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

```
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
```

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
```

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
```

Correctness

- Specification is secure
- Implementation $\iff$ specification

Correctness, safety, side channels

Correctness

- Specification is secure
- Implementation $\iff$ specification

Safety

- Termination
- Array accesses in bounds
- Arithmetic errors

Correctness, safety, side channels

Correctness

- Specification is secure
- Implementation $\iff$ specification

Safety

- Termination
- Array accesses in bounds
- Arithmetic errors

Constant time

Runtime does not depend on secrets

- Control flow
- Memory accesses

Correctness, safety, side channels

Correctness

- Specification is secure
- Implementation $\iff$ specification

Safety

- Termination
- Array accesses in bounds
- Arithmetic errors

Constant time

Runtime does not depend on secrets

- Control flow
- Memory accesses

Speculative constant time

CT even under speculative execution

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 2^64-1
    i += 1;
  }
  return i;
}
```

Safety - memory accesses

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

export
fn memset(reg u64 p, reg u8 c, reg u64 n) {
  reg u64 i;
  i = 0;
  while (i < n) {
    (u8)[p + i] = c;
    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) {
    reg u64 a b;
    a = [p + i * 8];
    b = [q + i * 8];
    r = one if a != b;
    i += 1;
  }
  #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;
  while (i < n) {
    reg u64 a b;
    a = [p + i * 8];
    b = [q + i * 8];
    i = n if a != b;
    i += 1;
  }
  r = #SETcc(i == n);
  #declassify x = (64u)r;
  return x;
}
```

Side-channel - strlen 1/2

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

  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;
  } (is_null) {
    i += 1;
  }

  return i;
}
```

Spectre attacks - strlen

```
fn strlen_sct(#transient reg u64 s) -> #public reg u64 {
  reg u64 msf i;
  msf = #init_msf(); i = 0;
  reg u8 is_null c;
  while {
    c = (u8)[s + i];
    #declassify is_null = #SETcc(c != 0);
    is_null = #protect_8(is_null, msf);
  } (is_null == 1) {
    msf = #update_msf(is_null == 1, msf);
    i += 1;
  }
  return i;
}
```

More online

FORMOSA CRYPTO

`formosa-crypto.org`

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

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

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