Practical Constant-Time Programming

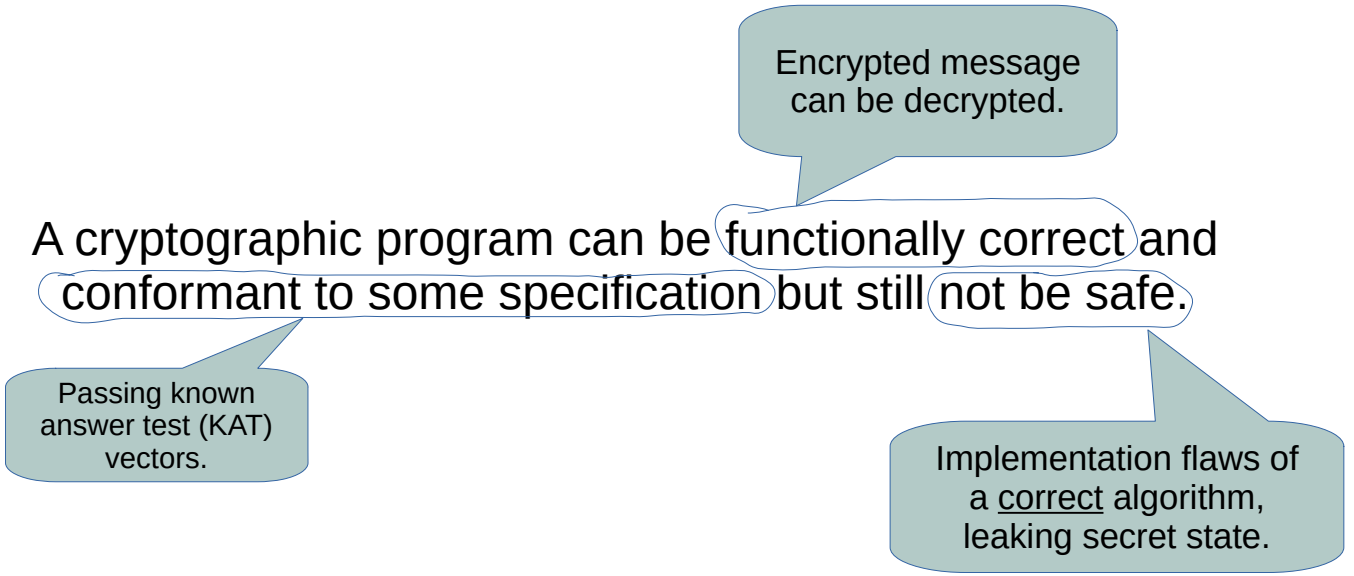
Cedarcrypt 2026, Cyprus
14th July, 2026

Anjan Roy

PhD @ DiS, Radboud University, The Netherlands
SWE @ Avail Project, UAE

Ex-Polygon Labs, Ex-TII

What is constant-timeness? (1/4)



What is constant-timeness? (2/4)

1) Passive observers looking for hints
2) Active attackers injecting faults
3) Compiler turning against us 🥲

What is constant-timeness? (3/4)

A functionally correct and conformant program can still leak its internal secret state, through timing.

You access memory or decide to (not) take a branch based on a secret value!

Instruction latency may depend on operand!

What is constant-timeness? (4/4)

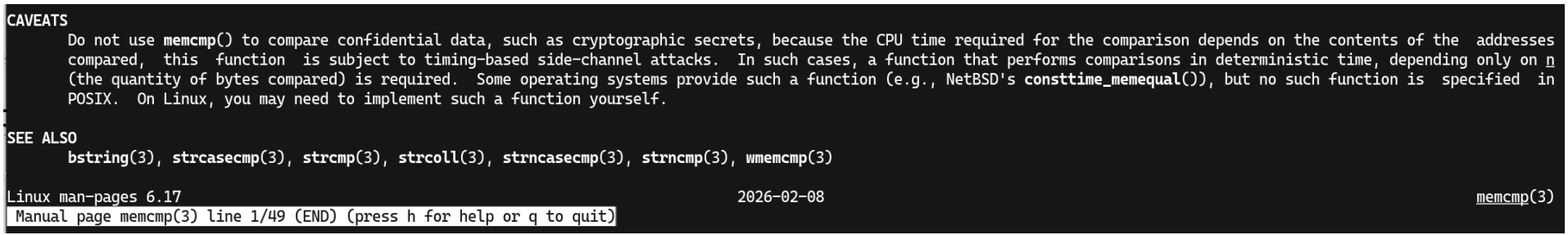
How long it takes to execute an instruction, which branches it takes, which memory addresses it touches – none of them should depend on a secret.

Example: Ascon-AEAD128 `decrypt`

```
54  [[nodiscard]]
55  forceinline constexpr bool
56  decrypt(std::span<const uint8_t, KEY_BYTE_LEN> key,
57          std::span<const uint8_t, NONCE_BYTE_LEN> nonce,
58          std::span<const uint8_t> associated_data,
59          std::span<const uint8_t> cipher,
60          std::span<uint8_t> text,
61          std::span<const uint8_t, TAG_BYTE_LEN> tag)
62  {
63    ascon_perm::ascon_perm_t state{};
64    std::array<uint8_t, TAG_BYTE_LEN> computed_tag{};
65
66    ascon_duplex_mode::initialize(state, key, nonce);
67    ascon_duplex_mode::process_associated_data(state, associated_data);
68    ascon_duplex_mode::process_ciphertext(state, cipher, text);
69    ascon_duplex_mode::finalize(state, key, computed_tag);
70
71    const uint32_t flg = ascon_common_utils::ct_eq_byte_array<TAG_BYTE_LEN>(tag, computed_tag);
72    ascon_common_utils::ct_conditional_memset(~flg, text, 0);
73
74    return static_cast<bool>(flg);
75  }
```

Unverified plaintext must not be released

Why not `memcmp`?



```
CAVEATS
       Do not use memcmp() to compare confidential data, such as cryptographic secrets, because the CPU time required for the comparison depends on the contents of the  addresses
       compared,  this  function  is subject to timing-based side-channel attacks.  In such cases, a function that performs comparisons in deterministic time, depending only on n
       (the quantity of bytes compared) is required.  Some operating systems provide such a function (e.g., NetBSD's consttime_memequal()), but no such function is  specified  in
       POSIX.  On Linux, you may need to implement such a function yourself.

SEE ALSO
       bstring(3), strcasecmp(3), strcmp(3), strcoll(3), strncasecmp(3), strncmp(3), wmemcmp(3)

Linux man-pages 6.17                                                    2026-02-08                                                                    memcmp(3)
 Manual page memcmp(3) line 1/49 (END) (press h for help or q to quit)
```

Convince me!

```
❯ ./build/ct_timing_leak_example
tag = 32 bytes, 16384 comparisons/batch, median of 51 (CPU cycles per comparison)

first mismatch at         naive_memcmp     std::memcmp       ct_memcmp
-------------------------------------------------------------------------
byte 0                             2.1             2.4            38.3
byte 4                             3.9             2.5            36.6
byte 8                             5.9             2.4            36.2
byte 12                            7.5             2.3            36.2
byte 16                            9.6             2.4            36.2
byte 20                           11.8             2.3            36.2
byte 24                           16.5             2.4            36.2
byte 28                           18.6             2.4            36.2
equal (full scan)                 21.4             2.4            36.2
```

On Intel x86_64, with GCC-15 `-O3`, timed with `rdtsc`

Nah, recover it!

```
❯ ./build/ct_timing_attack_example
Recovering a 16-byte secret tag through timing alone.
Brute force: up to 256^16 guesses. Timing: 256*16 = 4096 guessed positions.

INSECURE (naive_memcmp)
  secret    : 761857f5d68a75a732eb8b125d023181
  recovered : 761857f5d68a75a732eb8b125d023181
  => 16/16 bytes recovered

SECURE (subtle::ct_memcmp)
  secret    : 761857f5d68a75a732eb8b125d023181
  recovered : d4adb259b101cf13dd279d0018abc76a
  => 0/16 bytes recovered
```

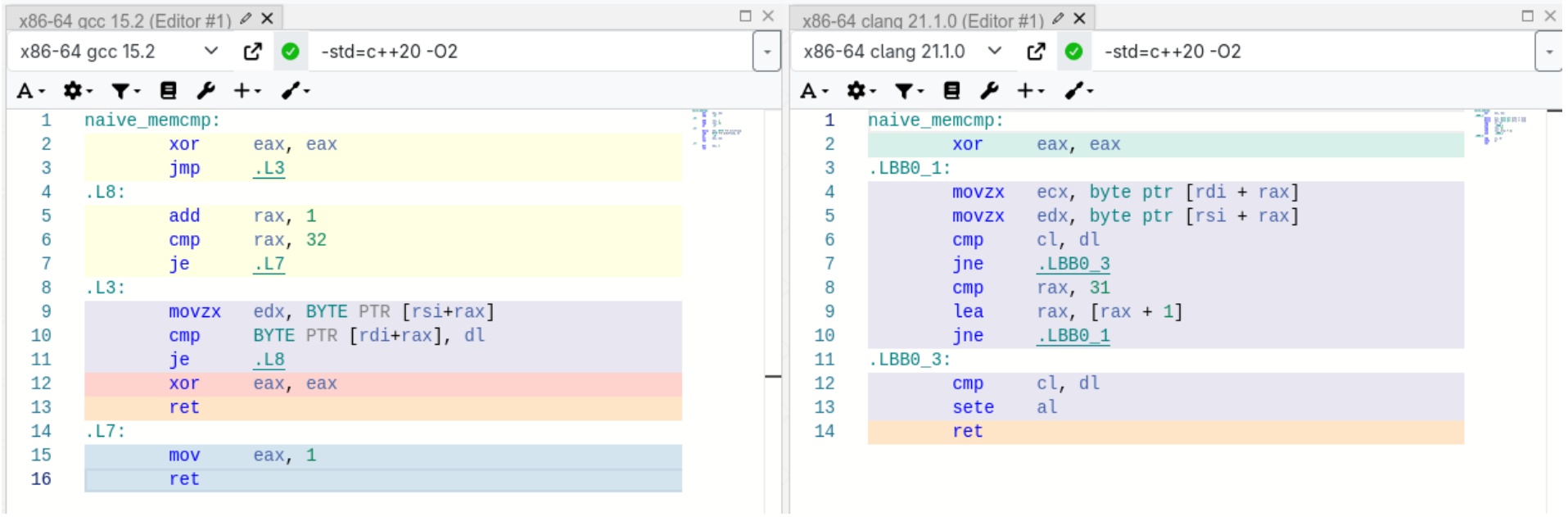
Hm, how?

```
129  template<typename Verify>
130  uint8_t
131  recover_byte(const Verify& verify, std::span<uint8_t, TAG_LEN> guess, size_t pos)
132  {
133    std::array<double, 256> cycles{};
134    cycles.fill(1e30);
135
136    for (size_t round = 0; round < ROUNDS; round++) {
137      for (size_t byte = 0; byte <= std::numeric_limits<uint8_t>::max(); byte++) {
138        guess[pos] = static_cast<uint8_t>(byte);
139        cycles[byte] = std::min(cycles[byte], measure_batch(verify, guess));
140      }
141    }
142
143    const auto* const slowest = std::ranges::max_element(cycles);
144    return static_cast<uint8_t>(slowest - cycles.begin());
145  }
```

Yeah, it is real.

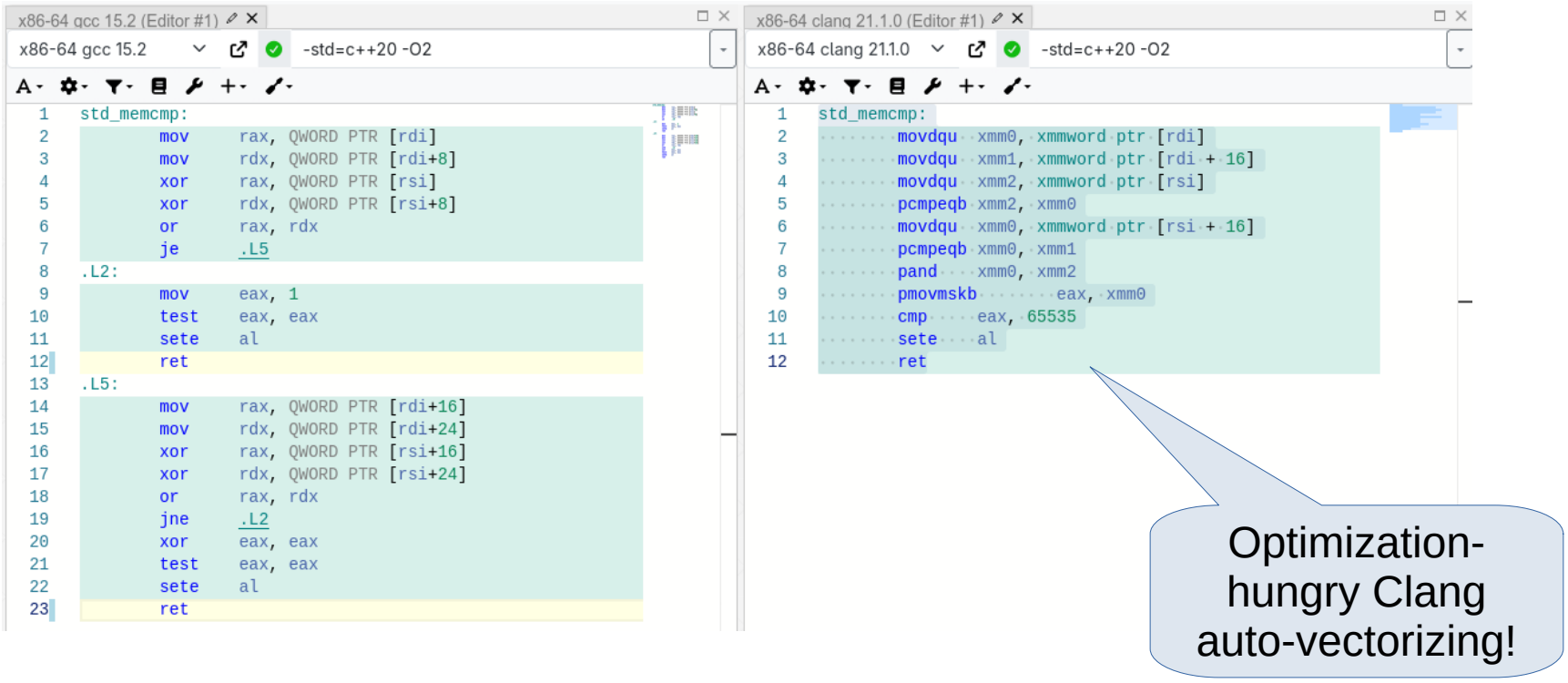
Constant-timeness for cryptographic implementations is important.

How to write `ct_memcmp`? (1/7)



`naive_memcmp`: Early exit. Exploitable. Flawed.

How to write `ct_memcmp`? (2/7)



`std::memcmp`: Old friend GCC exits early!

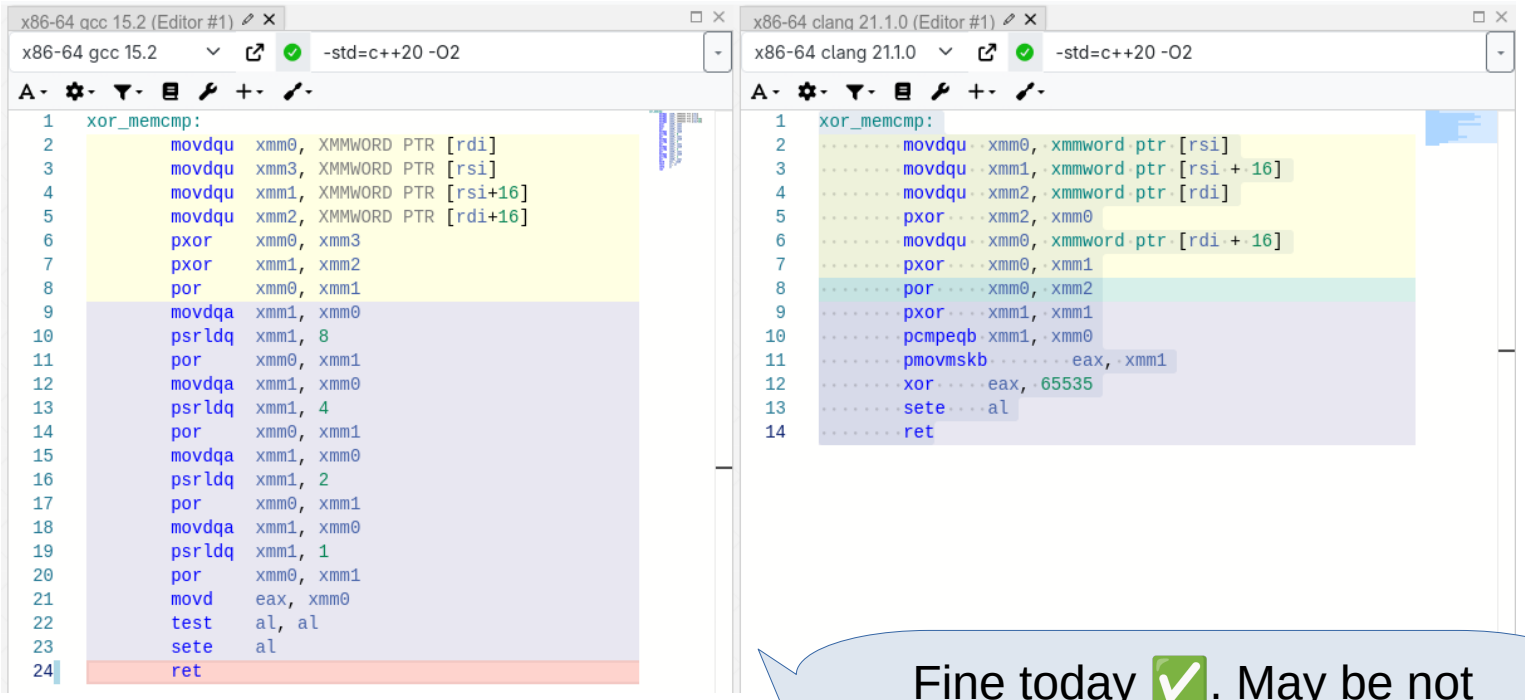
How to write `ct_memcmp`? (3/7)

```
33  extern "C" bool
34  xor_memcmp(std::span<const uint8_t, N> lhs, std::span<const uint8_t, N> rhs)
35  {
36    uint8_t diff = 0;
37    for (size_t i = 0; i < lhs.size(); i++) {
38      diff = static_cast<uint8_t>(diff | (lhs[i] ^ rhs[i]));
39    }
40
41    return diff == 0;
42  }
```

At i = 0, lhs = 1, rhs = 1
diff = 0

At i = 1, lhs = 1, rhs = 2
diff = 3

How to write `ct_memcmp`? (4/7)



Fine today ✅. May be not tomorrow. New compiler optimization-pass might break your “CT” fn.

How to write `ct_memcmp`? (5/7)

```
47  // Given two integers x, y of type operandT, this routine returns true ( if
48  // x == y ) or false ( in case x != y ) testing equality of two values.
49  //
50  // We represent truth value using maximum number that can be represented using
51  // returnT i.e. all bits of returnT are set to one. While for false value, we
52  // set all bits of returnT to zero.
53  template<typename operandT, typename returnT>
54  forceinline constexpr returnT
55  ct_eq(const operandT x, const operandT y)
56    requires(ct_operand<operandT> && std::is_unsigned_v<returnT>)
57  {
58    using Uop = std::make_unsigned_t<operandT>;
59
60    Uop a = static_cast<Uop>(static_cast<Uop>(x) ^ static_cast<Uop>(y));
61    ct_barrier(a);
62
63    const Uop b = static_cast<Uop>(a | static_cast<Uop>(-a));
64    const Uop c = static_cast<Uop>(b >> (std::numeric_limits<Uop>::digits - 1)); // select only MSB
65    const returnT d = static_cast<returnT>(c);
66    const returnT e = static_cast<returnT>(d - static_cast<returnT>(1));
67
68    return e;
69  }      You, 3 years ago • constant-time equality check of two unsigned in…
```

`ct_memcmp`: The future-proof way

How to write `ct_memcmp`? (6/7)

```
27  // Optimization barrier: prevents the compiler from reasoning about `val`,
28  // blocking pattern recognition that would restructure constant-time code into branches
29  // or secret-dependent memory accesses. Generates zero machine instructions.
30  template<typename T>
31  forceinline constexpr void
32  ct_barrier([[maybe_unused]] T& val)
33    requires(std::is_unsigned_v<T>)
34  {
35  #if !defined(SUBTLE_MSAN_ACTIVE_)
36    if (!std::is_constant_evaluated()) {
37  #if defined(_MSC_VER)
38      volatile T* vp = &val;
39      val = *vp;
40  #else
41      asm volatile("" : "+r"(val)); // NOLINT(hicpp-no-assembler)
42  #endif
43    }
44  #endif
45  }       You, 4 months ago • Add compiler barrier for preventing optimizatio…
```

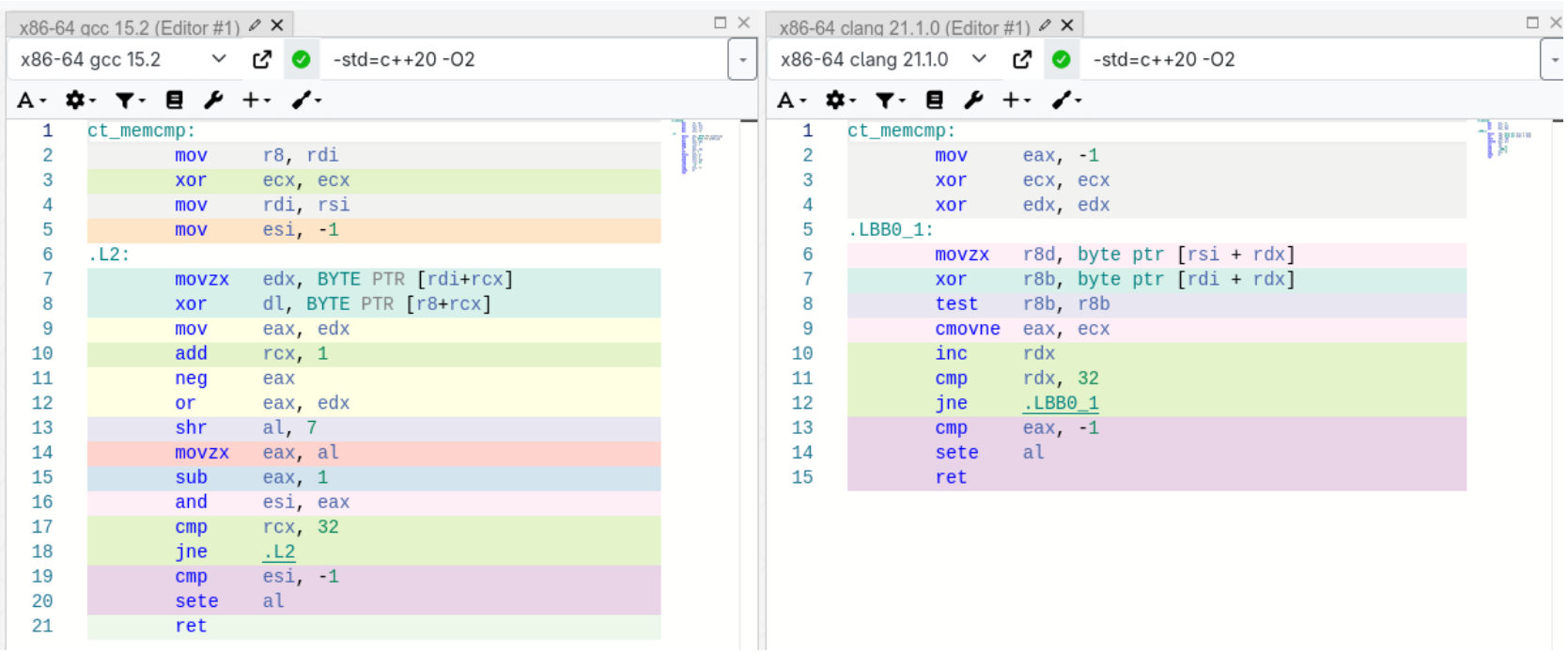
Barrier? Turns off compiler reasoning. At zero runtime cost 🤩

How to write `ct_memcmp`? (7/7)

```
303   // Constant-time comparison of two equal-length spans of integers.
304   // Returns truth value (all bits set) if the spans are element-wise equal,
305   // or false value (all bits zero) otherwise.
306   //
307   // Composes from ct_eq: AND-accumulates per-element equality results.
308   // Each ct_eq call contains an optimization barrier, making its return value
309   // opaque to the compiler — the loop cannot be short-circuited.
310   template<typename operandT, typename returnT, size_t N>
311   forceinline constexpr returnT
312   ct_memcmp(std::span<const operandT, N> lhs, std::span<const operandT, N> rhs)
313     requires(ct_operand<operandT> && std::is_unsigned_v<returnT>)
314   {
315     returnT acc = static_cast<returnT>(~returnT{ 0 });
316     for (size_t i = 0; i < lhs.size(); i++) {
317       acc = static_cast<returnT>(acc & ct_eq<operandT, returnT>(lhs[i], rhs[i]));
318     }
319
320     return acc;
321   }
```

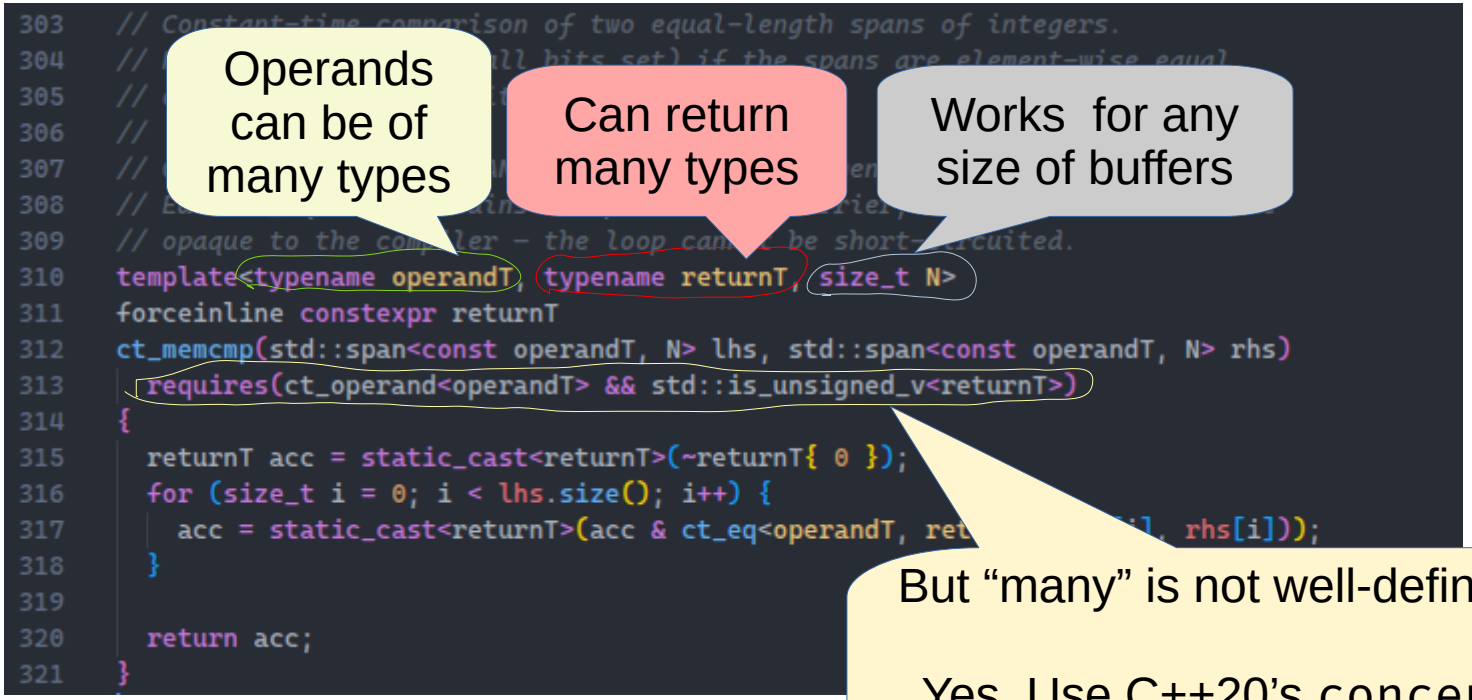
Plug-in. Get `ct_memcmp`.

Wait, is `ct_memcmp` actually CT?



Yes, across compilers and their optimization levels.

And this is ~~nicer~~ flexible




```
303  // Constant-time comparison of two equal-length spans of integers.
304  // [illegible] all bits set) if the spans are element-wise equal.
305  // [illegible]
306  // [illegible]
307  // [illegible]
308  // [illegible]
309  // opaque to the compiler - the loop cannot be short-circuited.
310  template<typename operandT, typename returnT, size_t N>
311  forceinline constexpr returnT
312  ct_memcmp(std::span<const operandT, N> lhs, std::span<const operandT, N> rhs)
313    requires(ct_operand<operandT> && std::is_unsigned_v<returnT>)
314  {
315    returnT acc = static_cast<returnT>(~returnT{ 0 });
316    for (size_t i = 0; i < lhs.size(); i++) {
317      acc = static_cast<returnT>(acc & ct_eq<operandT, ret[illegible]i], rhs[i]));
318    }
319
320    return acc;
321  }
```


Not the slide 😬

That’s all to constraint the compiler

```
21  // An operand type accepted by the constant-time routines below: any standard
22  // integer type -- signed or unsigned, of bit width 8, 16, 32 or 64 -- as well as
23  // the character types, but not bool.
24  template<typename T>
25  concept ct_operand = std::is_integral_v<T> && !std::is_same_v<std::remove_cv_t<T>, bool>;
```

A C++20 concept

Compile-time error messages!

```
 9    std::array<float, 32> arr_a{};
10    std::array<float, 32> arr_b{};
11
12    std::ranges::fill(arr_a, 0);
13    std::ranges::fill(arr_a, 1);
14
15    auto res = subtle::ct_memcmp<float,     // operand array element type
```

ct_timing_leak.cpp 1 of 1 problem

```
No matching function for call to 'ct_memcmp' clang(ovl_no_viable_function_in_call)

subtle.hpp(312, 1): Candidate template ignored: constraints not satisfied [with operandT = float, returnT = uint32_t, N = 32]
subtle.hpp(313, 12): Because 'float' does not satisfy 'ct_operand'
subtle.hpp(25, 22): Because 'std::is_integral_v<float>' evaluated to false
```

```
16                                uint32_t, // return value type
17                                32        // array size
18                                >(arr_a, arr_b);
```

Can’t misuse the API and get runtime errors!

Is `memcmp` the only one that needs to be CT?

No.

`memset`, but conditional

```
62  {
63      ascon_perm::ascon_perm_t state{};
64      std::array<uint8_t, TAG_BYTE_LEN> computed_tag{};
65
66      ascon_duplex_mode::initialize(state, key, nonce);
67      ascon_duplex_mode::process_associated_data(state, associated_data);
68      ascon_duplex_mode::process_ciphertext(state, cipher, text);
69      ascon_duplex_mode::finalize(state, key, computed_tag);
70
71      const uint32_t flg = ascon_common_utils::ct_eq_byte_array<TAG_BYTE_LEN>(tag, computed_tag);
72      ascon_common_utils::ct_conditional_memset(~flg, text, 0);      You, 14 months ago • Ascon a
73
74      return static_cast<bool>(flg);
75  }
```

Such that the condition itself is secret!

It better be constant-time ✌️

How to `memset` conditionally?

```
346  // Given a branch value br ( of type branchT ) holding either truth value (
347  // represented using value of type branchT s.t. all the bits are set to 1 ) or
348  // false value ( represented using value of type branchT s.t. all the bits are
349  // set to 0 ), a span dst of integers and a fill value val, this routine
350  // overwrites every element of dst with val if br is truth value. Otherwise dst
351  // retains its original contents.
352  //
353  // If br takes any value other than these two, this
354  template<typename branchT, typename operandT, si
355  forceinline constexpr void
356  ct_conditional_memset(const branchT br, std::span<operandT, N> dst, const operandT val)
357    requires(std::is_unsigned_v<branchT> && ct_operand<operandT>)
358  {
359    for (size_t i = 0; i < dst.size(); i++) {
360      dst[i] = ct_select<branchT, operandT>(br, val, dst[i]);
361    }
362  }
```

This is secret

Build on `ct_select`

ct_select: A reusable one

```cpp
template<typename branchT, typename operandT>
forceinline constexpr operandT
ct_select(const branchT br, const operandT x, const operandT y)
  requires(std::is_unsigned_v<branchT> && ct_operand<operandT>)
{
  using Uop = std::make_unsigned_t<operandT>;

  const branchT z = br >> (std::numeric_limits<branchT>::digits - 1); // select MSB
  Uop w = static_cast<Uop>(-static_cast<Uop>(z));
  ct_barrier(w);
  const Uop selected = static_cast<Uop>((static_cast<Uop>(x) & w) | static_cast<Uop>(static_cast<Uop>(y) & static_cast<Uop>(~w))); // br ? x : y

  return static_cast<operandT>(selected);
}      You, 3 years ago • constant-time selection ( like ternary operator…
```

result = condition ? a : b;

`ct_conditional_memcpy`: Built on `ct_select`

```
160     // line 9-12 of algorithm 17, in constant-time
161     using kdf_t = std::span<const uint8_t, shared_secret.size()>;
162     const uint32_t cond = ml_kem_utils::ct_memcmp(cipher, std::span<const uint8_t, ctlen>(c_prime));
163     ml_kem_utils::ct_cond_memcpy(cond, shared_secret, kdf_t(g_out_span0), kdf_t(j_out));      You, 4
164
165     ml_kem_utils::secure_zeroize(g_in);
166     ml_kem_utils::secure_zeroize(g_out);
167     ml_kem_utils::secure_zeroize(j_out);
168     ml_kem_utils::secure_zeroize(c_prime);
169   }
```

Fujisaki-Okamoto Transform in ML-KEM (FIPS 203)

What else to build on `ct_select`?

A look-up from a table, where the index to read from is a secret.

Call it `ct_lookup`.

ct_lookup = ct_eq + ct_select

```
364  // Given a secret index idx ( of type indexT ) and a table of integers,
365  // this routine returns table[idx] without ever using idx to address
366  // memory -- defeating cache-timing side channels that a plain table[idx] would expose.
367  //
368  // idx is expected to be in range [0, table.size()); an out-of-range idx matches
369  // no slot and yields a zero result.
370  template<typename indexT, typename operandT, size_t N>
371  forceinline constexpr operandT
372  ct_lookup(const indexT idx, std::span<const operandT, N> table)
373    requires(std::is_unsigned_v<indexT> && ct_operand<operandT>)
374  {
375    using Umask = std::make_unsigned_t<operandT>;
376
377    operandT result = operandT{ 0 };
378    for (size_t j = 0; j < table.size(); j++) {
379      const Umask mask = ct_eq<indexT, Umask>(static_cast<indexT>(j), idx);
380      result = ct_select<Umask, operandT>(mask, table[j], result);
381    }
382
383    return result;
384  }       You, 3 days ago • Add function for constant-time lookup from a ta…
```

`ct_swap`: Secret condition-based swapping

Classic textbook swap without a temporary

x = x ^ y

y = y ^ x
  = y ^ (x ^ y)
  = x ✅

x = x ^ y
  = (x ^ y) ^ x
  = y ✅

```
150  template<typename branchT, typename operandT>
151  forceinline constexpr void
152  ct_swap(const branchT br, operandT& x, operandT& y)
153    requires(std::is_unsigned_v<branchT> && ct_operand<operandT>)
154  {
155    using Uop = std::make_unsigned_t<operandT>;
156
157    Uop mask = static_cast<Uop>(-static_cast<Uop>(br & 1));
158    ct_barrier(mask);
159
160    Uop xu = static_cast<Uop>(x);
161    Uop yu = static_cast<Uop>(y);
162
163    xu = static_cast<Uop>(xu ^ static_cast<Uop>(mask & yu));
164    yu = static_cast<Uop>(yu ^ static_cast<Uop>(mask & xu));
     xu = static_cast<Uop>(xu ^ static_cast<Uop>(mask & yu));
167
168    x = static_cast<operandT>(xu);
169    y = static_cast<operandT>(yu);
   }      You, 3 years ago • implement constant-time swap based on condition…
```

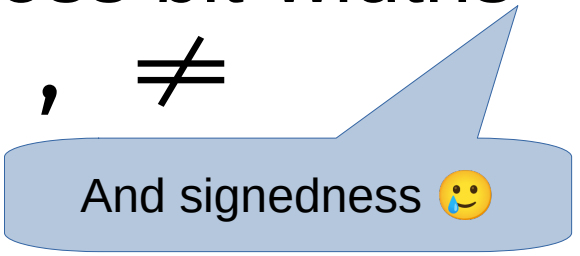
`ct_swap` inside ECDSA KeyGen

Instead use conditional swapping

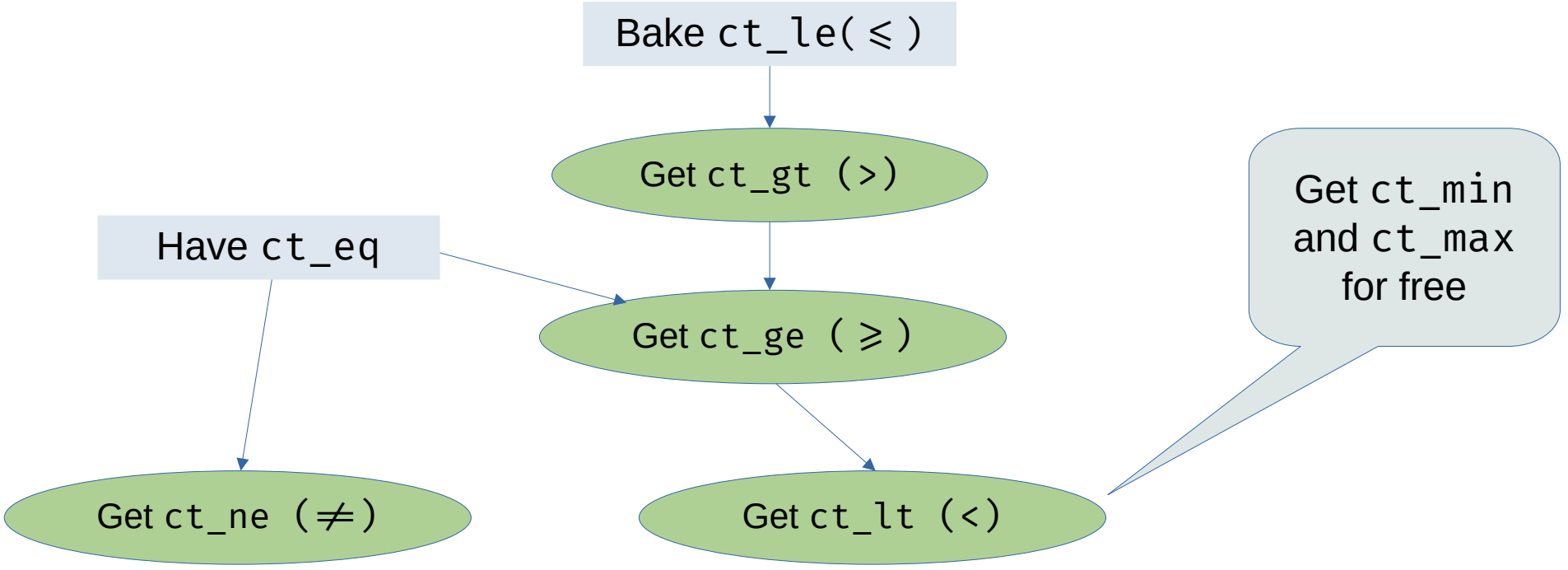
You might need more

Comparing integers across bit-widths

$<$, $>$, $\leqslant$, $\geqslant$, $\neq$



Build using composition



So, that’s all, right?

Not really.

Constant-time arithmetic operations (`+, -, x, /, %`)

In PQC, mostly over small integers and vectors, matrices of them

Elliptic curve cryptography (ECC) needs it over large bitwidth integers!

Do not forget me!
- Falcon (FN-DSA)

Ok, but how to test CT?

Write a lot of tests.

Runtime Analysis

```
30  #ifdef CT_USE_MSAN
31  #include <sanitizer/msan_interface.h>
32  // NOLINTNEXTLINE(cppcoreguidelines-macro-usage)
33  #define CT_POISON(addr, len) __msan_allocated_memory(addr, len)
34  #define CT_BACKEND_NAME "MSan"
35  #else
36  #include <valgrind/memcheck.h>
37  // NOLINTNEXTLINE(cppcoreguidelines-macro-usage)
38  #define CT_POISON(addr, len) VALGRIND_MAKE_MEM_UNDEFINED(addr, len)
39  #define CT_BACKEND_NAME "Valgrind"
40  #endif
```

LLVM Clang compile-time instrumentation

Valgrind runtime instrumentation

Static Analysis

```
==============================================
  clang++ / Release
==============================================
-- The CXX compiler identification is Clang 18.1.3
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.6s)
-- Generating done (0.0s)
-- Build files have been written to: /tmp/build_binsec_clangpp_release
[ 50%] Building CXX object CMakeFiles/subtle_ct_binsec.dir/tests/test_ct_binsec.cpp.o
[100%] Linking CXX executable subtle_ct_binsec
[100%] Built target subtle_ct_binsec
  binsec_ct_swap_span_i8          SECURE
---
Total: 1  Secure: 1  Insecure: 0  Unknown: 0
  RESULT: PASSED
```

Binsec

Case study: conditional memcpy (1/16)

```
131  template<typename branchT, typename operandT>
132  forceinline constexpr operandT
133  ct_select(const branchT br, const operandT x, const operandT y)
134    requires(std::is_unsigned_v<branchT> && ct_operand<operandT>)
135  {
136    // using Uop = std::make_unsigned_t<operandT>;
137
138    if ((br & 1) == 1) {
139      return x;
140    }
141
142    return y;
143
144    // const branchT z = br >> (std::numeric_limits<branchT>::digits - 1); // select MSB
145    // Uop w = static_cast<Uop>(-static_cast<Uop>(z));
146    // ct_barrier(w);
147    // const Uop selected = static_cast<Uop>((static_cast<Uop>(x) & w) | static_cast<Uop>(static_cast<Uop>(y) & static_cast<Uop>(~w))); // br ? x : y
148
149    // return static_cast<operandT>(selected);
150
```

Not CT

Case study: conditional memcpy (2/16)

```
        Start 705: CtConditionalMemcpyTest.Correctness<std::pair<unsigned short,unsigned char>>
690/831 Test #693: CtConditionalMemcpyTest.Correctness<std::pair<short,unsigned short>> .....   Passed    0.34 sec
        Start 706: CtConditionalMemcpyTest.Correctness<std::pair<unsigned int,unsigned char>>
691/831 Test #692: CtConditionalMemcpyTest.Correctness<std::pair<signed char,unsigned short>> ......   Passed    0.38 sec
        Start 707: CtConditionalMemcpyTest.Correctness<std::pair<unsigned long,unsigned char>>
692/831 Test #694: CtConditionalMemcpyTest.Correctness<std::pair<int,unsigned short>> .....   Passed    0.35 sec
        Start 708: CtConditionalMemcpyTest.Correctness<std::pair<unsigned char,unsigned short>>
693/831 Test #691: CtConditionalMemcpyTest.Correctness<std::pair<long,unsigned char>> .....   Passed    0.42 sec
        Start 709: CtConditionalMemcpyTest.Correctness<std::pair<unsigned short,unsigned short>>
```

Not constant-time, but functionally correct ! ✅

Case study: conditional memcpy (3/16)

```
❯ cmake -B build -DCMAKE_CXX_COMPILER=clang++ -DCMAKE_BUILD_TYPE=Release -DSUBTLE_BUILD_TESTS=ON -DSUBTLE_MSAN=ON -DSUBTLE_ENABLE_LTO=OFF
cmake --build build -j
ctest --test-dir build --output-on-failure -j
-- The CXX compiler identification is Clang 21.0.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /home/anjan/.local/share/swiftly/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.6s)
-- Generating done (0.0s)
-- Build files have been written to: /home/anjan/Documents/my_work/subtle/build
[ 50%] Building CXX object CMakeFiles/subtle_ct_tests.dir/tests/test_ct_dynamic.cpp.o
[100%] Linking CXX executable subtle_ct_tests
[100%] Built target subtle_ct_tests
Test project /home/anjan/Documents/my_work/subtle/build
    Start 1: subtle_ct_tests
1/1 Test #1: subtle_ct_tests .................   Passed    4.77 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   4.77 sec
```

Clang/MSan + −O3, does not detect ❌

Case study: conditional memcpy (4/16)

```
❯ cmake -B build -DCMAKE_CXX_COMPILER=g++ -DCMAKE_BUILD_TYPE=Release -DSUBTLE_BUILD_TESTS=ON -DSUBTLE_VALGRIND=ON -DSUBTLE_ENABLE_LTO=OFF
cmake --build build -j
ctest --test-dir build --output-on-failure -j
-- The CXX compiler identification is GNU 15.2.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.4s)
-- Generating done (0.0s)
-- Build files have been written to: /home/anjan/Documents/my_work/subtle/build
[ 50%] Building CXX object CMakeFiles/subtle_ct_tests.dir/tests/test_ct_dynamic.cpp.o
[100%] Linking CXX executable subtle_ct_tests
[100%] Built target subtle_ct_tests
Test project /home/anjan/Documents/my_work/subtle/build
    Start 1: subtle_ct_tests
1/1 Test #1: subtle_ct_tests ..................   Passed   65.83 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =  65.84 sec
```

GCC + −03 + Valgrind, does not detect ❌

Case study: conditional memcpy (5/16)

```
❯ cmake -B build -DCMAKE_CXX_COMPILER=clang++ -DCMAKE_BUILD_TYPE=Release -DSUBTLE_BUILD_TESTS=ON -DSUBTLE_VALGRIND=ON -DSUBTLE_ENABLE_LTO=OFF
cmake --build build -j
ctest --test-dir build --output-on-failure -j
-- The CXX compiler identification is Clang 21.0.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /home/anjan/.local/share/swiftly/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.5s)
-- Generating done (0.0s)
-- Build files have been written to: /home/anjan/Documents/my_work/subtle/build
[ 50%] Building CXX object CMakeFiles/subtle_ct_tests.dir/tests/test_ct_dynamic.cpp.o
[100%] Linking CXX executable subtle_ct_tests
[100%] Built target subtle_ct_tests
Test project /home/anjan/Documents/my_work/subtle/build
    Start 1: subtle_ct_tests
1/1 Test #1: subtle_ct_tests ..................***Failed   41.71 sec
==2480317== Memcheck, a memory error detector
==2480317== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==2480317== Using Valgrind-3.26.0 and LibVEX; rerun with -h for copyright info
==2480317== Command: /home/anjan/Documents/my_work/subtle/build/subtle_ct_tests
==2480317==
==2480317== Conditional jump or move depends on uninitialised value(s)
==2480317==    at 0x40014B3: main (in /home/anjan/Documents/my_work/subtle/build/subtle_ct_tests)
==2480317==  Uninitialised value was created by a client request
==2480317==    at 0x4001458: main (in /home/anjan/Documents/my_work/subtle/build/subtle_ct_tests)
==2480317==
```

...

Clang + −O3 + Valgrind, finally someone 🙌

Case study: conditional memcpy (6/16)

```
==2480317==
==2480317== More than 10000000 total errors detected.  I'm not reporting any more.
==2480317== Final error counts will be inaccurate.  Go fix your program!
==2480317== Rerun with --error-limit=no to disable this cutoff.  Note
==2480317== that errors may occur in your program without prior warning from
==2480317== Valgrind, because errors are no longer being displayed.
==2480317==
```

Clang + −O3 + Valgrind

Case study: conditional memcpy (7/16)

```
==========================================
  g++ / Release
==========================================
-- The CXX compiler identification is GNU 13.3.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.5s)
-- Generating done (0.0s)
-- Build files have been written to: /tmp/build_binsec_gpp_release
[ 50%] Building CXX object CMakeFiles/subtle_ct_binsec.dir/tests/test_ct_binsec.cpp.o
[100%] Linking CXX executable subtle_ct_binsec
[100%] Built target subtle_ct_binsec
[checkct:result] Instruction 0x401891 has control flow leak (0.028s)
[sse:info] Empty path worklist: halting ...
[sse:info] SMT queries
            Preprocessing simplifications
              total          1
              sat            1
              unsat          0
              time           0.00

            Satisfiability queries
              total          1
              sat            1
              unsat          0
              unknown        0
              time           0.01
              average        0.01

          Exploration
            total paths                     2
            completed/cut paths             2
            pending paths                   0
            discontinued paths              0
            failed assertions               0
            branching points                18
            max path depth                  30
            visited instructions (unrolled) 34
            visited instructions (static)   15


[checkct:result] Program status is : insecure (0.035)
  binsec_ct_conditional_memcpy_i32 INSECURE
---
Total: 1  Secure: 0  Insecure: 1  Unknown: 0
FAILED: 1 function(s) are not constant-time.
  RESULT: FAILED (exit 1)
```

```
==========================================
  clang++ / Release
==========================================
-- The CXX compiler identification is Clang 18.1.3
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.6s)
-- Generating done (0.0s)
-- Build files have been written to: /tmp/build_binsec_clangpp_release
[ 50%] Building CXX object CMakeFiles/subtle_ct_binsec.dir/tests/test_ct_binsec.cpp.o
[100%] Linking CXX executable subtle_ct_binsec
[100%] Built target subtle_ct_binsec
[checkct:result] Instruction 0x4017f8 has control flow leak (0.030s)
[sse:info] Empty path worklist: halting ...
[sse:info] SMT queries
            Preprocessing simplifications
              total          1
              sat            1
              unsat          0
              time           0.00

            Satisfiability queries
              total          1
              sat            1
              unsat          0
              unknown        0
              time           0.00
              average        0.00

          Exploration
            total paths                     2
            completed/cut paths             2
            pending paths                   0
            discontinued paths              0
            failed assertions               0
            branching points                1
            max path depth                  16
            visited instructions (unrolled) 21
            visited instructions (static)   17


[checkct:result] Program status is : insecure (0.031)
  binsec_ct_conditional_memcpy_i32 INSECURE
```

(GCC,Clang) + −03 + Binsec 🫡

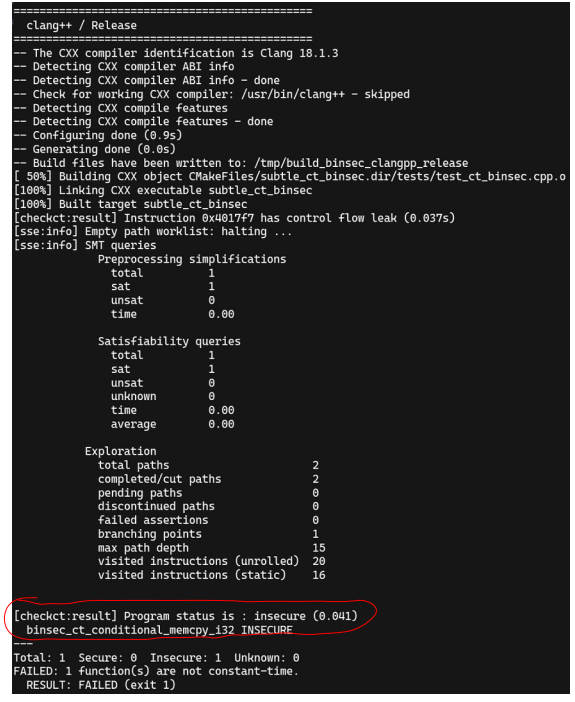
Case study: conditional memcpy (8/16)

```
131  template<typename branchT, typename operandT>
132  forceinline constexpr operandT
133  ct_select(const branchT br, const operandT x, const operandT y)
134    requires(std::is_unsigned_v<branchT> && ct_operand<operandT>)
135  {
136    using Uop = std::make_unsigned_t<operandT>;
137
138    const branchT z = br >> (std::numeric_limits<branchT>::digits - 1); // select MSB
139    Uop w = static_cast<Uop>(-static_cast<Uop>(z));
140    // ct_barrier(w);
141    const Uop selected = static_cast<Uop>((static_cast<Uop>(x) & w) | static_cast<Uop>(static_cast<Uop>(y) & static_cast<Uop>(~w))); // br ? x : y

       return static_cast<operandT>(selected);
  }
```

Disabled

Case study: conditional memcpy (9/16)

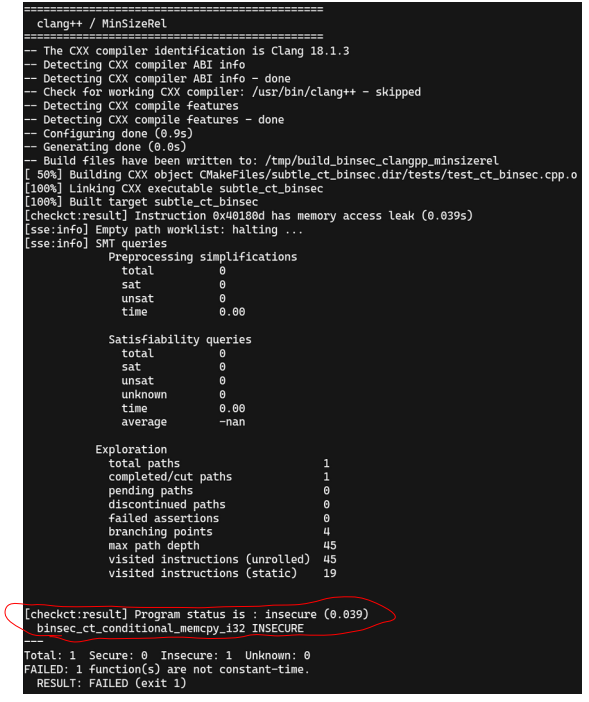


```
==============================================
  clang++ / Release
==============================================
-- The CXX compiler identification is Clang 18.1.3
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.9s)
-- Generating done (0.0s)
-- Build files have been written to: /tmp/build_binsec_clangpp_release
[ 50%] Building CXX object CMakeFiles/subtle_ct_binsec.dir/tests/test_ct_binsec.cpp.o
[100%] Linking CXX executable subtle_ct_binsec
[100%] Built target subtle_ct_binsec
[checkct:result] Instruction 0x4017f7 has control flow leak (0.037s)
[sse:info] Empty path worklist: halting ...
[sse:info] SMT queries
             Preprocessing simplifications
               total          1
               sat            1
               unsat          0
               time           0.00

             Satisfiability queries
               total          1
               sat            1
               unsat          0
               unknown        0
               time           0.00
               average        0.00

           Exploration
             total paths                      2
             completed/cut paths              2
             pending paths                    0
             discontinued paths               0
             failed assertions                0
             branching points                 1
             max path depth                   15
             visited instructions (unrolled)  20
             visited instructions (static)    16


[checkct:result] Program status is : insecure (0.041)
  binsec_ct_conditional_memcpy_i32 INSECURE
---
Total: 1  Secure: 0  Insecure: 1  Unknown: 0
FAILED: 1 function(s) are not constant-time.
  RESULT: FAILED (exit 1)
```



```
==============================================
  clang++ / MinSizeRel
==============================================
-- The CXX compiler identification is Clang 18.1.3
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.9s)
-- Generating done (0.0s)
-- Build files have been written to: /tmp/build_binsec_clangpp_minsizerel
[ 50%] Building CXX object CMakeFiles/subtle_ct_binsec.dir/tests/test_ct_binsec.cpp.o
[100%] Linking CXX executable subtle_ct_binsec
[100%] Built target subtle_ct_binsec
[checkct:result] Instruction 0x40180d has memory access leak (0.039s)
[sse:info] Empty path worklist: halting ...
[sse:info] SMT queries
             Preprocessing simplifications
               total          0
               sat            0
               unsat          0
               time           0.00

             Satisfiability queries
               total          0
               sat            0
               unsat          0
               unknown        0
               time           0.00
               average        -nan

           Exploration
             total paths                      1
             completed/cut paths              1
             pending paths                    0
             discontinued paths               0
             failed assertions                0
             branching points                 4
             max path depth                   45
             visited instructions (unrolled)  45
             visited instructions (static)    19


[checkct:result] Program status is : insecure (0.039)
  binsec_ct_conditional_memcpy_i32 INSECURE
---
Total: 1  Secure: 0  Insecure: 1  Unknown: 0
FAILED: 1 function(s) are not constant-time.
  RESULT: FAILED (exit 1)
```

Clang + (−O3, −Os) + Binsec

Case study: conditional memcpy (10/16)

```
❯ cmake -B build -DCMAKE_CXX_COMPILER=clang++ -DCMAKE_BUILD_TYPE=Release -DSUBTLE_BUILD_TESTS=ON -DSUBTLE_VALGRIND=ON -DSUBTLE_ENABLE_LTO=OFF
cmake --build build -j
ctest --test-dir build --output-on-failure -j
-- The CXX compiler identification is Clang 21.0.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /home/anjan/.local/share/swiftly/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.6s)
-- Generating done (0.0s)
-- Build files have been written to: /home/anjan/Documents/my_work/subtle/build
[ 50%] Building CXX object CMakeFiles/subtle_ct_tests.dir/tests/test_ct_dynamic.cpp.o
[100%] Linking CXX executable subtle_ct_tests
[100%] Built target subtle_ct_tests
Test project /home/anjan/Documents/my_work/subtle/build
    Start 1: subtle_ct_tests
1/1 Test #1: subtle_ct_tests ..................***Failed   41.69 sec
==2547111== Memcheck, a memory error detector
==2547111== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==2547111== Using Valgrind-3.26.0 and LibVEX; rerun with -h for copyright info
==2547111== Command: /home/anjan/Documents/my_work/subtle/build/subtle_ct_tests
==2547111==
==2547111== Conditional jump or move depends on uninitialised value(s)
==2547111==    at 0x40014B3: main (in /home/anjan/Documents/my_work/subtle/build/subtle_ct_tests)
==2547111==  Uninitialised value was created by a client request
==2547111==    at 0x4001458: main (in /home/anjan/Documents/my_work/subtle/build/subtle_ct_tests)
==2547111==
```

Clang + −O3 + Valgrind ✅

Case study: conditional memcpy (11/16)

```
❯ cmake -B build -DCMAKE_CXX_COMPILER=/usr/bin/clang++ -DCMAKE_BUILD_TYPE=MinSizeRel -DSUBTLE_BUILD_TESTS=ON -DSUBTLE_VALGRIND=ON -DSUBTLE_ENABLE_LTO=OFF
cmake --build build -j
ctest --test-dir build --output-on-failure -j
-- The CXX compiler identification is Clang 21.1.8
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.4s)
-- Generating done (0.0s)
-- Build files have been written to: /home/anjan/Documents/my_work/subtle/build
[ 50%] Building CXX object CMakeFiles/subtle_ct_tests.dir/tests/test_ct_dynamic.cpp.o
[100%] Linking CXX executable subtle_ct_tests
[100%] Built target subtle_ct_tests
Test project /home/anjan/Documents/my_work/subtle/build
    Start 1: subtle_ct_tests
1/1 Test #1: subtle_ct_tests ..................***Failed  111.21 sec
==59336== Memcheck, a memory error detector
==59336== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==59336== Using Valgrind-3.26.0 and LibVEX; rerun with -h for copyright info
==59336== Command: /home/anjan/Documents/my_work/subtle/build/subtle_ct_tests
==59336==
==59336== Conditional jump or move depends on uninitialised value(s)
==59336==    at 0x400141F: main (in /home/anjan/Documents/my_work/subtle/build/subtle_ct_tests)
==59336==  Uninitialised value was created by a client request
==59336==    at 0x40013C4: main (in /home/anjan/Documents/my_work/subtle/build/subtle_ct_tests)
==59336==
```

Clang + −Os + Valgrind ✅

Case study: conditional memcpy (12/16)

```
❯ cmake -B build -DCMAKE_CXX_COMPILER=clang++ -DCMAKE_BUILD_TYPE=Release -DSUBTLE_BUILD_TESTS=ON -DSUBTLE_MSAN=ON -DSUBTLE_ENABLE_LTO=OFF
cmake --build build -j
ctest --test-dir build --output-on-failure -j
-- The CXX compiler identification is Clang 21.0.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /home/anjan/.local/share/swiftly/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.5s)
-- Generating done (0.0s)
-- Build files have been written to: /home/anjan/Documents/my_work/subtle/build
[ 50%] Building CXX object CMakeFiles/subtle_ct_tests.dir/tests/test_ct_dynamic.cpp.o
[100%] Linking CXX executable subtle_ct_tests
[100%] Built target subtle_ct_tests
Test project /home/anjan/Documents/my_work/subtle/build
    Start 1: subtle_ct_tests
1/1 Test #1: subtle_ct_tests .................   Passed    4.76 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   4.77 sec
```

Clang + −O3 + MSan ❌

Case study: conditional memcpy (13/16)

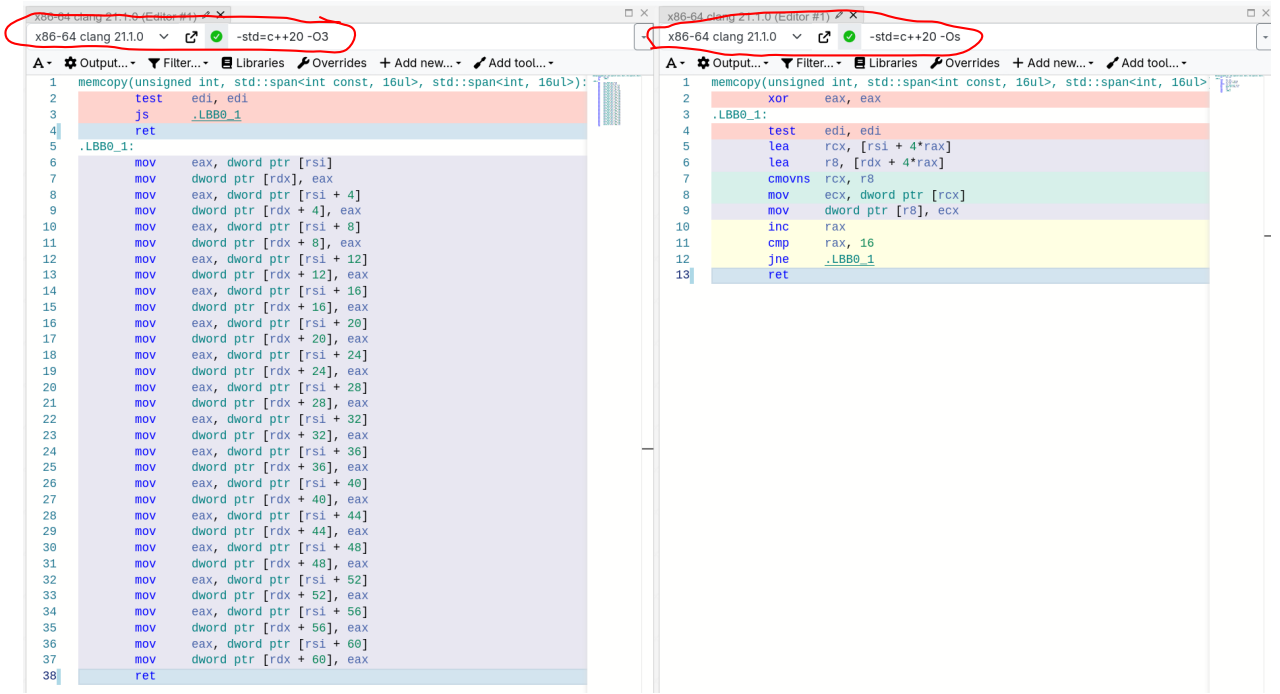
```
❯ cmake -B build -DCMAKE_CXX_COMPILER=clang++ -DCMAKE_BUILD_TYPE=MinSizeRel -DSUBTLE_BUILD_TESTS=ON -DSUBTLE_MSAN=ON -DSUBTLE_ENABLE_LTO=OFF
cmake --build build -j
ctest --test-dir build --output-on-failure -j
-- The CXX compiler identification is Clang 21.0.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /home/anjan/.local/share/swiftly/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.8s)
-- Generating done (0.0s)
-- Build files have been written to: /home/anjan/Documents/my_work/subtle/build
[ 50%] Building CXX object CMakeFiles/subtle_ct_tests.dir/tests/test_ct_dynamic.cpp.o
[100%] Linking CXX executable subtle_ct_tests
[100%] Built target subtle_ct_tests
Test project /home/anjan/Documents/my_work/subtle/build
    Start 1: subtle_ct_tests
1/1 Test #1: subtle_ct_tests ..................   Passed    4.13 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   4.13 sec
```

Clang + −0s + MSan ❌

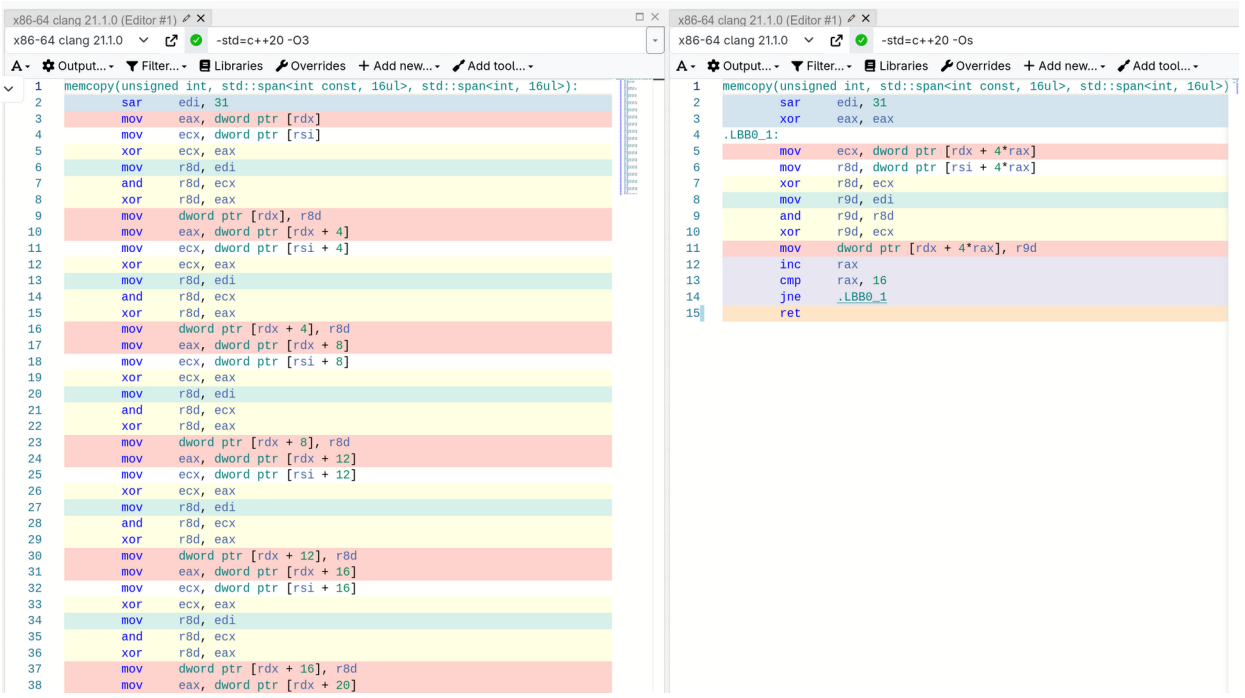
Case study: conditional memcpy (14/16)



`ct_barrier` is must have.

Else an eager to optimize compilers, like Clang, will just optimize it away.

Case study: conditional memcpy (15/16)



`ct_barrier` added back. Constant-time across optimization levels.

Case study: conditional memcpy (16/16)

Memory Sanitizer	Compile-time instrumentation ⭐
Valgrind	Runtime instrumentation ⭐⭐
Binsec	Symbolic analysis over binary ⭐⭐

Ok, so how to test CT?

Write a lot of tests.

And run them in different configurations with different tools.

There are more tools available

1) dudect
2) FlowTracker
3) ct-verif
4) ct-LLVM
5) Jasmin language
6) Fiat-Crypto
7) Constantine
...

n) `__builtin_ct_select`

None of them is a silverbullet !

But, what about Spectre 🐘?

1) Exploits “branch prediction” and “speculative execution” features
2) Compiler generated instruction is not what the CPU executes “exactly”
3) Yet another side-channel to extract secret key material
4) A constant-time program alone won’t save!

And what about DOITM?

1) Data Operand Independent Timing Mode
2) Mainly `div`/ `idiv`, `-ffast-math`
3) But no guarantees, just assumptions that `add`/ `xor`/ `and` will be CT

Recommendation: Use `subtle` for CT

Recommendation: Enable Spectre mitigations

```
#include <sys/prctl.h>
prctl(PR_SET_SPECULATION_CTRL, PR_SPEC_STORE_BYPASS, PR_SPEC_DISABLE, 0, 0);
```

Abstracted by Linux system headers

Recommendation: Enable Data Independent Timing (DIT)

```
asm volatile("msr DIT, #1");
```

On arm64, from userspace

Summary

1)Only timing is enough to extract secret.

2)Ops involving secret, must be constant-time.

3)There are tools to enforce CT, at scale.

4)None of them is perfect.

5)Compiler is a good friend, most of the time.

6)Do not forget to check h/w level guarantees.