CS 7480
Special Topics in PL

Formal Security for Cryptography

Joshua Gancher

This Class

- Seminar-style class on the following topic:

  How can we use formal methods to make cryptography more secure?

- Why would I care?
  - Increasingly important, practical area of research
  - Interdisciplinary area of research: many opportunities
  - Spans the range from highly applied to theoretical
  - Spans multiple **research styles**: Systems, PL, Applied Verification

Today

- Logistics, Introductions
- Course Overview
  - “SoK: Computer-Aided Cryptography”

- Goals for the class:
  - Bring you up to speed in this area of research (Computer-Aided Crypto)
  - Get practice critically reading research papers
  - Carry out a research project

- Background for this area: PL, Crypto, Verification
  - Nobody is expected to be an expert in all (or any) of these!
  - Only requirement from you: a willingness to learn
  - Supplemental background reading will be provided
  - When in doubt about a topic, ask me

Class Resources

- Course Webpage: https://gancher.dev/CS7480_Fall2024/class.html
- Course assignments, link to syllabus on Canvas
- Office Hours: by appointment
- Contact me:
  - j.gancher@northeastern.edu (include CS7480 in the subject line)
  - Office: WVH 360

Coursework

- Reading, responding to papers
  - Fill out a small questionnaire; ~15-20min after reading the paper
- In-class participation
- Self-directed Final Project

Grading Policy on syllabus:

40% paper responses, 40% final project, 20% participation

Please talk to me if you are feeling lost / need support in the class!

Class Format

- Come to class having read the paper, filled out questionnaire
- I may/may not give a brief background lecture
- Paper discussion, **guided by questionnaire responses**

Introductions

Course Overview

SoK: Computer-Aided Cryptography

Manuel Barbosa*, Gilles Barthe†‡, Karthik Bhargavan§, Bruno Blanchet§, Cas Cremers¶, Kevin Liao†‖, Bryan Parno**
*University of Porto (FCUP) and INESC TEC, †Max Planck Institute for Security & Privacy, ‡IMDEA Software Institute,
§INRIA Paris, ¶CISPA Helmholtz Center for Information Security, ‖MIT, **Carnegie Mellon University

Crypto is Essential

Use:

Encryption
Digital Signatures
Key Derivation Functions
...

to get

Confidentialy
Integrity
Authentication

Crypto is Complicated


```
while (1) {
        switch (st->write_state) {
        case WRITE_STATE_TRANSITION:
            if (cb != NULL) {
                /* Notify callback of an impending state change */
                if (s->server)
                    cb(ssl, SSL_CB_ACCEPT_LOOP, 1);
                else
                    cb(ssl, SSL_CB_CONNECT_LOOP, 1);
            }
            switch (transition(s)) {
            case WRITE_TRAN_CONTINUE:
                st->write_state = WRITE_STATE_PRE_WORK;
                st->write_state_work = WORK_MORE_A;
                break;
            case WRITE_TRAN_FINISHED:
                return SUB_STATE_FINISHED;
                break;
            …
            }
        case …:
          …
        }
}
```

Complex low-level state machines

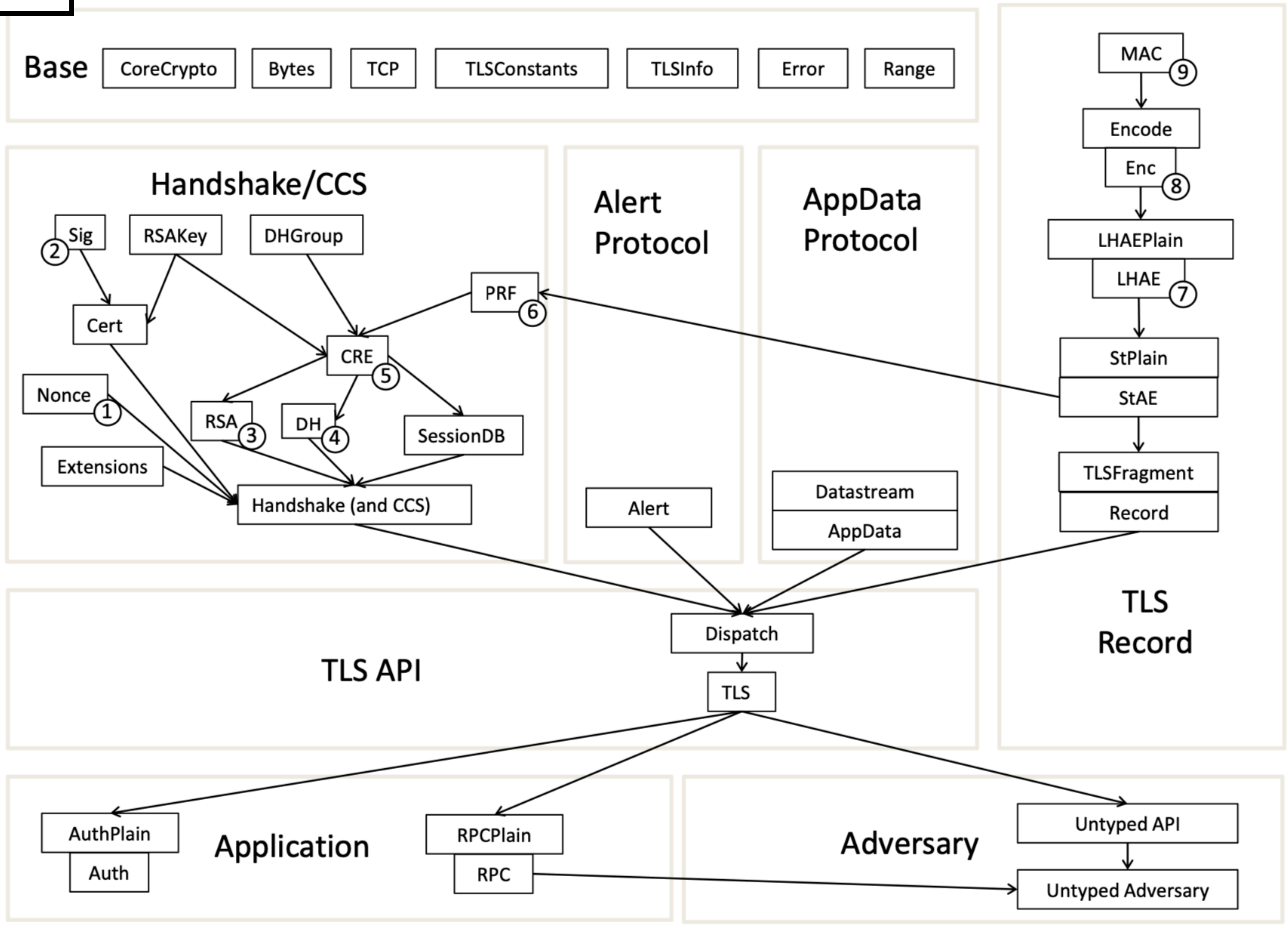
```
.text
.global _aes128_key_expansion
_aes128_key_expansion:
  movdqu 0(%rdi), %xmm1
  mov %rsi, %rdx
  movdqu %xmm1, 0(%rdx)
  aeskeygenassist $1, %xmm1, %xmm2
  pshufd $255, %xmm2, %xmm2
  vpslldq $4, %xmm1, %xmm3
  pxor %xmm3, %xmm1
  vpslldq $4, %xmm1, %xmm3
  pxor %xmm3, %xmm1
  vpslldq $4, %xmm1, %xmm3
  pxor %xmm3, %xmm1
  pxor %xmm2, %xmm1
```

Hand-optimized assembly

2013 IEEE Symposium on Security and Privacy

Implementing TLS with Verified Cryptographic Security

Karthikeyan Bhargavan*, Cédric Fournet†, Markulf Kohlweiss†, Alfredo Pironti*, Pierre-Yves Strub‡
*INRIA Paris-Rocquencourt, {karthikeyan.bhargavan,alfredo.pironti}@inria.fr
†Microsoft Research, {fournet,markulf}@microsoft.com
‡IMDEA Software, pierre-yves@strub.nu



Crypto can go wrong

ALPACA —

Hackers can mess with HTTPS connections by

This class: preventing vulnerabilities before they happen.

software to overseas customers

PS session

cookies, researchers say

ATTACK OF THE CLONES —

YubiKeys are vulnerable to cloning attacks thanks to newly discovered side channel

Sophisticated attack breaks security assurances of the most popular FIDO key.

DAN GOODIN - 9/3/2024, 1:58 PM

Yubico

https://arstechnica.com/security/2024/09/yubikeys-are-vulnerable-to-cloning-attacks-thanks-to-newly-discovered-side-channel/

```
Internet Engineering Task Force (IETF)                    E. Rescorla
Request for Comments: 8446                                    Mozilla
Obsoletes: 5077, 5246, 6961                               August 2018
Updates: 5705, 6066
Category: Standards Track
ISSN: 2070-1721


        The Transport Layer Security (TLS) Protocol Version 1.3
```

RFC Document: 160+ pages of English prose

Language Breakdown

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C	607,114	100,206	14.2%	92,580	799,900	62.4%
Perl	234,537	133,516	36.3%	78,208	446,261	34.8%

https://openhub.net/p/openssl/analyses/latest/languages_summary

What can go wrong?

Protocol Design

```
Internet Engineering Task Force (IETF)                       E. Rescorla
Request for Comments: 8446                                       Mozilla
Obsoletes: 5077, 5246, 6961                                  August 2018
Updates: 5705, 6066
Category: Standards Track
ISSN: 2070-1721


        The Transport Layer Security (TLS) Protocol Version 1.3
```

C, Asm

Protocol Implementation

Design-Level Vulnerabilities

- unsound invariants
- ciphertext leakages
- underspecification

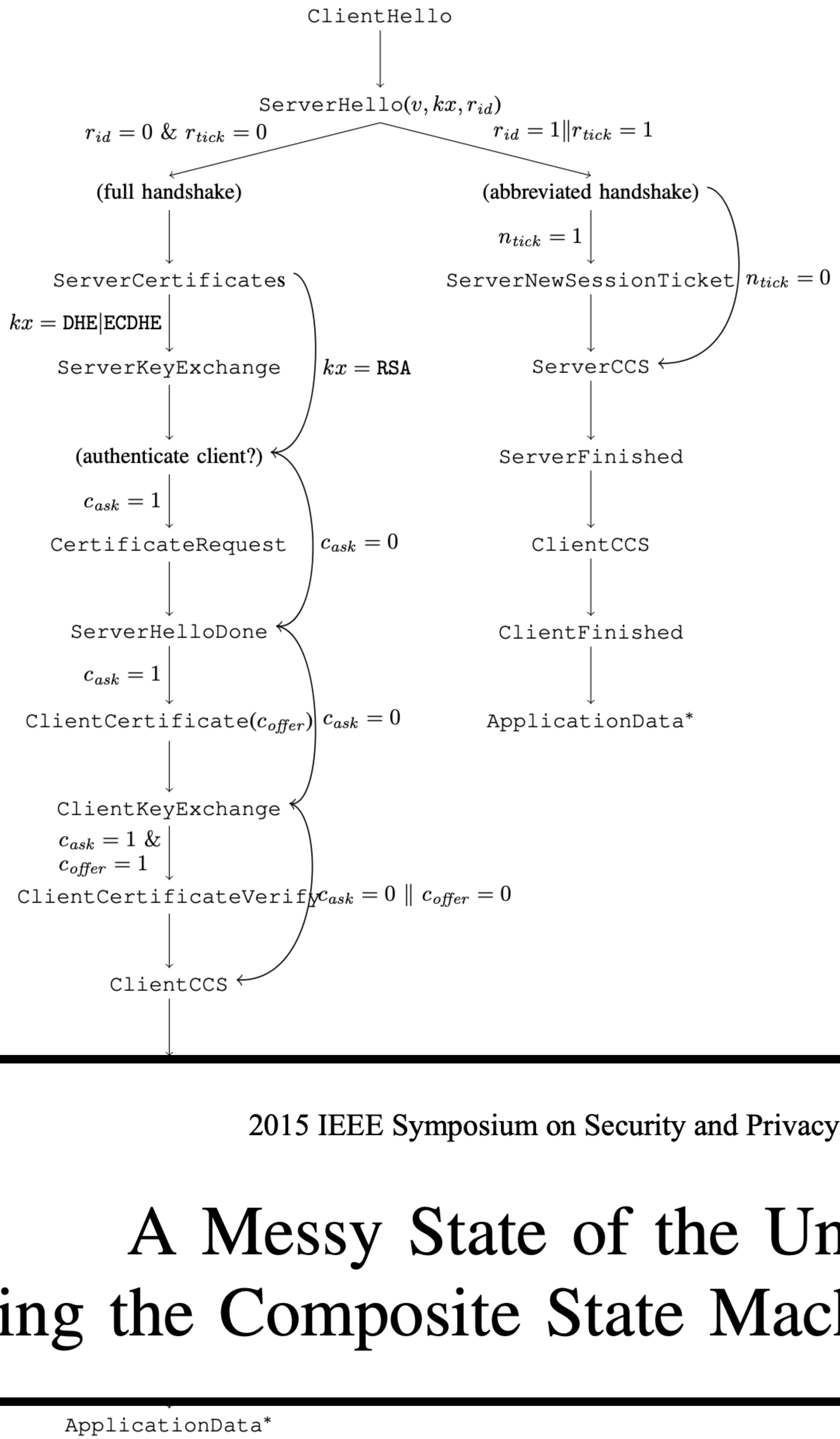
Implementation-Level Vulnerabilities

- incorrect implementations
- timing leakages
- buffer overflows

What can go wrong?
Design-Level Security

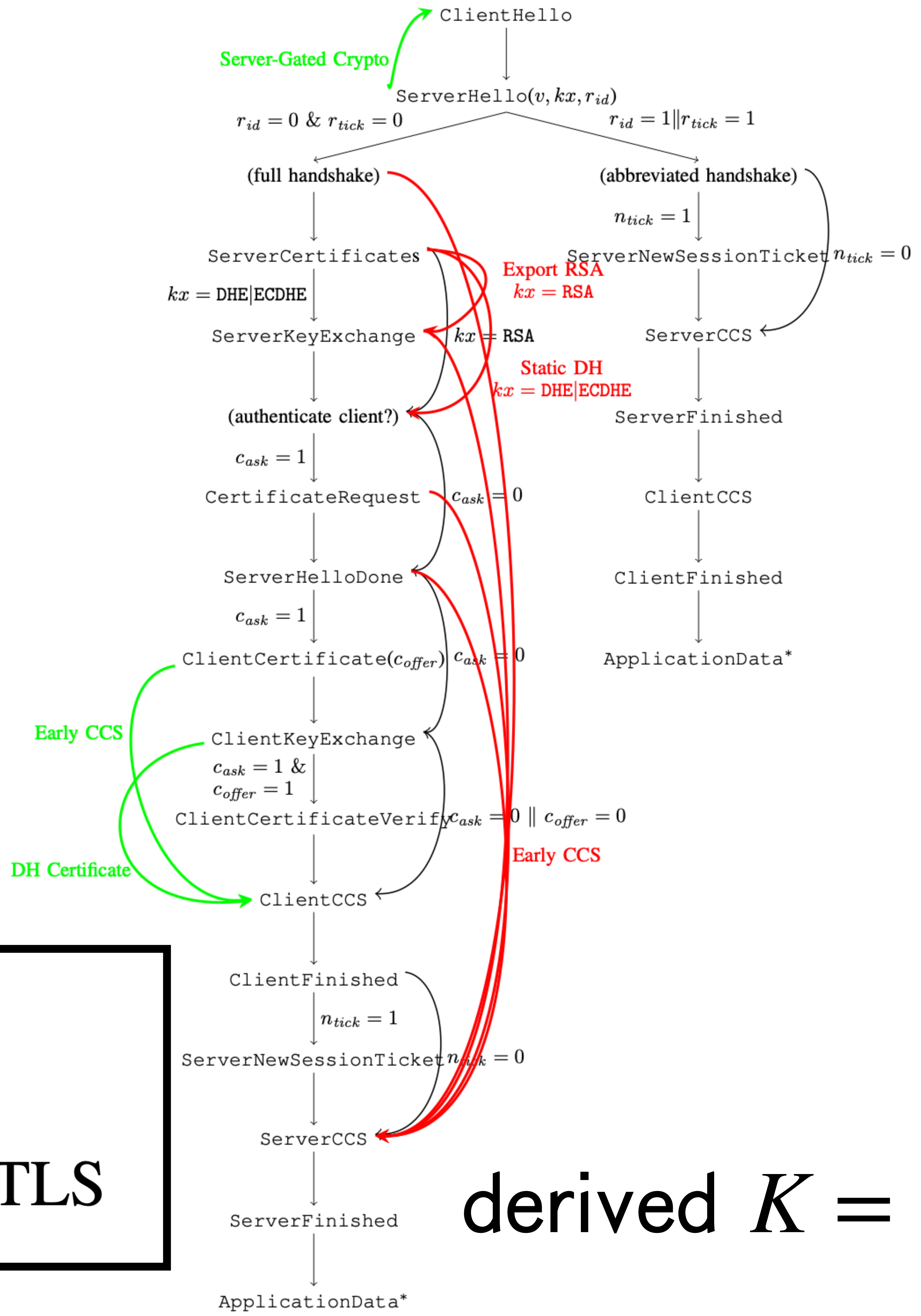
- The protocol can be insecure in the first place

  - Examples:
    - encrypting under the wrong key
    - confusing different clients
    - misunderstanding security guarantees of the crypto
    - format confusion attack



2015 IEEE Symposium on Security and Privacy

A Messy State of the Union: Taming the Composite State Machines of TLS

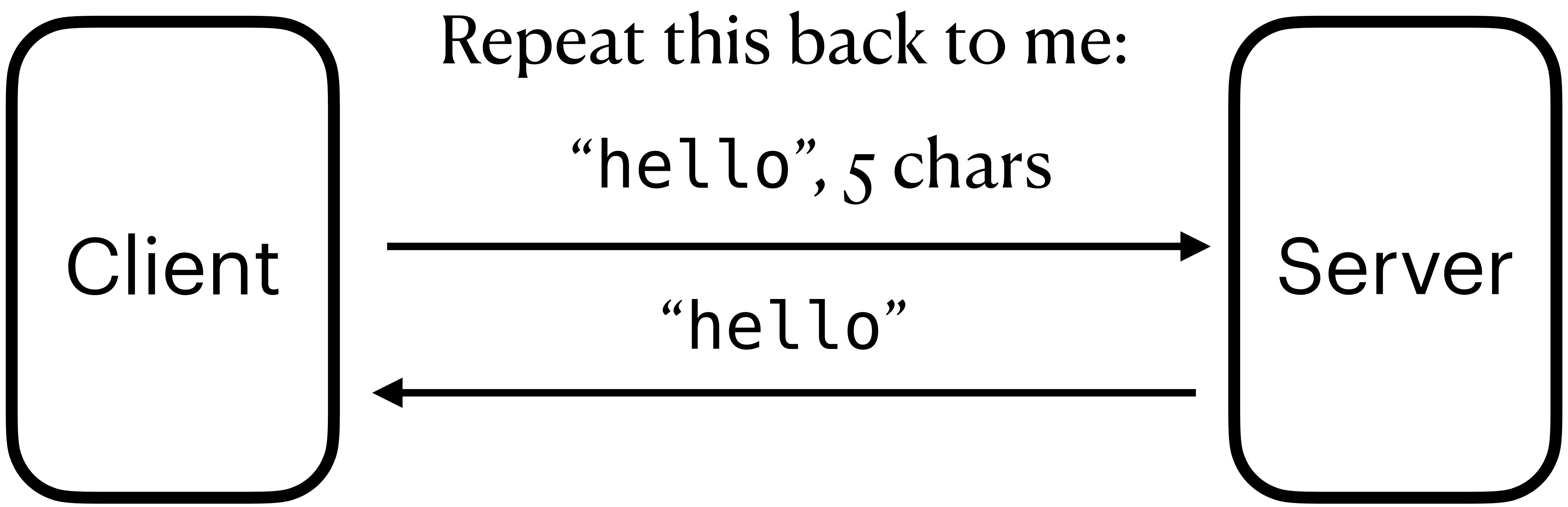


derived $K = 0$!

What can go wrong?
Functional Correctness

- The implementation can behave badly
  - Examples:
    - Concurrency-related bugs
    - Corner cases in state machines
    - Buffer overflows
    - Hard-to-notice errors in handwritten assembly

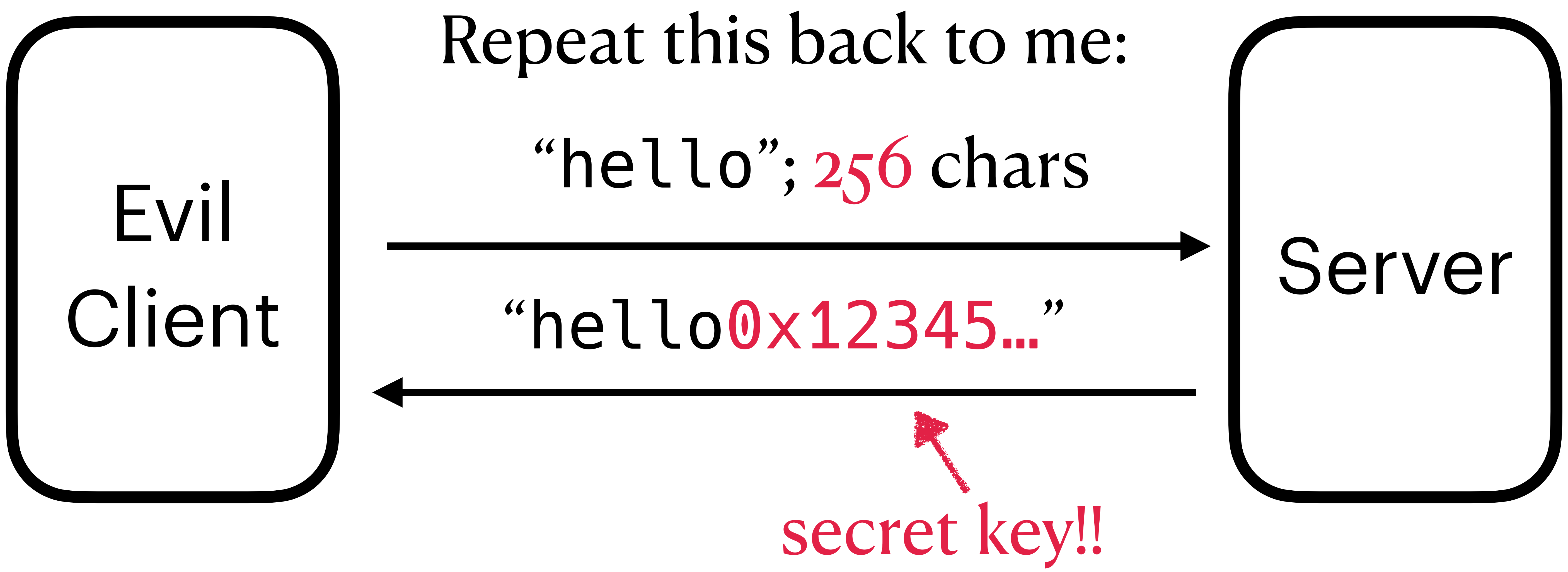
Heartbleed Attack




```
struct HeartbeatHello {
  uint16 length;
  bytes[payload] payload
 }
```

```
void ProcessHeartbeat(h) {
    netsend(h.payload, h.length);
}
```

Heartbleed Attack




```
struct HeartbeatHello {
   uint16 length;
   bytes[payload] payload
  }
```

```
void ProcessHeartbeat(h) {
    netsend(h.payload, h.length);
}
```

Heartbleed Attack

Repercussions

Need to update OpenSSL

Send those updates to entire internet

Locate compromised TLS certificates

Send certificate revocations

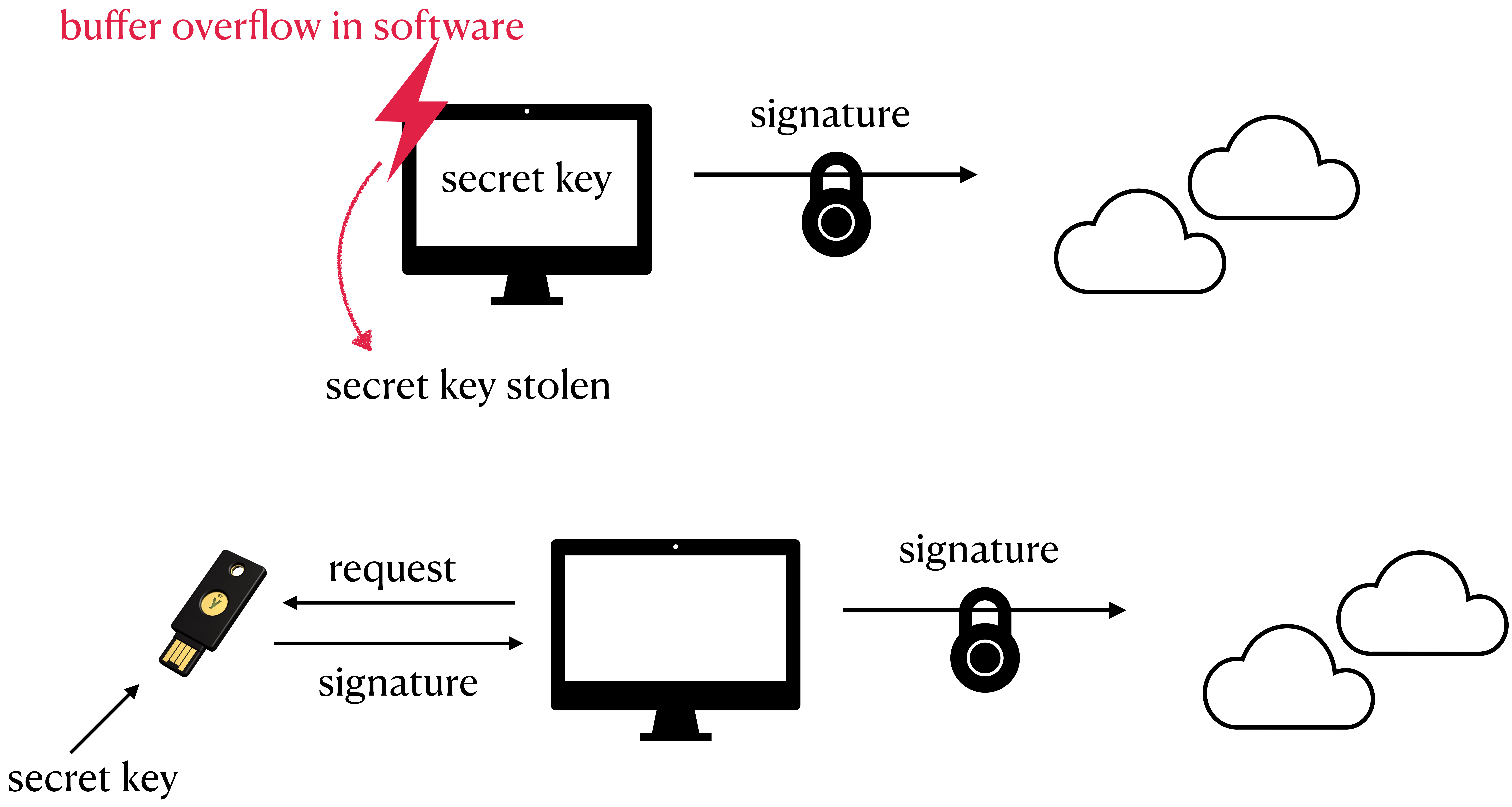
“supporting the traffic to deliver the CRL would have added $400,000USD to Globalsign's monthly bandwidth bill.”

CloudFlare, 2014

What can go wrong?
Side-Channel Leakages

- The implementation can be **insecure**: leak more than intended

- Examples:
  - Timing side channels:
    - `if lastBit(key) == 0 then doSlowThing else doFastThing`
    - YubiKey vulnerability
  - Memory side channels:
    - `A[secret] = 0:` can leave behind traces of the secret in cache
    - `Spectre, Meltdown`


buffer overflow in software
secret key
signature
secret key stolen
request
signature
signature
secret key

EUCLEAK

Side-Channel Attack on the YubiKey 5 Series

(Revealing and Breaking Infineon ECDSA Implementation on the Way)

Thomas ROCHE

NinjaLab, Montpellier, France
thomas@ninjalab.io

September 3^{rd}, 2024

ECDSA Signature:

- Long-term private key D
- To sign a message M:
  - Generate nonce K
  - Use D, K, M ==> generate signature
    - Involves computing (K^{-1} mod N)
  - **Throw away the nonce K**

Know K, M, signature ⟶ Can compute private key D

To compute K^{-1} mod N:

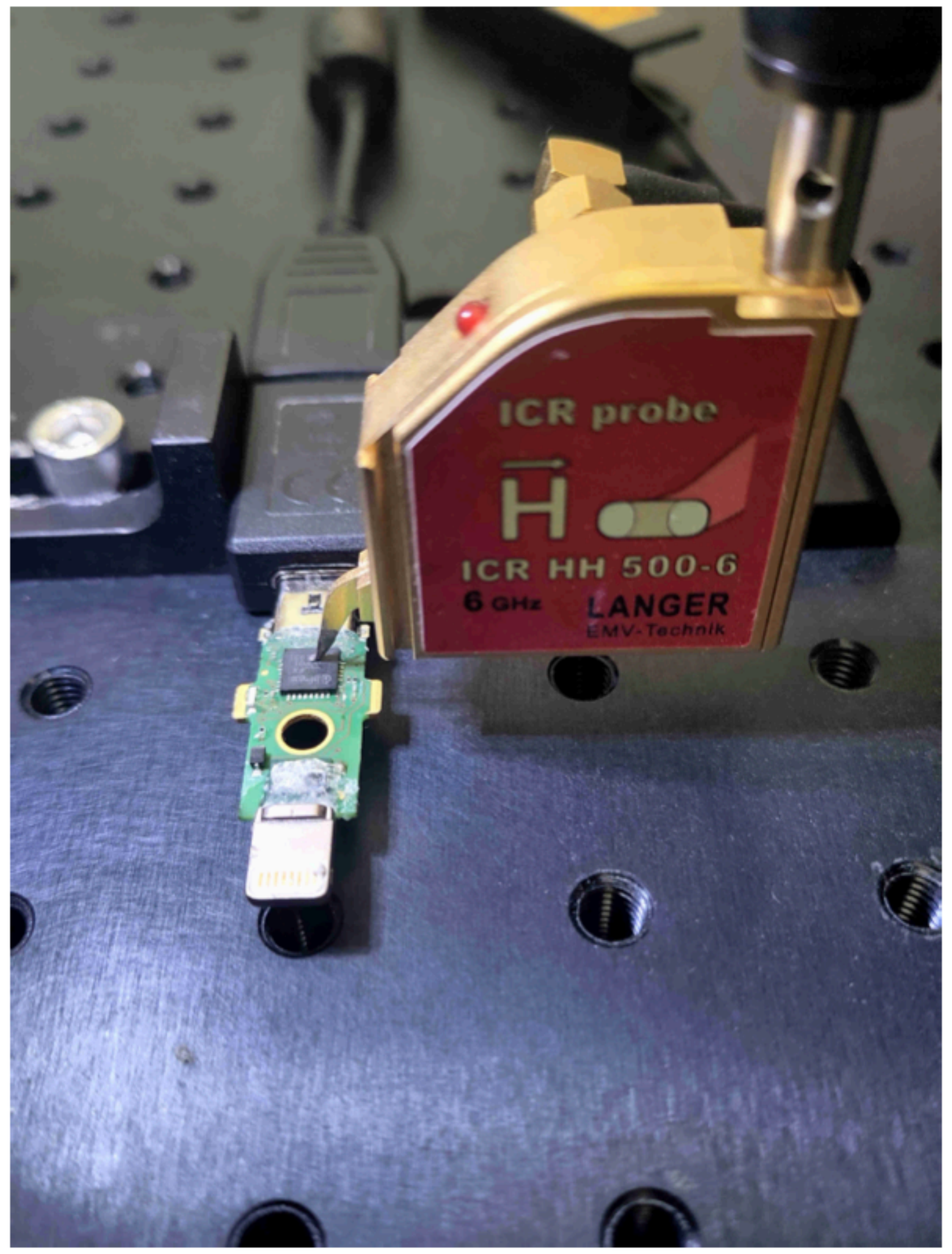
Algorithm 1: Extended Euclidean Algorithm for Modular Inversion

Input : v, n: two positive integers with $v \leq n$ and $gcd(v, n) = 1$
Output: $v^{-1} \bmod n$: the inverse of v modulo n

$r_0, r_1 \leftarrow n, v$
$t_0, t_1 \leftarrow 0, 1$
while $r_1 \neq 0$ **do**
  $q \leftarrow \texttt{div}(r_0, r_1)$
  $r_0, r_1 \leftarrow r_1, r_0 - q.r_1$
  $t_0, t_1 \leftarrow t_1, t_0 - q.t_1$
end
if $t_0 < 0$ **then**
  $t_0 \leftarrow t_0 + n$
end
return t_0

Number of loops depends on value of input!

Can **time** the code to deduce information about K



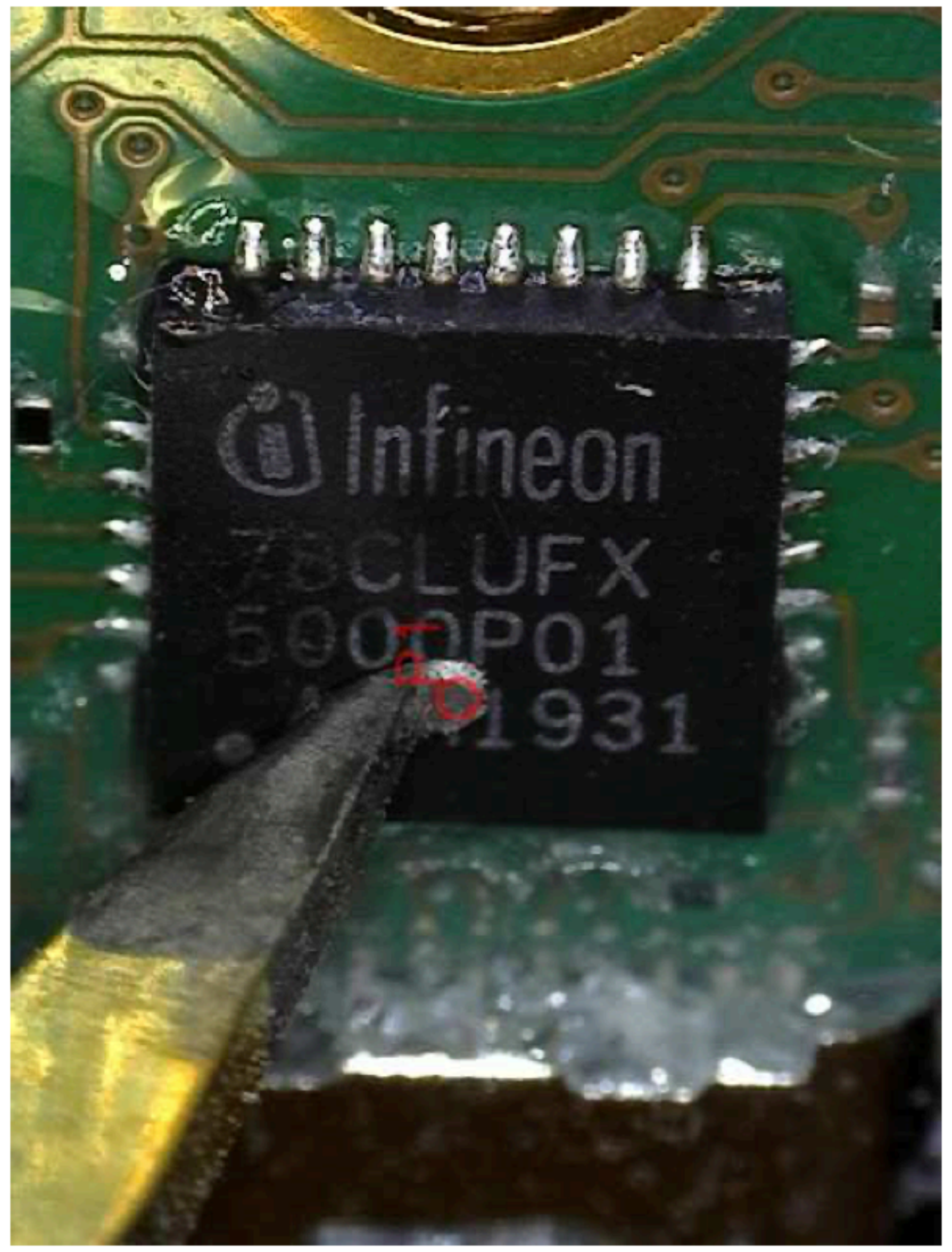
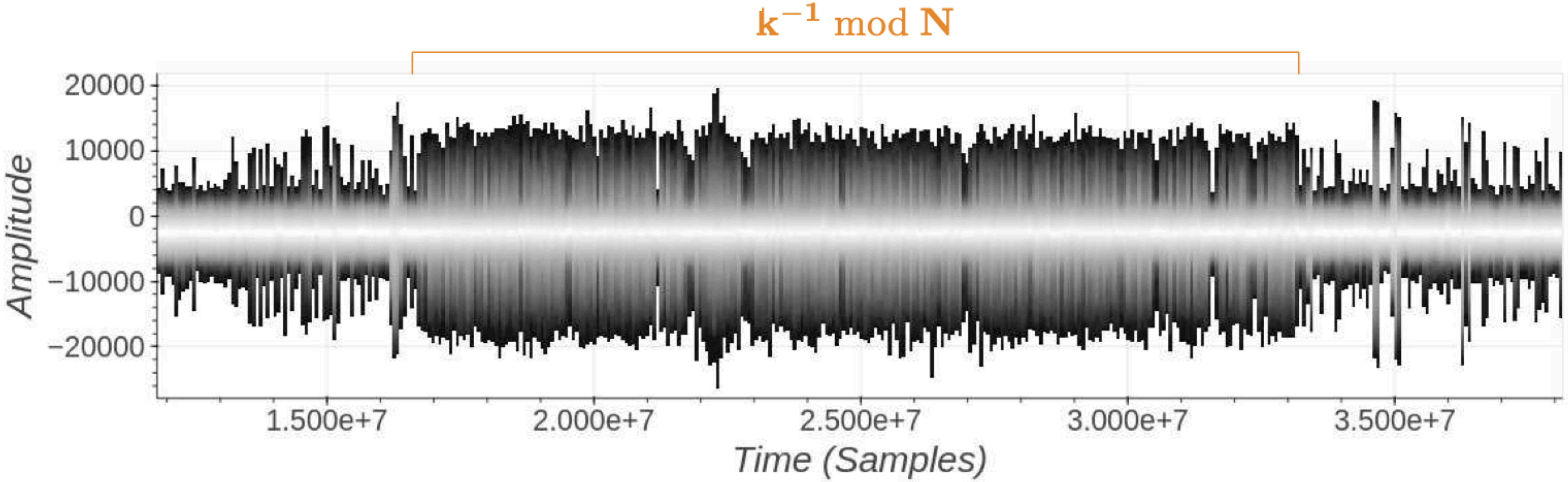


Figure 1.4: `YubiKey 5Ci` – EM Acquisition Setup

Along with (many) other tricks,

allows you to extract value of private key



What can we do?

- Use formal methods!
- **Mathematically prove** that cryptographic software isn't vulnerable

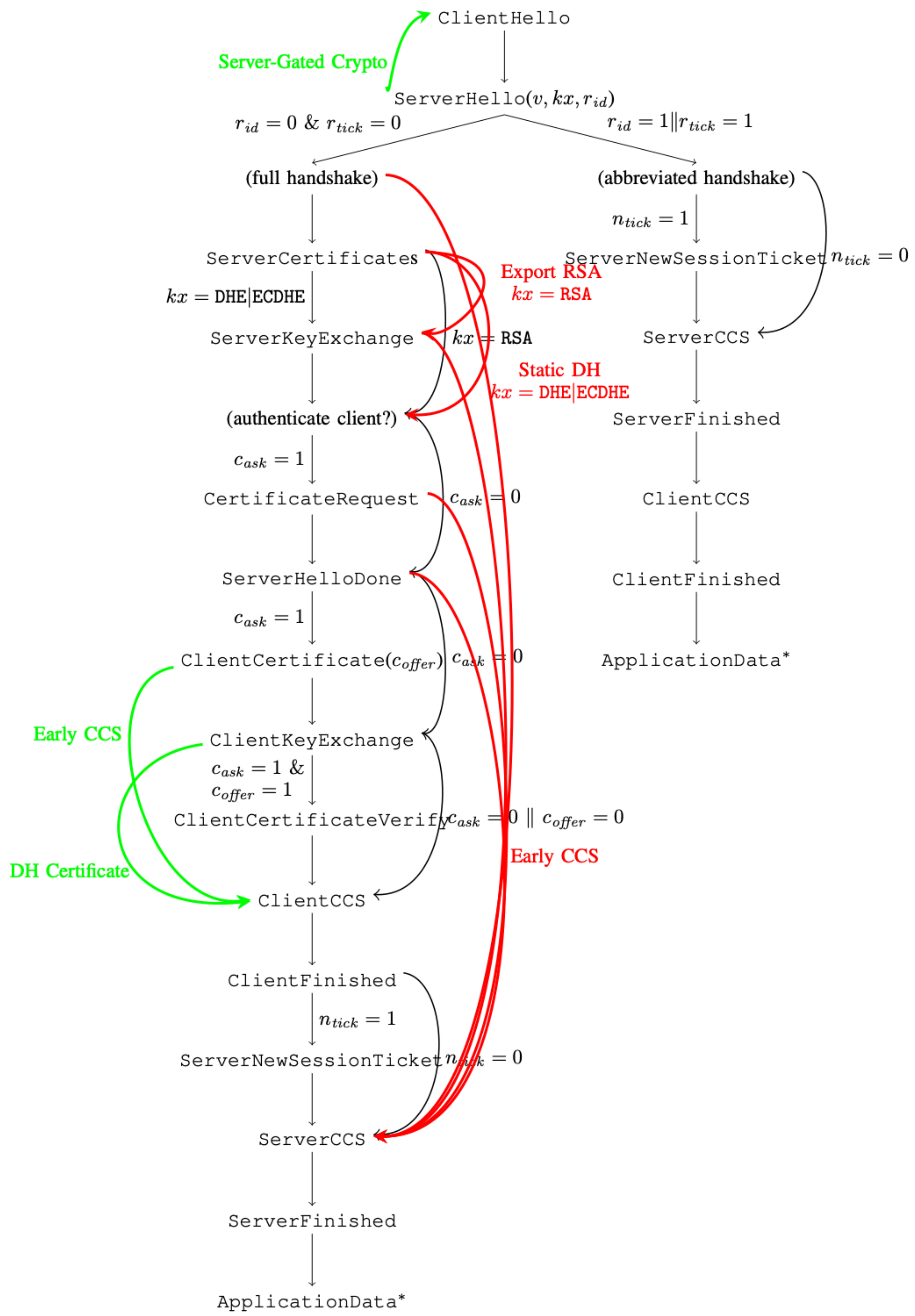
Type Systems

automatically **type check** the code

Theorem Provers

mechanize **formal proofs** about the code

Formal Methods to the Rescue



mechanized proofs

of security for protocols, state machines

mechanized proofs

of functional correctness,

memory safety

EUCLEAK

mechanized proofs

of side-channel resistance

Design-Level Security

Tool		Unbound	Trace	Equiv	Eq-thy	State	Link
CPSA▷	[16]	●	●	○	○	●	○
F7◇	[17]	●	●	○	◐	●	●
↳F5◇	[18]	●	●	○	◐	●	●
Maude-NPA▷	[19]	●	●	●d	●	○	○
ProVerif⋆†	[20]	●	●	●d	◐	○	○
↳fs2pv◇†	[21]	●	●	○	◐	○	●
↳GSVerif⋆†	[22]	●	●	○	◐	●	○
↳ProVerif-ATP⋆†	[23]	●	●	○	●	○	○
↳StatVerif⋆†	[24]	●	●	●d	◐	●	○
Scyther▷	[25]	●	●	○	○	○	○
scyther-proof▷‡§	[26]	●	●	○	○	○	○
Tamarin∗‡	[27]	●	●	●d	●	●	○
↳SAPIC⋆	[28]	●	●	○	●	●	○
CI-AtSe▷	[29]	○	●	○	●	●	○
OFMC▷†	[30]	○	●	○	◐	●	○
SATMC▷	[31]	○	●	○	○	●	○
AKISS⋆	[32]	○	○	●t	●	●	○
APTE⋆	[33]	○	○	●t	○	●	○
DEEPSEC⋆	[34]	○	○	●t	◐	●	○
SAT-Equiv⋆	[35]	○	○	●t	○	○	○
SPEC⋆,§	[36]	○	○	●o	○	○	○

Specification language
- ▷ – security protocol notation
- ⋆ – process calculus
- ∗ – multiset rewriting
- ◇ – general programming language

Miscellaneous symbols
- ↳ – previous tool extension
- † – abstractions
- ‡ – interactive mode
- § – independent verifiability

Equational theories (Eq-thy)
- ● – with AC axioms
- ◐ – without AC axioms
- ○ – fixed

Equivalence properties (Equiv)
- *t* – trace equivalence
- *o* – open bisimilarity
- *d* – diff equivalence

TABLE I

OVERVIEW OF TOOLS FOR SYMBOLIC SECURITY ANALYSIS. SEE SECTION II-B FOR MORE DETAILS ON COMPARISON CRITERIA.

Tool		RF	Auto	Comp	CS	Link	TCB
AutoG&P◇	[55]	◐	●	○	◐	○	self, SMT
CertiCrypt▷◇	[56]	◑	○	○	●	●	Coq
CryptHOL◇	[57]	◑	○	●	◐	○	Isabelle
CryptoVerif⋆◇	[58]	◐	●	○	●	●	self
EasyCrypt▷◇	[59]	◑	○	●	◐	●	self, SMT
F7◇	[17]	◐	○	●	○	●	self, SMT
F*◇	[60]	◐	○	●	○	●	self, SMT
FCF◇	[61]	◑	○	●	◐	●	Coq
ZooCrypt◇	[62]	◐	●	○	●	○	self, SMT

Reasoning Focus (RF)
- ◐ – automation focus
- ◑ – expressiveness focus

Concrete security (CS)
- ● – security + efficiency
- ◐ – security only
- ○ – no support

Specification language
- ⋆ – process calculus
- ▷ – imperative
- ◇ – functional

TABLE II

OVERVIEW OF TOOLS FOR COMPUTATIONAL SECURITY ANALYSIS. SEE SECTION II-D FOR MORE DETAILS ON COMPARISON CRITERIA.

Symbolic Security

Computational Security

Functional Correctness

Tool		Memory safety	Automation	Parametric verification	Input language	Target(s)	TCB
Cryptol + SAW	[97]	●	◐	○	C, Java	C, Java	SAT, SMT
CryptoLine	[98]	○	●	○	CryptoLine	C	Boolector, MathSAT, Singular
Dafny	[99]	●	◐	○	Dafny	C#, Java, JavaScript, Go	Boogie, Z3
F*	[60]	●	◐	○	F*	OCaml, F#, C, Asm, Wasm	Z3, typechecker
Fiat Crypto	[6]	●	○	●	Gallina	C	Coq, C compiler
Frama-C	[100]	●	◐	○	C	C	Coq, Alt-Ergo, Why3
gfverif	[101]	○	●	○	C	C	g++, Sage
Jasmin	[102]	●	◐	○	Jasmin	Asm	Coq, Dafny, Z3
Vale	[103], [104]	●	◐	●	Vale	Asm	Dafny or F*, Z3
VST	[105]	●	○	○	Gallina	C	Coq
Why3	[106]	○	◐	○	WhyML	OCaml	SMT, Coq

Automation

● – automated ◐ – automated + interactive ○ – interactive

TABLE III

OVERVIEW OF TOOLS FOR FUNCTIONAL CORRECTNESS. SEE SECTION III-B FOR MORE DETAILS ON COMPARISON CRITERIA.

Side-Channel Security

Tool		Target	Method	Synthesis	Sound	Complete	Public inputs	Public outputs	Control flow	Memory access	Variable-time op.
ABPV13	[132]	C	DV	○	●	●	●	○	●	●	○
CacheAudit	[133]	Binary	Q	○	●	○	○	○	●	●	○
ct-verif	[134]	LLVM	DV	○	●	●	●	●	●	●	●
CT-Wasm	[135]	Wasm	TC	○	●	○	●	○	●	●	●
FaCT	[136]	LLVM	TC	●	●	○	●	○	●	●	●
FlowTracker	[137]	LLVM	DF	○	●	○	●	○	●	●	○
Jasmin	[102]	asm	DV	○	●	●	●	●	●	●	○
KMO12	[138]	Binary	Q	○	●	○	○	○	○	●	○
Low*	[139]	C	TC	○	●	○	●	○	●	●	●
SC Eliminator	[140]	LLVM	DF	●	●	○	●	○	●	●	○
Vale	[103]	asm	DF	○	●	○	●	●	●	●	●
VirtualCert	[141]	x86	DF	○	●	○	●	○	●	●	○

Method			
TC – type-checking	DF – data-flow analysis	DV – deductive verification	Q – Quantitative

TABLE V

OVERVIEW OF TOOLS FOR SIDE-CHANNEL RESISTANCE. SEE SECTION IV-B FOR MORE DETAILS ON TOOL FEATURES.

Some Case Studies

Implementation(s)		Target(s)	Tool(s) used	Computational security	Functional correctness	Efficiency	Side-channel resistance
RSA-OEAP	[172]	C	EasyCrypt, Frama-C, CompCert	●	●	○	●
Curve25519 scalar mult. loop	[114]	asm	Coq, SMT	—	●	●	○
SHA-1, SHA-2, HMAC, RSA	[131]	asm	Dafny, BoogieX86	—	●	●	○
HMAC-SHA-2	[173]	C	FCF, VST, CompCert	●	●	○	○
MEE-CBC	[174]	C	EasyCrypt, Frama-C, CompCert	●	●	○	●
Salsa20, AES, ZUC, FFS, ECDSA, SHA-3	[175]	Java, C	Cryptol, SAW	○	◐	◐	○
Curve25519	[176]	OCaml	F*, Sage	—	◐	○	◐
Salsa20, Curve25519, Ed25519	[102]	asm	Jasmin	○	○	●	●
SHA-2, Poly1305, AES-CBC	[103]	asm	Vale	○	●	○	●
HMAC-DRBG	[177]	C	FCF, VST, CompCert	●	●	○	○
HACL*[1]	[5]	C	F*	◐	◐	◐	◐
HACL*[1]	[5]	C	F*, CompCert	◐	●	○	●
HMAC-DRBG	[178]	C	Cryptol, SAW	○	◐	◐	○
SHA-3	[69]	asm	EasyCrypt, Jasmin	●	●	●	●
ChaCha20, Poly1305	[117]	asm	EasyCrypt, Jasmin	○	●	●	●
BGW multi-party computation protocol	[179]	OCaml	EasyCrypt, Why3	●	◐	○	○
Curve25519, P-256	[6]	C	Fiat Crypto	—	◐	◐	○
Poly1305, AES-GCM	[104]	asm	F*, Vale	○	●	●	●
Bignum code[4]	[98]	C	CryptoLine	—	●	◐	○
WHACL*[1], LibSignal*	[180]	Wasm	F*	◐	●	◐	●
EverCrypt[2]	[7]	C	F*	○	◐	◐	◐
EverCrypt[3]	[7]	asm	F*, Vale	○	●	●	●

Computational security	Functional correctness	Efficiency	Side-channel resistance
● – verified	● – target-level	● – comparable to asm ref	● – target-level
◐ – partially verified	◐ – source-level	◐ – comparable to C ref	◐ – source-level
○ – not verified	○ – not verified	○ – slower than C ref	○ – not verified
— – not applicable			

[1](ChaCha20, Salsa20, Poly1305, SHA-2, HMAC, Curve25519, Ed25519) [2](MD5, SHA-1, SHA-2, HMAC, Poly1305, HKDF, Curve25519, ChaCha20)
[3](AES-GCM, ChaCha20, Poly1305, SHA-2, HMAC, HKDF, Curve25519, Ed25519, P-256) [4](In NaCl, wolfSSL, OpenSSL, BoringSSL, Bitcoin)

TABLE VI

VERIFIED CRYPTOGRAPHIC IMPLEMENTATIONS AND THEIR FORMAL GUARANTEES.

Simple High-Level Code For Cryptographic Arithmetic – With Proofs, Without Compromises

Andres Erbsen Jade Philipoom Jason Gross Robert Sloan Adam Chlipala
MIT CSAIL,
Cambridge, MA, USA
{andreser, jadep, jgross}@mit.edu, rob.sloan@alum.mit.edu, adamc@csail.mit.edu

Integrated into BoringSSL → Chrome browser
Integrated into BoringSSL → Android

roughly **half** of all HTTPs connections mediated by verified code

Class Plan

- Part 1: Background and Overview
  - Today and next class
  - Get you up to speed for Part 2
- Part 2: Protocol Security
  - Verifying high-level designs of cryptographic protocols
- Part 3: Implementation Security
  - Functional Correctness, side-channel security of low-level crypto
- Part 4: Additional Topics, subject to interest

Next Class (Sep 10)

- Introduction to some of the technical ideas in the class

- Verification Bootcamp:
  - Specifying languages via syntax + semantics
  - Formal logic and type systems
  - Verification tools (Dafny and Coq)

- Provable Security:
  - Foundations:
    - Polynomial-Time Algorithms, Hardness Assumptions
    - The Symbolic Model of Cryptography
  - Cryptographic Games: Encryption, Digital Signatures, Hash Functions
  - Specifying Security for Protocols (TLS, WireGuard, …)

First Paper (Friday, Sep 13)

A Comprehensive Symbolic Analysis of TLS 1.3

Cas Cremers
University of Oxford, UK

Marko Horvat
MPI-SWS, Germany

Jonathan Hoyland
Royal Holloway, University of London, UK

Sam Scott
Royal Holloway, University of London, UK

Thyla van der Merwe
Royal Holloway, University of London, UK

Supplementary Reading:

Security Protocol Verification:

Symbolic and Computational Models