FaCT: A DSL for Timing-Sensitive Computation

Sunjay Cauligi, Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad Wahby,
John Renner, Benjamin Gregoire, Gilles Barthe, Ranjit Jhala, Deian Stefan

Presented by Mengjia Yan

MIT 6.888 Fall 2020

Based on slides from Sunjay Cauligi

Goal: Constant-Time Code

- Constant-time code: timing is independent of secrets
  - Variable-time instruction
  - Memory accesses
  - Conditional branches
  - Early termination

```
if (sec)
    x = a;
} else {
    x = b;
}
```

Motivation: Constant-Time Code is Messy

- Existing techniques include using bitmasks, CMOVs, ORAM, etc.
- The problem:
  - Manually optimized code is messy/unreadable/difficult to reason about correctness
  - Automatically obfuscated code incurs high performance overhead

```
if (sec)
    x = a;
} else {
    x = b;
}
```

```
x = (sec & a) | (~mask & b)
```

```
(sec) CMOV x = a
(!sec) CMOV x = b
```

Rane et. al. Raccoon: closing digital side-channels through obfuscated execution. SEC'15

Threat Model

- Attacker can observe execution time of target programs
- Not concretely stated in the paper
  - Instruction execution “trace” should be independent from secrets
- However, execution time is determined by micro-arch states
  - Thus, miss a computer architecture model, characterized by which kinds of instructions can leak information and which can not, e.g., arithmetic instructions

Overview

- A DSL for writing readable constant-time code

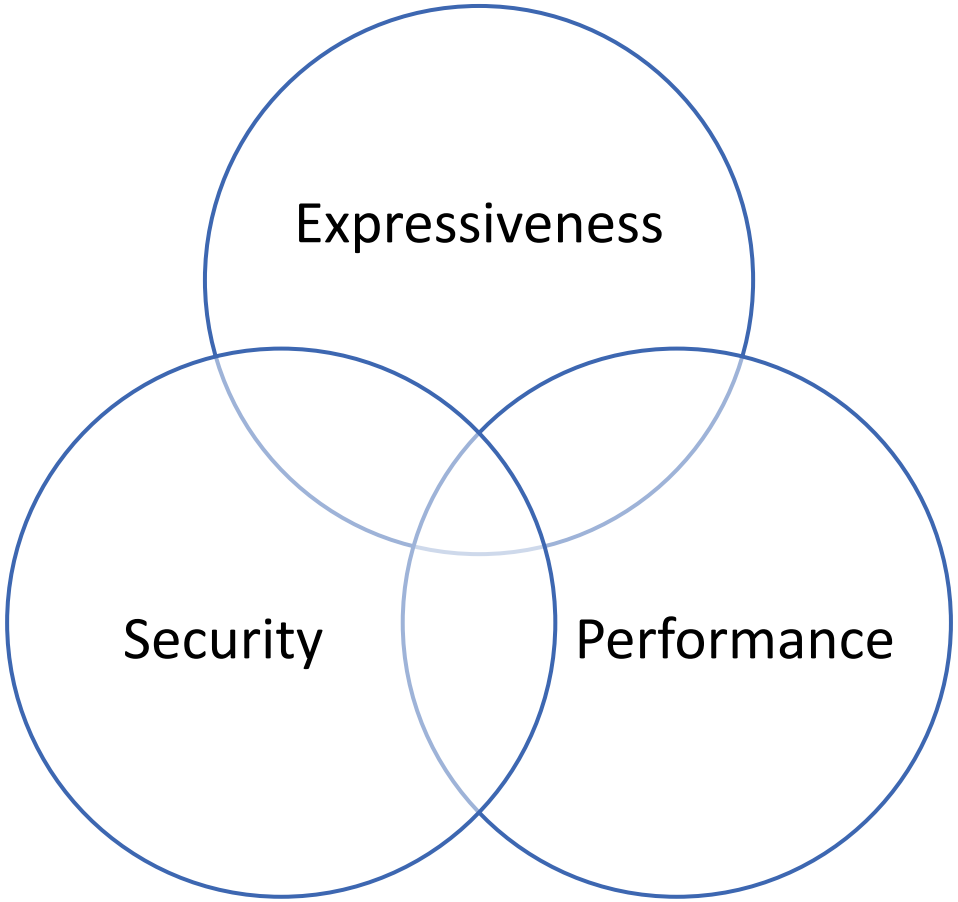
- Transform **secret control flow** to constant-time
  - Transform code that leaks secret via early return, conditional branch
  - Reject programs that leak secret via memory accesses, loop iterations, variable-time instructions

- Ensure transformations can be performed safely

A DSL Trade-offs Among

An example:

To address the imprecision problem of static information flow analysis, remove pointers and disallow recursive typed references



Strengths (Potential Long-term Impacts)

- Provide a great abstraction
  - For SW developers, easy to write constant-time programs
  - For compiler developers, use different techniques to achieve the constant-time goal
    - **ctselect** compiles to a series of bitmasks or the CMOV instruction on x86_64
  - For HW people, performance optimization for execution on public data
- Well-defined typing systems for information flow tracking and formal verification
- A user study to show how easy to write programs using FaCT

A Controversial Contribution

- Reject programs that leak secret via memory accesses, loop iterations, variable-time instructions
  - Put the pressure on programmers. What about AES? Is it really a good trade-off?
  - How much time is spent on manually fixing these problems?

O(1)

```
x = buffer[secret_index];
```

O(n)

```
for (uint32 i = 0 to len buffer) {
    if (i == secret_index) {
        x = buffer[i];
    }
}
```

Limitations/Questions

- Impacts of compiler optimizations of FaCT generated code
  - Security evaluation using *deduct* is not sufficient
  - More information about generated binary sizes may help reason about the performance improvements

- It would be helpful to elaborate more on the trade-offs/reasons for picking the specific design choice in the paper

Reparaz et al. Dude, is my code constant time? DATE'17

FaCT Technique Details

Explicit Secrecy and Information Flow Tracking

- How to handle `st(sec_val, pub_addr)` ?

```
secret uint32 decrypt(
        secret uint32 key,
        public uint32 msg) {

    if (key > 40) {
        ...
    }


    ...

}
```

```
secret uint32 decrypt(
        secret uint32 key,
        public uint32 msg) {

    if (key > 40) {
        ...
    }


    ...

}
```

Type system detects leaks via...

- Conditional branches
- Early termination
- Function side effects

FaCT transforms these

- Memory access patterns
- Direct assignment
- …

FaCT disallows these

Transform Secret Conditionals

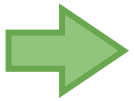
```
if (s) {
    x = 40;
} else {
    x = 19;
    y = x + 2;
}
```

```
x = -s & 40 | (s-1) & x;
```

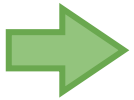
```
x = (s-1) & 19 | -s & x;
y = (s-1) & (x + 2) | -s & y;
```

Transform Secret Returns

```
if (s) {
  return 40;
}
```



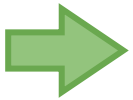
```
if (s) {
  if (!done) {
    rval = 40;
    done = true;
  }
}
```



```
rval = (-s & (done-1)) & 40 | ...
done = (-s & (done-1)) & true | ...
```

Transform Conditional Functions

```
void foo(secret mut uint32 x) {
    x = 5;
}
...
if (sec) {
    foo(x);
}
```



```
void foo(secret mut uint32 x, secret bool
callCtx) {
        x = ctselect(callCtx, 5, x);
}
...
foo(x, sec);
```

Unsafe transformations

```
if (j < secret_len) {
    x = arr[j];
}
```

```
x = -(j < secret_len) & arr[j]
      | ((j < secret_len)-1) & x;
```

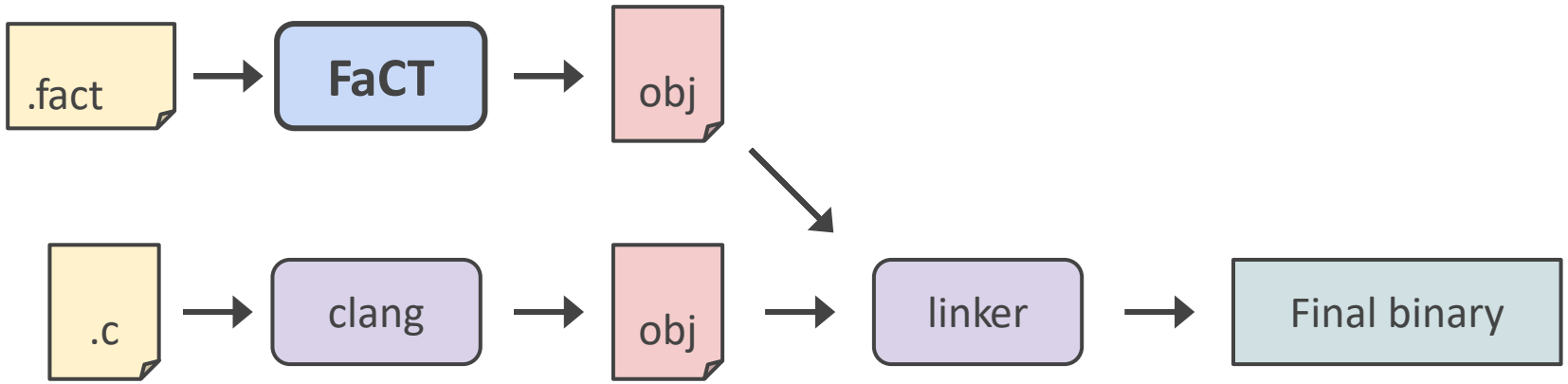
What if j > len arr?

Out of bounds access!

Check for out-of-bounds accesses; Solve constraints using Z3

Porting code to FaCT

- Rewrite the whole library
- Rewrite a function (and callees)
- Rewrite a chunk of code



Real Code Needs Escape Aatches

- **Declassify** secrets to public
  - secretbox:

```
if (!declassify(crypto_verify(...))
  return false;
```

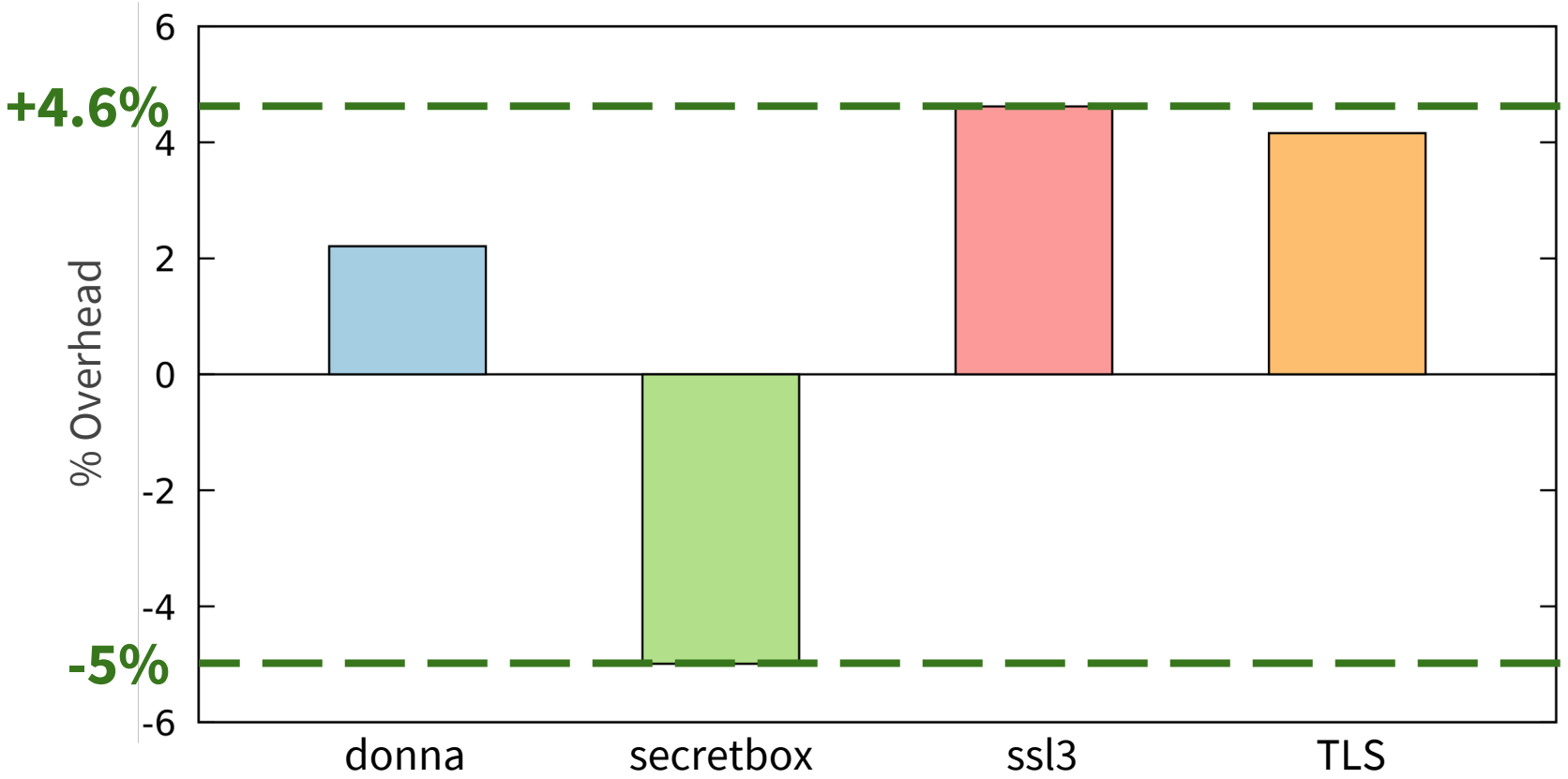
  - TLS:

```
b = pmac[declassify(i)];
```

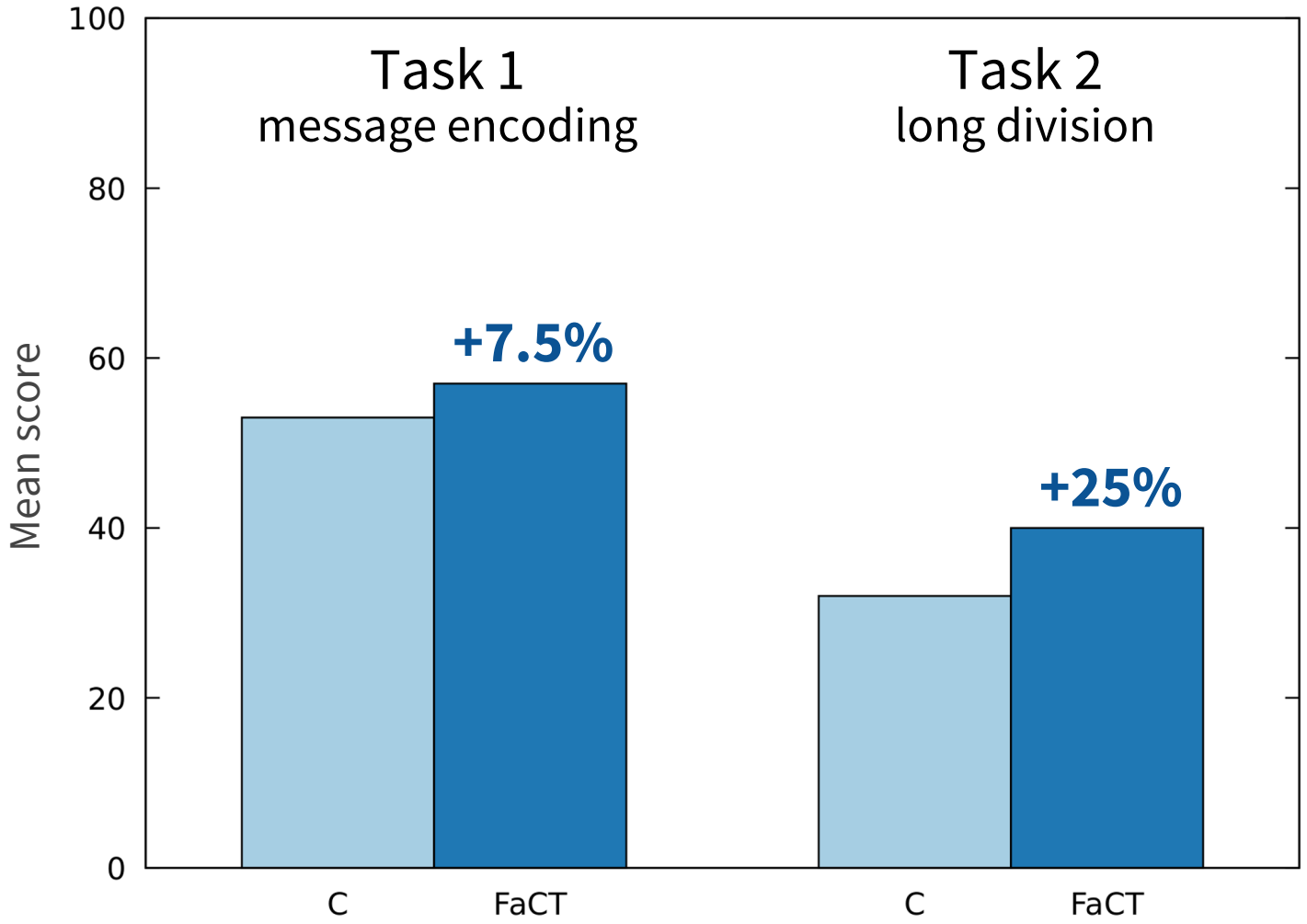
- **Assume** constraints for solver
  - Function preconditions
  - Invariants for mutable variables
- **Extern** function declarations
  - OpenSSL: AES + SHA1 implementations

Performance Evaluation

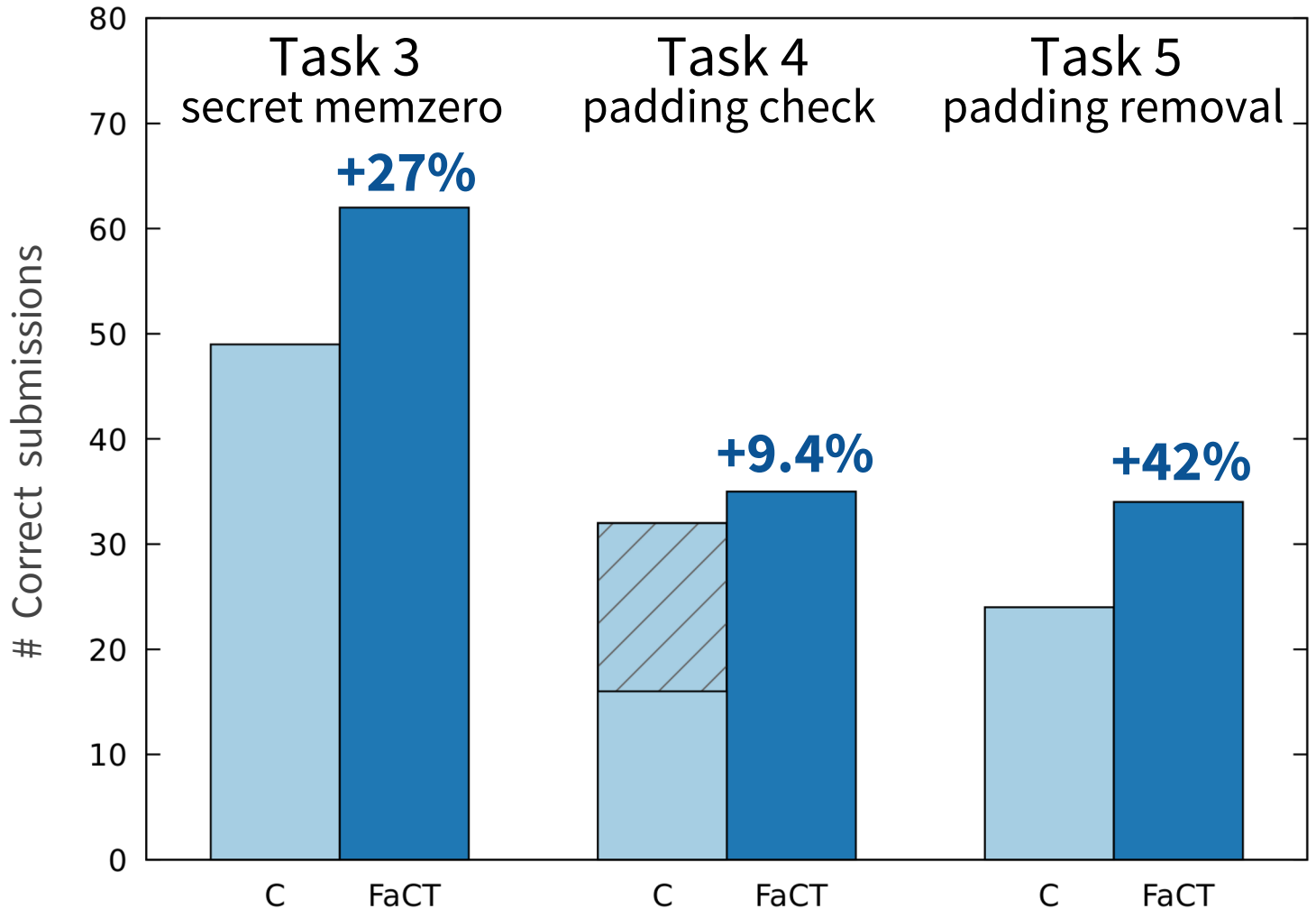
- Optimized with **same optimization flags**
- **Empirically tested** to be constant-time



Understanding constant-time code



Writing constant-time code



Discussion Questions on HW/SW

- Given modern computers have execution units that may not be constant time (specifically division), even a static flow of instructions may not execute with constant total time. What would it take to make sure said execution units operate in a constant time? Division is rare in crypt, so maybe just avoiding it altogether?

- What other processor optimizations exist that will make constant-time operation hard or impossible?

- If a given piece of code is made timing-insensitive, is it possible for power side-channels to still be present?

Discussion Questions on Code Transformation

- Would a lower level solution to the constant time problem be more effective?
- Could we further extend such constant-time reasoning to the optimizer to formally verify the entire compilation flow?
- Is there a more efficient way for the front-end compiler to operate than return statements -> conditionals and then conditionals -> constant time code?

Discussion Questions on Usage

- Are there any cryptographic constructs which are unable to be expressed in FaCT?

- Has there been any further user studies done? If so, what have they shown? If not, what could we expect to see?

- How well do secrets propagate through the type system in practice? For example, if I as an inexperience cryptographer produce a cipher where I mark my salt, my key, and my plaintext as secret, is this sufficient? Is it overkill?