FaCT: A DSL for Timing-Sensitive Computation

Sunjay Cauligi, UC San Diego

Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad Wahby, John Renner, Benjamin Gregoire, Gilles Barthe, Ranjit Jhala, Deian Stefan
What does this code do?

```plaintext
for (i = 0; i < n; i++) {
    d |= x[i] ^ y[i];
}
return (1 & ((d - 1) >> 8)) - 1;
```
What does *this* code do?

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```plaintext
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```java
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}

return 0;
```
It compares two buffers.

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```java
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```plaintext
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```java
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```java
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```python
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```java
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```java
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
} return 0;
```
It compares two buffers.

```java
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```

<table>
<thead>
<tr>
<th>pwd:</th>
<th>guess:</th>
<th>5ms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pwd:</th>
<th>guess:</th>
<th>8ms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

pwd: `guess:`
It compares two buffers.

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```
It compares two buffers.

```plaintext
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```

Exiting early based on contents → leak!
Must not exit early

for (i = 0; i < n; i++) {
  if (x[i] != y[i])
    return -1;
}
return 0;

for (i = 0; i < n; i++) {
  d ^= x[i] ^ y[i];
}
return (1 & ((d - 1) >> 8)) - 1;
Must not exit early

for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;

for (i = 0; i < n; i++) {
    d |= x[i] ^ y[i];
}
return (1 & ((d - 1) >> 8)) - 1;
Must not exit early

**Constant-time code**

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;

for (i = 0; i < n; i++) {
    d |= x[i] ^ y[i];
}
return (1 & ((d - 1) >> 8)) - 1;
```
Must not exit early

**Constant-time code**
Timing is independent of secrets

```plaintext
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```

```plaintext
for (i = 0; i < n; i++) {
    d |= x[i] ^ y[i];
}
return (1 & ((d - 1) >> 8)) - 1;
```
for (j = 0; j < md_block_size; j++) {
    uint8_t b = data[j];
    uint8_t is_past_c = is_block_a & constant_time_ge_8_s(j, c);
    uint8_t is_past_cp1 = is_block_a & constant_time_ge_8_s(j, c + 1);
    b = constant_time_select_8(is_past_c, 0x80, b);
    b = b & ~is_past_cp1;
    b &= ~is_block_b | is_block_a;
    if (j >= md_block_size - md_length_size) {
        b = constant_time_select_8(is_block_b,
                                    length_bytes[j - (md_block_size - md_length_size)], b);
    }
    block[j] = b;
}
Constant-time code is messy

```c
for (j = 0; j < md_block_size; j++) {
    uint8_t b = data[j];
    uint8_t is_past_c = is_block_a & constant_time_ge_8_s(j, c);
    uint8_t is_past_cp1 = is_block_a & constant_time_ge_8_s(j, c + 1);
    b = constant_time_select_8(is_past_c, 0x80, b);
    b = b & ~is_past_cp1;
    b &= ~is_block_b | is_block_a;
    if (j >= md_block_size - md_length_size) {
        b = constant_time_select_8(is_block_b, length_bytes[j - (md_block_size - md_length_size)], b);
    }
    block[j] = b;
}
```
Constant-time code is hard to write

OpenSSL padding oracle attack

Constant-time code is hard to write

OpenSSL padding oracle attack

Constant-time code is hard to write

Lucky 13 timing attack

Constant-time code is hard to write

Further refinements

Removing all measurable timing differences
Goal: Write readable code

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```

```c
for (i = 0; i < n; i++) {
    d |= x[i] ^ y[i];
}
return (1 & ((d - 1) >> 8)) - 1;
```
Goal: Write readable code

```c
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```

```c
for (i = 0; i < n; i++) {
    d |= x[i] ^ y[i];
}
return (1 & ((d - 1) >> 8)) - 1;
```
Goal: Write readable code

for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;

FaCT

for (i = 0; i < n; i++) {
    d |= x[i] ^ y[i];
}
return (1 & ((d - 1) >> 8)) - 1;
Goal: Write readable code

for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;

Transforms readable code into constant-time code
Transforming to constant-time

● What to transform?
● How to transform?
● What *not* to transform?
● Evaluation
Transforming to constant-time

- **What to transform?**
- **How to transform?**
- **What *not* to transform?**
- **Evaluation**
Transform everything?

```java
if (secret) {
    x = 19;
}
```
Transform everything?

```java
if (secret) {
    x = 19;
} else {
    x = -(secret & 19) | ((secret - 1) & x);
}
```
Transform everything?

```c
if (secret) {
    x = 19;
}
```

```
x = -secret & 19 | (secret-1) & x;
```

Slower but necessary
Transform everything?

```java
if (secret) {
    x = 19;
}

if (public) {
    y = 42;
}

x = -secret & 19 | (secret-1) & x;

y = -public & 42 | (public-1) & y;
```

Slower but necessary
Transform everything?

```java
if (secret) {
    x = 19;
}

if (public) {
    y = 42;
}

x = -secret & 19 | (secret-1) & x;

y = -public & 42 | (public-1) & y;
```

Slower but necessary

Slower and unnecessary!
Transform everything?

if (secret) {
    x = 19;
}

if (public) {
    y = 42;
}

Only transform if code leaks secret values

\[
x = \neg \text{secret} \& 19 \mid (\text{secret} - 1) \& x;
\]

\[
y = \neg \text{public} \& 42 \mid (\text{public} - 1) \& y;
\]

Slower but necessary

Slower and unnecessary!
Explicit secrecy in the type system

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

    if (key > 40) {
        ...
    }

    ...

    ...

    }
```
Explicit secrecy in the type system

```c
secret uint32 decrypt(
    secret uint32 key,
    public uint32 msg) {
    if (key > 40) {
        ...
    }
    ...
}
```
Explicit secrecy in the type system

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

    if (key > 40) {
        ...
    }

    We can detect secret leakage!
    ...
}
```
Type system detects leaks via...

- Conditional branches
- Early termination
- Function side effects
- Memory access patterns
- Direct assignment
- ...

FaCT: A DSL for Timing-Sensitive Computation
Type system detects leaks via...

- Conditional branches
- Early termination
- Function side effects
- Memory access patterns
- Direct assignment
- ...

FaCT transforms these
Type system detects leaks via...

- Conditional branches
- Early termination
- Function side effects
- Memory access patterns
- Direct assignment
- ...

FaCT transforms these

FaCT disallows these
Transforming to constant-time

- What to transform?
- How to transform?
- What *not* to transform?
- Evaluation
Transforming to constant-time

- What to transform?
- **How to transform?**
- What *not* to transform?
- Evaluation
Transforming control flow

- Conditional branches
- Early termination
- Function side effects
Transforming control flow

- Conditional branches
- Early termination
- Function side effects
Transform secret conditionals

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

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

```plaintext
x = -s & 40 | (s-1) & x;
```
Transform secret conditionals

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

```
x = -(s) & 40 \| (s-1) \& x;
```
Transform secret conditionals

```latex
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 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;
```
Secret returns are conditionals too

```java
if (s) {
    return 40;
}
```
Secret returns are conditionals too

if (s) {
    return 40;
}

if (s) {
    if (!done) {
        rval = 40;
        done = true;
    }
    
    return rval;
}
Secret returns are conditionals too

```
if (s) {
    if (!done) {
        rval = 40;
        done = true;
    } 
    return 40;
}
```
Secret returns are conditionals too

if (s) {
    return 40;
}

if (s) {
    if (!done) {
        rval = 40;
        done = true;
    }
}
Secret returns are conditionals too

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

if (s) {
    if (!done) {
        rval = 40;
        done = true;
    }
}
```
Transforming to constant-time

- What to transform?
- How to transform?
- What \textit{not} to transform?
- Evaluation
Transforming to constant-time

- What to transform?
- How to transform?
- What \textit{not} to transform?
- Evaluation
Not all transformations are good

- May produce **inefficient** code
- May produce **unsafe** code
Not all transformations are good

- May produce **inefficient** code
- May produce **unsafe** code

**Type system rejects** such programs
Inefficient transformations

```c
x = buffer[secret_index];
```
Inefficient transformations

```c
x = buffer[secret_index];

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

0(1)  
x = buffer[secret_index];

0(n)  
for (uint32 i from 0 to len buffer) {
    if (i == secret_index) {
        x = buffer[i];
    }
}
Inefficient transformations

0(1)

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

0(n)

```c
for (uint32_t i = 0; i < len_buffer; i++) {
  if (i == secret_index) {
    x = buffer[i];
  }
}
```
Inefficient transformations

\[
x = \text{buffer}[\text{secret_index}];
\]

0(1) \quad \text{for} \quad \begin{array}{c}
\text{uint32} \quad i \quad \text{from} \quad 0 \quad \text{to} \quad \text{len} \quad \text{buffer} \end{array} \quad \begin{array}{c}
\quad \text{if} \quad (i = \text{secret_index}) \quad \{ \\
\quad \quad x = \text{buffer}[i]; \\
\quad \}
\end{array}
\]

O(n) \quad \text{Reject if transformation is inefficient}
Unsafe transformations

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

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

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

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

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

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

What if \( j > \text{len arr} \)?
Unsafe transformations

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

What if $j > \text{len arr}$?

**Out of bounds** access!
Type system checks safety

Check for **out-of-bounds accesses**

**Solve constraints** using Z3

Path sensitive **except secret branches**

**Reject** if transformation is **unsafe**
Type system checks safety

Check for **out-of-bounds accesses**

Solve constraints using Z3

Path sensitive **except secret branches**

Reject if transformation is **unsafe**
Type system checks safety

Check for **out-of-bounds accesses**

*Solve constraints* using Z3

Path sensitive *except secret branches*

*Reject* if transformation is *unsafe*
Type system checks safety

Check for **out-of-bounds** accesses

**Solve constraints** using Z3

Path sensitive **except secret branches**

**Reject** if transformation is **unsafe**
Type system checks safety

Check for **out-of-bounds accesses**

**Solve constraints** using Z3

Path sensitive **except secret branches**

**Reject** if transformation is **unsafe**
Transforming to constant-time

● What to transform?
● How to transform?
● What *not* to transform?
● Evaluation
Transforming to constant-time

- What to transform?
- How to transform?
- What *not* to transform?
- Evaluation
Evaluating FaCT

- Can FaCT express real code?

- Is FaCT code as fast as C?

- Is FaCT more readable than C?
Evaluating FaCT

- Can FaCT express real code?
- Is FaCT code as fast as C?
- Is FaCT more readable than C?
Porting code to FaCT

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

- Rewrite the **whole library**
- Rewrite a function (and callees)
- Rewrite a chunk of code
Porting code to FaCT

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

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

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

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

```
.fact
FaCT
.obj
.c
clang
.obj
 linker
Final binary
```
Porting code to FaCT

- Rewrite the whole library: **donna curve25519**
- Rewrite a function (and callees): **libsodium secretbox**
- Rewrite a chunk of code: **OpenSSL ssl3/TLS record verification**
Porting code to FaCT

- Rewrite the whole library: *donna curve25519*
- Rewrite a function (and callees): *libsodium secretbox*
- Rewrite a chunk of code: *OpenSSL ssl3/TLS record verification*

![Bar chart showing lines of code for different libraries and languages.](chart.png)
Real code needs escape hatches

- Declassify
- Assume
- Extern
Real code needs escape hatches

- **Declassify** secrets to public

- Assume

- Extern
Real code needs escape hatches

- **Declassify** secrets to public
  - secretbox: if (!declassify(crypto_verify(...))) return false;

- Assume

- Extern
Real code needs escape hatches

- **Declassify** secrets to public
  - `secretbox`: 
    ```
    if (!declassify(crypto_verify(...))
        return false;
    ```
  - `TLS`: 
    ```
    b = pmac[declassify(i)];
    ```

- **Assume**

- **Extern**
Real code needs escape hatches

- **Declassify** secrets to public
  - secretbox: `if (!declassify(crypto_verify(...))) return false;`
  - TLS: `b = pmac[declassify(i)];`

- **Assume** constraints for solver

- **Extern**
Real code needs escape hatches

- **Declassify** secrets to public
  - secretbox: 
    ```
    if (!declassify(crypto_verify(...))
        return false;
    ```
  - TLS: 
    ```
    b = pmac[declassify(i)];
    ```

- **Assume** constraints for solver
  - Function preconditions

- **Extern**
Real code needs escape hatches

- **Declassify** secrets to public
  - secretbox:  
    ```java
    if (!declassify(crypto_verify(...))
        return false;
    ```
  - TLS:  
    ```java
    b = pmac[declassify(i)];
    ```

- **Assume** constraints for solver
  - Function preconditions
  - Invariants for mutable variables

- **Extern**
Real code needs escape hatches

- **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
Real code needs escape hatches

- **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
Evaluating FaCT

● Can FaCT express real code?

● Is FaCT code as fast as C?

● Is FaCT more readable than C?
Performance vs. C

- Optimized with same optimization flags
- Empirically tested to be constant-time
Performance vs. C

- Optimized with **same optimization flags**
- Empirically tested to be constant-time
Performance vs. C

- Optimized with **same optimization flags**
- **Empirically tested** to be constant-time
Performance vs. C

- Optimized with **same optimization flags**
- **Empirically tested** to be constant-time
Performance vs. C

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

---

![Bar chart showing performance comparison](chart.png)
Evaluating FaCT

- Can FaCT express real code?

- Is FaCT code as fast as C?

- Is FaCT more readable than C?
User study: FaCT vs. C

- 77 undergraduates
- Understanding constant-time code
- Writing constant-time code
User study: FaCT vs. C

- 77 undergraduates
- **Understanding** constant-time code
- **Writing** constant-time code
Understanding constant-time code

- **Task 1**: message encoding
- **Task 2**: long division

Bar chart showing performance comparison between C and FaCT for each task.
Understanding constant-time code

Task 1
message encoding

Task 2
long division

Mean score

+7.5%

+25%

C  FaCT
C  FaCT
Writing constant-time code

![Bar chart showing correct submissions for different tasks and languages]

- Task 3: secret memzero
- Task 4: padding check
- Task 5: padding removal

Languages compared:
- C
- FaCT
Writing constant-time code

- Task 3: secret memzero (+27%)
- Task 4: padding check (+9.4%)
- Task 5: padding removal (+42%)
Writing constant-time code

<table>
<thead>
<tr>
<th>Task 3</th>
<th>Task 4</th>
<th>Task 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>secret memzero</td>
<td>padding check</td>
<td>padding removal</td>
</tr>
<tr>
<td>+27%</td>
<td>+9.4%</td>
<td>+42%</td>
</tr>
</tbody>
</table>

Bar chart showing:
- Task 3: secret memzero, C: 50 submissions, FaCT: 57 submissions, +27%
- Task 4: padding check, C: 14 submissions, FaCT: 15.9 submissions, +9.4%
- Task 5: padding removal, C: 29 submissions, FaCT: 42 submissions, +42%
Evaluating FaCT

- FaCT can express real code
- FaCT code is as fast as C
- FaCT is more readable than C
Summary

● DSL for writing **readable** constant-time code

● **Transform secret control flow** to constant-time

● Ensure transformations **can be performed safely**

https://fact.programming.systems
FACT: A DSL for Timing-Sensitive Computation

Sunjay Cauligi

PLDI 2019
Comparing two buffers in FaCT

```c
secret int32 crypto_verify_n(
    secret uint8[] x,
    secret uint8[] y) {

    assume(len x == len y);
    for (uint64 i from 0 to len x) {
        if (x[i] != y[i]) {
            return -1;
        }
    }
    return 0;
}
```
Message encoding in FaCT

```rust
for (uint64 j from 0 to md_block_size) {
    secret mut uint8 b = 0;
    b = data[j];
    if (is_block_a) {
        if (j == c) {
            b = 0x80;
        } else if (j > c) {
            b = 0;
        }
    }
    if (is_block_b) {
        if (!is_block_a) {
            b = 0;
        } else if (j >= md_block_size - md_length_size) {
            b = length_bytes[j - (md_block_size - md_length_size)];
        }
    }
    block[j] = b;
}
```