Jasmin: high-assurance high-speed cryptography

Miguel Quaresma Santiago Arranz Olmos

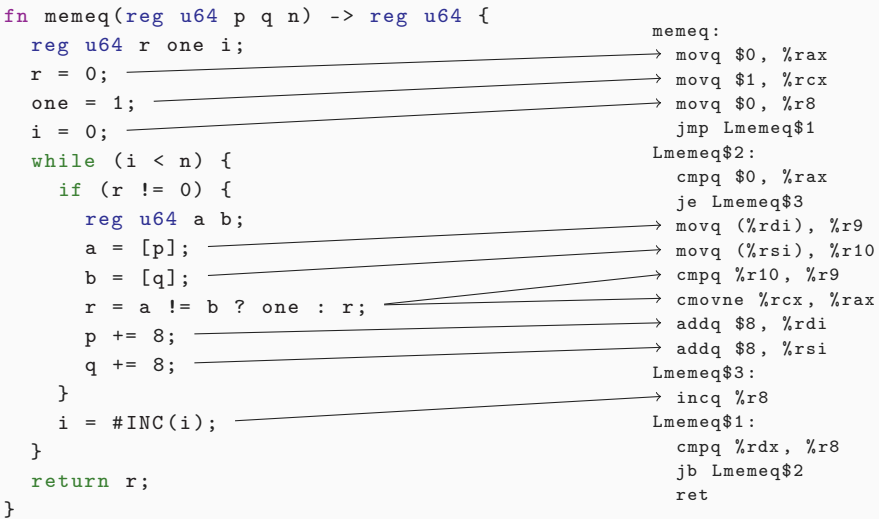
September 4, 2024

Max Planck Institute for Security and Privacy

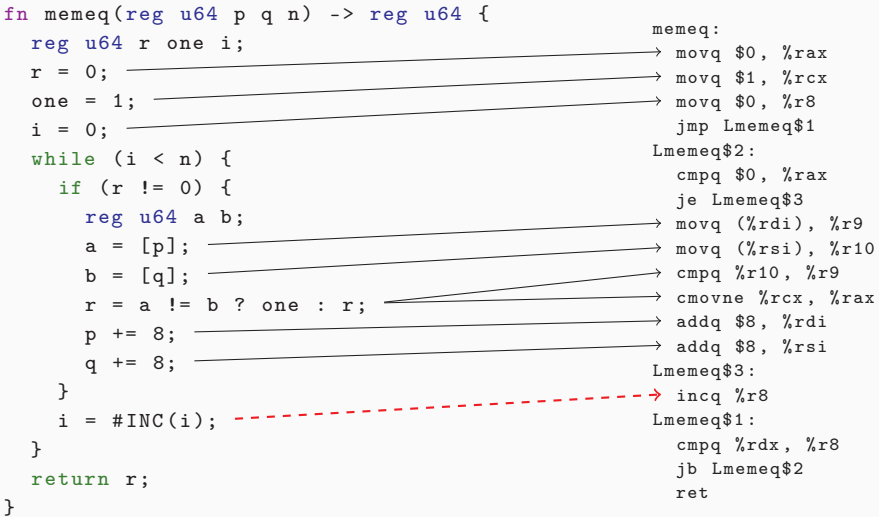
Efficient, correct, safe, and secure

```
fn memeq(reg u64 p q n) -> reg u64 {
  reg u64 r one i;
  r = 0;
  one = 1;
  i = 0;
  while (i < n) {
    if (r != 0) {
      reg u64 a b;
      a = [p];
      b = [q];
      r = a != b ? one : r;
      p += 8;
      q += 8;
    }
    i = #INC(i);
  }
  return r;
}
```

Efficient, correct, safe, and secure



Efficient, correct, safe, and secure



Efficient, correct, safe, and secure

```
fn memeq(reg u64 p q n) -> reg u64 {
  reg u64 r one i;
  r = 0;
  one = 1;
  i = 0;
  while (i < n) {
    if (r != 0) {
      reg u64 a b;
      a = [p];
      b = [q];
      r = a != b ? one : r;
      p += 8;
      q += 8;
    }
    i = #INC(i);
  }
  return r;
}
```

```
memeq:
  movq $0, %rax
  movq $1, %rcx
  movq $0, %r8
  jmp Lmemeq$1
Lmemeq$2:
  cmpq $0, %rax
  je Lmemeq$3
  movq (%rdi), %r9
  movq (%rsi), %r10
  cmpq %r10, %r9
  cmovne %rcx, %rax
  addq $8, %rdi
  addq $8, %rsi
Lmemeq$3:
  incq %r8
Lmemeq$1:
  cmpq %rdx, %r8
  jb Lmemeq$2
  ret
```

Safety - uninitialized values

```
export
fn uninitialized() -> reg u64 {
  reg u64 x;
  x = x + 1; // Uninitialized read from x.
  return x;
}
```

Safety - division by zero

```
export
fn arithmetic(reg u64 x y) -> reg u64 {
  x = x / y; // y could be zero.
  return x;

}
```

Safety - out of bounds access

```
export
fn index(reg u64 x) -> reg u64 {
  stack u64[1] s;
  s[x] = 0; // x could be out of bounds.
  x = s[0]; // s[0] could be uninitialized
  return x;
}
```

Safety - termination

```
export
fn termination(reg u64 n) -> reg u64 {
  reg u64 i;
  i = 0;
  while (i <= n) { // n could be 255
    i += 1;
  }
  return i;
}
```

Safety

```
export
fn alignment(reg u64 p) {
  [#aligned p] = 0; // p needs to be 64bit-aligned.
}

// Write c to the first n bytes of p.
// Run with -safetyparam 'memset>p;n'.
export
fn memset(reg u64 p, reg u8 c, reg u64 n) {
  reg u64 i;
  i = 0;
  while (i < n) {
    (u8)[p + i] = c; // 0 <= i < n
    i += 1;
  }
}
```

Side-channel - memeq 1/2

```
export
fn memeq(#public reg u64 p q n) -> #public reg u64 {
  reg u64 r one i;
  r = 0;
  one = 1;
  i = 0;

  while (i < n) {
    // Invariant: if r = 0 then p[0:i-1] = q[0:i-1] else p != q
    reg u64 a b;
    a = [p + i * 8];
    b = [q + i * 8];
    r = one if a != b;
    i += 1;
  }

  /* The result of the function depends on the contents of the buffer,
  which is secret, so we tell the CT checker that r (only one bit of
  information) is public. */
  #declassify r = r;
  return r;
}
```

Side-channel - memeq 2/2

```
fn memeq_early_abort(#public reg u64 p q n) -> #public reg u64 {
  reg u64 i x
  reg u8 r;
  i = 0;

  /* Since we stop early if the two buffers differ, we leak
  information through timing side channels. Thus, the CT
  checker rejects this program. */
  while (i < n) {
    // Invariant: if i <= n then p[0:i-1] = q[0:i-1] else p != q
    reg u64 a b;
    a = [p + i * 8];
    b = [q + i * 8];
    i = n if a != b;
    i += 1;
  }

  r = #SETcc(i == n); /* Set r to be 1 or 0 depending on this condition. SETcc
  sets only the lower byte of the register. */
  x = (64u)r; // So we need to zero extend.
  #declassify x = x;
  return x;
}
```

Side-channel - strlen 1/2

```
// Length of a null-terminated string.
fn strlen(#public reg u64 s) -> #public reg u64 {
  reg u64 i;
  i = 0;

  /* The condition of this loop leaks whether the character is zero.
  Declassifying the character means that we tell the CT checker that every
  character in the string is public, which is bad. */
  reg u8 c;
  while {
    c = (u8)[s + i];
  } (c != 0) {
    i += 1;
  }

  return i;
}
```

Side-channel - strlen 2/2

```
fn strlen_ct(#public reg u64 s) -> #public reg u64 {
  reg u64 i;
  i = 0;

  reg bool is_null;
  while {
    reg u8 c;
    c = (u8)[s + i];
    #declassify is_null = c != 0; /* We can declassify only the bit of
    information we are returning anyways:
    the ZF flag. */
  } (is_null) {
    i += 1;
  }

  return i;
}
```

More online

Jasmin: `github.com/jasmin-lang/jasmin`

EasyCrypt specifications: `github.com/formosa-crypto/crypto-specs`

Libjade: `github.com/formosa-crypto/libjade`