





VeCT: Secure and Efficient Constant-Time Code Rewriting with Vector Extensions

Qisheng Jiang and Danfeng Zhang

Duke University

Duke

* This work is supported by NSF and Intel.

Constant-Time (CT) Programming



The Security Goal

Program execution must be independent of secret data:

- No secret-dependent branches
- No secret-dependent memory accesses



Industry Adoption

Constant-Time (CT) programming is no longer niche.

It is the fundamental defense against timing side-channels in:

- Cryptographic libraries
  - OpenSSL / BearSSL / mbed TLS
- Privacy-sensitive software

Duke

The Heavy Cost of Security

Performance Overhead of Current SOTA CT Rewriters



Root cause: *CT often increases memory footprint and creates massive CPU stalls.*

Duke

VeCT: Leveraging Vector Extensions



Single Instruction, Multiple Data (SIMD)

- Allows executing the same operation across multiple data simultaneously.
- Hardware Support: Intel AVX-512, ARM SVE, RISC-V 'V' Extension.

***Our Key Insight*:**

Use **Vector Extensions** to **parallelize memory accesses** for CT code rewriting

Duke

Challenges



Security

Do vector instructions break CT guarantees?

- A Statistical Hypothesis Testing Framework
- CT Hardware Contract



Efficiency

How to maximize hardware utilization?

- Instruction-Level Rewriting
- Program-Level Optimization

Duke

Results: A Leap in Efficiency

Overhead Reductions on Average v.s. Constantine



98.9%

MAX REDUCTION IN OVERHEAD

VeCT makes CT code rewriting practical for high-throughput security-critical systems.

Duke

Conclusion

- VeCT bridges the gap between **Security Guarantees** and **High Performance** by revealing the Constant-Time Hardware *Contract* and introducing both *Instruction*-Level and *Program*-Level *Vectorization*.

- Questions?
- Contact: qisheng.jiang@duke.edu
- Open Source: QR Code

Duke