)

unmodified, insecure

| | Baseline | Verified | Overhead |
|---|---|---|---|
| *Delay w/ FPU (ns)* | *4.20* | *4.20* | *0%* |
| *Delay w/o FPU (ns)* | *1.64* | *1.66* | *1.2%* |
| *Area ($\mu^2$)* | *399400* | *402079* | *0.7%* |
| *Power (mW)* | *575.5* | *575.6* | *0.02%* |

**Enabled by the SecVerilog type system**

# SW-level overhead




**9% overhead on average**

**same cache area ⇒ smaller effective cache**

# Unavoidable leakage

- Program execution time can depend on secrets:

```
int{H} nsecrets;
boolean{L} done;
for (i = 0; i < nsecrets; i++) {…}
done = true;
```

- Idea: **mitigate** timing leakage dynamically [CCS'10, CCS'11, PLDI'12] — asymptotically bound information leakage