Security Verification with F*

Cédric Fournet
Catalin Hritcu
Aseem Rastogi
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

---

**Services & Applications**

- Edge
- cURL
- WebKit
- Skype
- IIS
- Apache
- Nginx

---

**Certiﬁcation Authority**

**X.509**

**ASN.1**

**TLS**

**Network buffers**

**Untrusted network (TCP, UDP, ...)**

---

**Crypto Algorithms**

- RSA
- SHA
- ECDH
- 4Q

---

**Clients**

**Servers**
Verified Components for the HTTPS Ecosystem

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

Services & Applications

- Edge
- cURL
- WebKit
- Skype
- IIS
- Apache
- Nginx

Crypto Algorithms

- RSA
- ECDH
- SHA
- AES

Network buffers

Untrusted network (TCP, UDP, ...)

Certification Authority

HTTPS

X.509

ASN.1

TLS

Network buffers

Untrusted network (TCP, UDP, ...)

湘潭 Z3
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 😌

Software safety & security (implementation)
- Focus on performance, error handling, operational security
- Security vulnerabilities & patches

RFCs (informal specs)
- Focus on wire format, flexibility, and interoperability
- Security is considered, not specified

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

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

kreMLin

C/C++

gcc, compcert, clang, msvc

production code
The TLS/HTTPS ecosystem

- **X.509**
- **ASN.1**
- **TLS**
- **HTTPS**
- **RSA**
- **ECDH**
- **SHA**
- **4Q**
- **Network buffers**

Crypto Algorithms
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
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

Hello
- Protocol negotiation
  - Agree on version
  - Agree on ciphersuite
  - Determines all crypto algos

Keying
- Authenticated Key Exchange
  - Verify server/client identity
  - Generate master secret
  - Derive connection keys

Finished
- Key & transcript confirmation
  - Completes authentication
  - Matches transcripts
  - Authenticated encryption

AppData
- Application data streams
  - Full duplex channel
  - Authenticated encryption
Many configurations *(some of them broken)*

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello</td>
<td>7 protocol versions</td>
</tr>
<tr>
<td></td>
<td>100s of ciphersuites</td>
</tr>
<tr>
<td></td>
<td>10s of extensions</td>
</tr>
<tr>
<td>Keying</td>
<td>RSA key transport or</td>
</tr>
<tr>
<td></td>
<td>DHE/ECDHE exchange</td>
</tr>
<tr>
<td></td>
<td>RSA/DSA/ECDSA signatures</td>
</tr>
<tr>
<td>Finished</td>
<td></td>
</tr>
<tr>
<td>AppData</td>
<td>HMAC with AES-CBC</td>
</tr>
<tr>
<td></td>
<td>HMAC with RC4</td>
</tr>
<tr>
<td></td>
<td>AES-GCM, Chacha-Poly1305</td>
</tr>
</tbody>
</table>
miTLS (2013—...)  
a first verified reference implementation

1. Internet Standard compliance & interoperability  
supporting SSL 3.0—TLS 1.2

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

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

---

Excluding crypto algorithms, X.509, ...

Not fully mechanized  
(paper proofs too)

Not production code  
(poor performance)
Triple handshake attack (2014)

flaw in the standard now patched in TLS

1. new session (sync keys)
2. resumption (sync transcript)
3. renegotiation (forwarding)

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)
- clients that accept ServerKeyExchange in RSA (state machine bug)

From 40% to 8.5% almost all browsers have been patched

Factoring in 7-10h

Similar attack, different crypto:
LOGJAM (2015) downgrade to weak groups
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
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 1.2: Two roundtrips before sending application data

TLS 1.3: One roundtrip before sending application data

TLS 1.3: Zero roundtrip before sending application data
IETF WG 95.99
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 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

Project Wycheckproof
December 19, 2016

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

We're excited to announce the release of Project Wycheckproof, 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 Wycheckproof, the smallest mountain in the world. The smaller the mountain the easier it is to climb it!
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

Application Security: https://
# Unsolved issues with HTTPS

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

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

Crypto failures
- Debian OpenSSL entropy bug
- Bleichenbacher’s e=3 attack on PKCS#1 signatures
- HashClash rogue CA (MD5 collision) Stevens et al.
- Flame malware NSA/GCHQ attack against Windows CA
- 512 bit Korean School CAs

Basic constraints not enforced (recurring catastrophic bug)
- Name constraints failures
- OpenSSL null prefix
- Usage-unrestricted VeriSign certificates
- GnuTLS X509v1

VeriSign CAs failures
- VeriSign hack
- Comodo hack
- DigiNotar hack
- TÜRKTRUST

Other CAs failures
- NetDiscovery
- The SHAppening
- BERSerk (MSR—Inria)
- DROWN KeyUsage
- OpenSSH CVE-2015-1793

Formatting & semantics
- OpenSSH entropy bug
- Basic constraints not enforced (recurring catastrophic bug)
- Name constraints failures
- Usage-unrestricted VeriSign certificates

CA failures
- Comodo hack
- DigiNotar hack
- TÜRKTRUST

ANSSI
- VeriSign hack
- The SHAppening
Side Channel Challenge (Attacks)

<table>
<thead>
<tr>
<th>Protocol-level side channels</th>
<th>Traffic analysis</th>
<th>Timing attacks against cryptographic primitives</th>
<th>Memory &amp; Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS messages may reveal information about the internal protocol state or the application data</td>
<td>Combined analysis of the time and length distributions of packets leaks information about the application</td>
<td>A remote attacker may learn information about crypto secrets by timing execution time for various inputs</td>
<td>Memory access patterns may expose secrets, in particular because caching may expose sensitive data (e.g. by timing)</td>
</tr>
</tbody>
</table>

- 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

Remote timing attacks are practical

Bleichenbacher

Vaudenay

AES cache timing

Side-channel leaks in Web applications

ECDSA timing

Tag size

CRIME

Lucky13

BREACH

DROWN ->
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
We integrate miTLS & its verified crypto with Internet Explorer.

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

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

Master Process

Worker Thread

Virtual Server

Certificate

TLS

Virtual Server

Certificate

TLS

Virtual Server

Certificate

TLS

Shared Cache

Unix sockets

FastCGI

HTTP

Sendfile

AIO

mmap

Backends

Application servers
Memory Caches

Local Files