Engineering Cryptographic Software

Introduction

Peter Schwabe

Winter 2026/27

Who we are



- Lecturer: **Peter Schwabe**
- E-Mail: peter@cryptojedi.org
- Professor at Radboud University (Nijmegen, NL)
- Since 2020 part time
- Scientific Director at Max Planck Institute for Security and Privacy (MPI-SP)
- Tutor: **Melchisedech Mbeng**
- E-Mail: melchisedechmbeng7@gmail.com

MPI-SP



- Located in **Bochum**
- Founded in 2019
- Currently 13 PIs

- Aim to have
  - 6 Departments
  - 12 Research Groups
  - Around 250 people total

Assignment + short oral exam



- Programming assignment on embedded CPU
- Work in groups of 2
- Short (10–15 minute) oral exam about assignment submission
- Each student needs to be able to answer questions about code
- Oral exam only for groups who submitted assignment ticking all boxes

On the use of LLMs

On the use of LLMs

- Use them.

On the use of LLMs

- Use them.
- Use LLMs for things **you know in principle how to do yourself**

On the use of LLMs

- Use them.
- Use LLMs for things **you know in principle how to do yourself**
- You need to understand code written by Claude and friends
- You need to be able to explain this code!
- You need to document use of LLMs (see assignment PDF)

Course schedule

Date	Lecture/Lab sessions	Assignment
Nov 6	Lecture	
Nov 13	Lecture	
Nov 20	Lecture	
Nov 27	Lecture	
Dec 4	Lecture	
Dec 11	Lecture	
Dec 18	Lecture	
Dec 21		Feedback Deadline SW Asignment
Jan 20		Final Deadline SW Assigment

Cryptographic software



Is cryptographic software *special*?

Cryptographic software



Is cryptographic software *special*?

In some sense it's not. . .

- We can implement crypto in pretty much any language
- We expect typical properties like correctness, efficiency, maintainability. . .

Cryptographic software



Is cryptographic software *special*?

In some sense it's not. . .

- We can implement crypto in pretty much any language
- We expect typical properties like correctness, efficiency, maintainability. . .

. . . but in many ways it is

- Code is typically very small

Cryptographic software



Is cryptographic software *special*?

In some sense it's not. . .

- We can implement crypto in pretty much any language
- We expect typical properties like correctness, efficiency, maintainability. . .

. . . but in many ways it is

- Code is typically very small
- Even small performance improvements matter

Cryptographic software



Is cryptographic software *special*?

In some sense it's not. . .

- We can implement crypto in pretty much any language
- We expect typical properties like correctness, efficiency, maintainability. . .

. . . but in many ways it is

- Code is typically very small
- Even small performance improvements matter
- We typically have a full functional specification

Cryptographic software



Is cryptographic software *special*?

In some sense it's not...

- We can implement crypto in pretty much any language
- We expect typical properties like correctness, efficiency, maintainability...

...but in many ways it is

- Code is typically very small
- Even small performance improvements matter
- We typically have a full functional specification
- Bugs are essentially always security critical

Cryptographic software



Is cryptographic software *special*?

In some sense it's not. . .

- We can implement crypto in pretty much any language
- We expect typical properties like correctness, efficiency, maintainability. . .

. . . but in many ways it is

- Code is typically very small
- Even small performance improvements matter
- We typically have a full functional specification
- Bugs are essentially always security critical
- **Crypto operates on secret data**, must not leak this!

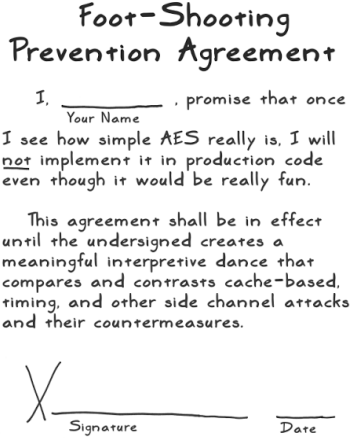
Cryptographic software



Cryptographic software is **small**, highly **performance critical**, highly **security critical**, and typically operates **in adversarial environments**.

“Don’t roll your own crypto”

- Crypto is hard to get right
- Crypto software is hard to get right
- Need extensive independent review

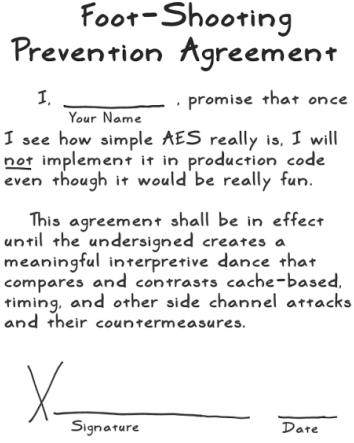


From *A Stick Figure Guide to the Advanced Encryption Standard (AES)*

https://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

“Don’t roll your own crypto”

- Crypto is hard to get right
- Crypto software is hard to get right
- Need extensive independent review **before being used**



From *A Stick Figure Guide to the Advanced Encryption Standard (AES)*

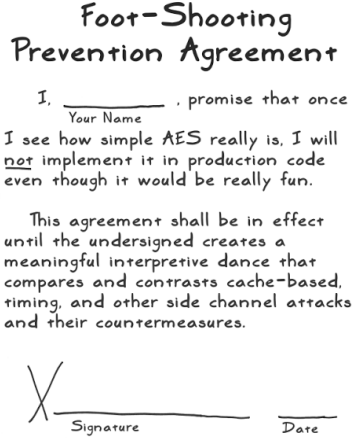
https://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

“Don’t roll your own crypto”

- Crypto is hard to get right
- Crypto software is hard to get right
- Need extensive independent review **before being used**

My take:

- *Roll your own crypto!*
- Write your own crypto software!



From *A Stick Figure Guide to the Advanced Encryption Standard (AES)*

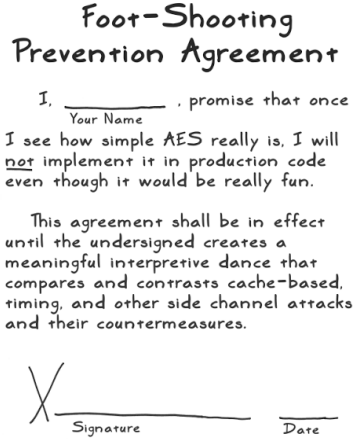
https://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

“Don’t roll your own crypto”

- Crypto is hard to get right
- Crypto software is hard to get right
- Need extensive independent review **before being used**

My take:

- *Roll your own crypto!*
- Write your own crypto software!
- Get it wrong, be told, learn
- Get better, keep learning



From *A Stick Figure Guide to the Advanced Encryption Standard (AES)*

https://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

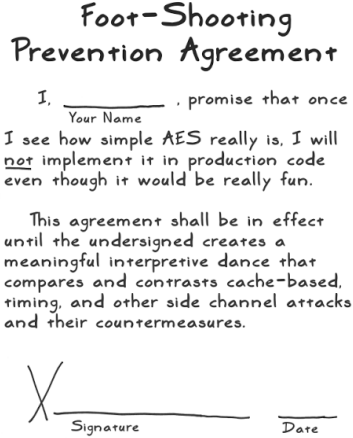
“Don’t roll your own crypto”

- Crypto is hard to get right
- Crypto software is hard to get right
- Need extensive independent review **before being used**

My take:

- *Roll your own crypto!*
- Write your own crypto software!
- Get it wrong, be told, learn
- Get better, keep learning

Just don’t use your own crypto (software).



From *A Stick Figure Guide to the Advanced Encryption Standard (AES)*

https://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

The past of this course



- Taught since 2014 at Radboud University (Nijmegen, The Netherlands)
- Fundamentals of crypto software
- Symmetric crypto examples
- Elliptic-curve crypto
- Assignments in C/assembly + PDF explaining submission
- Lectures using code/examples from many (micro-)architectures
- Assignments: Optimize code on embedded microcontroller (ARM Cortex-M4)
- Close to state-of-the-art in real-world crypto software

The use of C/assembly for crypto software



Some downsides of C

- No memory safety
- Finicky semantics
  - Undefined behavior
  - Implementation-specific behavior
  - Context-specific behavior
- No mandatory initialization
- No (optional) runtime checks

The use of C/assembly for crypto software



Some downsides of C

- No memory safety
- Finicky semantics
  - Undefined behavior
  - Implementation-specific behavior
  - Context-specific behavior
- No mandatory initialization
- No (optional) runtime checks

but... Rust!

- Memory safe
- More clear semantics (?)
- Mandatory variable initialization
- (Optional) runtime checks for, e.g., overflows

The use of C/assembly for crypto software

Lack of security features

"Security engineers have been fighting with C compilers for years."

—Simon, Chisnall, Anderson, 2018[a]

- **No concept of secret vs. public data**
- Compilers introduce vulnerabilities!
- Cat-and-mouse game **against your own tools!**

[a] *What you get is what you C: Controlling side effects in mainstream C compilers.* EuroS&P 2018

It's getting worse!

Breaking Bad: How Compilers Break Constant-Time Implementations

Moritz Schneider
moritz.schneider@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Daniele Lain
daniele.lain@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Ivan Puddu
ivan.puddu@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Nicolas Dutly
ndutly@student.ethz.ch
ETH Zurich
Zurich, Switzerland

Srdjan Čapkun
srdjan.capkun@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Abstract

The implementations of most hardened cryptographic libraries use defensive programming techniques for side-channel resistance. These techniques are usually specified as guidelines to developers on specific code patterns to use or avoid. Examples include performing arithmetic operations to choose between two variables instead of executing a secret-dependent branch. However, such techniques are only meaningful if they persist across compilation. In this paper, we investigate how optimizations used by modern compilers break

Keywords

Constant time code, cryptographic implementations, compilers

ACM Reference Format:

Moritz Schneider, Daniele Lain, Ivan Puddu, Nicolas Dutly, and Srdjan Čapkun. 2025. Breaking Bad: How Compilers Break Constant-Time Implementations. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '25), August 25–29, 2025, Hanoi, Vietnam.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3708821.3733909

It's getting worse!

Breaking Bad: How Compilers Break Constant-Time Implementations

Moritz Schneider
moritz.schneider@inf.ethz.
ETH Zurich
Zurich, Switzerland

Abstract

The implementations of most harde
use defensive programming technique
These techniques are usually specifie
on specific code patterns to use or avoi
ing arithmetic operations to choose be
of executing a secret-dependent branc
are only meaningful if they persist acr

Do Compilers Break Constant-time Guarantees?

Lukas Gerlach[1], Robert Pietsch[2], and Michael Schwarz[1]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
[2] Saarland University, Saarbrücken, Germany

Abstract. Side-channel attacks are a significant concern for the implementation of cryptographic algorithms. Data-oblivious programming is a discipline that helps mitigate side-channel attacks by preventing data leakage over side channels. However, due to various optimizations in modern compilers, data-obliviousness cannot be guaranteed in high-level languages. This work investigates to which extent compiler optimizations violate data-obliviousness. To this end, we present data-oblivious compiler checker (DOCC), an automated binary testing pipeline for detecting data-obliviousness violations under different compiler configura-

It's getting worse!

Breaking Bad: How Compilers Break Constant-Time Implementations

Moritz Schneider
moritz.schneider@inf.ethz.
ETH Zurich
Zurich, Switzerland

Abstract

The implementations of most harde
use defensive programming technique
These techniques are usually specifie
on specific code patterns to use or avoi
ing arithmetic operations to choose be
of executing a secret-dependent branc
are only meaningful if they persist acr

Do Compilers Break Constant-time Guarantees?

Lukas Gerl

[1] CISPA Helmholtz
[2] Sa

Abstract. Side-c
plementation of c
is a discipline th
data leakage over
modern compilers
languages. This
tions violate data-obliviousness. To this end, we present data-oblivious
compiler checker (DOCC), an automated binary testing pipeline for de-
tecting data-obliviousness violations under different compiler configura-

Fun with flags: How Compilers Break and Fix Constant-Time Code

Antoine Geimer
Univ. Lille, CNRS, Inria
Univ. Rennes, CNRS, IRISA
antoine.geimer@inria.fr

Clémentine Maurice
Univ. Lille, CNRS, Inria
clementine.maurice@inria.fr

***Abstract*—Developers rely on constant-time programming to prevent timing side-channel attacks. But these efforts can be undone by compilers, whose optimizations may silently reintroduce leaks. While recent works have measured the extent of such leakage, they leave developers without actionable insights: which optimization passes are responsible, and how to disable them without modifying the compiler remains unclear.**

can re-implement critical functions in assembly snippets for each targeted architecture – a time-consuming task that risk introducing more bugs. On the other hand they can purposefully complexify their code to counter the compiler's optimizations – hardly a resilient approach as compilers improve.

Problem. While a mix of both approaches is generally

It's getting worse!

Breaking Bad: How Compilers Break Constant-Time Implementations

Moritz Schneider
moritz.schneider@inf.ethz.
ETH Zurich
Zurich, Switzerland

Abstract

The implementations of most harde
use defensive programming technique
These techniques are usually specifie
on specific code patterns to use or avoi
ing arithmetic operations to choose be
of executing a secret-dependent branc
are only meaningful if they persist acr

Do Compilers Break Constant-time Guarantees?

Lukas Gerl

[1] CISPA Helmholtz
[2] Sa

Abstract. Side-c
plementation of c
is a discipline th
data leakage over
modern compilers
languages. This
tions violate data-obliviousness. To this end,
compiler checker (DOCC), an automated bin
tecting data-obliviousness violations under d

Fun with flags: How Compilers Break and Fix Constant-Time Code

Antoine
Univ. Lille,
Univ. Rennes,
antoine.gei

***Abstract*—Developers rely on const
prevent timing side-channel attacks.
undone by compilers, whose optimiza
duce leaks. While recent works have m
leakage, they leave developers without
optimization passes are responsible,
without modifying the compiler remai

CT-LLVM: Automatic Large-Scale Constant-Time Analysis

Zhiyuan Zhang, Gilles Barthe

MPI-SP, Bochum, Germany
IMDEA Software Institute, Madrid, Spain

Abstract

Constant-time (CT) is a popular programming discipline to protect cryptographic libraries against micro-architectural timing attacks. One appeal of the CT discipline lies in its conceptual simplicity: a program is CT iff it has no secret-dependent data-flow, control-flow or variable-timing operation. Thanks to its simplicity, the CT discipline is supported by dozens of analysis tools. However, a recent user study demonstrates that these tools are seldom used due to poor usability and maintainability (Jancar et al. IEEE SP 2022).

Problems Identification. We identify two main reasons for not closing the gap between the CT discipline and the practice. The first reason is the low adoption of CT analysis tools in real-world development. A recent study [24] shows that developers do not routinely use CT analysis tools because of poor usability. First, most available tools are difficult to install, due to complex dependencies and reliance on deprecated software. Second, once installed, the overwhelming majority of the tools are still hard to use. For instance, they may require complex setups for each use of the tool. Third, analysis results may be difficult to interpret, due to the underlying analysis



Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*

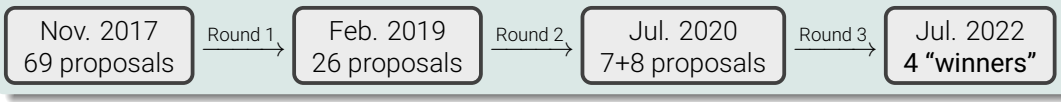
Peter W. Shor†

Abstract

A digital computer is generally believed to be an efficient universal computing device; that is, it is believed able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. This paper considers factoring integers and finding discrete logarithms, two problems which are generally thought to be hard on a classical computer and which have been used as the basis of several proposed cryptosystems. Efficient randomized algorithms are given for these two problems on a hypothetical quantum computer. These algorithms take a number of steps polynomial in the input size, e.g., the number of digits of the integer to be factored.

The post-quantum (re-)volution

NIST PQC



“The public-key encryption and key-establishment algorithm that will be standardized is CRYSTALS-KYBER. The digital signatures that will be standardized are CRYSTALS-Dilithium, FALCON, and SPHINCS$^+$. While there are multiple signature algorithms selected, NIST recommends CRYSTALS-Dilithium as the primary algorithm to be implemented”

—NIST IR 8413 (July 2022)

(Some) PQC deployment



From https://radar.cloudflare.com/adoption-and-usage#post-quantum-encryption-adoption

(Some) PQC deployment



From https://radar.cloudflare.com/adoption-and-usage#post-quantum-encryption-adoption

(Some) PQC deployment



From https://radar.cloudflare.com/adoption-and-usage#post-quantum-encryption-adoption

(Some) PQC deployment



From https://radar.cloudflare.com/adoption-and-usage#post-quantum-encryption-adoption







(Some) PQC deployment



From https://radar.cloudflare.com/adoption-and-usage#post-quantum-encryption-adoption









The AI revolution



- Software development is now LLM-assisted
- (So is research in CS, fun dinner topic)

The AI revolution



- Software development is now LLM-assisted
- (So is research in CS, fun dinner topic)
- Conjecture: LLMs will create a software-quality divide
  - Good developers will produce better software faster
  - Bad developers will produce worse software faster

The AI revolution



- Software development is now LLM-assisted
- (So is research in CS, fun dinner topic)
- Conjecture: LLMs will create a software-quality divide
  - Good developers will produce better software faster
  - Bad developers will produce worse software faster
- LLMs allow us to scale expensive software-development techniques
  - Can formally verify software *at scale*
  - Eventually: write specs instead of code?

The AI revolution



- Software development is now LLM-assisted
- (So is research in CS, fun dinner topic)
- Conjecture: LLMs will create a software-quality divide
  - Good developers will produce better software faster
  - Bad developers will produce worse software faster
- LLMs allow us to scale expensive software-development techniques
  - Can formally verify software *at scale*
  - Eventually: write specs instead of code?
- Last year allowed use of AI/LLMs for assignments
- Saw *massively* different solutions from previous years
  - Much faster software
  - Large “boring” changes instead of small clever ones
  - Changes that were less connected to course content

The future of this course

- Course at Radboud only every second year

The future of this course



- Course at Radboud only every second year
- Running example of secure-channel protocol using
  - AES-GCM and Keccak as symmetric primitives
  - X-Wing (hybrid of X25519 and ML-KEM-768)
  - Bird-of-prey hybrid of Ed25519 and ML-DSA-65)

The future of this course

- Course at Radboud only every second year
- Running example of secure-channel protocol using
  - AES-GCM and Keccak as symmetric primitives
  - X-Wing (hybrid of X25519 and ML-KEM-768)
  - Bird-of-prey hybrid of Ed25519 and ML-DSA-65)
- Get rid of C. Move all code (slides and assignments) to Jasmin

The future of this course

- Course at Radboud only every second year
- Running example of secure-channel protocol using
  - AES-GCM and Keccak as symmetric primitives
  - X-Wing (hybrid of X25519 and ML-KEM-768)
  - Bird-of-prey hybrid of Ed25519 and ML-DSA-65)
- Get rid of C. Move all code (slides and assignments) to Jasmin
- Eventually four parts:
  1. Embedded microcontrollers (ARM Cortex-M4 and M7)

The future of this course

- Course at Radboud only every second year
- Running example of secure-channel protocol using
  - AES-GCM and Keccak as symmetric primitives
  - X-Wing (hybrid of X25519 and ML-KEM-768)
  - Bird-of-prey hybrid of Ed25519 and ML-DSA-65)
- Get rid of C. Move all code (slides and assignments) to Jasmin
- Eventually four parts:
  1. Embedded microcontrollers (ARM Cortex-M4 and M7)
  2. Larger processors (Intel and AMD 64-bit CPUs)

The future of this course



- Course at Radboud only every second year
- Running example of secure-channel protocol using
  - AES-GCM and Keccak as symmetric primitives
  - X-Wing (hybrid of X25519 and ML-KEM-768)
  - Bird-of-prey hybrid of Ed25519 and ML-DSA-65)
- Get rid of C. Move all code (slides and assignments) to Jasmin
- Eventually four parts:
  1. Embedded microcontrollers (ARM Cortex-M4 and M7)
  2. Larger processors (Intel and AMD 64-bit CPUs)
  3. Purpose-built hardware for crypto (`https://pavona.org`)

The future of this course



- Course at Radboud only every second year
- Running example of secure-channel protocol using
  - AES-GCM and Keccak as symmetric primitives
  - X-Wing (hybrid of X25519 and ML-KEM-768)
  - Bird-of-prey hybrid of Ed25519 and ML-DSA-65)
- Get rid of C. Move all code (slides and assignments) to Jasmin
- Eventually four parts:
  1. Embedded microcontrollers (ARM Cortex-M4 and M7)
  2. Larger processors (Intel and AMD 64-bit CPUs)
  3. Purpose-built hardware for crypto (`https://pavona.org`)
  4. Formally proving crypto software

The future of this course



- Course at Radboud only every second year
- Running example of secure-channel protocol using
  - AES-GCM and Keccak as symmetric primitives
  - X-Wing (hybrid of X25519 and ML-KEM-768)
  - Bird-of-prey hybrid of Ed25519 and ML-DSA-65)
- Get rid of C. Move all code (slides and assignments) to Jasmin
- Eventually four parts:
  1. Embedded microcontrollers (ARM Cortex-M4 and M7)
  2. Larger processors (Intel and AMD 64-bit CPUs)
  3. Purpose-built hardware for crypto (`https://pavona.org`)
  4. Formally proving crypto software
- Obviously *way* too much for 3-EC course. . .

The present of this course



The first five weeks

- Focus on Cortex-M4 as target platform
  - All example code in assembly or Jasmin for Cortex-M4
  - Close connection between lectures and assignment
- Assignments completely in Jasmin
- Still focus on (some) symmetric crypto and ECC
- Cover everything that's needed for the assignments

The present of this course



The first five weeks

- Focus on Cortex-M4 as target platform
  - All example code in assembly or Jasmin for Cortex-M4
  - Close connection between lectures and assignment
- Assignments completely in Jasmin
- Still focus on (some) symmetric crypto and ECC
- Cover everything that's needed for the assignments

The last two weeks

- Intro to larger CPUs (what is different)
- Intro to implementing PQC
- Intro/overview of high-assurance crypto