# Engineering Cryptographic Software

## Introduction

Peter Schwabe

January 2026

# What got us here?



# What got us here?



- First time in Mauritius in 2017

# What got us here?



- First time in Mauritius in 2017
- Met Logan Velvindron in 2024
- Learned about Cyberstorm
- Got in contact with Anwar Chutoo
- Gave a talk at MoU
- External examiner for BA program since 2025
- Idea of block lecture came up

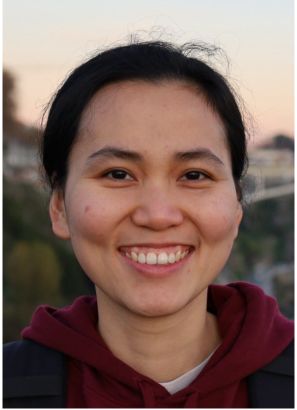
# What got us here?



- First time in Mauritius in 2017
- Met Logan Velvindron in 2024
- Learned about Cyberstorm
- Got in contact with Anwar Chutoo
- Gave a talk at MoU
- External examiner for BA program since 2025
- Idea of block lecture came up
- Convinced Hien and Amin to join
- Convinced them to do most of the work ;-)

# Hien, Amin, and Peter





**Hien Pham**
PhD student @ MPI-SP
nguyenhien.phamhoang@gmail.com

**Amin Abdulrahman**
PhD student @ MPI-SP
amin@abdulrahman.de

**Peter Schwabe**
Scientific Director @ MPI-SP
peter@cryptojedi.org

# MPI-SP



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

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

# MPI-SP



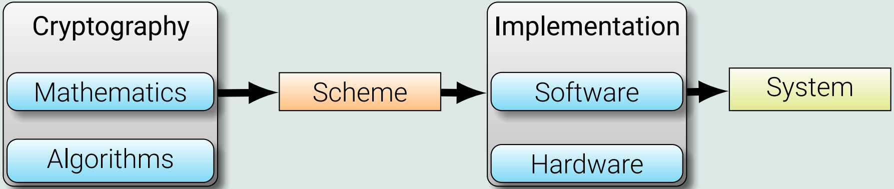
# The Cryptographic Engineering (EPOQUE) Department



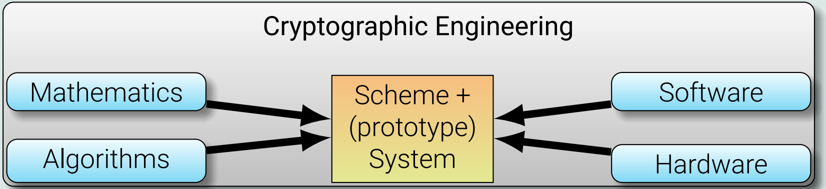
*“Cryptography [. . . ] is the practice and study of techniques for secure communication in the presence of adversarial behavior. [. . . ] Modern cryptography exists at the intersection of the disciplines of mathematics, computer science, information security, electrical engineering, digital signal processing, physics, and others.”*

—Wikipedia on *Cryptography*

## The traditional approach



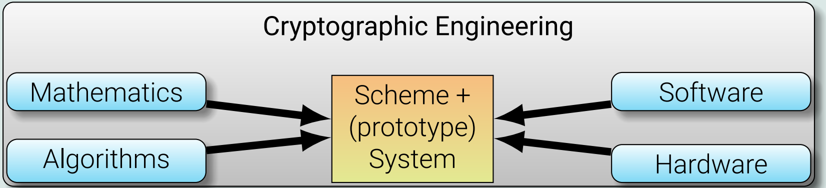
## A holistic approach



# Cryptographic Engineering



## A holistic approach




Motivation from real-world problems – aim to make real-world impact

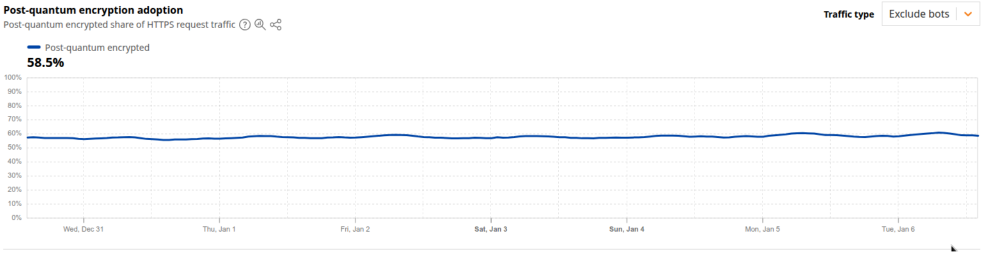
# Real-world impact: PQC deployment



[A very quick demo]

# Real-world impact: PQC deployment






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

- Hundreds of billions of connections per day at Cloudflare alone
- Also used in secure messaging (Signal, iMessage)
- Also in cloud infrastructure (AWS)

# 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

## 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

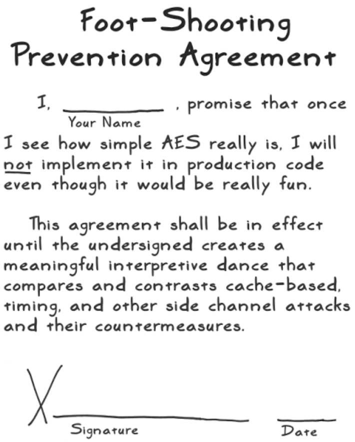


Cryptographic software is **small**, highly **performance critical**, highly **security critical**, and typically operates **in adversarial environments**.

# "Don't roll your own crypto"



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



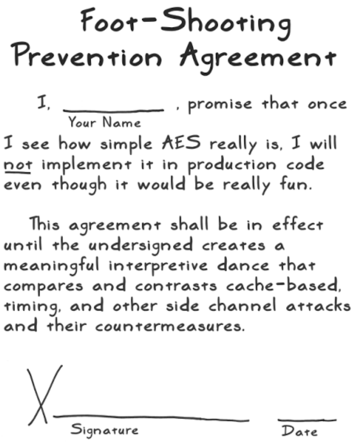

From *A Stick Figure Guide to the Advanced Encryption Standard (AES)*

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

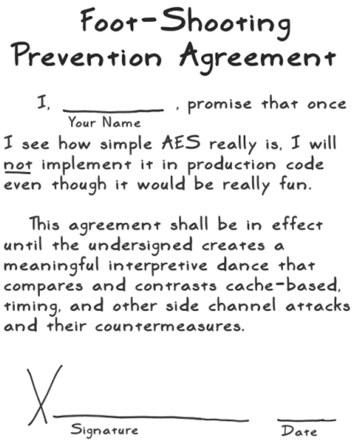
# "Don't roll your own crypto"



- 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

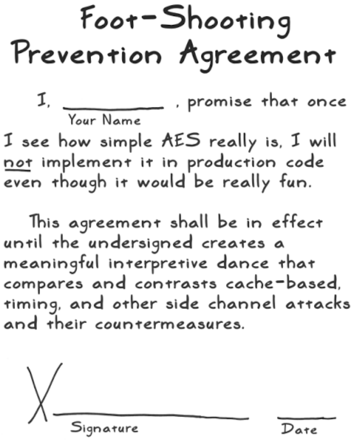
# “Don’t roll your own crypto”



- 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”

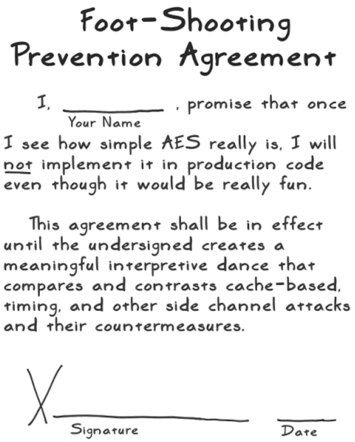


- Crypto is hard to get right
- Crypto software is hard to get right
- Need extensive independent review **before being used**

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 traditional approach



Most production crypto software in use today is written in C/assembly

# The traditional approach



## 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 traditional approach



## 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 traditional approach



## 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,

**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

ndu

Zı

**Abstract**
The implementations of most harde
use defensive programming technique
These techniques are usually specifiec
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 acro

# 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

ndu

Z

**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-
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-
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

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

Antoin
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

# Origins and idea for this course



## Origins

*"Engineering Cryptographic Software"* course at Radboud University (NL) since 2014

- Fundamentals of crypto software
- Symmetric crypto examples
- Elliptic-curve crypto
- Assignments in C/assembly
- Optimize on embedded microcontroller

# Origins and idea for this course



## Origins

*"Engineering Cryptographic Software"* course at Radboud University (NL) since 2014

- Fundamentals of crypto software
- Symmetric crypto examples
- Elliptic-curve crypto
- Assignments in C/assembly
- Optimize on embedded microcontroller

## Idea

- Modernize this course
- Get rid of C/assembly
- Move to dedicated crypto toolchain
- Teaching close to ongoing research
- Incorporate post-quantum crypto

# Plan for this one-week course



**6 Lectures**

- Cryptography on the Arm Cortex-M4
- The Jasmin Framework
- Scalar Multiplication
- Elliptic-curve Arithmetic
- Multiprecision Arithmetic
- More Cryptographic Software

# Plan for this one-week course



**6 Lectures**

- Cryptography on the Arm Cortex-M4
- The Jasmin Framework
- Scalar Multiplication
- Elliptic-curve Arithmetic
- Multiprecision Arithmetic
- More Cryptographic Software

**4 “Assignments”**

- Getting set up
- Adding up 1000 integers
- ChaCha20
- Elliptic-curve Diffie-Hellman

# Plan for this one-week course



6 Lectures

- Cryptography on the Arm Cortex-M4
- The Jasmin Framework
- Scalar Multiplication
- Elliptic-curve Arithmetic
- Multiprecision Arithmetic
- More Cryptographic Software

4 “Assignments”

- Getting set up
- Adding up 1000 integers
- ChaCha20
- Elliptic-curve Diffie-Hellman

It’s all flexible – we’re all learning here!

# Course website



https://cryptojedi.org/peter/teaching/engineering-crypto-software-mru2026.shtml