Let's DOIT: Using Intel's Extended HW/SW Contract for Secure Compilation of Crypto Code

Santiago Arranz-Olmos[1], Gilles Barthe[1,2], Benjamin Grégoire[3],
Jan Jancar[4], Vincent Laporte[3], Tiago Oliveira[5], Peter Schwabe[1,6]

1 MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY
2 institute IMDEA software
3 Inria
4 CROCS
5 SANDBOXAQ
6 Radboud Universiteit

CHES 2025

Constant-time

What does *constant-time* even mean?

- Execution time independent of secret data
- **Constant-time programming** [1]

Constant-time

What does *constant-time* even mean?

- Execution time independent of secret data
- **Constant-time programming** [1]
  - Avoid branches on secret-dependent data
  - Avoid memory lookups on secret-dependent data
  - Avoid secret-dependent inputs to variable-time instructions

Constant-time

What does *constant-time* even mean?

- Execution time independent of secret data
- **Constant-time programming** [1]
  - Avoid branches on secret-dependent data
  - Avoid memory lookups on secret-dependent data
  - Avoid secret-dependent inputs to variable-time instructions

Constant-time

What does *constant-time* even mean?

- Execution time independent of secret data
- **Constant-time programming** [1]
  - Avoid branches on secret-dependent data
  - Avoid memory lookups on secret-dependent data
  - Avoid secret-dependent inputs to variable-time instructions
- More complexity
  - Compiler interference [2]
  - Speculative execution [3]
  - Cache design [4]
  - Dynamic frequency scaling [5]

Constant-time

Consequences

KyberSlash [6]

- Best Paper Award (~1 hour ago)
- Division by a constant compiled to soft or hard division
- Leaked enough for key recovery

Constant-time

Variable-time instructions

Which instructions are actually constant-time?

- Micro-architecture-dependent
- Previously:
  - Ad-hoc measurements/documentation/micro-benchmarks
  - Sometimes `MUL`, sometimes `DIV`

Constant-time

Variable-time instructions

Which instructions are actually constant-time?

- Micro-architecture-dependent
- Previously:
  - Ad-hoc measurements/documentation/micro-benchmarks
  - Sometimes `MUL`, sometimes `DIV`
- Now:
  - Intel DOIT
  - Arm DIT (+ Apple)
  - RISC-V Zkt/Zvkt

Constant-time

Variable-time instructions

Which instructions are actually constant-time?

- Micro-architecture-dependent
- Previously:
  - Ad-hoc measurements/documentation/micro-benchmarks
  - Sometimes `MUL`, sometimes `DIV`
- Now:
  - Intel DOIT
  - Arm DIT (+ Apple)
  - RISC-V Zkt/Zvkt
- *Future-proof constant-time*

Intel DOIT, Arm DIT, RISC-V Zkt/Zvkt

- List of instructions guaranteed to execute in input-independent timing

Intel DOIT, Arm DIT, RISC-V Zkt/Zvkt

- List of instructions guaranteed to execute in input-independent timing
- **If** the mode is enabled:
  - DOIT: Model Specific Register (MSR) set from kernel mode
  - DIT: Process State register (PSTATE) set from user mode

Intel DOIT, Arm DIT, RISC-V Zkt/Zvkt

- List of instructions guaranteed to execute in input-independent timing
- **If** the mode is enabled:
  - DOIT: Model Specific Register (MSR) set from kernel mode
  - DIT: Process State register (PSTATE) set from user mode
- **Always**:
  - Zkt/Zvkt

Intel DOIT

- **D**ata **O**perand **I**ndependent **T**iming
- *"DOIT mode may have a performance impact, and Intel expects the performance impact of this mode may be significantly higher on future processors*"
- Disables data-dependent prefetcher and fast-store-forwarding prefetcher
- Guarantees data-independence of timing:
  - on newer architectures **if enabled**
  - on older architectures
- Notably **not included**
  - Division
  - Rotate
  - Floating point operations
- No changes

Arm DIT

- **D**ata **I**ndependent **T**iming
- Arm v8.4-A and beyond (+ Apple Silicon)
- Guarantees data-independence of timing on supported architectures **if enabled**
- Changes often:
  - 2021-12: removal of `RET`,
  - 2023-03: removal of `SQCVTN`, `SQCVTUN`, `SQRSHRN`, `SQRSHRUN`, `UQCVTN`, `UQRSHRN`,
  - 2023-09: removal of all `WHILE*` instructions,
  - 2023-12: removal of `PEXT`,
  - 2024-06: removal of `CNTP`, `PUNPKHI`, `PUNPKLO`,
  - 2024-12: removal of all `RCW*` instructions.

RISC-V Zkt/Zvkt

- Zkt: scalar instructions , Zvkt: vector instructions
- Guarantees data-independence of timing if advertised
- No changes
- Notably **included**
  - seed CSR

Jasmin

- Language + compiler for high-assurance high-speed cryptography
  - Low-level (assembly like) -> control & performance
  - Formally verified -> assurance & security
  - Proven to preserve constant-time
- Active maintenance over 10 years
- x86_64 and ARMv7-M

```
fn strlen(reg u64 s) -> reg u32 {
    reg u32 len;
    reg u8 c;
    len = 0;
    while {
        c = [:u8 s];
    } (c != 0) {
        s += 1;
        len += 1;
    }
    return len;
}
```

```
strlen:
    movl    $0, %eax
    jmp     Lstrlen$1
Lstrlen$2:
    incq    %rdi
    incl    %eax
Lstrlen$1:
    movb    (%rdi), %cl
    cmpb    $0, %cl
    jne     Lstrlen$2
    ret
```

Jasmin

(Speculative) constant-time checker

- Type-based (speculative) constant-time checker
- Requires **`#secret`** and **`#public`** annotation on inputs
- CLI using `jasmin-ct`

```
fn foo(#secret reg u64 key, #public reg u64 nonce) -> #public reg u64 { ... }
```

Jasmin

(Speculative) constant-time checker

- Type-based (speculative) constant-time checker
- Requires **`#secret`** and **`#public`** annotation on inputs
- CLI using `jasmin-ct`

```
export fn passes(#public reg u8 key) -> #public reg u8 {
    reg u8 res;
    if (key == 5) { ✅
        res = 0;
    } else {
        res = 1;
    }
    return res;
}
```

Jasmin

(Speculative) constant-time checker

- Type-based (speculative) constant-time checker
- Requires **`#secret`** and **`#public`** annotation on inputs
- CLI using `jasmin-ct`

```
export fn fails(#secret reg u8 key) -> #secret reg u8 {
    reg u8 res;
    if (key == 5) { ✗
        res = 0;
    } else {
        res = 1;
    }
    return res;
}
```

Jasmin

(Speculative) constant-time checker

- Type-based (speculative) constant-time checker
- Requires **`#secret`** and **`#public`** annotation on inputs
- CLI using `jasmin-ct`

```
export fn would_pass(#secret reg u32 x) -> #secret reg u32 {
    reg u32 d q;
    x = x;
    d = 3;
    q = x / d;
    return q;
}
```

Jasmin

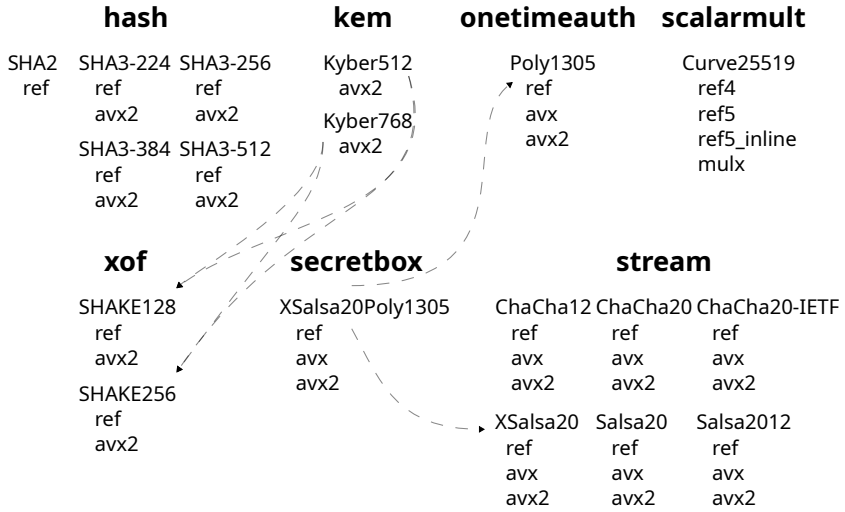
Making it DOIT

- Extract list of DOIT instructions
- Map to Jasmin instructions
- Modify typing rule for instructions
  - If `instr` is not DOIT, then all arguments must be **`#public`**
- Voilà!

Libjade

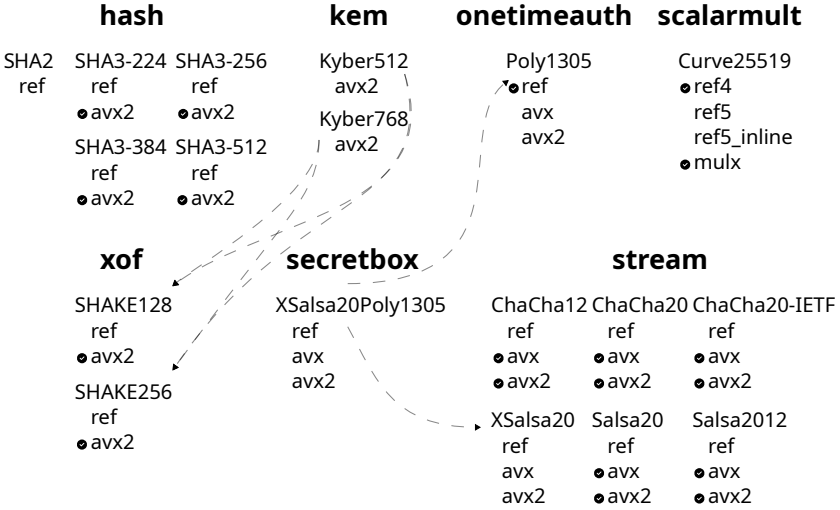
- Collection of popular crypto primitives in Jasmin
- Targets x86_64
- (Optimized) variants for reference/AVX/AVX2

Libjade


hash
kem
onetimeauth
scalarmult
SHA2
ref
SHA3-224
ref
avx2
SHA3-256
ref
avx2
SHA3-384
ref
avx2
SHA3-512
ref
avx2
Kyber512
avx2
Kyber768
avx2
Poly1305
ref
avx
avx2
Curve25519
ref4
ref5
ref5_inline
mulx
xof
secretbox
stream
SHAKE128
ref
avx2
SHAKE256
ref
avx2
XSalsa20Poly1305
ref
avx
avx2
ChaCha12
ref
avx
avx2
ChaCha20
ref
avx
avx2
ChaCha20-IETF
ref
avx
avx2
XSalsa20
ref
avx
avx2
Salsa20
ref
avx
avx2
Salsa2012
ref
avx
avx2

Libjade

Making it DOIT



Libjade

Making it DOIT

- Some are already DOIT

Libjade

Making it DOIT

- Some are already DOIT
- Enable DOIT, run `jasmin-ct`, fix errors, repeat…

Libjade

Making it DOIT

- Some are already DOIT
- Enable DOIT, run `jasmin-ct`, fix errors, repeat…
- Takeaways and **pain points**:

Libjade

Making it DOIT

- Some are already DOIT
- Enable DOIT, run `jasmin-ct`, fix errors, repeat…
- Takeaways and **pain points**:
  - Missing rotates: 2 shifts and an OR, extra register

```
rol32:
    roll $5, %eax
```

```
rol32_doit:
    movl    %eax, %ecx
    shll    $5, %eax
    shrl    $27, %ecx
    orl     %ecx, %eax
```

Libjade

Making it DOIT

- Some are already DOIT
- Enable DOIT, run `jasmin-ct`, fix errors, repeat…
- Takeaways and **pain points**:
  - Missing rotates: 2 shifts and an OR, extra register
  - BSWAP missing: Shifts, ANDs, ORs…

```
bswap32:
    bswapl %eax
```

```
bswap32_doit:
    movl    %eax, %ecx
    shrl    $24, %ecx
    movl    %eax, %edx
    shll    $24, %edx
    orl     %edx, %ecx
    movl    %eax, %edx
    shrl    $8, %edx
    andl    $65280, %edx
    orl     %edx, %ecx
    shll    $8, %eax ...
```

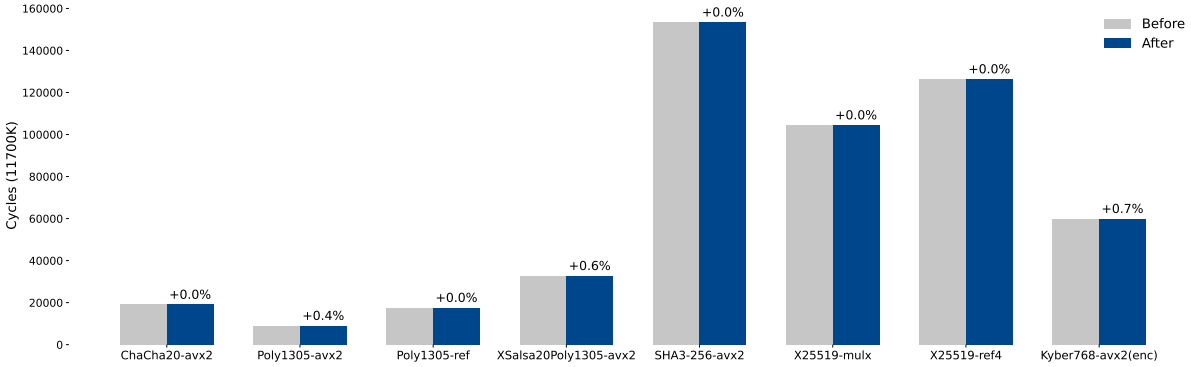
Libjade

Making it DOIT

- Some are already DOIT
- Enable DOIT, run `jasmin-ct`, fix errors, repeat…
- Takeaways and **pain points**:
  - Missing rotates: 2 shifts and an OR, extra register
  - BSWAP missing: Shifts, ANDs, ORs…
  - Miscellaneous vector loads, etc.

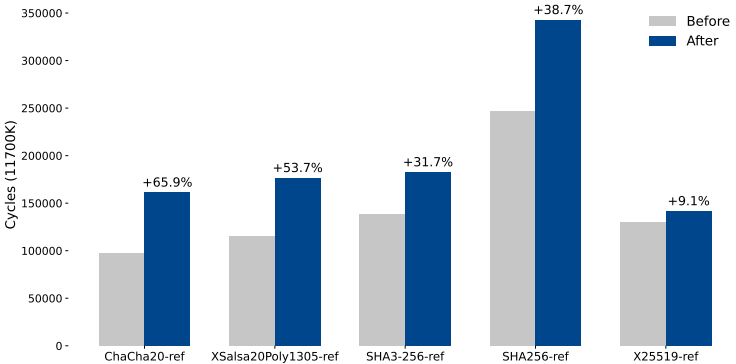
Libjade

Performance impacts: The good



Libjade

Performance impacts: The bad



Recommendations

Platform/ISA developer

- Be as specific as possible
- Use machine-readable format
- Consider use-cases in cryptography
- Limit changes to the guarantees

Recommendations

Platform/ISA developer

- Be as specific as possible
- Use machine-readable format
- Consider use-cases in cryptography
- Limit changes to the guarantees

Crypto developer

- Move toward DOIT/DIT/Zkt implementations
- Enable the DOIT/DIT mode

Recommendations

Platform/ISA developer

- Be as specific as possible
- Use machine-readable format
- Consider use-cases in cryptography
- Limit changes to the guarantees

Crypto developer

- Move toward DOIT/DIT/Zkt implementations
- Enable the DOIT/DIT mode

Kernel developer

- Offer user-space API for DOIT control
- Same as crypto developer…

Recommendations

Platform/ISA developer

- Be as specific as possible
- Use machine-readable format
- Consider use-cases in cryptography
- Limit changes to the guarantees

Kernel developer

- Offer user-space API for DOIT control
- Same as crypto developer…

Crypto developer

- Move toward DOIT/DIT/Zkt implementations
- Enable the DOIT/DIT mode

Compiler/tool developer

- Add option to limit/validate generated assembly for DOIT/DIT/Zkt

Conclusions

There is hope for constant-time.

Conclusions

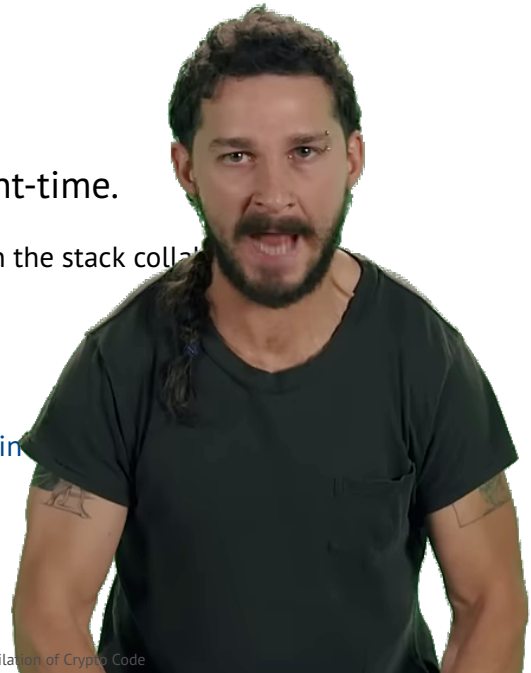
There is hope for constant-time.

- If people at various levels in the stack collaborate
  - Processor
  - Compiler
  - Developer
- The pessimist case [7]
- Check-out our artifact, Jasmin and Libjade

Conclusions

There is hope for constant-time.

- If people at various levels in the stack coll
  - Processor
  - Compiler
  - Developer
- The pessimist case [7]
- Check-out our artifact, Jasmin

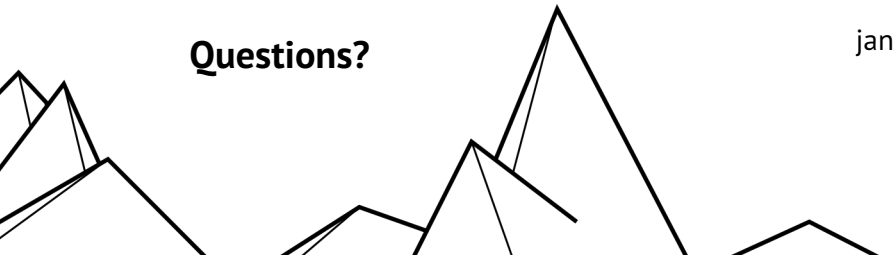


Let's DOIT: Using Intel's Extended HW/SW Contract for Secure Compilation of Crypto Code

Santiago Arranz-Olmos, Gilles Barthe, Benjamin Grégoire,
Jan Jancar, Vincent Laporte, Tiago Oliveira, Peter Schwabe

Questions?

J08nY
jan@neuromancer.sk



References

- Santiago Arranz-Olmos, Gilles Barthe, Benjamin Grégoire, Jan Jancar, Vincent Laporte, Tiago Oliveira & Peter Schwabe:
  Let's DOIT: Using Intel's Extended HW/SW Contract for Secure Compilation of Crypto Code

References (cont.)

1. Paul C. Kocher: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems
2. Moritz Schneider, Daniele Lain, Ivan Puddu, Nicolas Dutly & Srdjan Capkun: Breaking Bad: How Compilers Break Constant-Time Implementations
3. Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz & Yuval Yarom: Spectre attacks: exploiting speculative execution
4. Shuai Wang, Pei Wang, Xiao Liu, Danfeng Zhang & Dinghao Wu: CacheD: Identifying Cache-Based Timing Channels in Production Software
5. Yingchen Wang, Riccardo Paccagnella, Alan Wandke, Zhao Gang, Grant Garrett-Grossman, Christopher W. Fletcher, David Kohlbrenner & Hovav Shacham: DVFS Frequently Leaks Secrets: Hertzbleed Attacks Beyond SIKE, Cryptography, and CPU-Only Data
6. Daniel J. Bernstein, Karthikeyan Bhargavan, Shivam Bhasin, Anupam Chattopadhyay, Tee Kiah Chia, Matthias J. Kannwischer, Franziskus Kiefer, Thales B. Paiva, Prasanna Ravi & Goutam Tamvada: KyberSlash: Exploiting secret-dependent divisiontimings in Kyber implementations
7. Thomas Pornin: Constant-Time Code: The Pessimist Case

Other

- Icons from **Font Awesome**