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

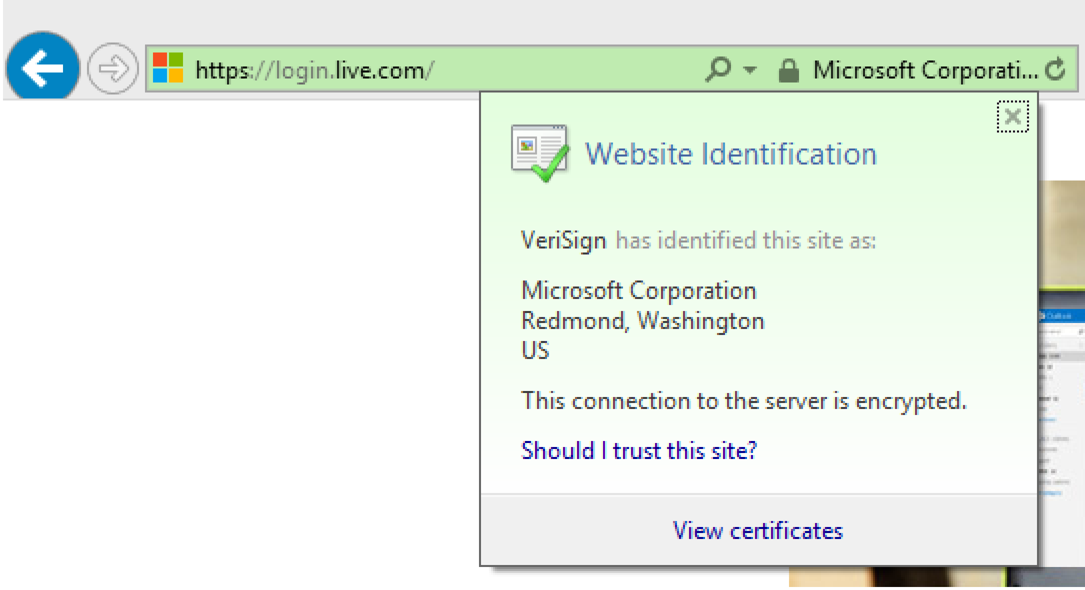
Security Verification with F*

Cédric Fournet
Catalin Hritcu
Aseem Rastogi

Microsoft Research

Inria INVENTEURS DU MONDE NUMÉRIQUE

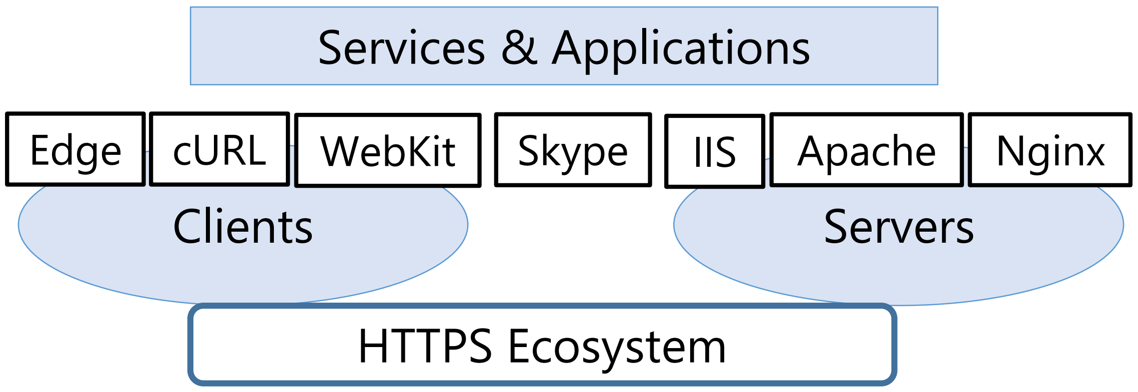
Microsoft Research - Inria JOINT CENTRE



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

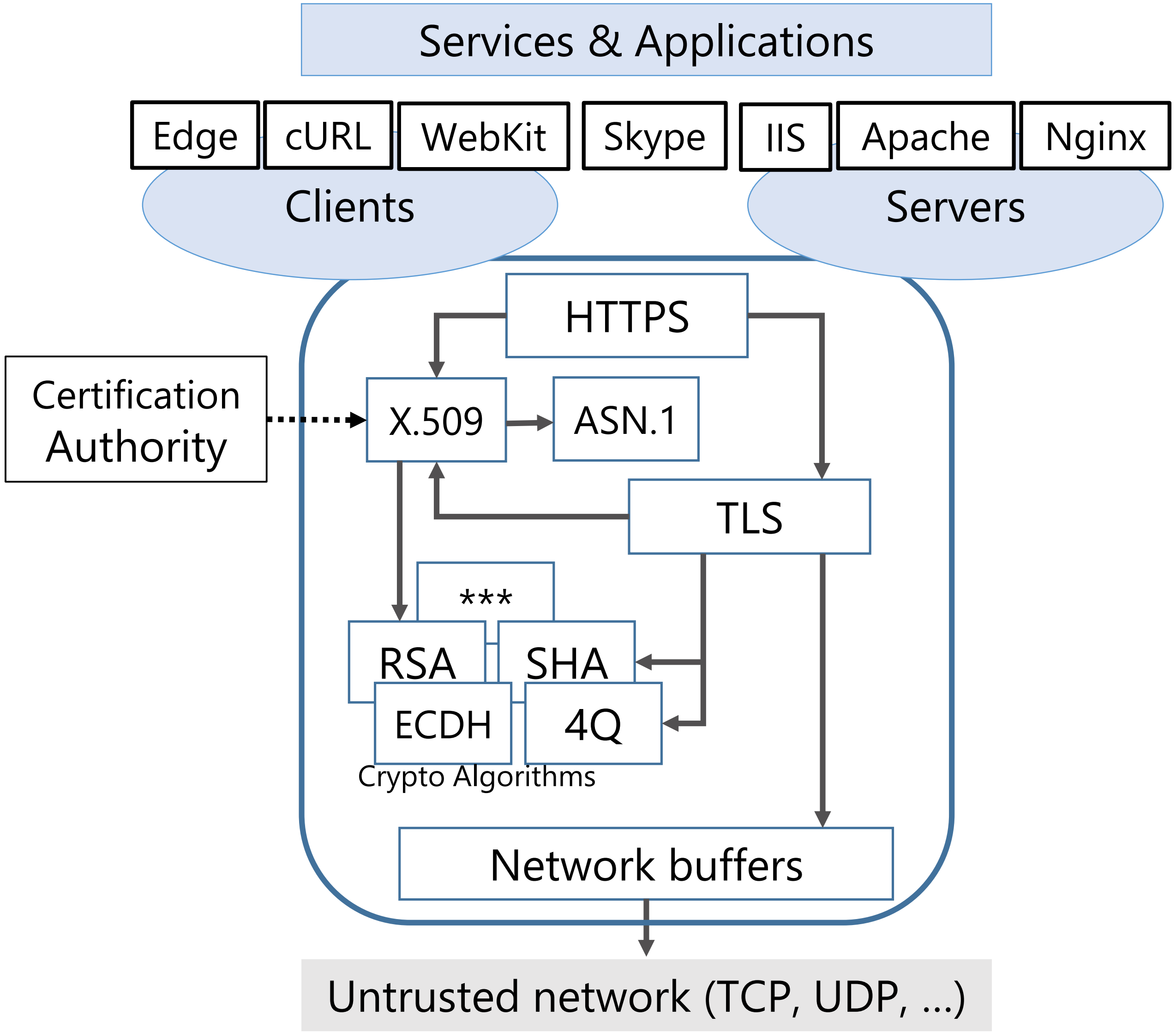
*the Everest VERified End-to-end Secure Transport

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

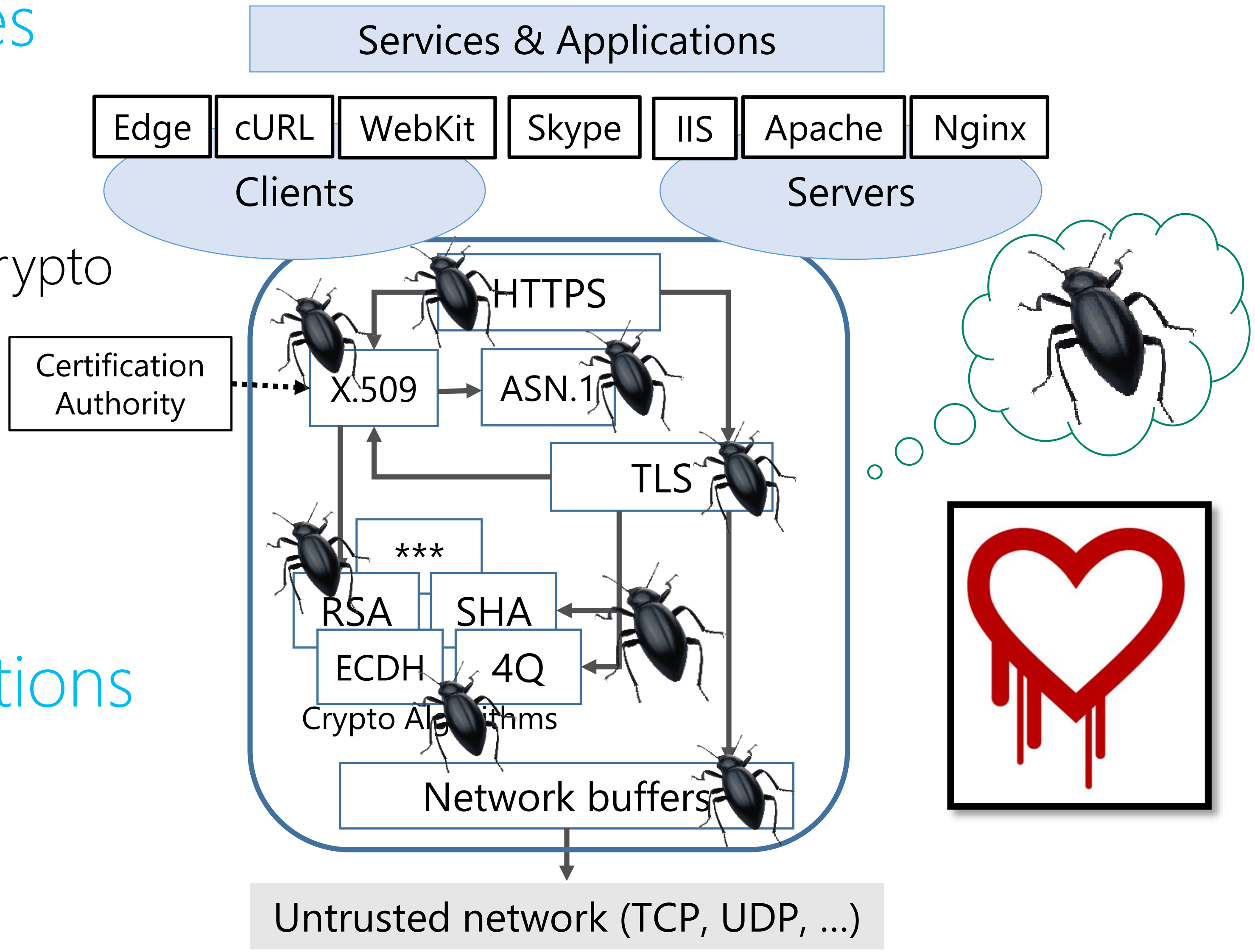
- 20 years of attacks & fixes

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

  Implicit security goals
  Dangerous APIs
  Flawed standards

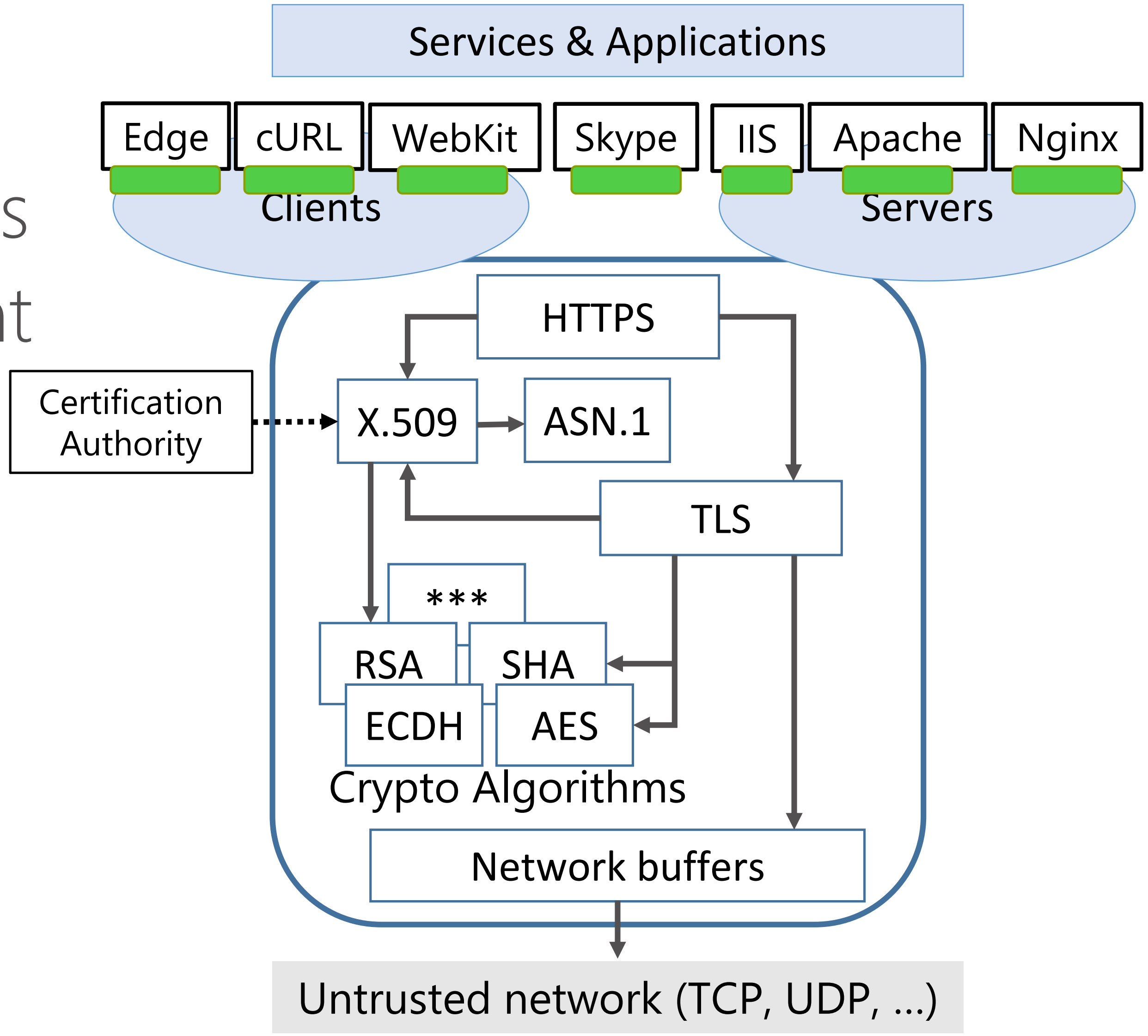
- Mainstream implementations

  OpenSSL, SChannel, NSS, …
  Monthly security patches

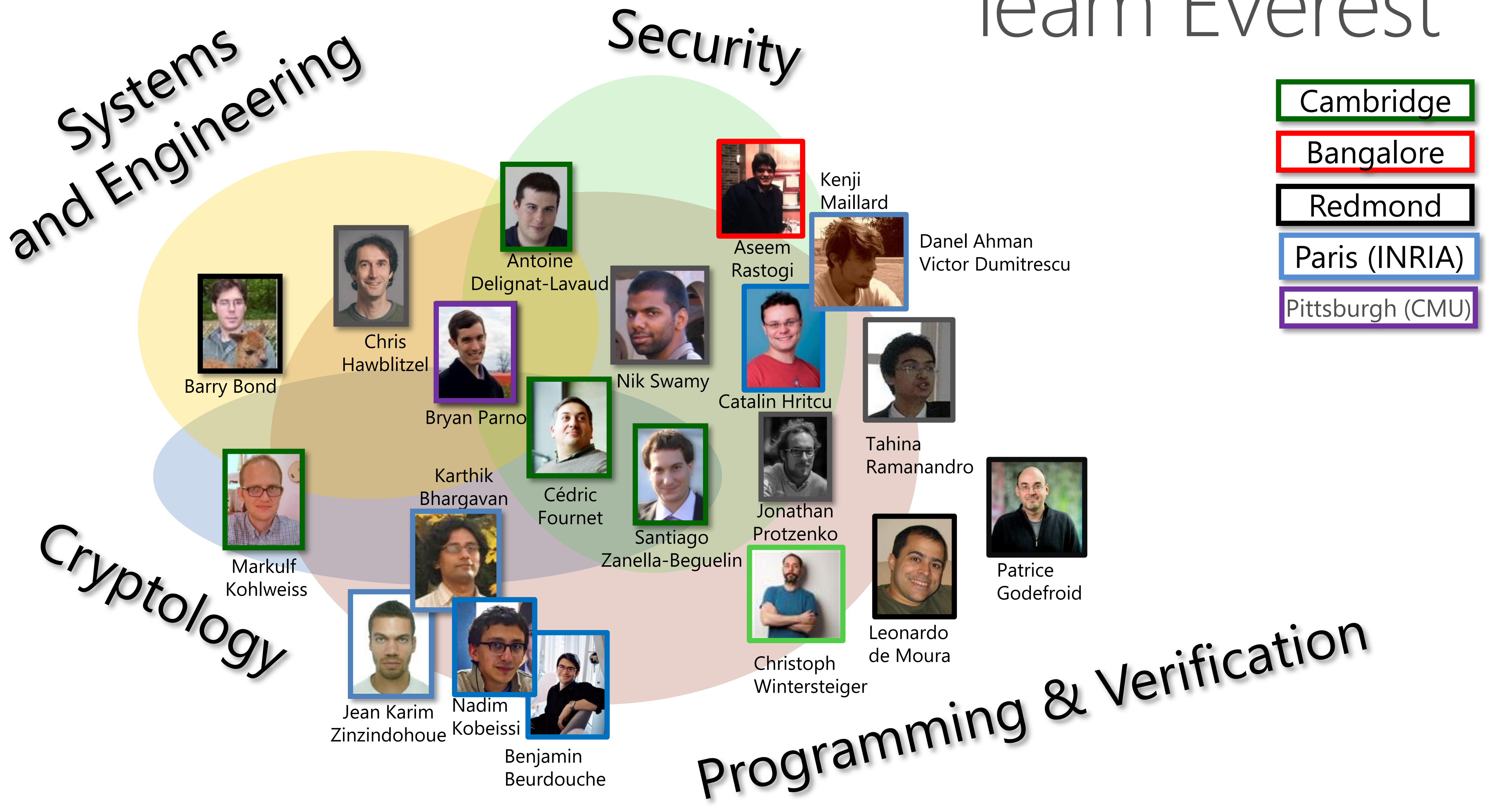


Verified Components for the HTTPS Ecosystem

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



Team Everest



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

kreMLin

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

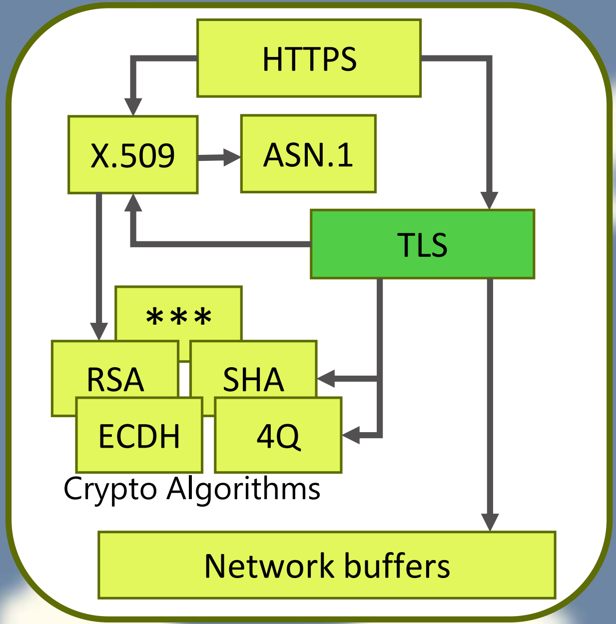
C/C++

gcc, compcert, clang, msvc

interop with rest of TLS/HTTPS ecosystem

production code

The TLS/HTTPS ecosystem



TLS Standards & Implementations

Internet Standard

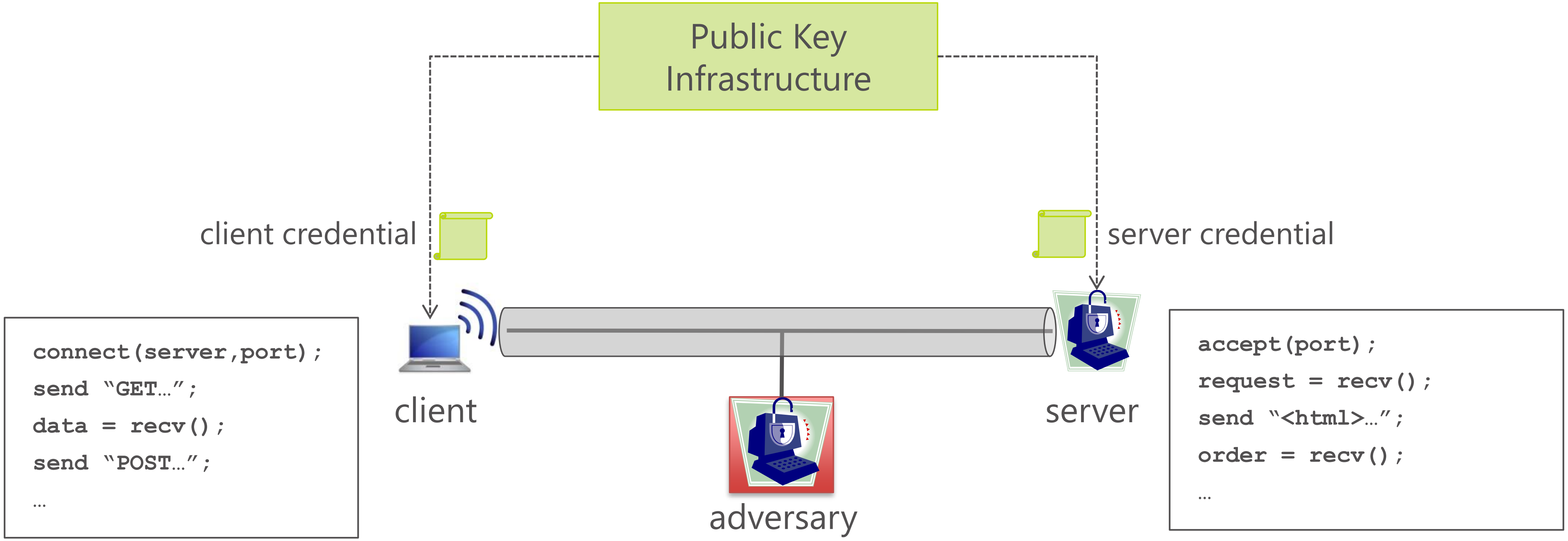
1994	Netscape's Secure Sockets Layer
1995	SSL3
1999	TLS 1.0 (≈SSL3)
2006	TLS 1.1
2008	TLS 1.2
2017?	TLS 1.3

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

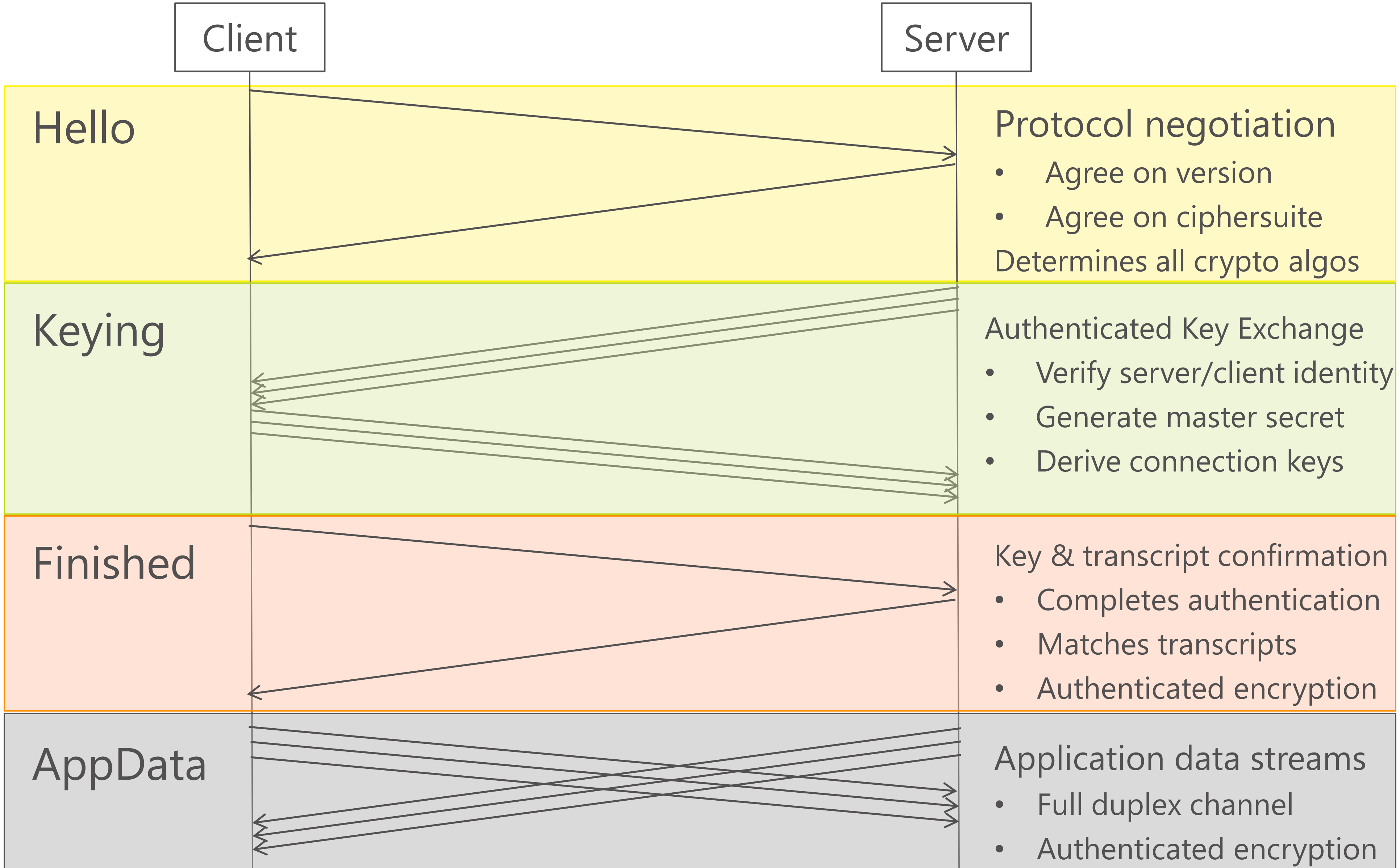
TLS Verification Goal: Secure Channel



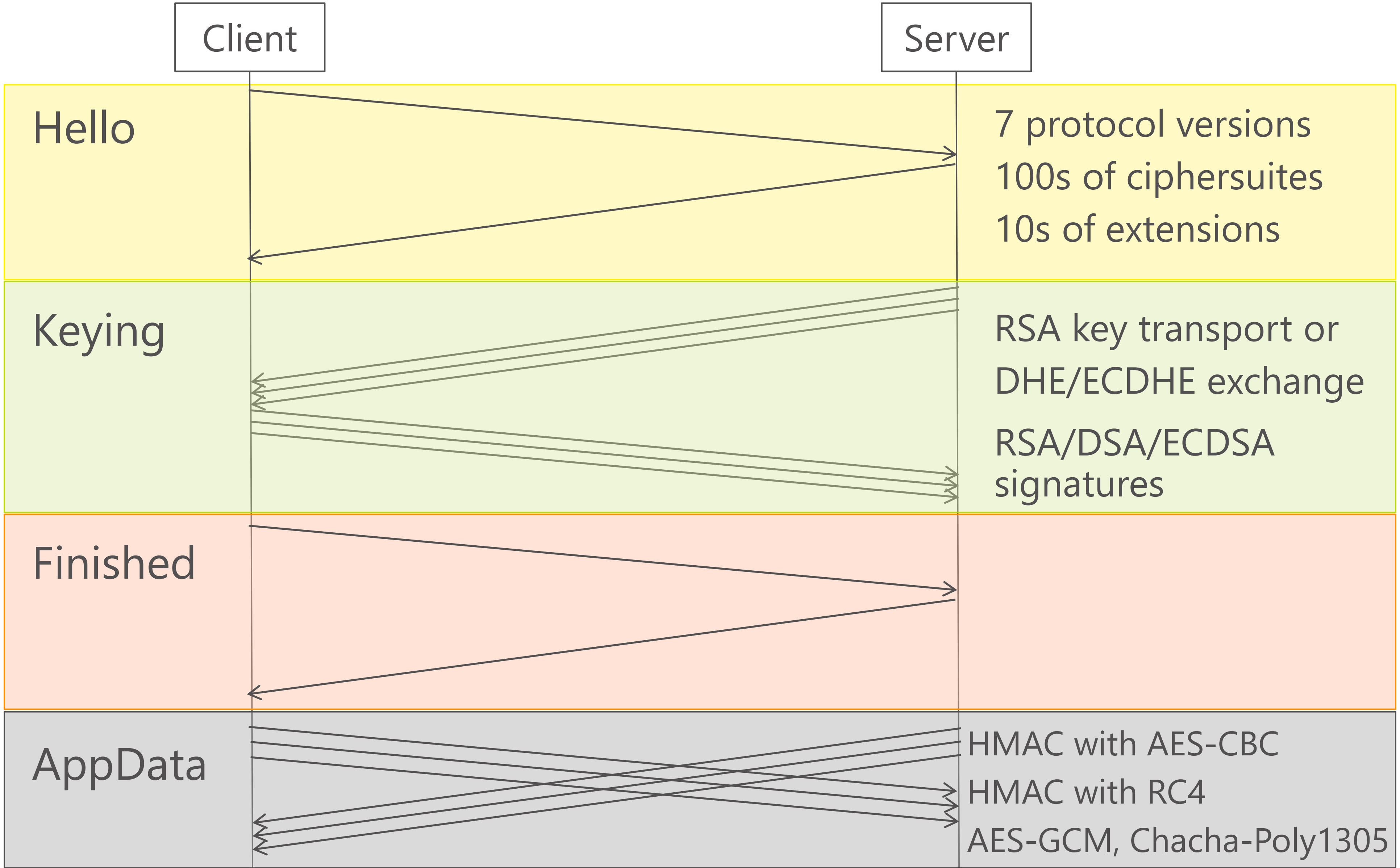
Security Goal: As long as the adversary does not control the long-term credentials of the client and server, it cannot

- 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

Microsoft Research - Inria
JOINT CENTRE

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,...)

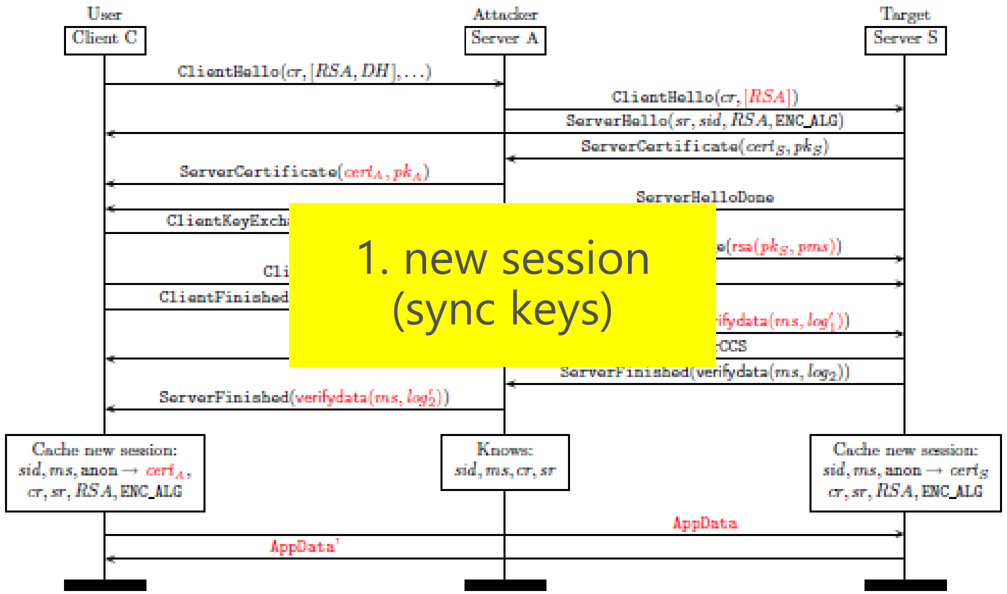
Not fully mechanized (paper proofs too)

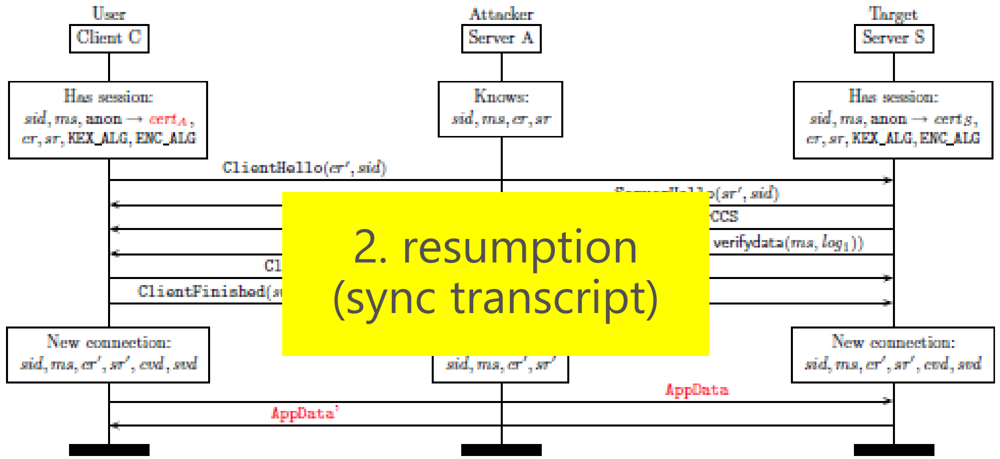
3. Experimental platform:
   for testing corner cases, trying out attacks, analysing extensions and patches, ...

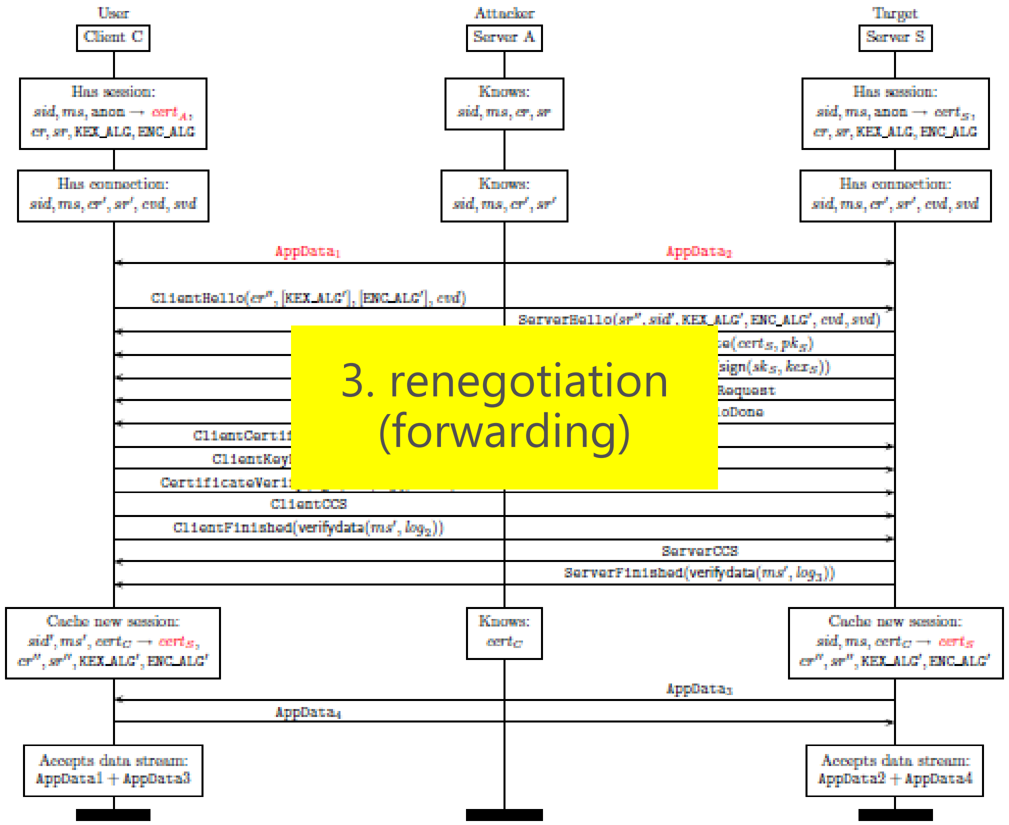
Not production code (poor performance)

Triple handshake attack (2014)

flaw in the standard
now patched in TLS







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

Systematically testing the TLS state machine

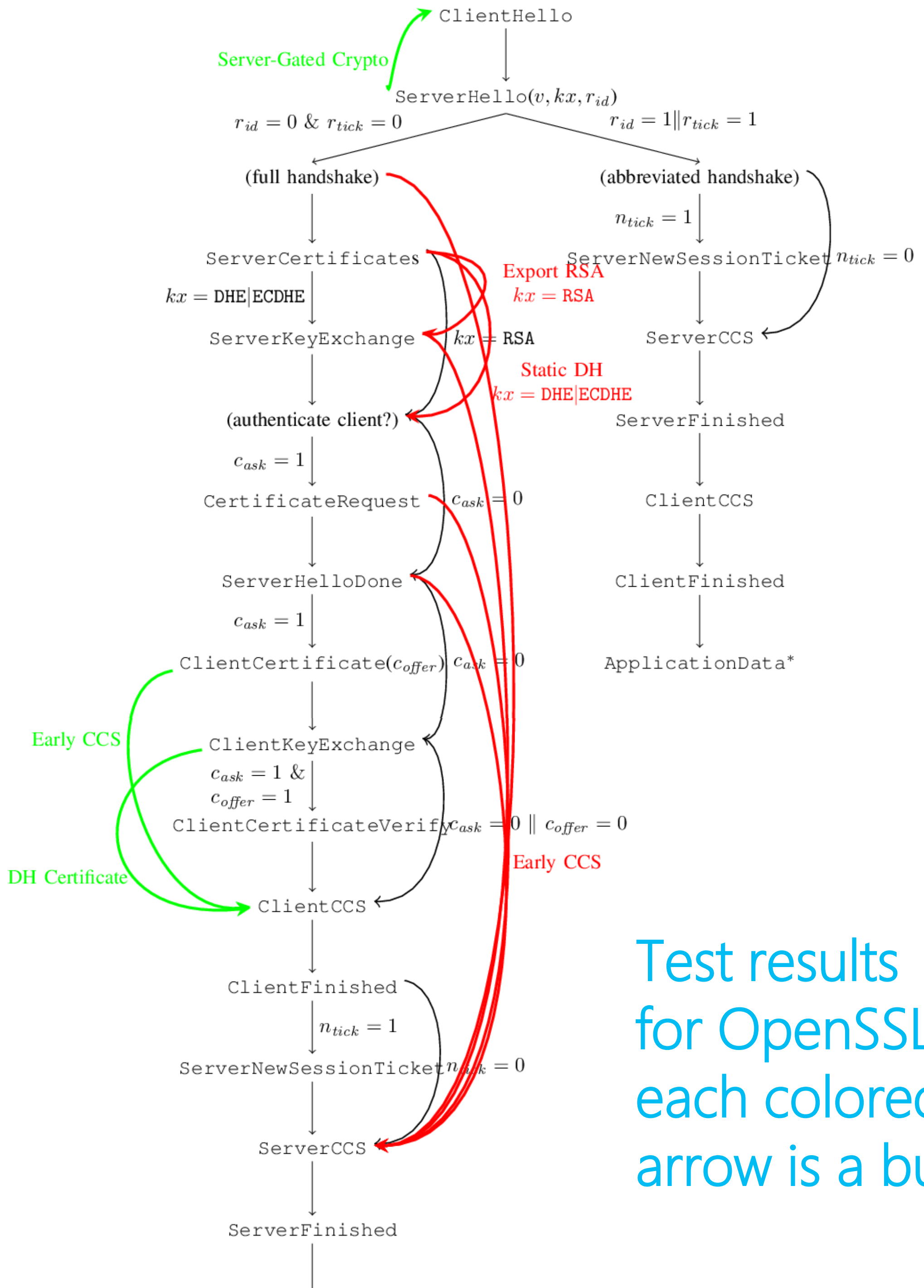
new attacks against all mainstream implementations

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

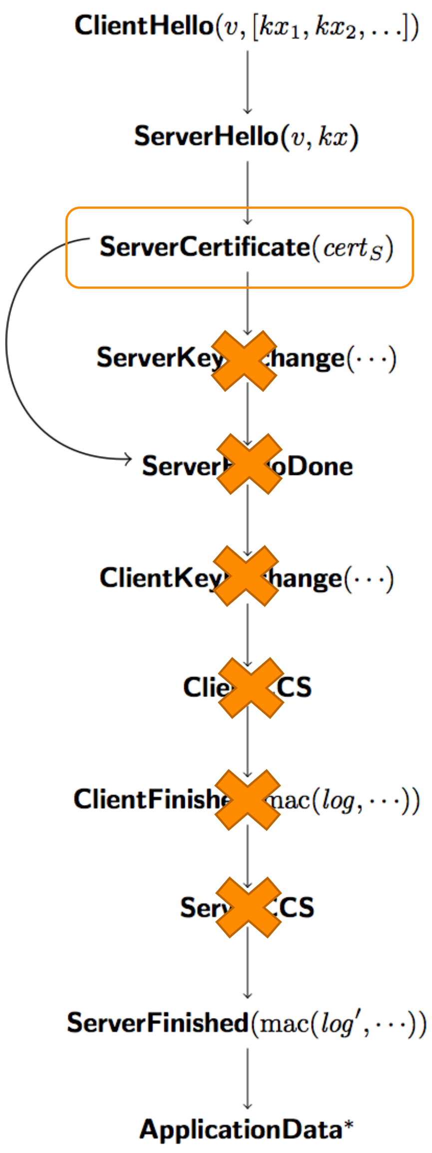
Systematically testing the TLS state machine

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)

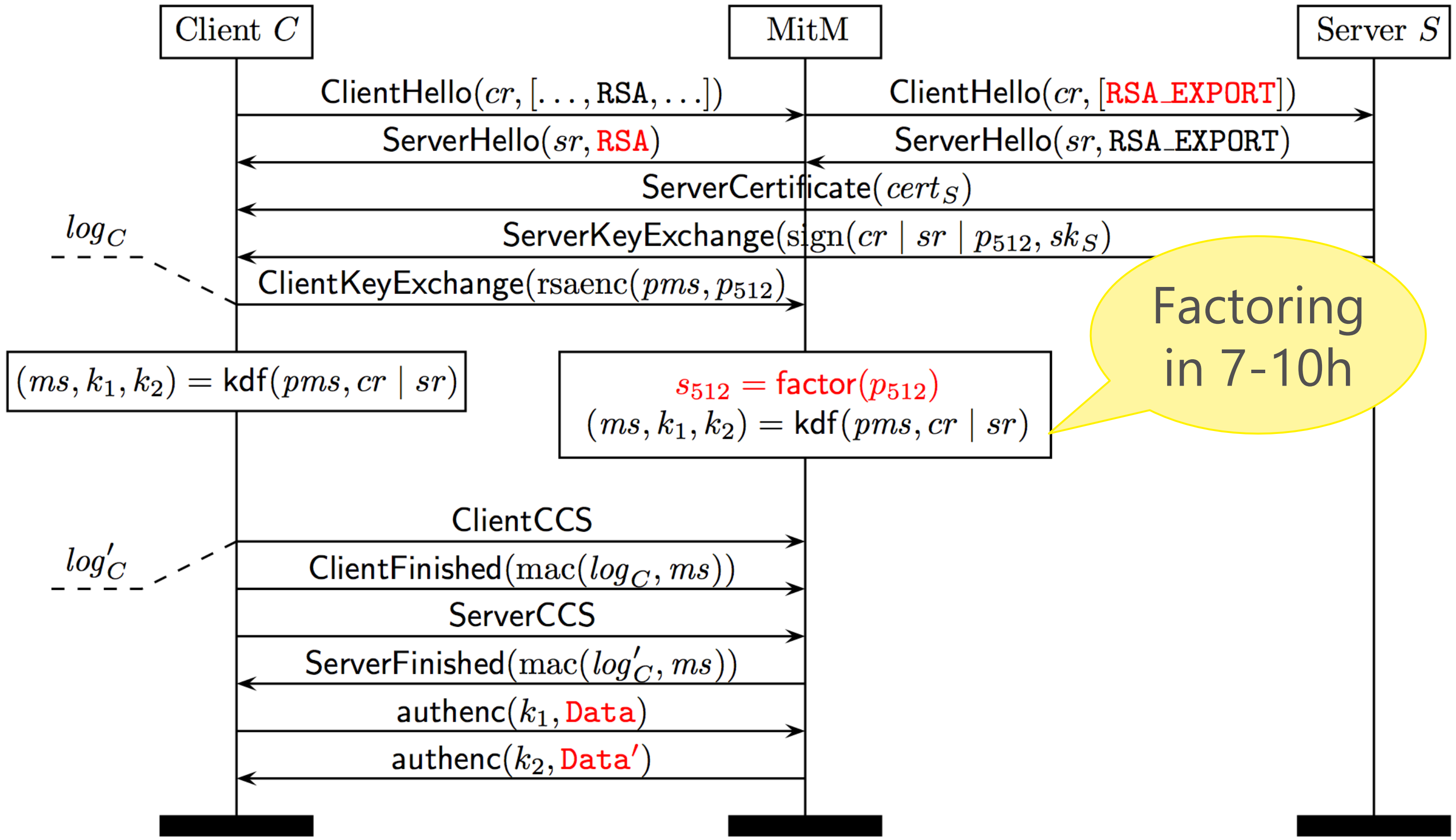
We skip 6 messages

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:

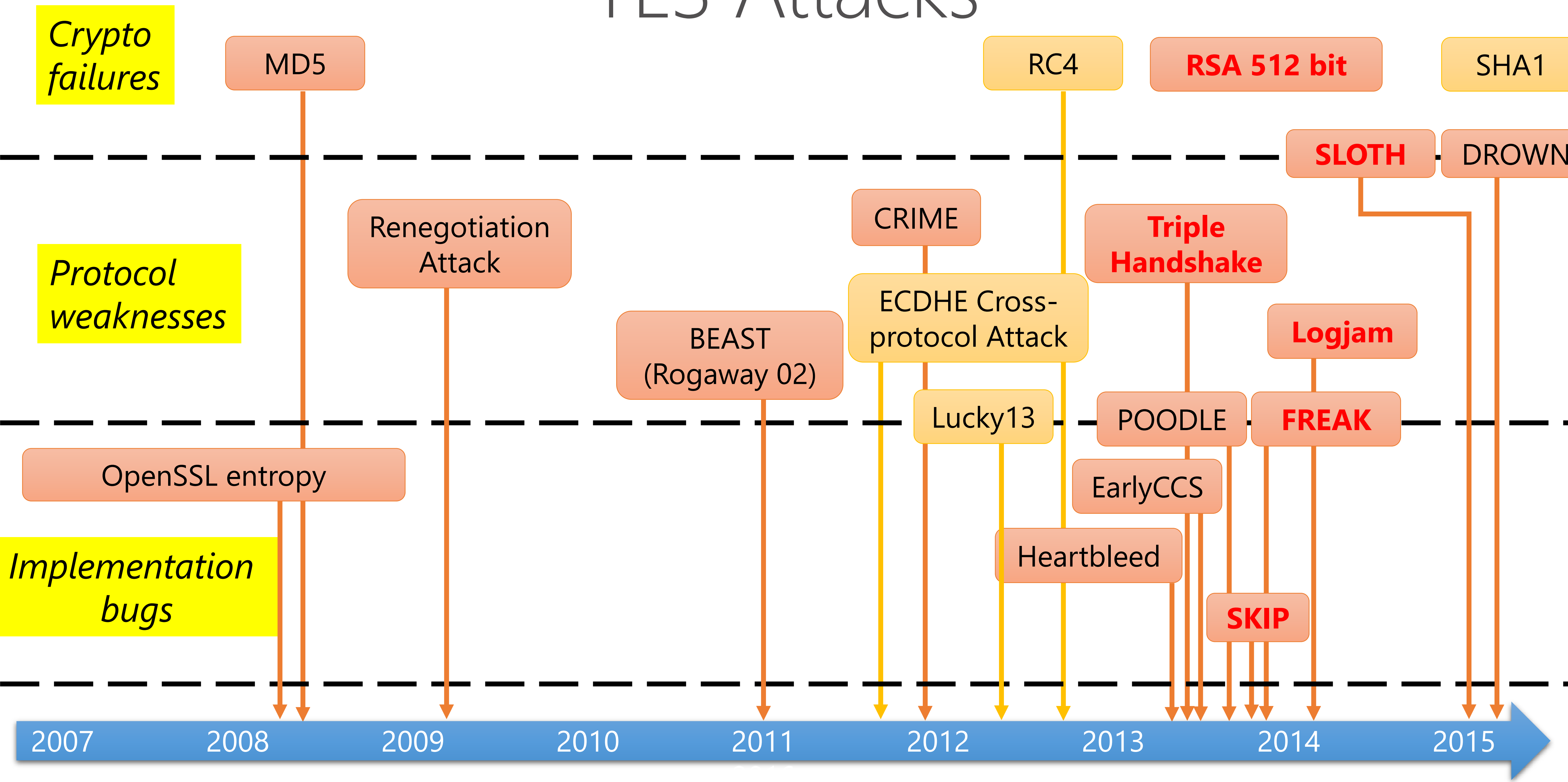
- servers that support RSA_EXPORT (512bit keys obsoleted in 2000) from 40% to 8.5%
- clients that accept ServerKeyExchange in RSA (state machine bug) almost all browsers have been patched



Similar attack, different crypto:

LOGJAM (2015) downgrade to weak groups

TLS Attacks


Crypto failures
MD5
RC4
RSA 512 bit
SHA1
SLOTH
DROWN
Protocol weaknesses
Renegotiation Attack
CRIME
Triple Handshake
ECDHE Cross-protocol Attack
BEAST (Rogaway 02)
Logjam
Lucky13
POODLE
FREAK
Implementation bugs
OpenSSL entropy
EarlyCCS
Heartbleed
SKIP
2007
2008
2009
2010
2011
2012
2013
2014
2015

TLS 1.3: a new hope

Much discussions

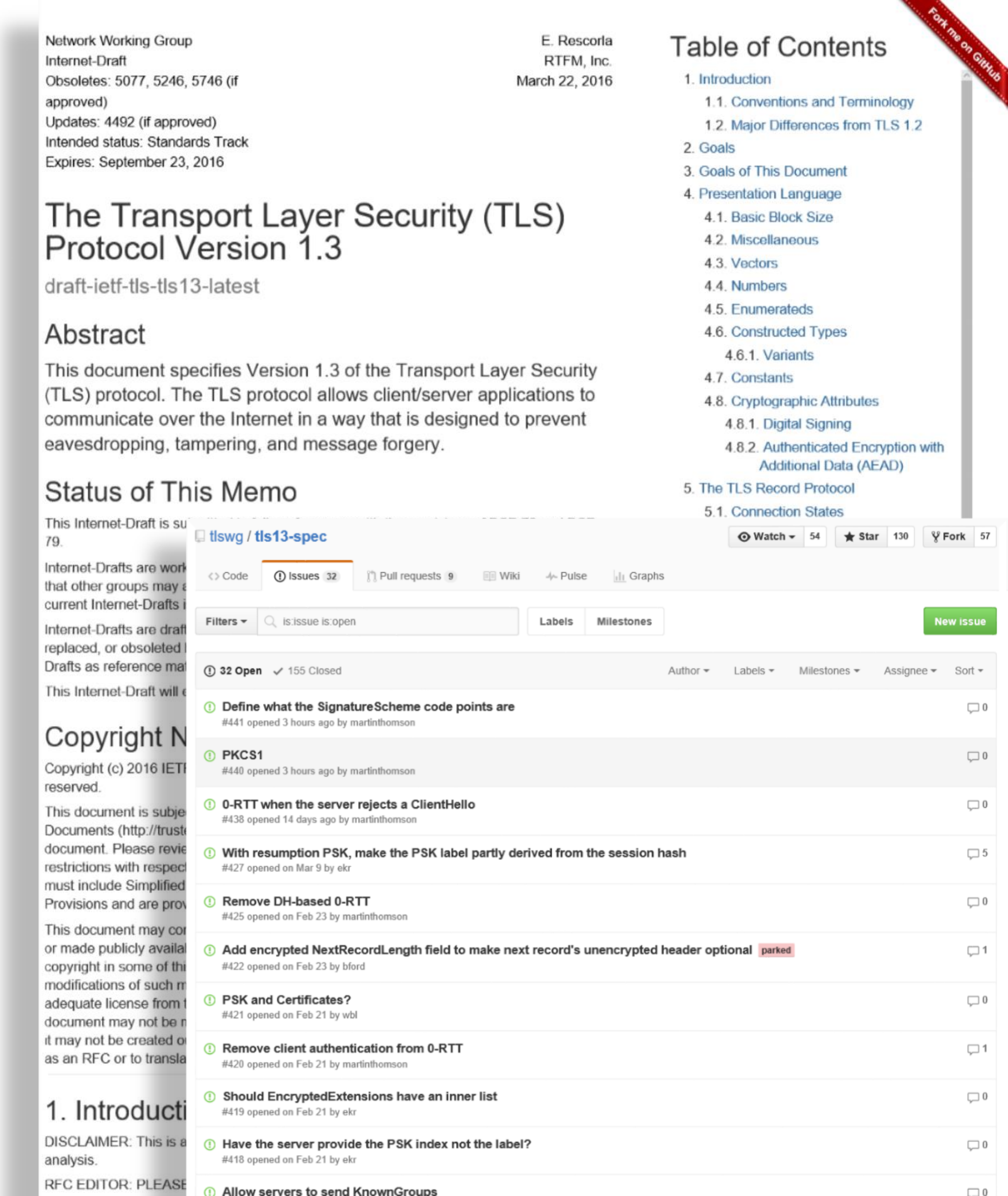
IETF, Google, Mozilla, Microsoft, CDNs, cryptographers, network engineers, …

Much improvements

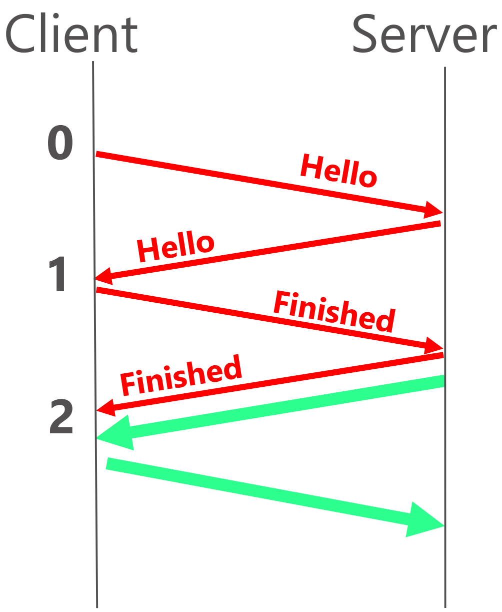
- Modern design
- Fewer roundtrips
- Stronger security

New implementations required for all

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

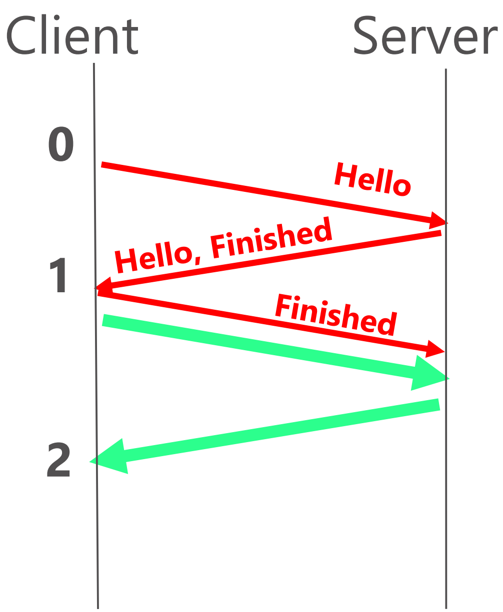


Saving roundtrips for new connections



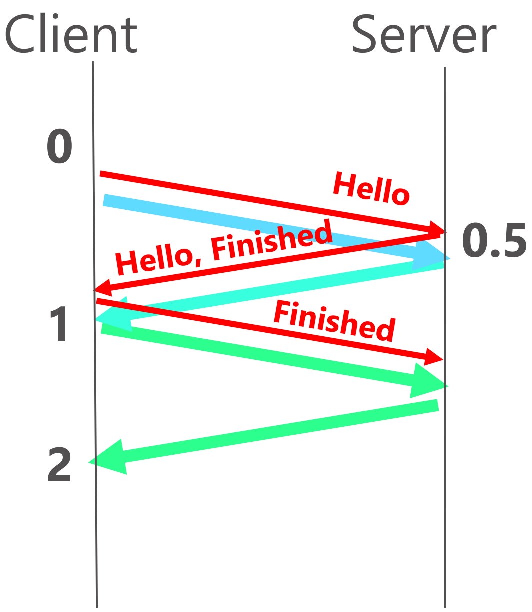
TLS 1.2

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

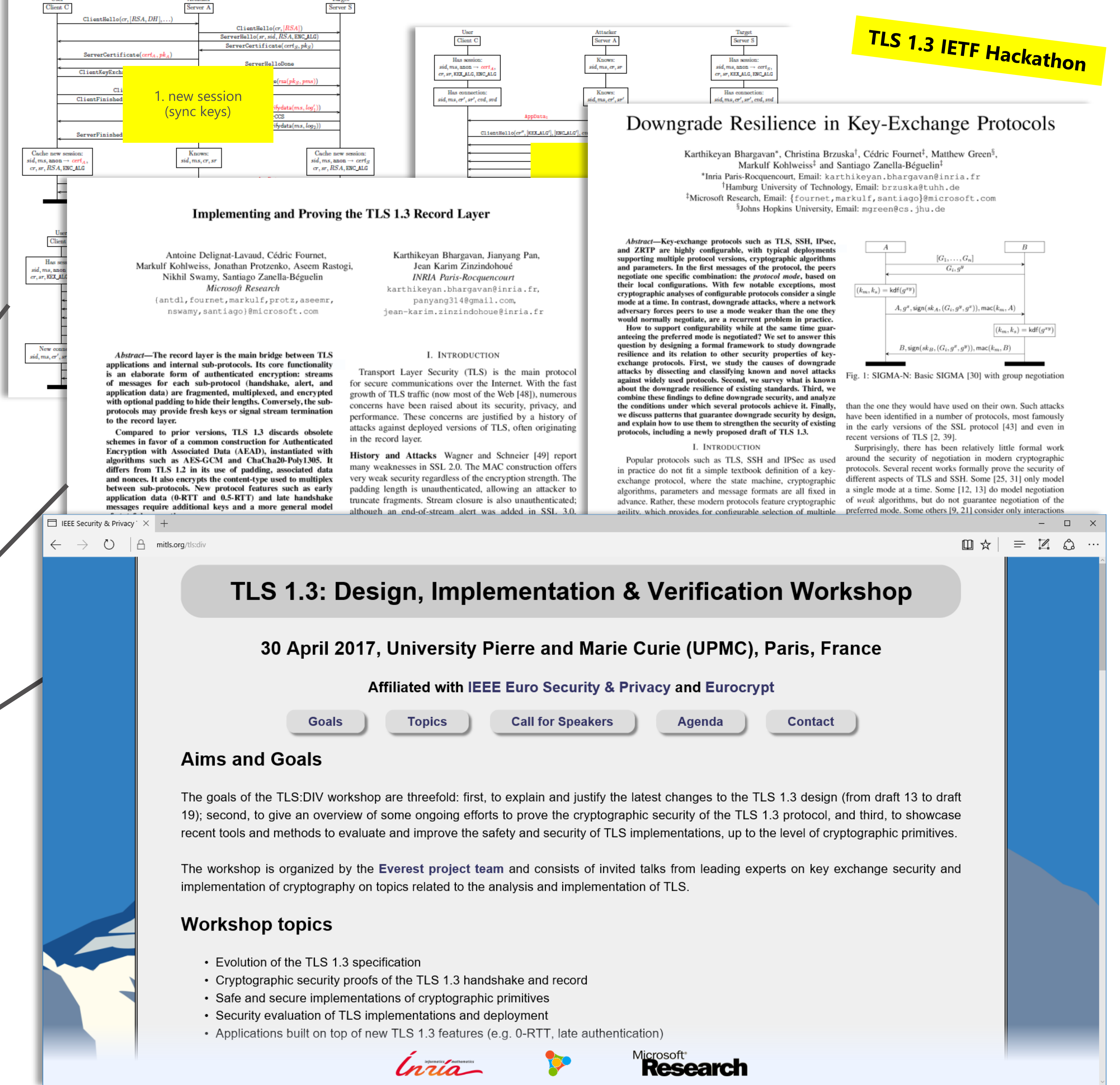
TLS 1.3: status

IETF WG ~~95~~ 99

~~13~~ 21st 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
	pre-shared-key 0RTT
#18	PSK binding (fixing an attack)

RFC finalized this month?



Cryptographic Algorithms for HTTPS

Algorithms get broken & replaced over time

Security relies on probabilistic cryptographic assumptions (who knows?)

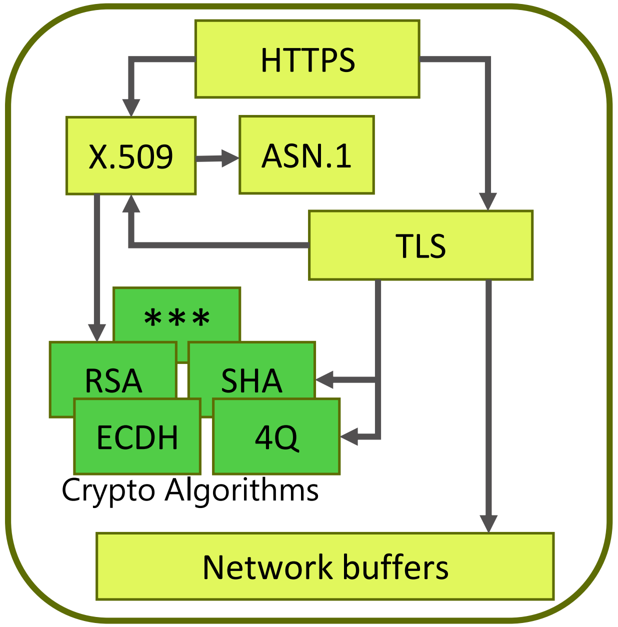
Modern design & implementations select between
various algorithms & implementations for the same core functionality

~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

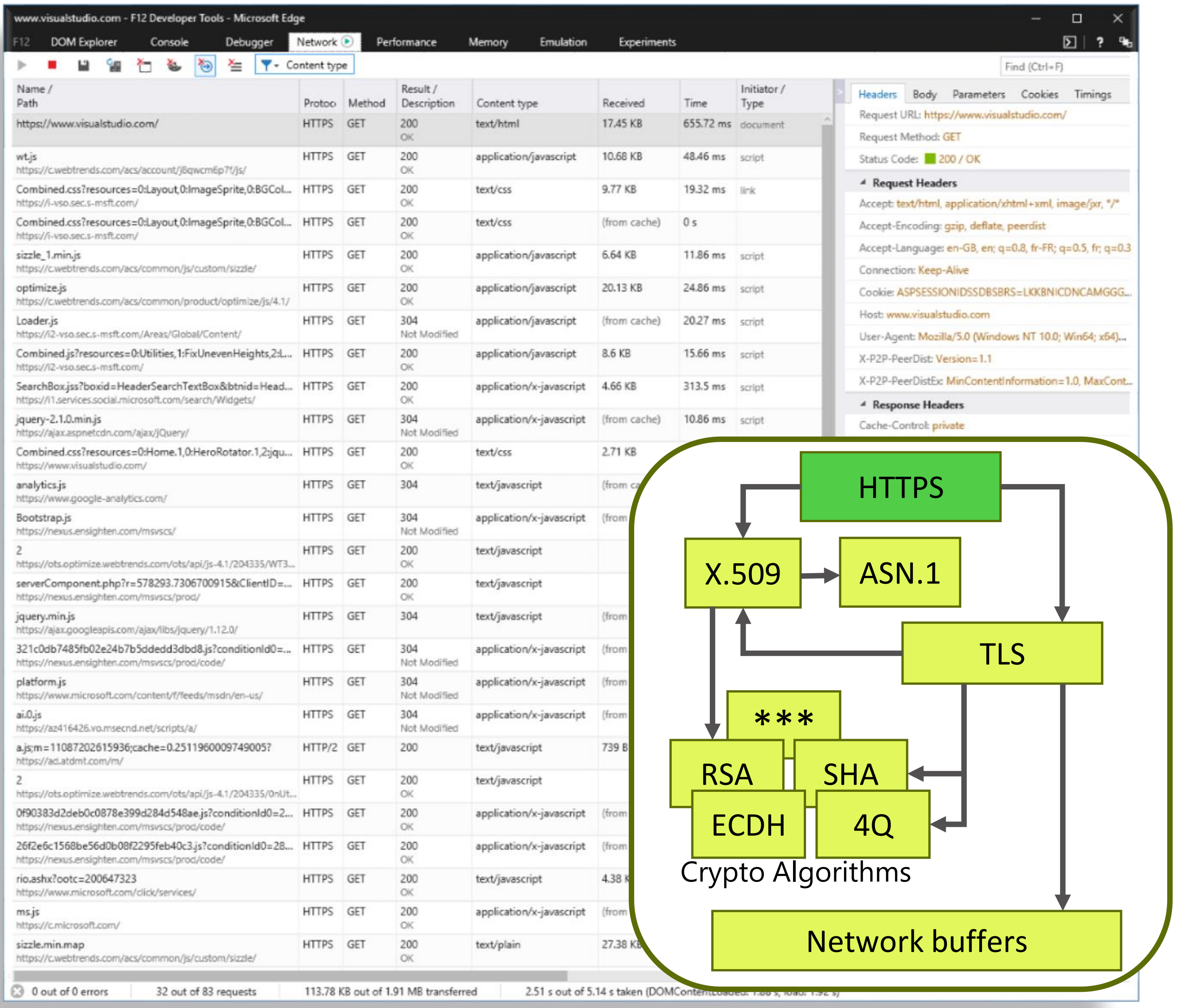
We're excited to announce the release of Project Wycheproof, a set of security tests that check cryptographic software libraries for known weaknesses. We've developed over 80 test cases which have uncovered more than 40 security bugs (some tests or bugs are not open sourced today, as they are being fixed 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://

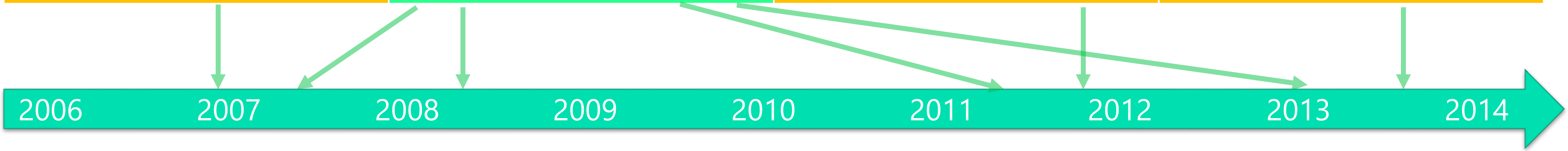
Example: tracing
https://www.visualstudio.com/

- **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



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.



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?

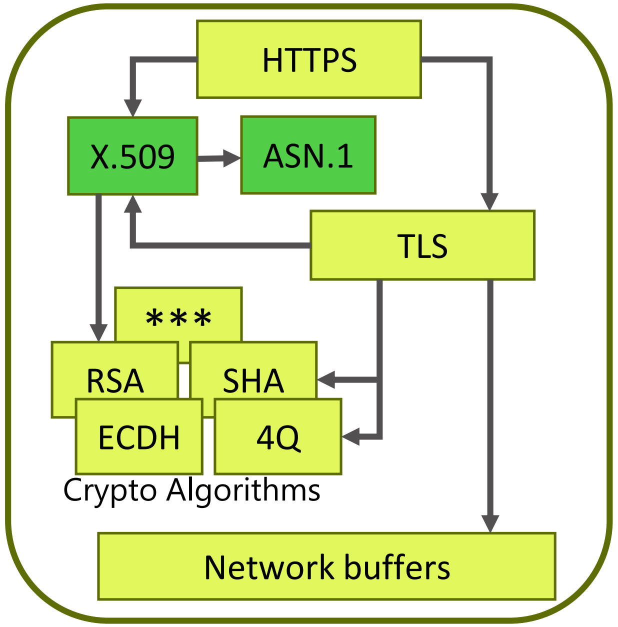
ASN.1 grammar; many extensions and interpretations
50% of "TLS attacks" are in fact X.509 attacks

Recent initiatives

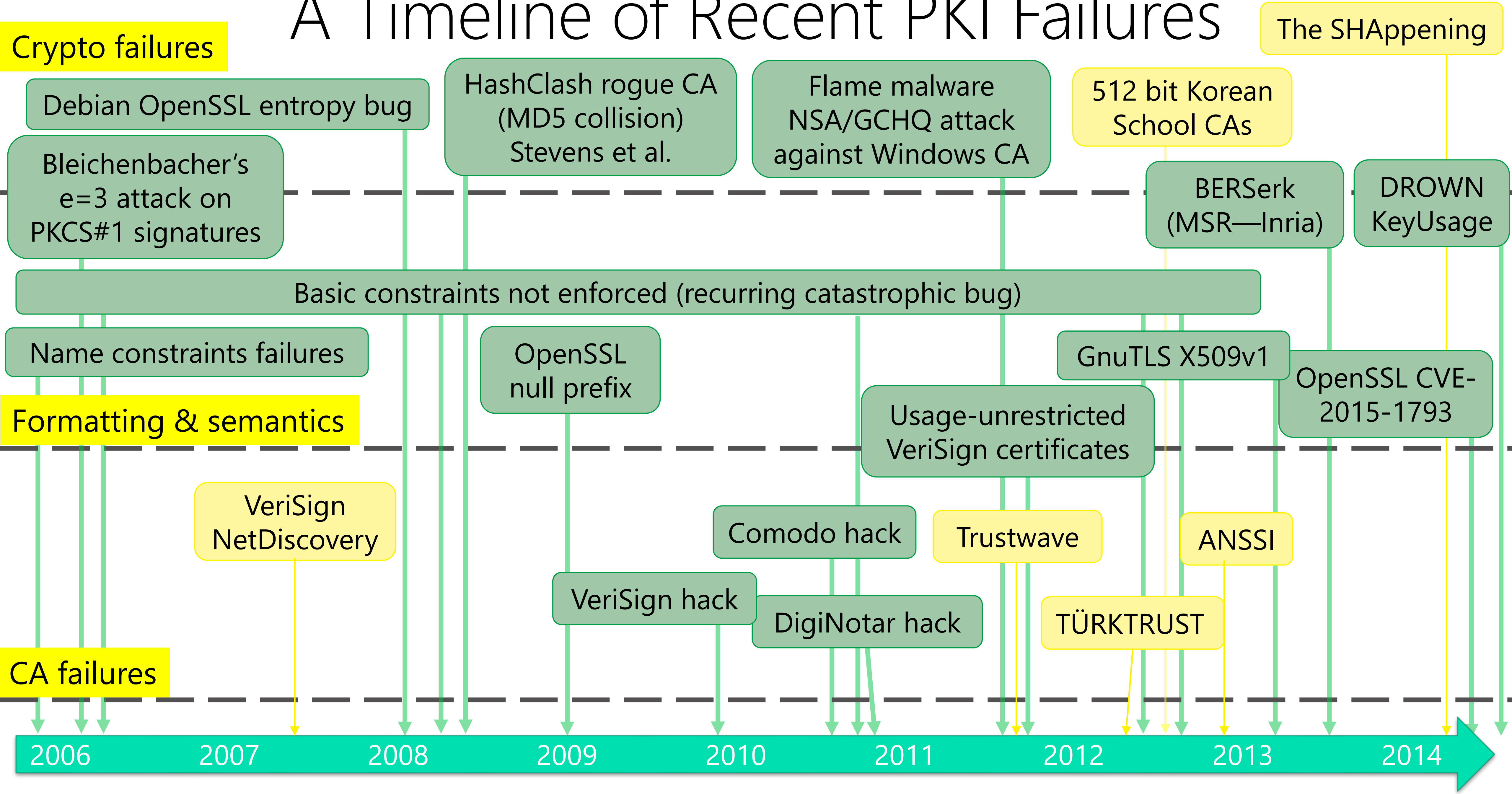
Global scans for millions of certificates
Certificate pinning & transparency
Let's encrypt! https://letsencrypt.org/

Verification?

Complex ambiguous format
Certificate issuance and revocation policies

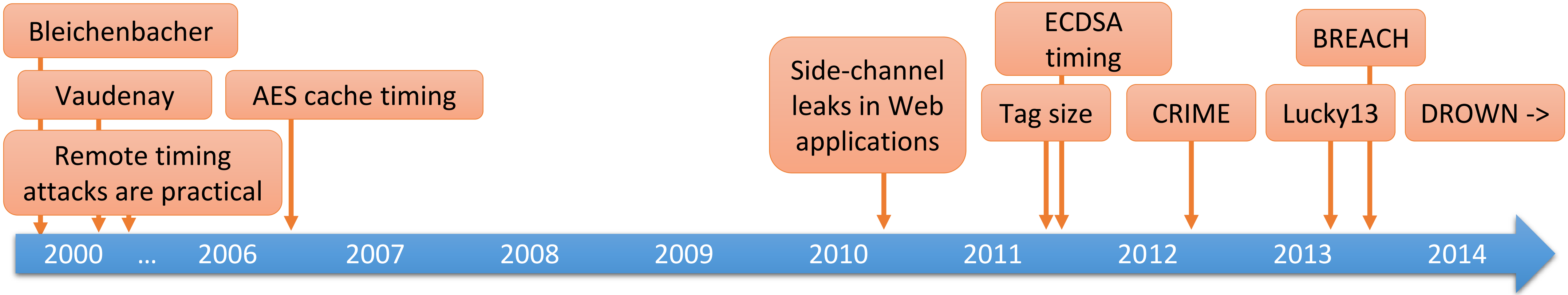


A Timeline of Recent PKI Failures



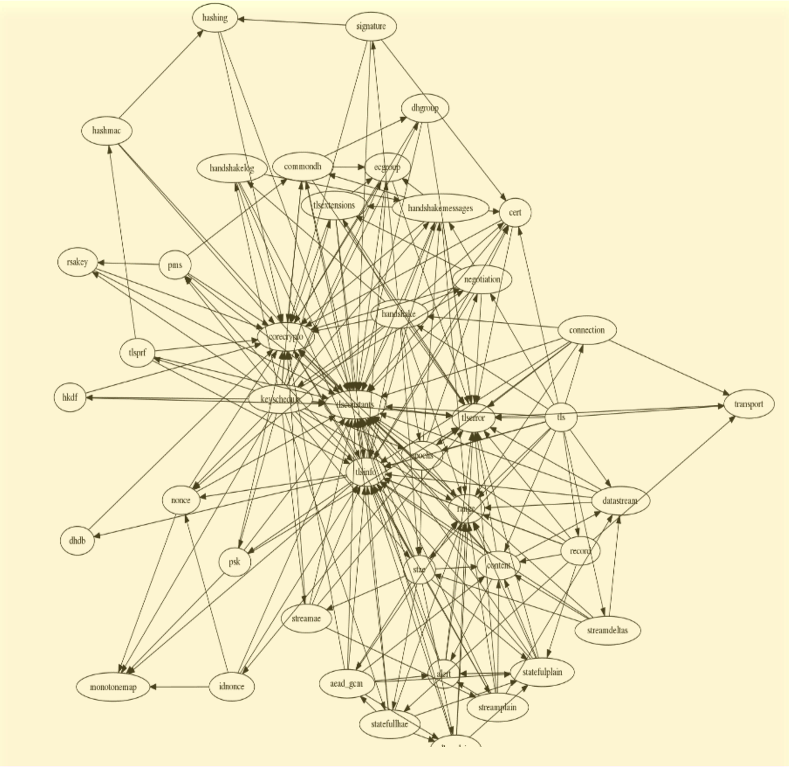
Side Channel Challenge (Attacks)

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)
• Hello message contents (e.g. time in nonces, SNI) • Alerts (e.g. decryption vs. padding alerts) • Record headers	• CRIME/BREACH (adaptive chosen plaintext attack) • User tracking • Auto-complete input theft	• Bleichenbacher attacks against PKCS#1 decryption and signatures • Timing attacks against RC4 (Lucky 13)	• OpenSSL key recovery in virtual machines • Cache timing attacks against AES

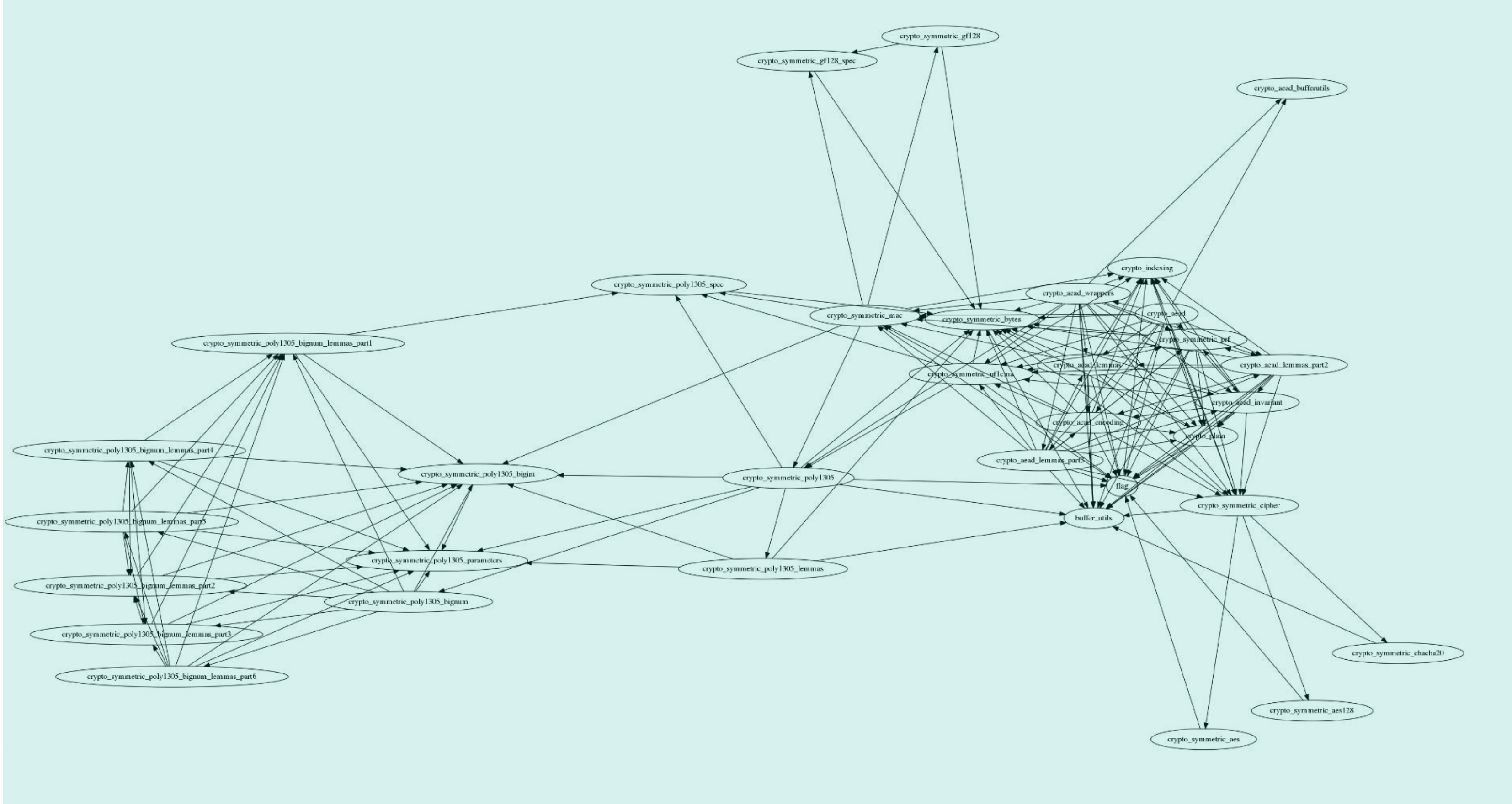


Demo

miTLS in F* today

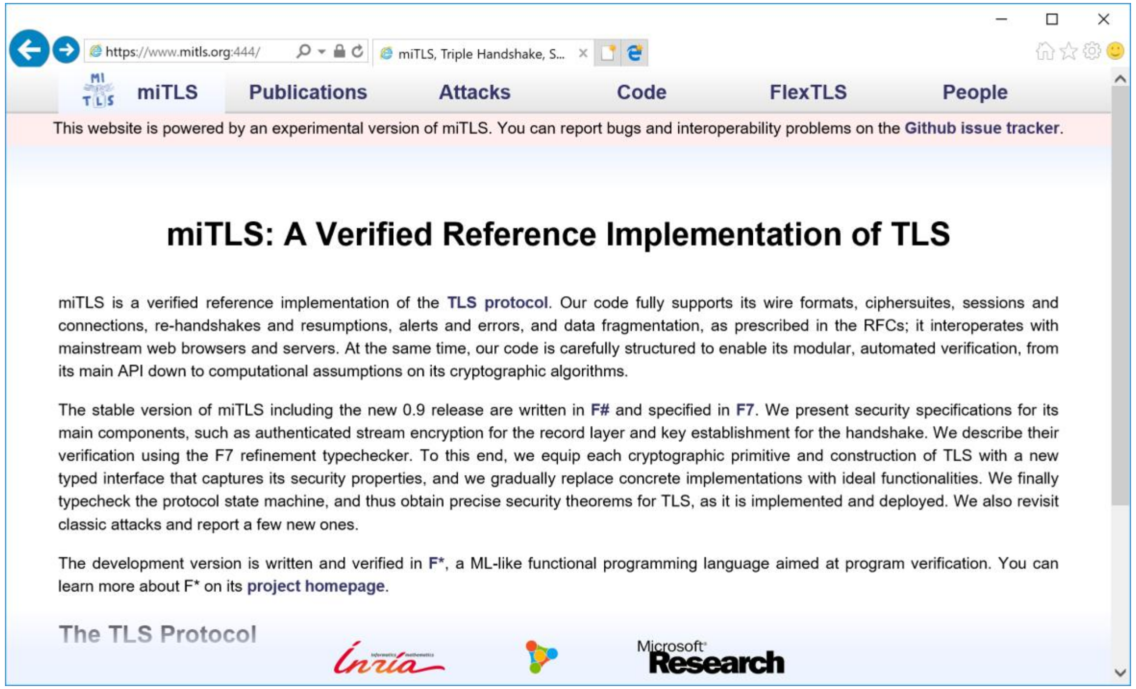


miTLS, protocol layer:
16K lines of code and proofs
Compiled to Ocaml.
Partially verified.



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

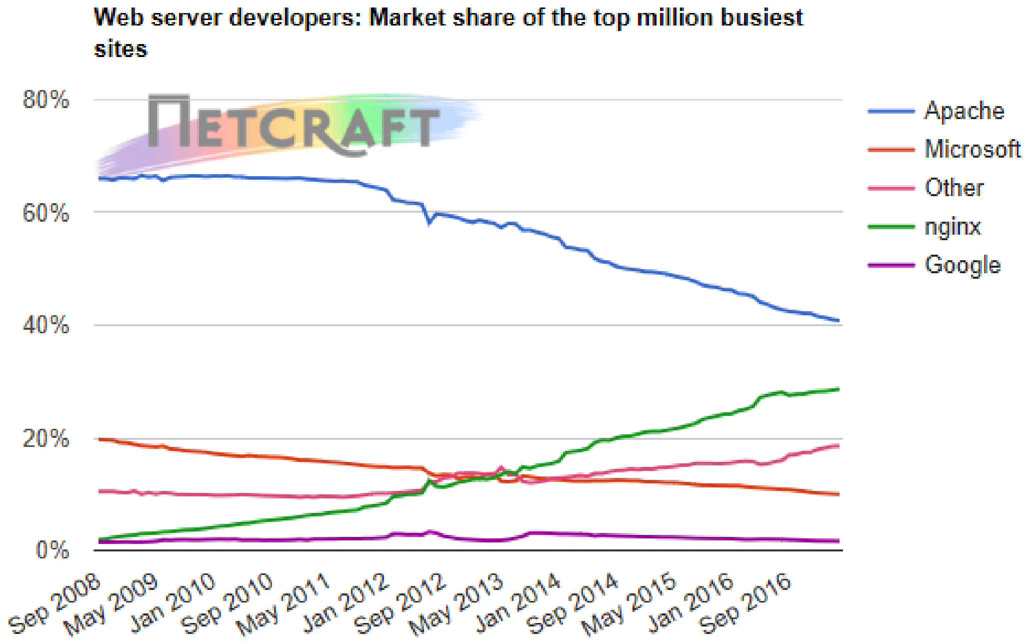
Client: IE



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