

Computer-Aided Security Proofs, Aarhus, Oct 9–13 2017

Security Verification with F*

Cédric Fournet

Catalin Hritcu

Aseem Rastogi

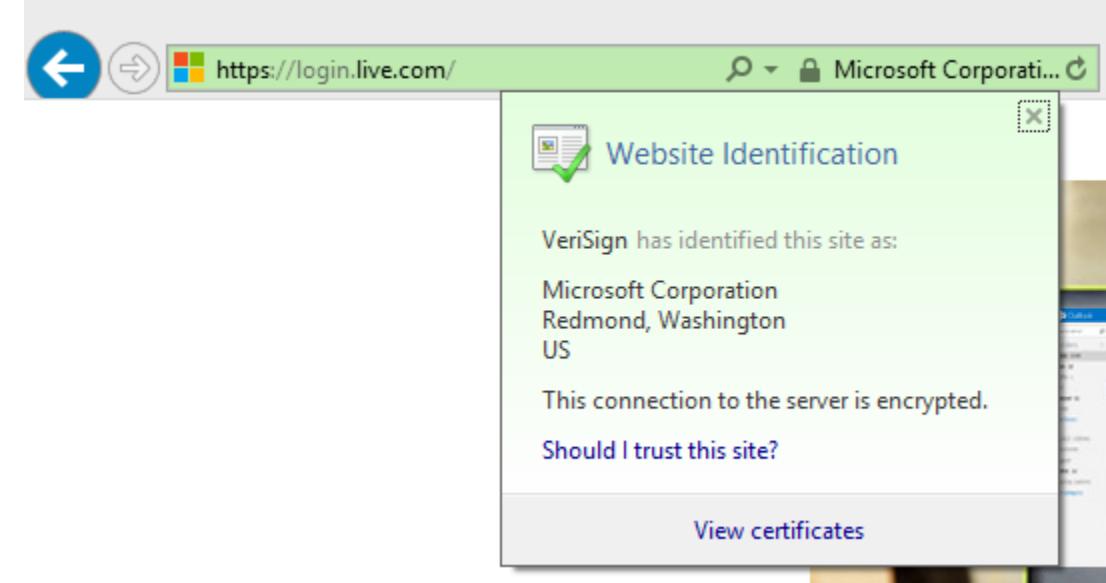


Microsoft®

Research



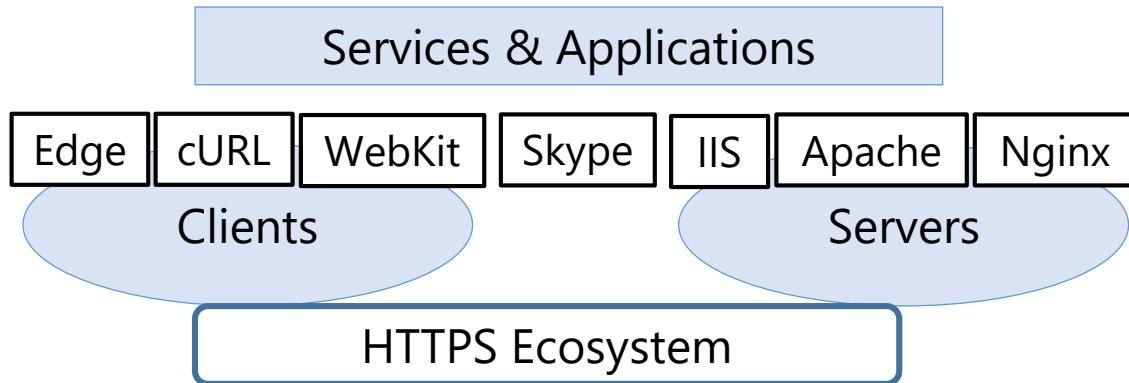
Microsoft Research - Inria
JOINT CENTRE



Everest*: Verified Drop-in Replacements for TLS/HTTPS

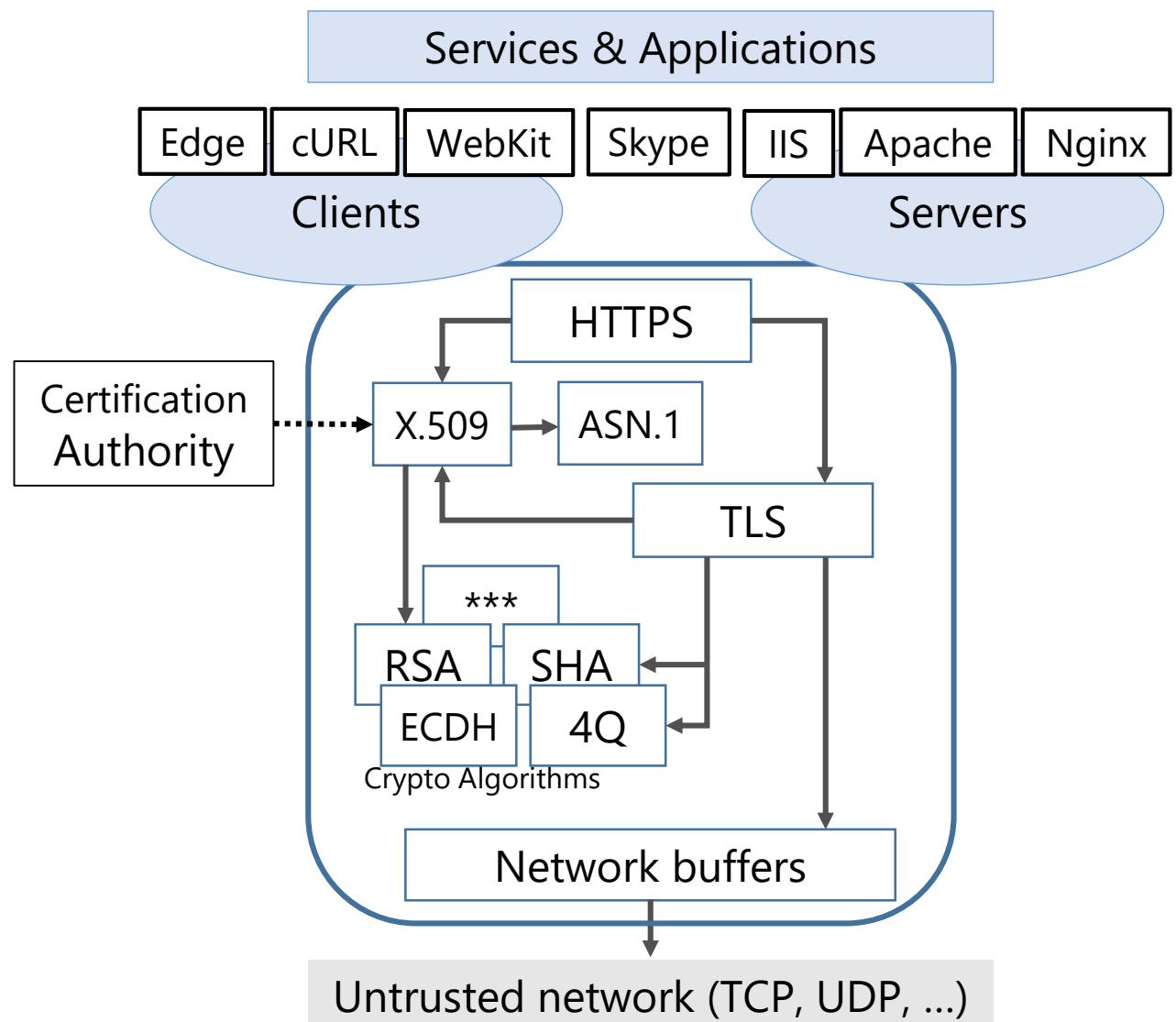


The HTTPS Ecosystem is critical



- Default protocol—trillions of connections
- Most of Internet traffic (+40%/year)
- Web, cloud, email, VoIP, 802.1x, VPNs, IoT...

The HTTPS Ecosystem is complex



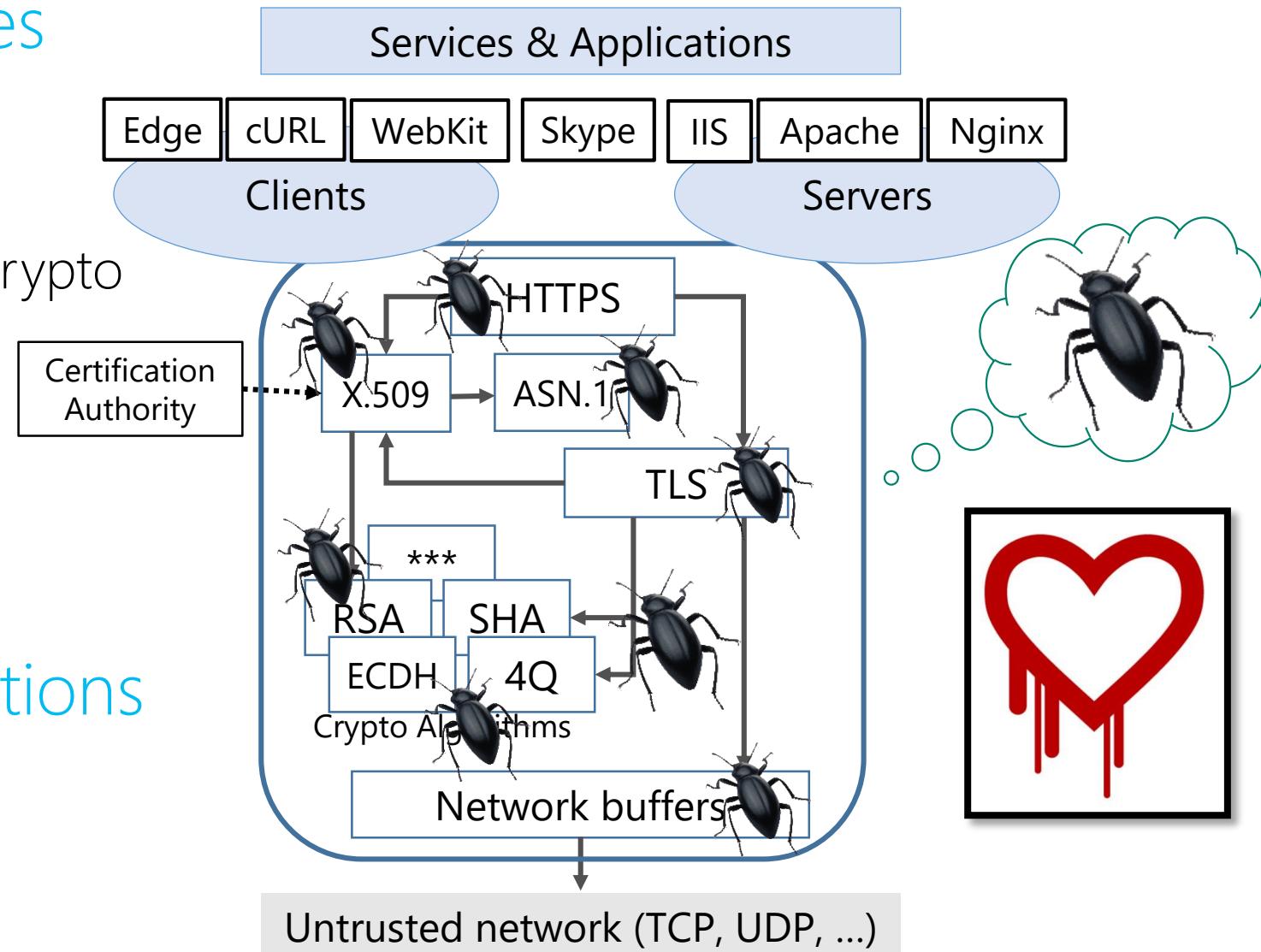
The HTTPS Ecosystem is broken

- 20 years of attacks & fixes

Buffer overflows
Incorrect state machines
Lax certificate parsing
Weak or poorly implemented crypto
Side channels

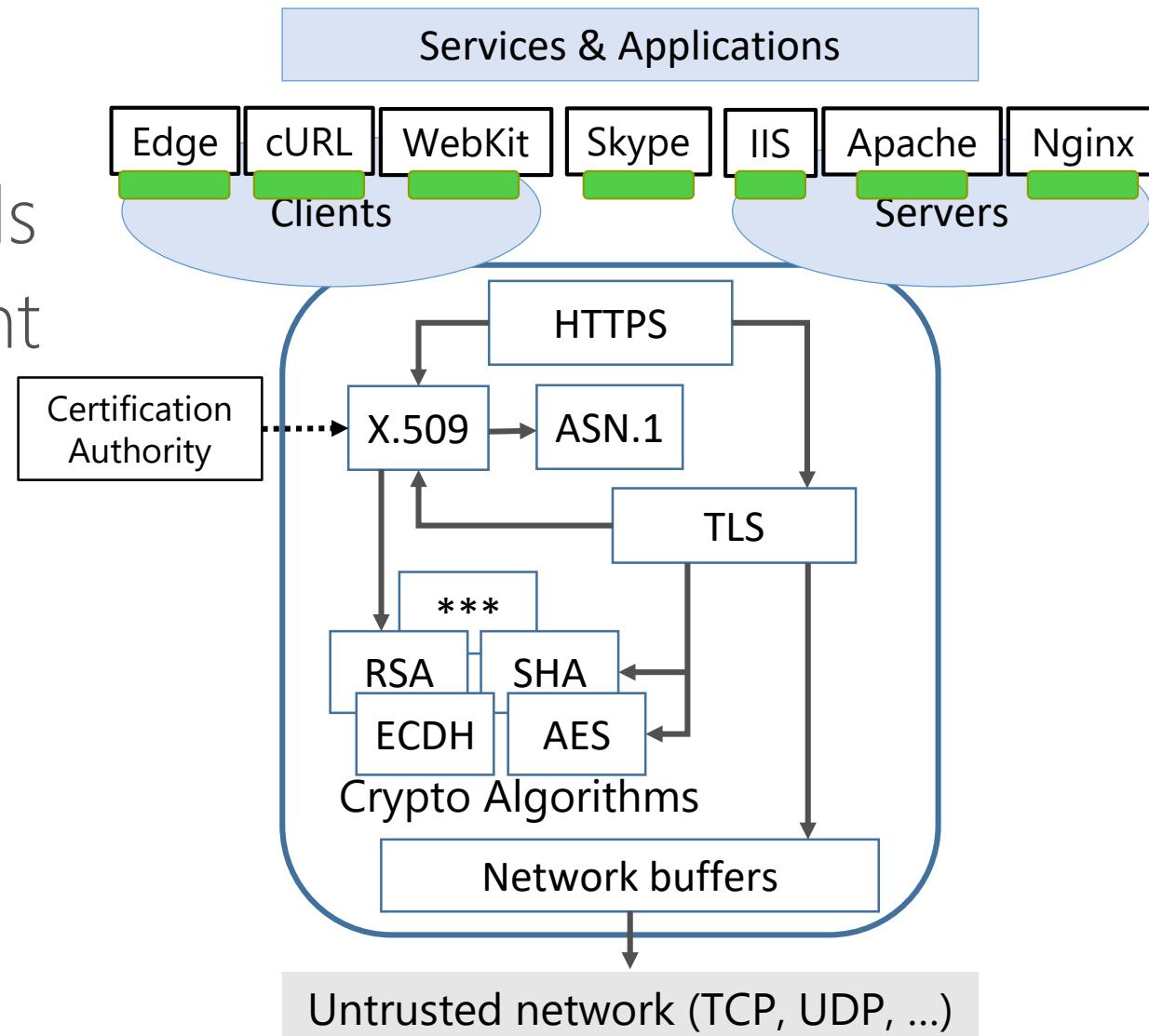
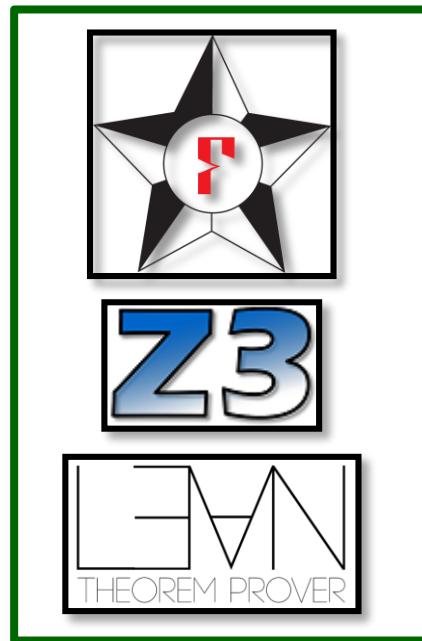
Implicit security goals
Dangerous APIs
Flawed standards

- Mainstream implementations
OpenSSL, SChannel, NSS, ...
Monthly security patches



Verified Components for the HTTPS Ecosystem

- Strong verified safety & security
- Trustworthy, usable tools
- Widespread deployment



Team Everest

and Systems
and Engineering

Security

Programming & Verification

Cryptology



- Cambridge
- Bangalore
- Redmond
- Paris (INRIA)
- Pittsburgh (CMU)

TLS/HTTPS: Just a Secure Channel?

Crypto provable security (core model)

One security property at a time
—simple definitions vs composition

Intuitive informal proofs
Omitting most protocol details

New models & assumptions required 😞

RFCs (informal specs)

Focus on wire format,
flexibility, and interoperability

Security is considered, not specified

Software safety & security (implementation)

Focus on performance, error handling,
operational security

Security vulnerabilities & patches

Application security (interface)

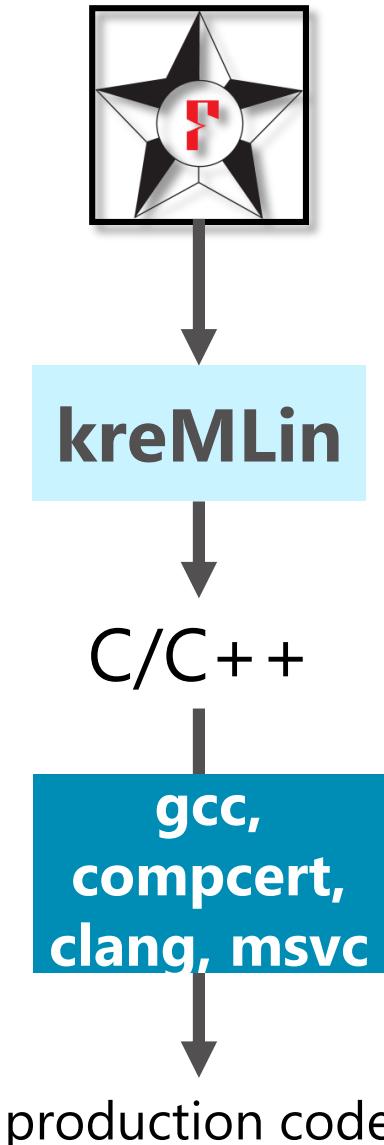
Lower-level, underspecified, implementation-specific.
Poorly understood by most users.

Weak configurations, policies, and deployments

Everest: verified secure usable components for the HTTPS stacks

By implementing
standardized components
and proving them secure,
we validate both their
design and our code.

source code, specs, security definitions,
crypto games & constructions, proofs...

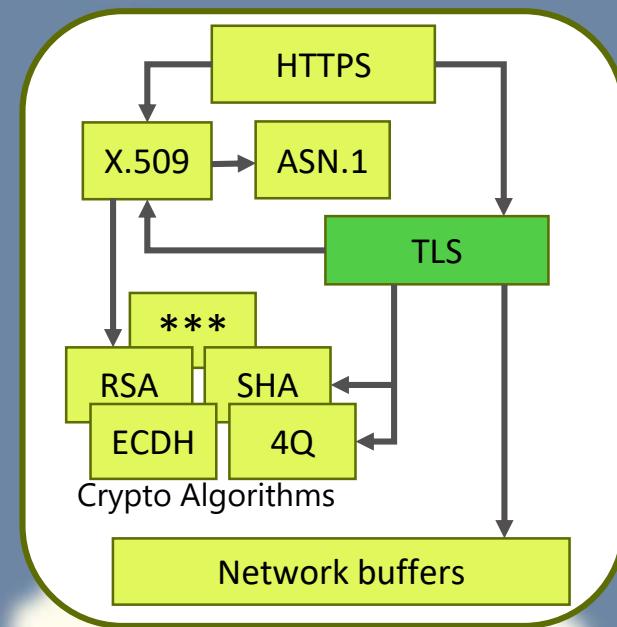


verify all properties
(using automated provers)
then **erase** all proofs

extract low-level code,
with good performance &
(some) side-channel protection

interop with rest of
TLS/HTTPS ecosystem

The TLS/HTTPS ecosystem



TLS Standards & Implementations

Internet Standard

1994	Netscape's Secure Sockets Layer
1995	SSL3
1999	TLS 1.0 (\approx SSL3)
2006	TLS 1.1
2008	TLS 1.2
2017?	TLS 1.3

Implementations:

OpenSSL sChannel NSS SecureTransport PolarSSL JSSE GnuTLS miTLS

Large C++ codebase (400K LOC), many forks <https://github.com/openssl/openssl>

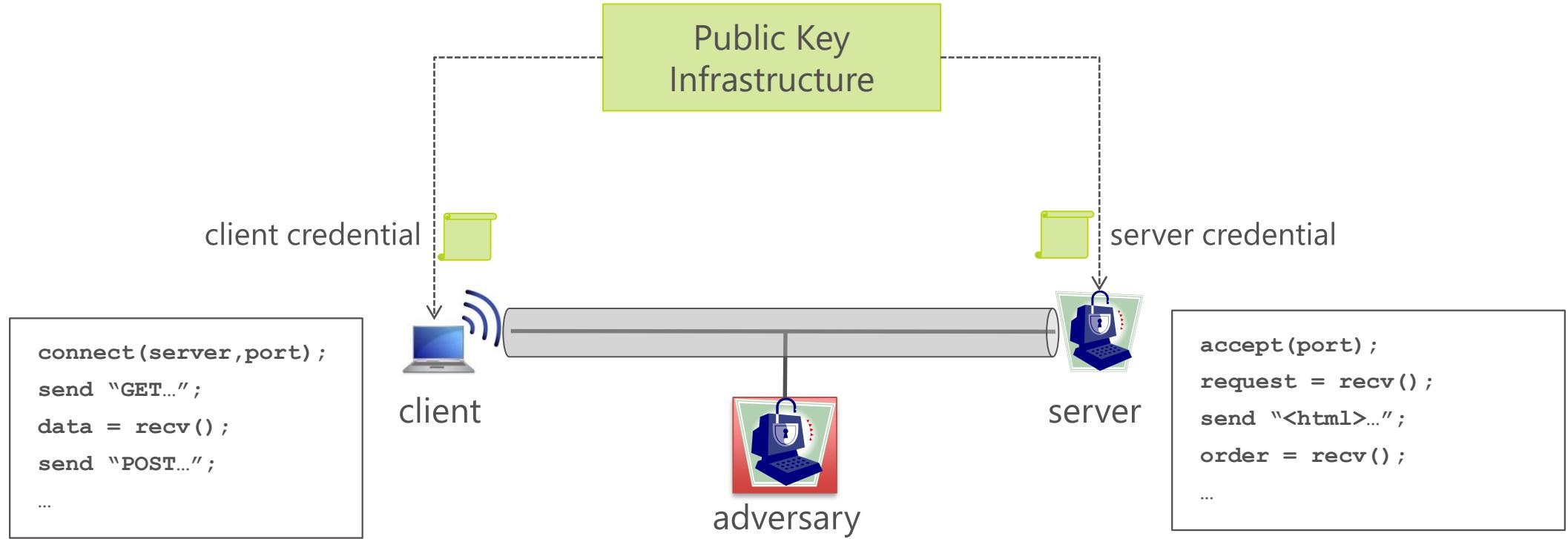
Optimized cryptography for 50 platforms

Terrible API

Frequent critical patches <https://openssl.org/news/vulnerabilities.html>

Never secure so far

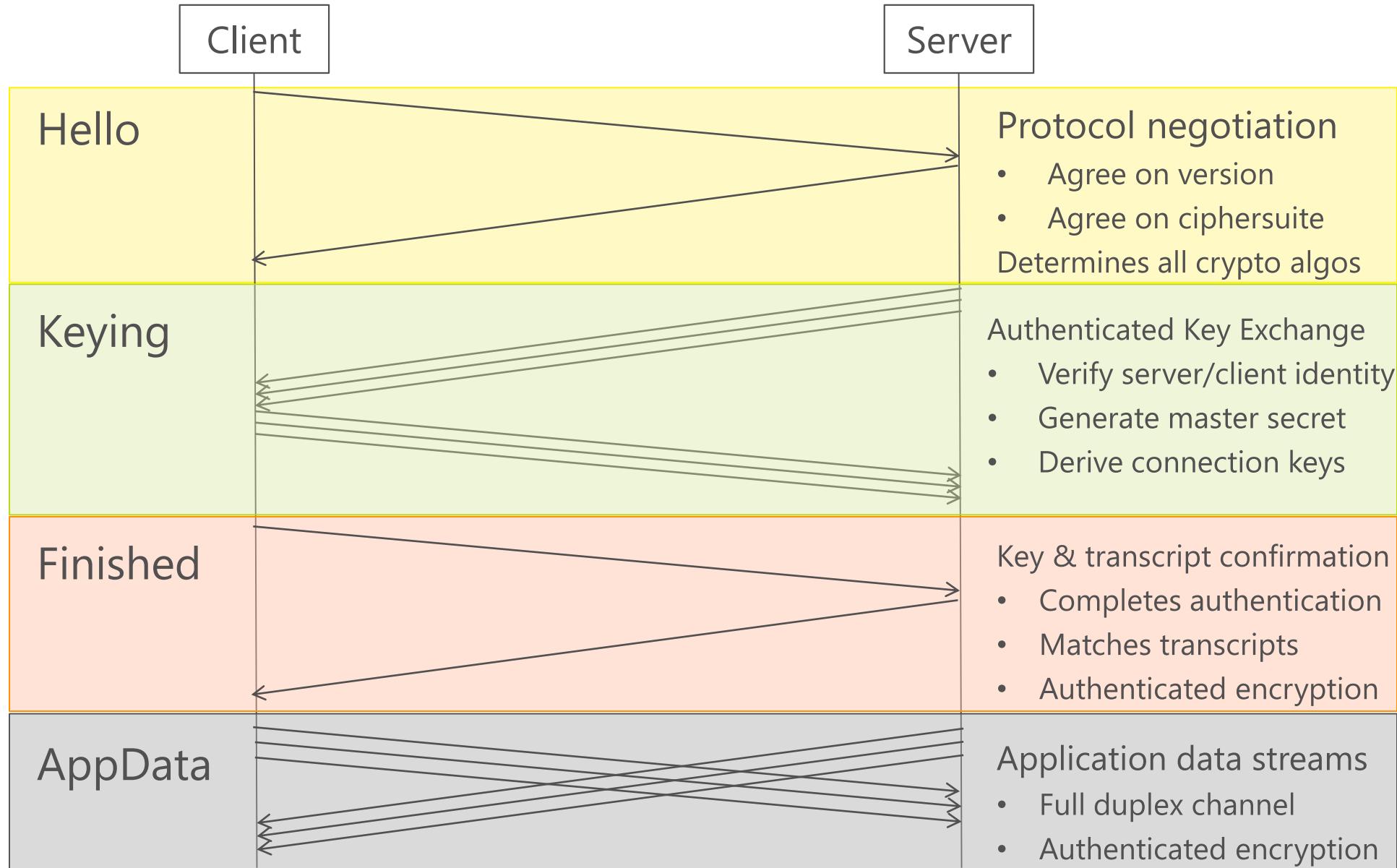
TLS Verification Goal: Secure Channel



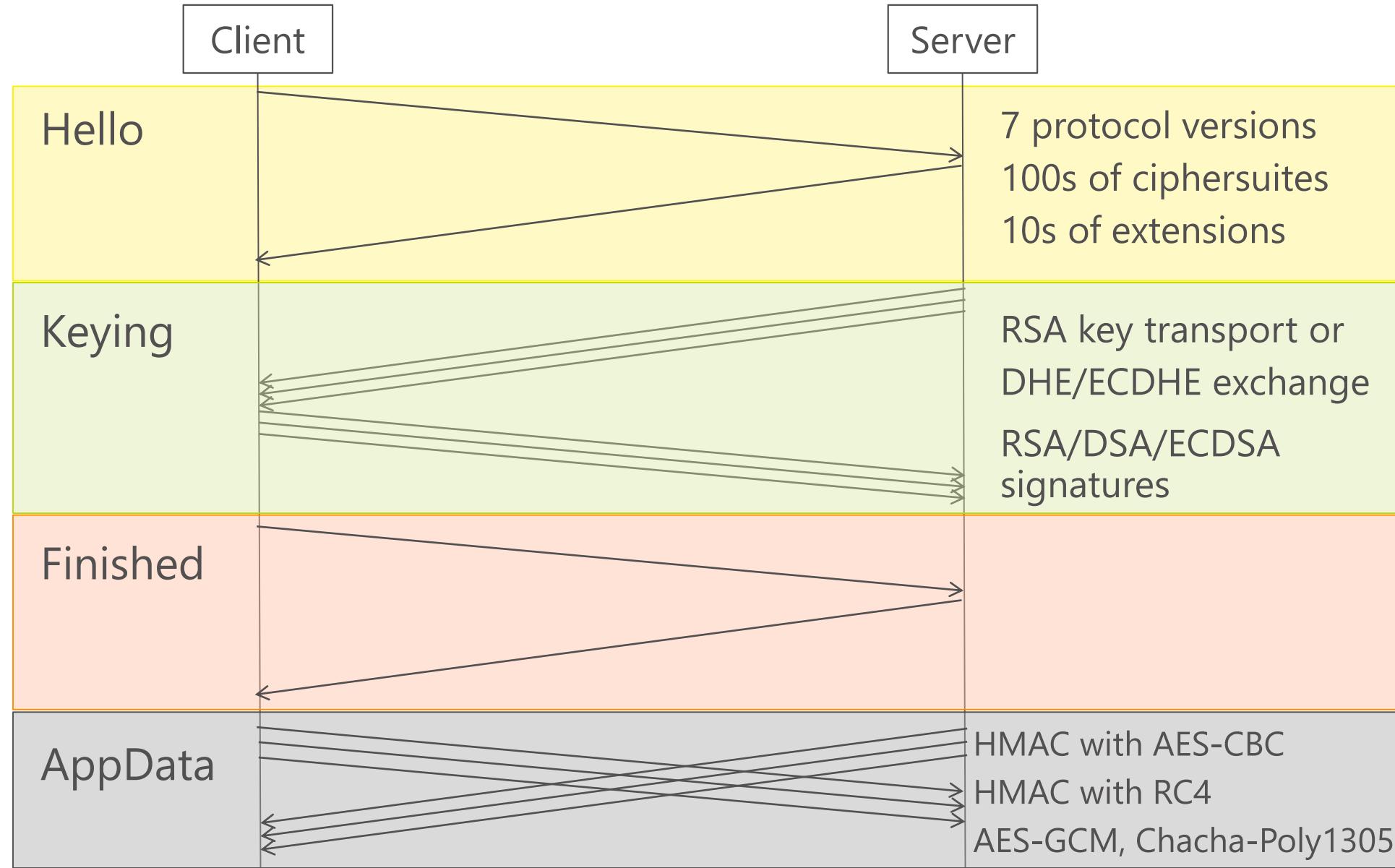
Security Goal: As long as the adversary does not control the long-term credentials of the client and server, it cannot

- Inject forged data into the stream (authenticity)
- Distinguish the data stream from random bytes (confidentiality)

TLS protocol overview



Many configurations (some of them broken)



miTLS (2013—...)

a first verified reference implementation

1. Internet Standard compliance & interoperability

supporting SSL 3.0—TLS 1.2

Excluding crypto
algorithms, X.509, ...

2. Verified security:

we structured our code to enable its
modular cryptographic verification,
from its main API down to concrete
algorithms (RSA, AES,...)

Not fully mechanized
(paper proofs too)

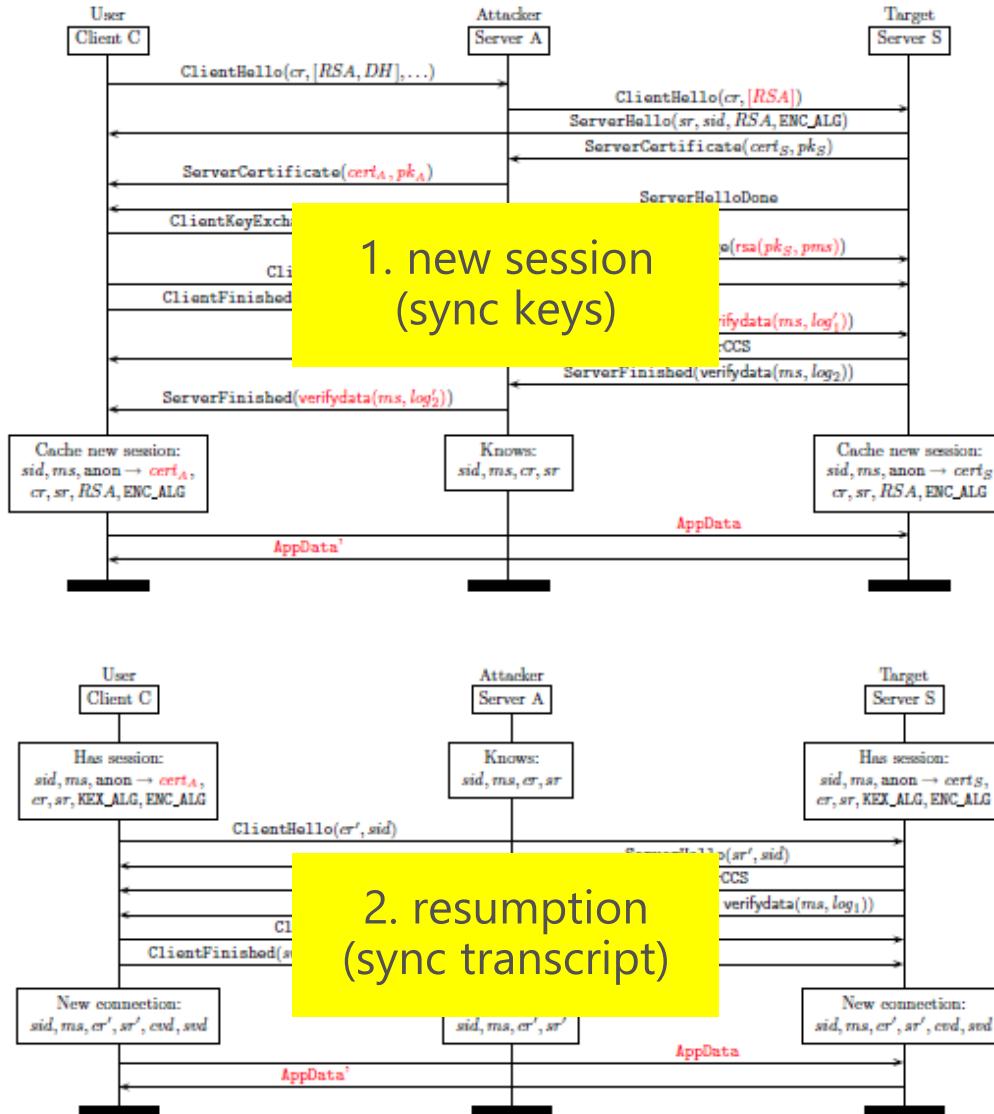
3. Experimental platform:

for testing corner cases, trying out attacks,
analysing extensions and patches, ...

Not production code
(poor performance)

Triple handshake attack (2014)

flaw in the standard
now patched in TLS



<https://www.secure-resumption.com/>

Systematically testing the TLS state machine

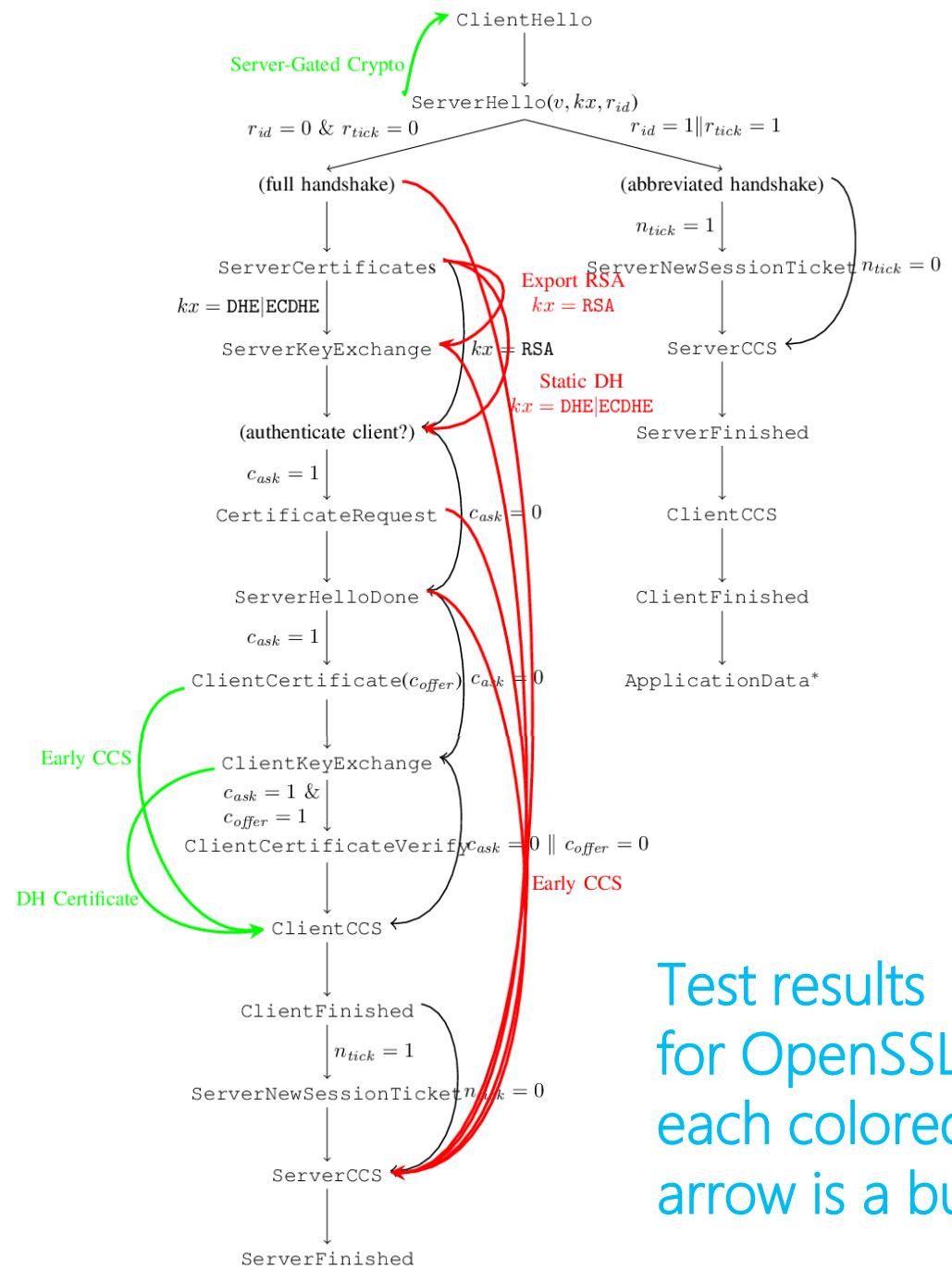
new attacks against all mainstream implementations

TLS offers many ciphersuites, optional messages, extensions... sharing the same state machine.

miTLS provides a verified TLS state machine.

We systematically generated and tested
deviant traces against other implementation
(skipping, inserting, reordering valid messages)

We found many many exploitable bugs



**Test results
for OpenSSL:**
each colored
arrow is a bug

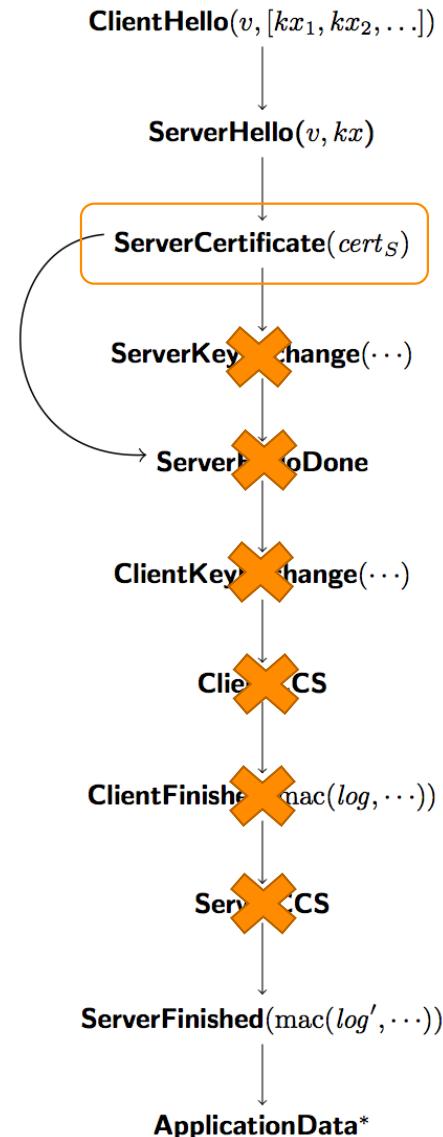
Systematically testing the TLS state machine

new attacks against all mainstream implementations

TLS offers many ciphersuites, optional messages, extensions... sharing the same state machine.

miTLS provides a verified TLS state machine.

We systematically generated and tested
deviant traces against other implementation
(skipping, inserting, reordering valid messages)



An attack
against TLS
Java Library
(open for
10 years)

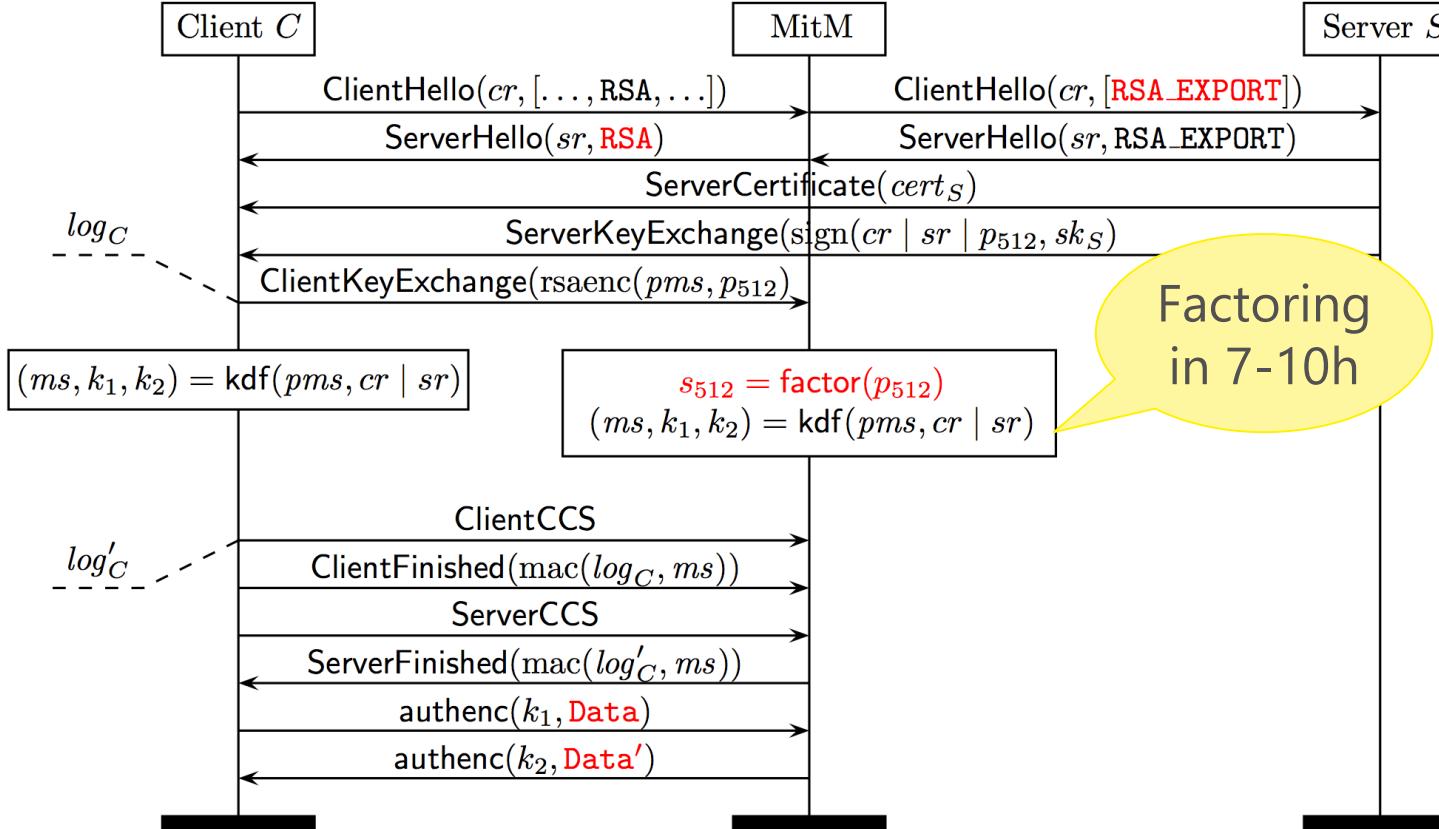
We skip 6 messages

JSSE's client assumes
the key exchange
is finished, uses
uninitialized
0x000000...
as session key!

FREAK: downgrade to RSA_EXPORT (2015)

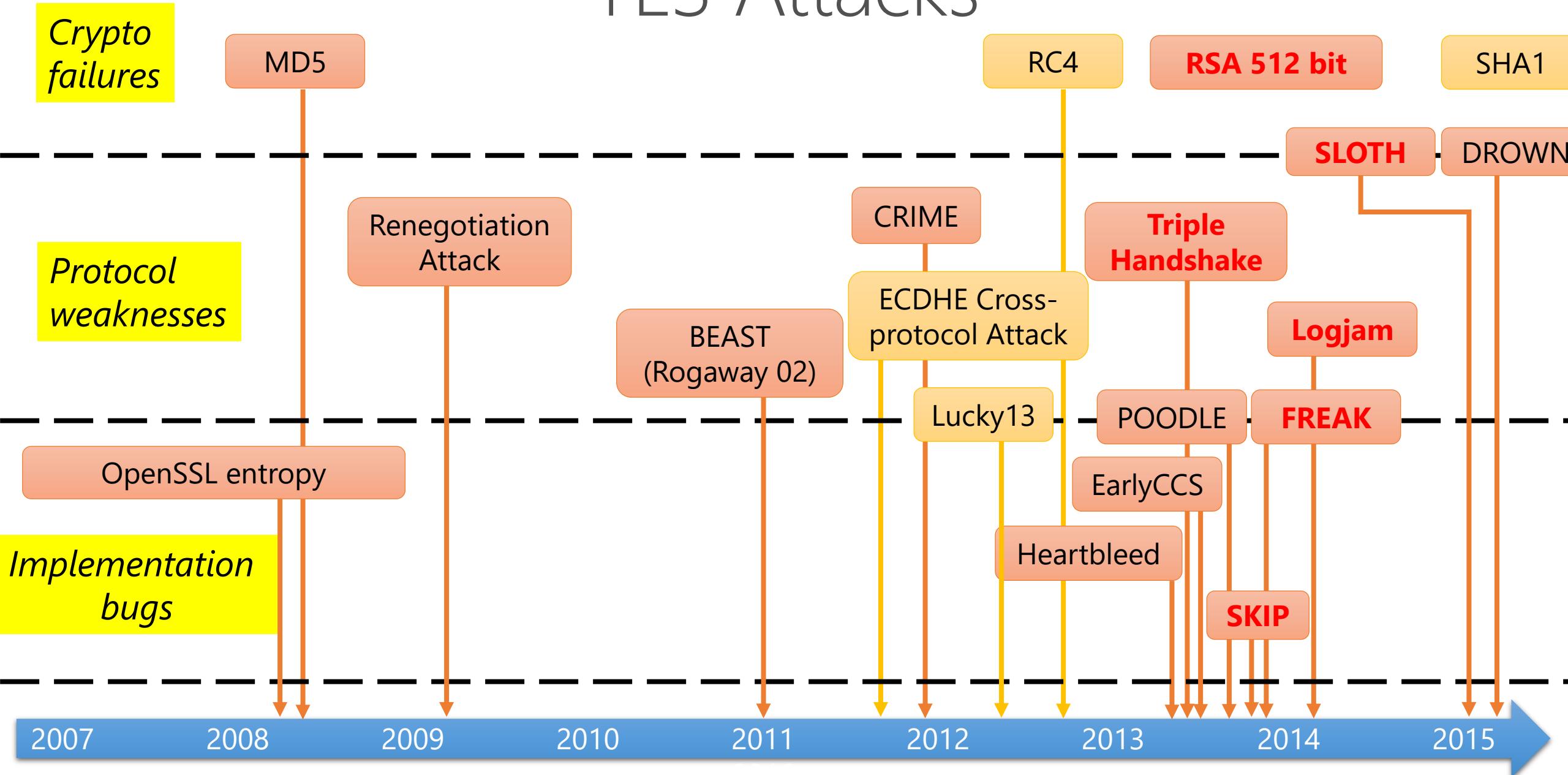
Man-in-the-middle attack against:

- servers that support RSA_EXPORT (512bit keys obsoleted in 2000) from 40% to 8.5%
- clients that accept ServerKeyExchange in RSA (state machine bug) almost all browsers have been patched



Similar attack,
different crypto:
LOGJAM (2015)
downgrade to
weak groups

TLS Attacks



TLS 1.3: a new hope

Much discussions

IETF, Google, Mozilla, Microsoft, CDNs,
cryptographers, network engineers, ...

Much improvements

- Modern design
 - Fewer roundtrips
 - Stronger security

New implementations required for all

- Be first & verified too!
 - Find & fix flaws before it's too late

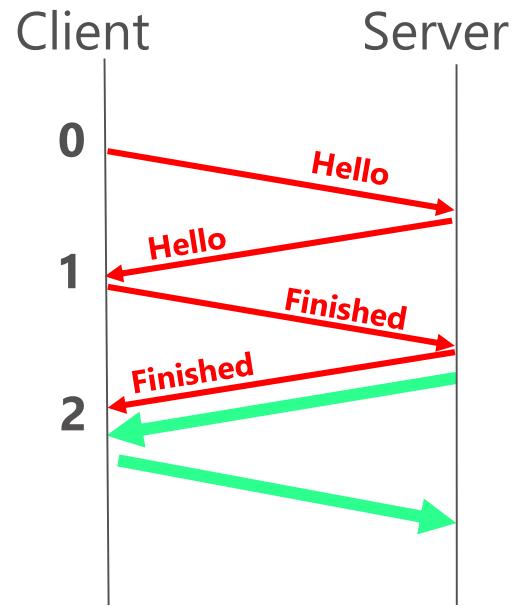
Network Working Group
Internet-Draft
Obsoletes: 5077, 5246, 5746 (if approved)
Updates: 4492 (if approved)
Intended status: Standards Track
Expires: September 23, 2016

E. Rescorla
RTFM, Inc.
March 22, 2016

Table of Contents

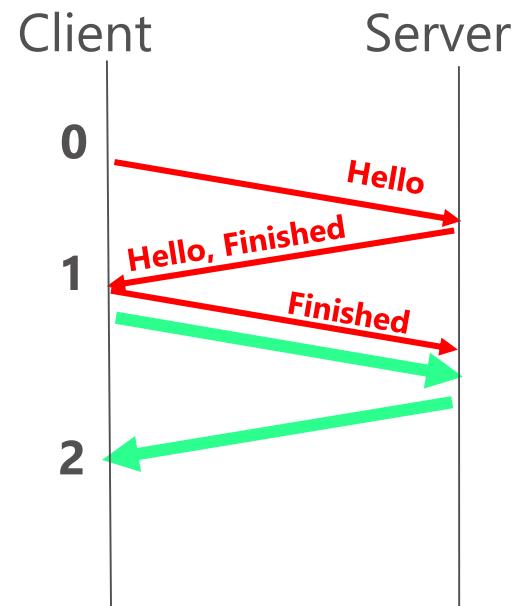
1. Introduction
 - 1.1. Conventions and Terminology
 - 1.2. Major Differences from TLS 1.2
 2. Goals
 3. Goals of This Document
 4. Presentation Language
 - 4.1. Basic Block Size
 - 4.2. Miscellaneous
 - 4.3. Vectors
 - 4.4. Numbers
 - 4.5. Enumerateds
 - 4.6. Constructed Types
 - 4.6.1. Variants
 - 4.7. Constants
 - 4.8. Cryptographic Attributes
 - 4.8.1. Digital Signing
 - 4.8.2. Authenticated Encryption with Additional Data (AEAD)
 5. The TLS Record Protocol
 - 5.1. Connection States

Saving roundtrips for new connections



TLS 1.2

Two roundtrips
before sending
application data



TLS 1.3

One roundtrip
before sending
application data



TLS 1.3

Zero roundtrip
before sending
application data

Client has no
guarantee
the server is
present or unique.

Server has no
guarantee the
client agrees on
the connection

Trading
performance
for security

TLS 1.3: status

IETF WG9⁵⁹⁹

1321st draft including
some of our proposals

#4

log-based key separation
extended session hashes
(fixing attacks we found on 1.2)

#11

stream terminators
(eventually fixing an attack)

#14

downgrade resilience

#15

session ticket format

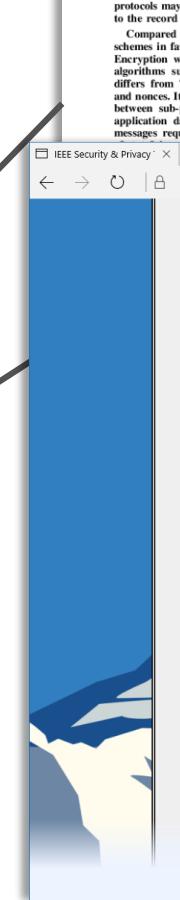
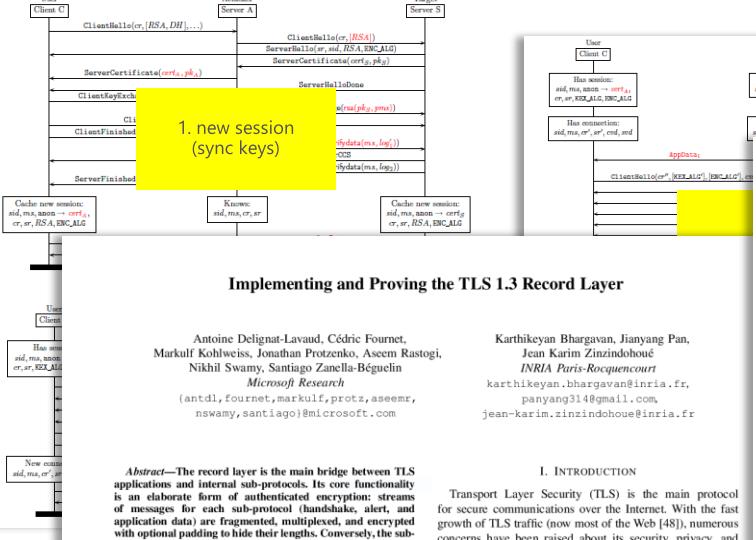
#17

simplified key schedule

#18

pre-shared-key 0RTT
PSK binding (fixing an attack)

RFC finalized this month?



Downgrade Resilience in Key-Exchange Protocols

Karthikeyan Bhargavan^a, Christina Brzuska^b, Cédric Fournet^b, Matthew Green^b, Markulf Kohlweiss^c and Santiago Zanella-Béguelin^d

^aInria Paris-Rocquencourt, Email: karthikeyan.bhargavan@inria.fr

^bHamburg University of Technology, Email: brzuska@uhh.de

^cMicrosoft Research, Email: {fournet, markulf, santiago}@microsoft.com

^dJohns Hopkins University, Email: mgreen@cs.jhu.edu

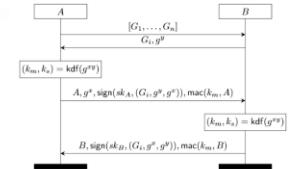
Abstract—Key-exchange protocols such as TLS, SSH, IPsec, and ZRTP are highly configurable, with typical deployments supporting multiple cipher-suites, cryptographic primitives, and key-exchange parameters. In the first message of the handshake, the peers negotiate one specific combination: the *protocol mode*, based on their local configurations. With few notable exceptions, most cryptographic analyses of configurable protocols consider a single mode at a time. In contrast, downgrade attacks, where a network adversary can peer into the communication channel and change the mode normally negotiate, are a recurring problem in practice.

How to support configurability while at the same time guaranteeing the preferred mode is negotiated? We set to answer this question by designing a formal framework to study downgrade resilience and its relation to other security properties of key-exchange protocols. First, we study the causes of downgrade attacks by dissecting and classifying known and novel attacks against widely used protocols. Second, we survey what is known about the downgrade resilience of existing standards. Third, we combine these results to define the security properties, and analyze the constraints under which standard protocols act. Finally, we discuss patterns that guarantees downgrade security by design, and explain how to use them to strengthen the security of existing protocols, including a newly proposed draft of TLS 1.3.

I. INTRODUCTION

Transport Layer Security (TLS) is the main protocol for secure communications over the Internet. With the fast growth of TLS traffic (now most of the Web [48]), numerous concerns have been raised about its security, privacy, and performance. These concerns are justified by a history of attacks against deployed versions of TLS, often originating in the record layer.

Compared to prior versions, TLS 1.3 discards obsolete schemes in favor of a common construction for Authenticated Encryption with Associated Data (AEAD), instantiated with algorithms such as AES-GCM and ChaCha20-Poly1305. It differs significantly in its use of additional handshake data and messages. It also encodes the content-type used in multiples between sub-protocols. New protocol features such as early application data (0-RTT and 0.5-RTT) and late handshake messages require additional keys and a more general model



than the one they would have used on their own. Such attacks have been identified in a number of protocols, most famously in the early versions of the SSL protocol [43] and even in recent versions of TLS [2, 39].

Surprisingly, there has been relatively little formal work around the security of negotiation in modern cryptographic protocols. Several recent works formally prove the security of different aspects of TLS and SSH. Some [25, 31] only model a single mode at a time. Some [12, 13] do model negotiation of weak algorithms, but do not guarantee negotiation of the preferred mode. Some others [9, 21] consider only interactions

TLS 1.3: Design, Implementation & Verification Workshop

30 April 2017, University Pierre and Marie Curie (UPMC), Paris, France

Affiliated with IEEE Euro Security & Privacy and Eurocrypt

Goals Topics Call for Speakers Agenda Contact

Aims and Goals

The goals of the TLS:DIV workshop are threefold: first, to explain and justify the latest changes to the TLS 1.3 design (from draft 13 to draft 19); second, to give an overview of some ongoing efforts to prove the cryptographic security of the TLS 1.3 protocol, and third, to showcase recent tools and methods to evaluate and improve the safety and security of TLS implementations, up to the level of cryptographic primitives.

The workshop is organized by the [Everest project team](#) and consists of invited talks from leading experts on key exchange security and implementation of cryptography on topics related to the analysis and implementation of TLS.

Workshop topics

- Evolution of the TLS 1.3 specification
- Cryptographic security proofs of the TLS 1.3 handshake and record
- Safe and secure implementations of cryptographic primitives
- Security evaluation of TLS implementations and deployment
- Applications built on top of new TLS 1.3 features (e.g. 0-RTT, late authentication)



Cryptographic Algorithms for HTTPS

Algorithms get broken & replaced over time

Security relies on probabilistic cryptographic assumptions (who knows?)

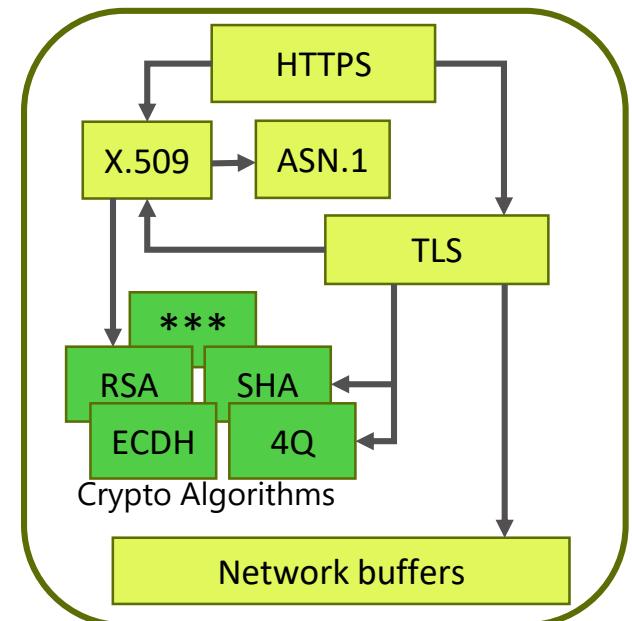
Modern design & implementations select between various algorithms & implementations for the same core functionality

~30 standard algorithms

- Hash and key-derivation functions (SHA256)
- Symmetric cryptography (AES_GCM, AES_CBC)
- Public-key encryption and signing
- Elliptic curves (NIST, 25519, 4Q)

High-performance

AES_GCM takes 0.46 cycle/byte on Intel Skylake
Hand-tuned, low-level, architecture-specific



Testing for known bugs in 3rd-party code



Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Project Wycheproof

December 19, 2016

Posted by Daniel Bleichenbacher, Security Engineer and Thai Duong, Security Engineer

We're excited to announce the release of [Project Wycheproof](#), a set of security tests that check cryptographic software libraries for known weaknesses. We've developed over 80 test cases which have uncovered more than [40 security bugs](#) (some tests or bugs are not open sourced today, as they are being fixed by vendors). For example, we found that we could recover the private key of widely-used [DSA](#) and [ECDHC](#) implementations. We also provide ready-to-use tools to check [Java Cryptography Architecture](#) providers such as [Bouncy Castle](#) and the default providers in [OpenJDK](#).

The main motivation for the project is to have an achievable goal. That's why we've named it after the Mount Wycheproof, the [smallest mountain in the world](#). The smaller the mountain the easier it is to climb it!

Application Security: https://

Example: tracing
https://www.visualstudio.com/

- Trust is transitive
 - each page involves connections to many servers (different origins)
- Trust is implicit
 - 17 concurrent TLS connections, configurations, certificate chains
- Trust is a matter of state
 - cookies, caches, configurations, proxies

The diagram illustrates the HTTPS handshake process. It starts with a green box labeled "HTTPS" at the top, which connects to a yellow box labeled "X.509". An arrow points from "X.509" to "ASN.1", which then connects to a yellow box labeled "TLS". The "TLS" box has two downward arrows: one to a stack of four boxes labeled "RSA", "ECDH", "SHA", and "4Q" (with three asterisks between RSA and ECDH), and another to a green box labeled "Network buffers" at the bottom. The screenshot of the Microsoft Edge F12 Developer Tools Network tab shows a list of 83 requests for various resources on www.visualstudio.com, including files like wt.js, Combined.css, and sizzle.js. The "Content type" column shows various file types like text/html, application/javascript, and text/css. The "Received" column shows file sizes like 17.45 KB and 10.68 KB. The "Time" column shows response times like 655.72 ms and 48.46 ms. The "Initiator / Type" column indicates the source of each request, such as "document", "script", or "link". On the right side of the screenshot, there are tabs for Headers, Body, Parameters, Cookies, and Timings, along with detailed status information for the current request.

Unsolved issues with HTTPS

SSL Stripping (Marlinspike)	Cookie-based Attacks (various variants)	CRIME / BREACH (Rizzo, Duong et al.)	Virtual Host Confusion (Delignat-Lavaud)
TLS is optional in HTTP and can be disabled by an active attacker	Shared cookie database for HTTP and HTTPS can be used to mount various session fixation and login CSRF attacks.	Attackers can easily mount adaptive chosen-plaintext attacks. Encryption after compression can leak secrets through length.	HTTPS servers do not correlate transport-layer and HTTP identities, leading to origin confusion
Mitigated by correct use of HTTP Strict Transport Security (HSTS)	Mitigated by new binding proposals (ChannelID, Token Binding). Mitigation is not widely implemented.	Mitigated by refreshing secrets (e.g. CSRF tokens). Some protocol-specific mitigations (QUICK, HTTP2)	Mitigated by configuration of HTTPS servers with strict host rules
Mitigation not widely used, and vulnerability is still widespread in practice.	Difficult to mitigate in browsers with current technologies. Can be used to attack many websites.	Ad-hoc mitigation; attack is still widespread in practice as HTTP compression remains popular.	Ad-hoc mitigation. Attack still widespread in practice.

The timeline diagram illustrates the progression of SSL stripping and other attacks over time. It features a horizontal arrow at the bottom representing the timeline from 2006 to 2014. Four green arrows point downwards from the table rows to specific years on the timeline: 2007, 2008, 2011, and 2013. The row for 'SSL Stripping' points to 2007. The row for 'Cookie-based Attacks' points to 2008. The row for 'CRIME / BREACH' points to 2011. The row for 'Virtual Host Confusion' points to 2013.

2006 2007 2008 2009 2010 2011 2012 2013 2014

Long-term identities: X.509

Public-Key Infrastructure (Certificate Chains)

Designed in 1984; widely criticized but hard to replace
HTTPS is just one application

Same complexity as TLS?

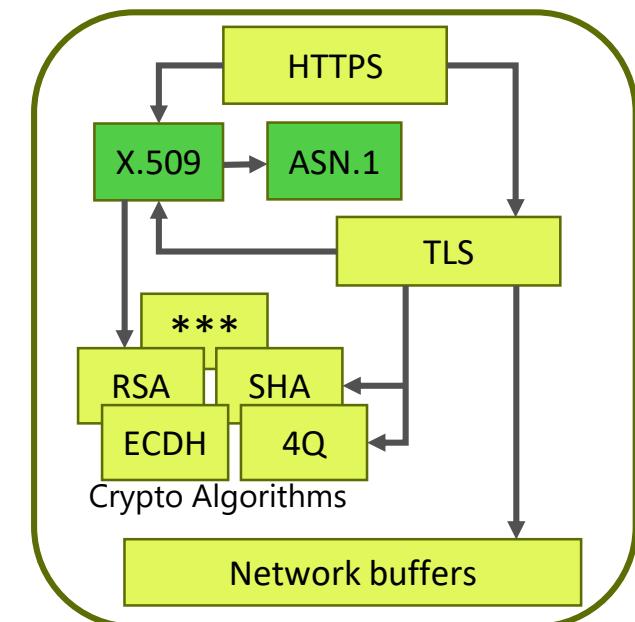
ASN.1 grammar; many extensions and interpretations
50% of "TLS attacks" are in fact X.509 attacks

Recent initiatives

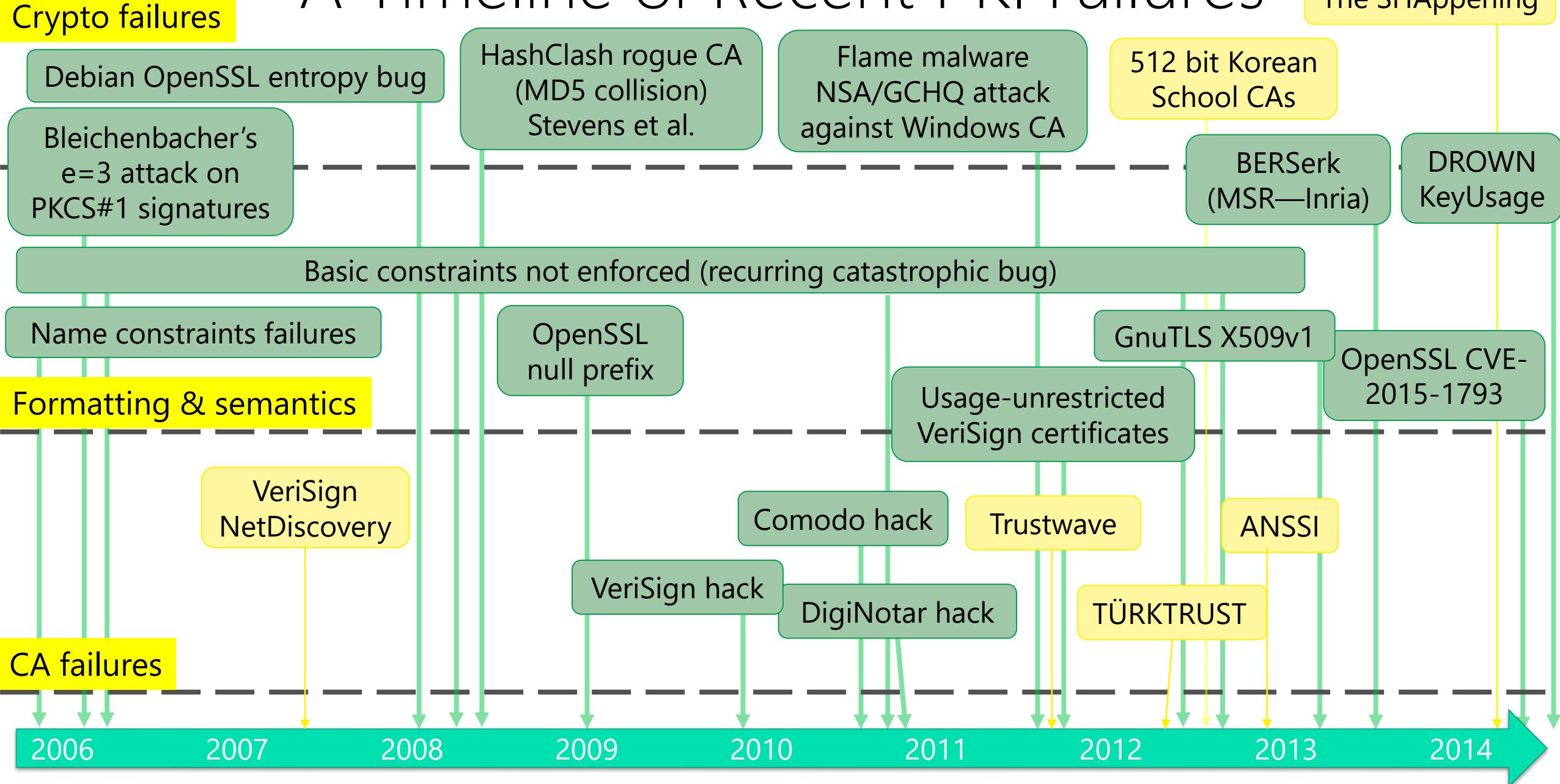
Global scans for millions of certificates
Certificate pinning & transparency
Let's encrypt! <https://letsencrypt.org/>

Verification?

Complex ambiguous format
Certificate issuance and revocation policies

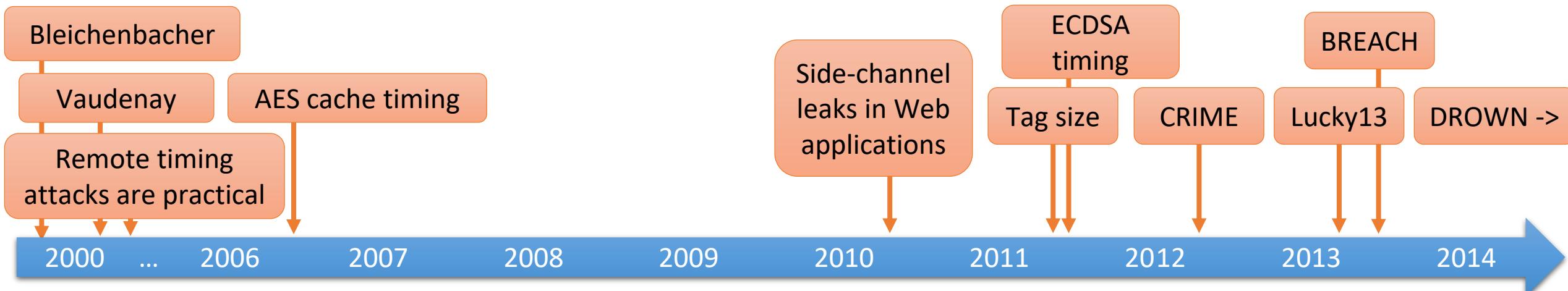


A Timeline of Recent PKI Failures



Side Channel Challenge (Attacks)

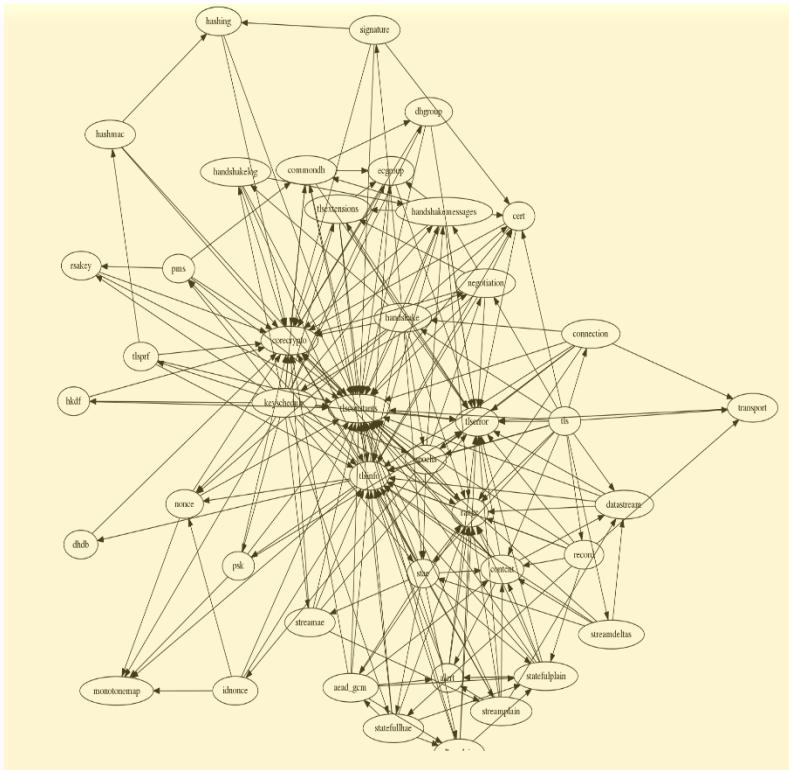
Protocol-level side channels	Traffic analysis	Timing attacks against cryptographic primitives	Memory & Cache
TLS messages may reveal information about the internal protocol state or the application data	Combined analysis of the time and length distributions of packets leaks information about the application	A remote attacker may learn information about crypto secrets by timing execution time for various inputs	Memory access patterns may expose secrets, in particular because caching may expose sensitive data (e.g. by timing)
<ul style="list-style-type: none"> • Hello message contents (e.g. time in nonces, SNI) • Alerts (e.g. decryption vs. padding alerts) • Record headers 	<ul style="list-style-type: none"> • CRIME/BREACH (adaptive chosen plaintext attack) • User tracking • Auto-complete input theft 	<ul style="list-style-type: none"> • Bleichenbacher attacks against PKCS#1 decryption and signatures • Timing attacks against RC4 (Lucky 13) 	<ul style="list-style-type: none"> • OpenSSL key recovery in virtual machines • Cache timing attacks against AES



Demo

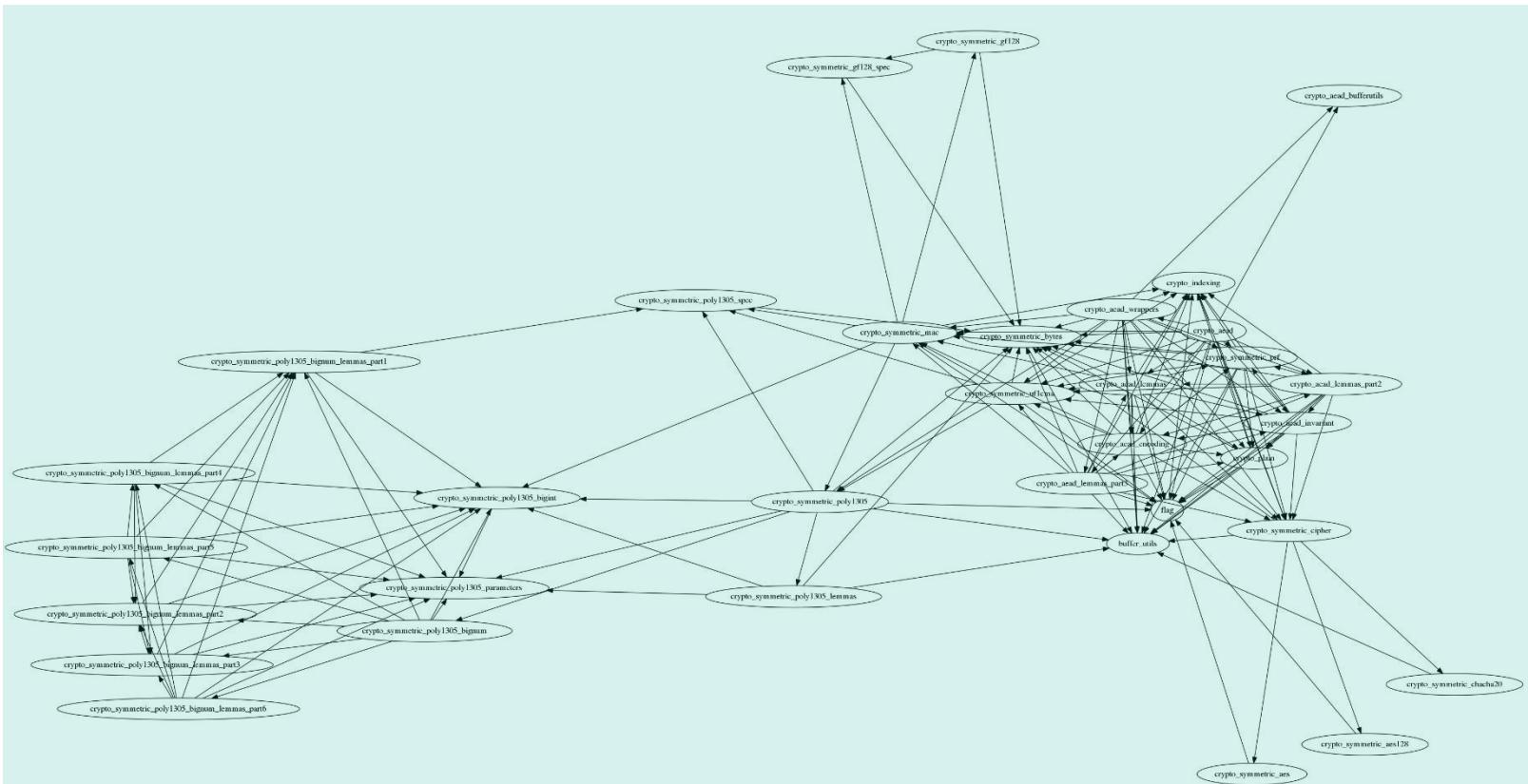


miTLS in F* today

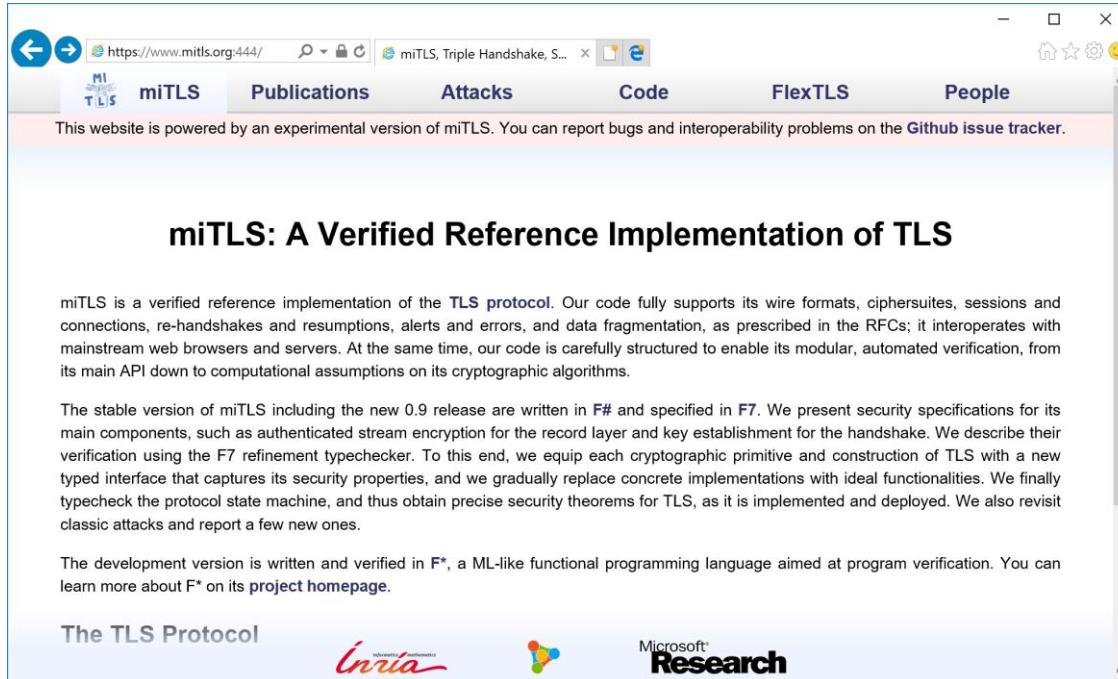


AEAD record-layer crypto 14K lines of code and proofs Verified & compiled to C

**miTLS, protocol layer:
16K lines of code and proofs
Compiled to Ocaml.
Partially verified.**



Client: IE

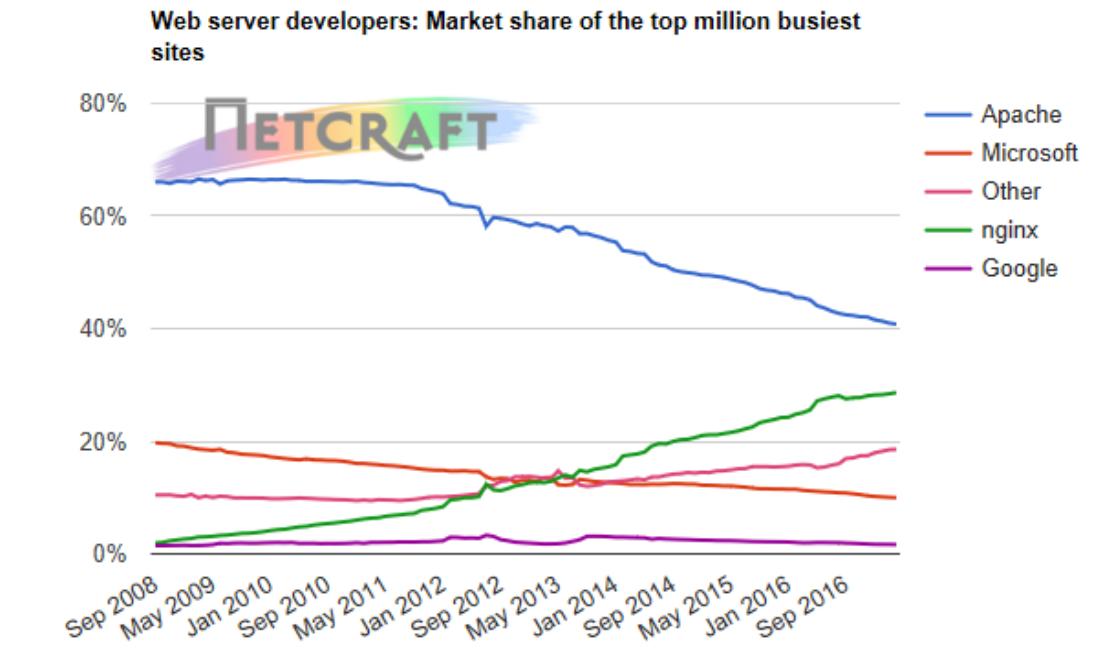


A screenshot of a Microsoft Internet Explorer browser window displaying the miTLS website. The address bar shows the URL <https://www.mitls.org:444/>. The page title is "miTLS: A Verified Reference Implementation of TLS". The content discusses miTLS as a verified reference implementation of the TLS protocol, written in F# and specified in F7. It mentions the F7 refinement typechecker and the verification of main components like authenticated stream encryption and key establishment. The development version is written in F*, a ML-like functional programming language for program verification. Logos for Inria and Microsoft Research are visible at the bottom.

We integrate miTLS & its verified crypto with Internet Explorer.

We run TLS 1.3 sessions with 0RTT without changing their application code.

Server: nginx



A high performance server for HTTP, reverse proxy, mail,...

We replace OpenSSL with miTLS & its crypto: the modified server supports TLS 1.3 with tickets and 0-RTT requests.

Nginx Architecture

