DWOLLA

TLS in Practice including the road to 1.3

TLS in practice including the road to 1.3

whoami + Dwolla

Basic Cryptography + TLS Overview

TLS History (SSLv1 to TLS 1.3)

Use Cases

Opportunities

Best Practices

TLS 1.3

Evaluation Criteria/Questions

**TLS is a massive topic - will cite sources and inspiration for this talk throughout. Thanks to CloudFlare's TLS leadership, FeistyDuck's Bulletproof SSL and TLS book and SSLlabs.com*

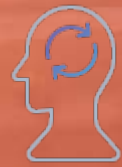
Who am I?

Ben Schmitt - VP of Information Security

I have the privilege of leading InfoSec @ Dwolla with a motivated, creative, technical and very smart group of colleagues.

Focuses: Security by design, API/Web application hardening and security @ scale in a cloud environment founded in continuous security monitoring and adaptive defense.

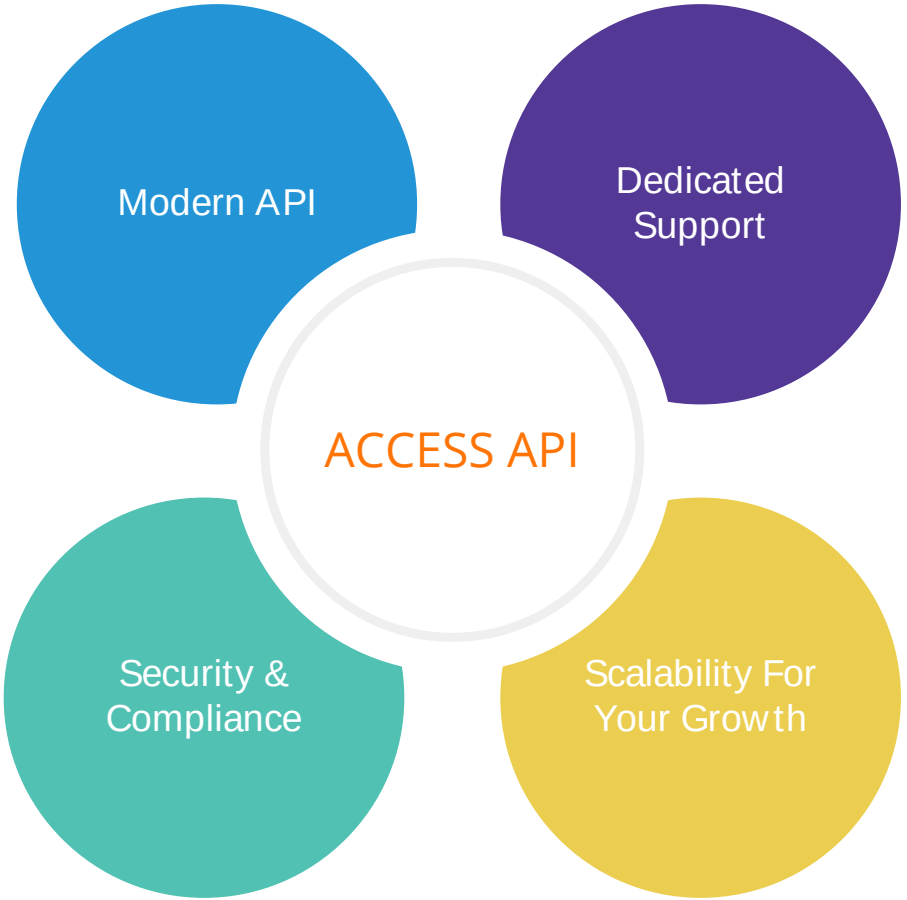
SecDSM Board Member



Security is Never Done

Dwolla's Access API

TLS is a significant part of our API's security model.



Dwolla's Engineering Team

My team is embedded (seated) within the engineering team.

My team sits about 5 feet from the Engineer on the right of the photo.

Meet Chen

Security is no longer the team in the corner, it is an embedded partner of the team which creates and implements technology.

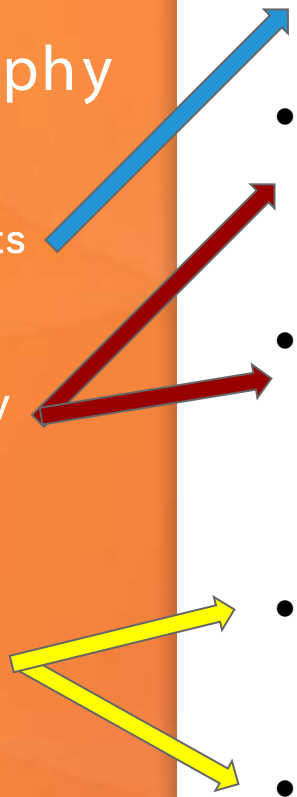
Chen's focus is on continuous security monitoring, behavioral analysis and special projects. He loves cryptography and ping pong and is currently analyzing our TLS edge.

Basic Cryptography

Shared Secrets

Confidentiality with Integrity

Integrity



Cryptographic solutions involved in TLS include the following:

- Key Exchange Algorithm
  - Forms of Diffie-Hellman, RSA (legacy)
- Asymmetric Cryptographic Systems
  - RSA and forms of ECC
  - Preference for RSA 2048 or greater keys or NIST P-curves + Curve25519
- Symmetric Cryptographic Systems
  - AES and other private key block and stream ciphers
  - Preference for AES 256 GCM
  - Authenticated Encryption with Associated Data (AEAD)
    - AES GCM, ChaCha20-Poly1305
- Digital Signatures/Hashing Algorithms
  - DSA/RSA/ECDSA
  - SHA 2 Family
- Message Authentication Code (MAC)
  - Preference for HMAC based on SHA2

TLS Overview

SSL and TLS used interchangeably. This is and isn't ok.

- Vendors should get it right.
- Most mean TLS when they say SSL but the context matters.

Primary use-case: Client/Server Application Security (data in-transit) providing Confidentiality and Integrity.

A solution to the classic Alice <> Eve <> Bob problem:

"TLS has exactly one performance problem: it isn't used widely enough. Everything else can be optimized."[1]

[1] *https://istlsfastyet.com/*

TLS Overview*

**Also an interview question when I applied at Dwolla*

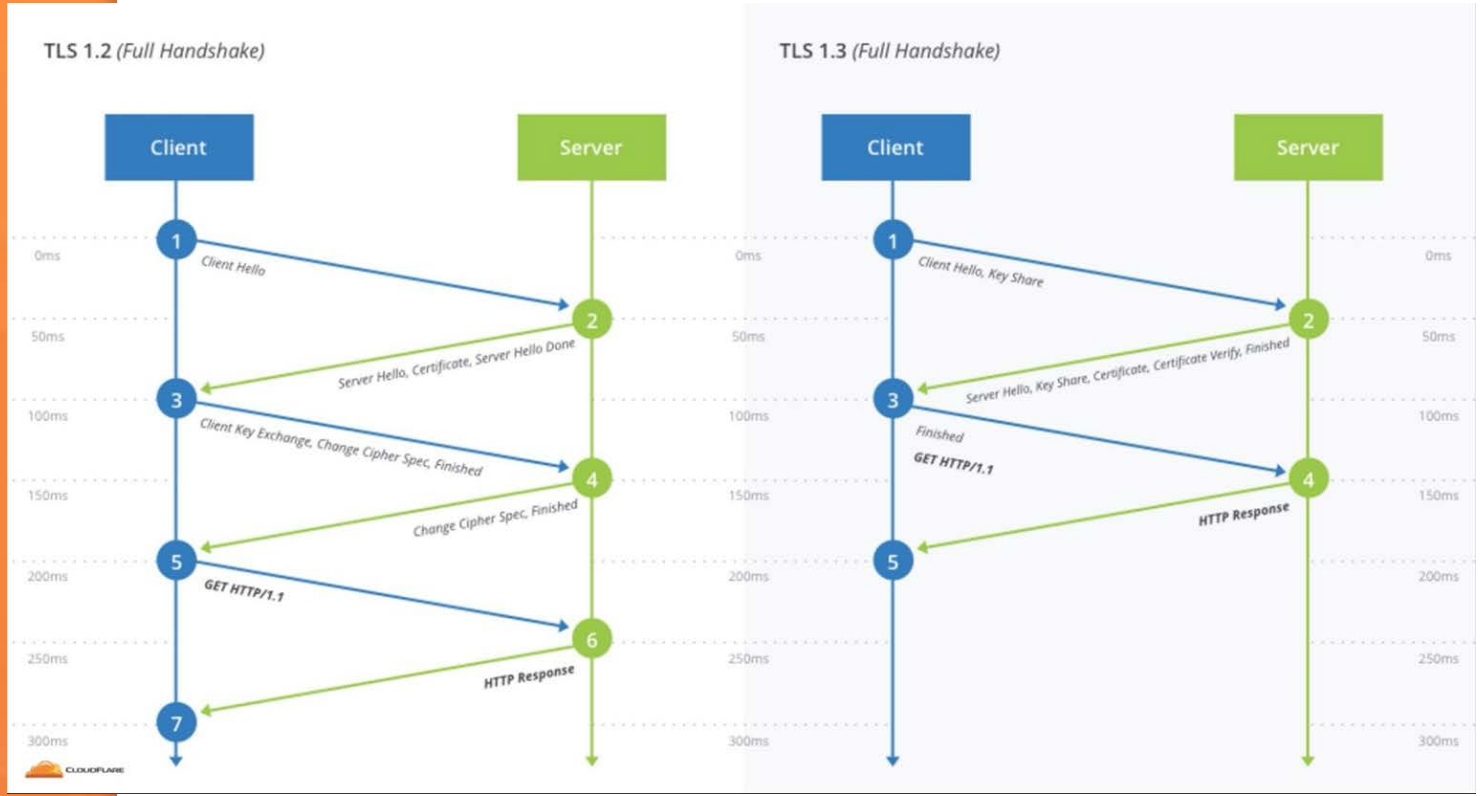
Simplified TLS session setup to exchange cryptographic keys including both asymmetric and symmetric keys.

1. Client resolves a remote hostname
2. Client HELLO:
    a. Client connects to remote host to obtain a public key presented via a Digital Certificate. Typically large-ish RSA or equivalent ECC key.
    b. Supported ciphers
3. Server HELLO:
    a. Cipher Suite
    b. Keyshare
    c. Certificate *pedigree* evaluated - proceed or not based on configuration.
4. Client Finished
5. Server Finished
6. TLS ensues

This is an oversimplification but sufficient for today's discussion. We will discuss problems with this handshake sequence and solutions in 13 to simplify and drive secure performance.

TLS Overview

What does TLS look like? From Cloudflare:



https://blog.cloudflare.com/introducing-tls-1-3/

TLS Overview

What does TLS look like? From my laptop:

75	1.798306	192.168.126.135	198.41.214.162	TLSv1.3	571	Client Hello
85	1.892961	198.41.214.162	192.168.126.135	TLSv1.3	1434	Server Hello, Application Data
89	1.894157	198.41.214.162	192.168.126.135	TLSv1.3	647	Application DataApplication Data, Application Data
90	1.894162	198.41.214.162	192.168.126.135	TLSv1.3	103	Application Data
93	1.894768	192.168.126.135	198.41.214.162	TLSv1.3	112	Application Data
95	1.923592	198.41.214.162	192.168.126.135	TLSv1.3	218	Application Data
97	1.924978	198.41.214.162	192.168.126.135	TLSv1.3	218	Application Data

```
➜  ~ sudo tcpdump -i en0 -vv port 443 -c2 -X
tcpdump: listening on en0, link-type EN10MB (Ethernet), capture size 262144 bytes
16:09:35.413004 IP (tos 0x2,ECT(0), ttl 64, id 44307, offset 0, flags [DF], proto TCP (6)
, length 1076)
    10.10.10.113.62280 > r-199-59-150-40.twttr.com.https: Flags [P.], cksum 0x45b6 (corre
ct), seq 3786681872:3786682896, ack 4148498986, win 4096, options [nop,nop,TS val 6544908
31 ecr 609457356], length 1024
        0x0000:  4502 0434 ad13 4000 4006 17d0 0a0a 0a71  E..4..@.@......q
        0x0010:  c73b 9628 f348 01bb e1b4 2e10 f745 122a  .;.(.H.......E.*
        0x0020:  8018 1000 45b6 0000 0101 080a 2702 bccf  ....E.......'...
        0x0030:  2453 94cc 1703 0300 3a30 6e50 8849 6dd8  $S......:0nP.Im.
        0x0040:  04f8 0a43 87d2 6010 9d9f 4712 6541 99ef  ...C..`...G.eA..
        0x0050:  0944 f10a 681c 1654 c4dd def5 4260 ced2  .D..h..T....B`..
        0x0060:  30d4 e96d cc96 fe17 a8a1 076f 0dcb fecb  0..m.......o....
        0x0070:  85eb 0f17 0303 121c 306e 5088 496d d805  ........0nP.Im..
        0x0080:  0e01 3b12 934d 7ace 378a 8b95 3dae 7367  ..;..Mz.7...=.sg
        0x0090:  8835 34ad 9b2b 500c 0fcd 7556 a849 2e95  .54..+P...uV.I..
        0x00a0:  7bd3 3e5d 51e6 2761 a361 5c00 6ae2 fc04  {.>]Q.'a.a\.j...
        0x00b0:  df0e 3ee3 40ec a1a3 075c ae7c ab11 9beb  ..>.@....\.|....
        0x00c0:  a0bd a5cb cd8e 36c5 e45f dd04 b7cf 14e0  ......6._......
        0x00d0:  bbd5 5926 2c7e 7743 5a32 777e 5f60 8798  ..Y&,~wCZ2w~_`..
        0x00e0:  9369 b934 ab64 6152 e310 542e 5eac 8a82  .i.4.daR..T.^...
        0x00f0:  f636 042b 177a cabf f3a2 f366 2d4d af52  .6.+.z.....f-M.R
        0x0100:  4f21 d0e1 f1f3 cd0f c007 5dc5 dea2 1447  O!........]....G
        0x0110:  ba43 18b9 82da 022b 9425 0f5f 80e0 f376  .C.....+.%._...v
        0x0120:  713d 3d1d a809 0190 8631 efac 20bf 637a  q==......1....cz
        0x0130:  cb45 9007 3cd8 4196 f63d 39e1 fb76 0e00  .E..<.A..=9..v..
        0x0140:  6170 b652 7d4b 6fa6 44a8 c8bc 1fa6 db74  ap.R}Ko.D......t
        0x0150:  46ca c839 e0f1 37a0 395e 19ea bc75 5b51  F..9..7.9^...u[Q
        0x0160:  0ad6 724e 6e27 6b16 bddf b951 d284 2533  ..rNn'k....Q..%3
        0x0170:  be7c 53ee e824 8aae 87f3 477c 1fb8 1d3f  .|S..$....G|...?
```

TLS Overview

Digital Certificates are created and certified by a Certificate Authority such as Comodo, Symantec, GoDaddy, Let's Encrypt, etc.

A Certificate Authority operates a Public Key Infrastructure (PKI).

Certificates are obtained via a Certificate Signing Request (CSR). Typically PKCS#10 format:

- CSR contains information about the applicant such as Distinguished Name, Business Name, etc.
- May be accompanied by additional information to verify the authenticity of the request.
- Signed by applicant's private key, includes public key.

Certificate is provided - typically x509 format.

- Includes public key, identity and CA signature
- Can be self-signed which isn't necessarily bad.

Pedigree of Digital Certificate provided by Certificate Chain to a root of trust.

TLS History

Over 20 years old.

- ~ 1994: Started as an internal project @ Netscape
- 1994: SSLv2
- 1995: SSLv3
- 1999: TLS 1.0
- 2006: TLS 1.1
- 2008: TLS 1.2
- 2010: HTTP Strict transport Security (HSTS)/SPDY
- 2011: Forward Secrecy
- 2012: Content Security Policy (CSP)
- 2013: ChaCha20-Poly1305
- 2015: Let's Encrypt
- 2017: SHA1 deprecation
- 2018: TLS 1.0/1.1 deprecation

https://www.feistyduck.com/ssl-tls-and-pki-history/

TLS History

Notable Vulnerabilities:

- Heartbleed: OpenSSL vulnerability allowing memory to be leaked
- CRIME: Abusing compression to extract secrets (session cookies)
- LogJam: TLS downgrade attack to export-level Diffie-Hellman parameters.
- DROWN: SSLv2 downgrade attacks against vulnerable servers

Some Use Cases

Web Applications

> Browsers
> Web/Application Servers
> APIs
> Software Update Services

Armoring Protocols

> FTP
> SMTP

VPN

> Cisco AnyConnect
> OpenVPN
> NetScaler

Mutual Authentication

> Servers authentication + Client authentication

Opportunities

Downgrade Attacks

Weak cryptography

Static or long-lived keys

Weak key exchange parameters

Vulnerable components

Compression weaknesses

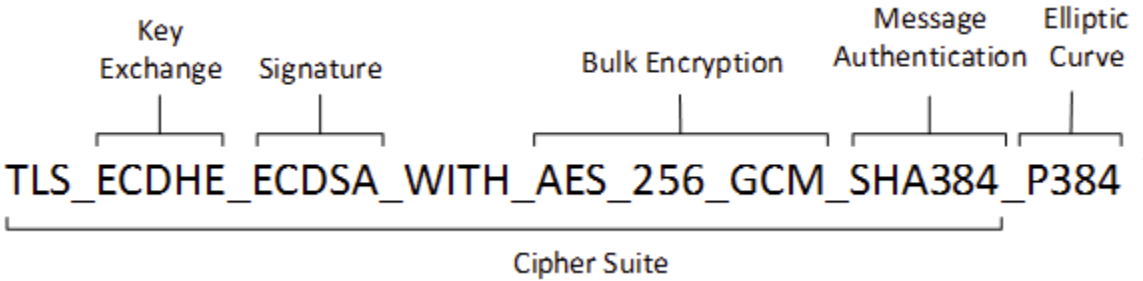
Conversion to 12 and beyond

Complexity

Opportunities

Let's look @ some poor cipher suites which may be negotiated by clients.

Great Example [1] (courtesy of Microsoft):



Really Bad Examples:

TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
SSL_RC4_128_WITH_MD5
SSL_DES_192_EDE3_CBC_WITH_MD5
SSL_RC2_CBC_128_CBC_WITH_MD5
SSL_DES_64_CBC_WITH_MD5
SSL_RC4_128_EXPORT40_WITH_MD5

[1] https://msdn.microsoft.com/en-us/library/windows/desktop/aa374757(v=vs.85).aspx

Best Practices

TLS all things: HTTP redirected to HTTPS

>= TLS 1.2

Certificate key strength >=2048 bits

Forward Secrecy: each connection individually protected with different, rotating keys.

AEAD Ciphers: AES GCM / ChaCha20-Poly1305

HSTS: header in encrypted responses instructing the client to fail closed.

CSP: header directing client behavior with a focus on XSS mitigation

Monitoring of TLS edge.

SSLLabs.com score

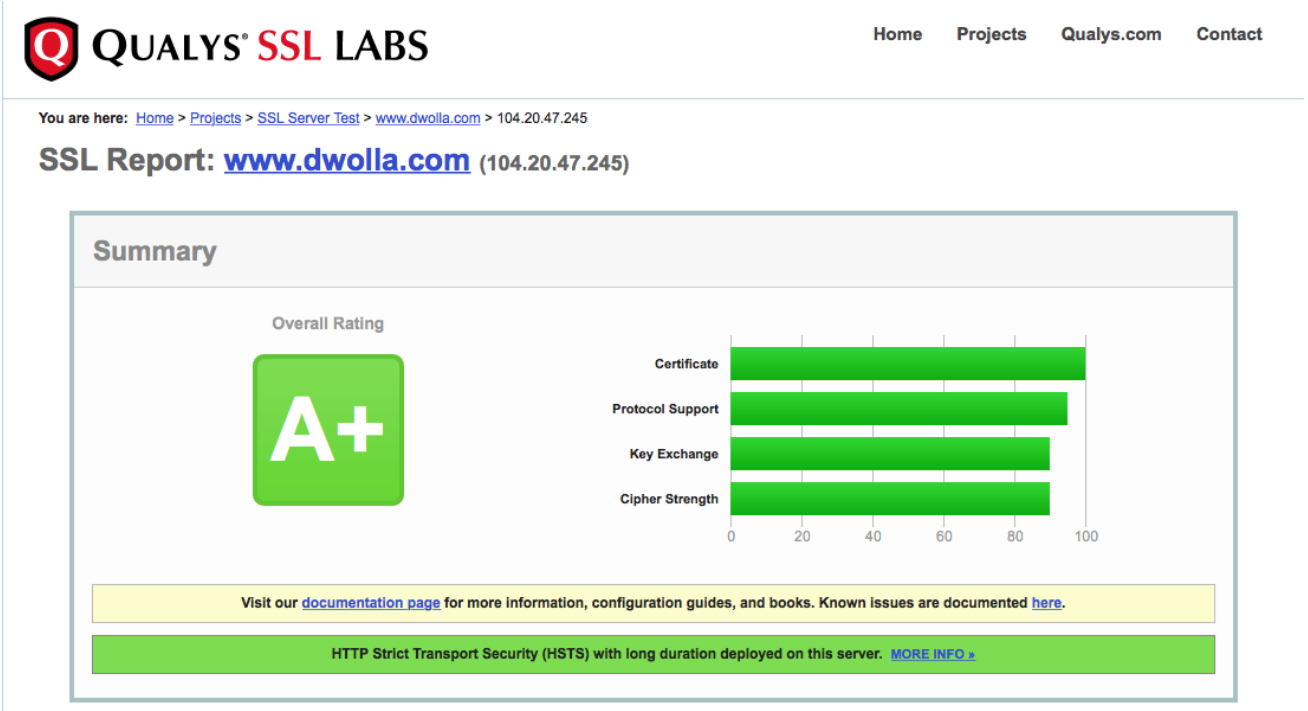
Best Practices

Big thank you to Qualys for the sponsorship of ssllabs.com

It has an API :-)

SSLLabs.com (slight irony..) is a wonderful tool.

- https://www.hardenize.com/ - similar and by the original author of ssllabs.com.
- https://securityheaders.io - focus on server-controlled headers



Best Practices

What about your stack?

We've focused on what a server can do...what about the client?

- Microsoft .NET Framework >= v4.5. Just use 4.6
- Java >= 7. Just use 8
- OpenSSL - that's tricky. >=v1.01

Browsers:

- IE >=v11
- Edge
- Chrome >=v30
- Firefox >= v27
- Safari >= 7

Operating Systems:

- Windows >= 7
- RHEL/CentOS >=6
- iOS >=5
- Android >= 5.x

Best Practices

What about your stack?

- Check your browser configuration @ssllabs.com:

SSL/TLS Capabilities of Your Browser

User Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36

Other User Agents »

Protocol Support

Your user agent has good protocol support.
Your user agent supports TLS 1.2, which is recommended protocol version at the moment.

- Check your servers/CDN/PaaS @istlsfastyet.com

	Session identifiers	Session tickets	OCSP stapling	Dynamic record sizing	ALPN	Forward secrecy	HTTP/2	TLS 1.3	TLS 1.3 0-RTT
Akamai	yes	yes	no	configurable (static)	yes	yes	yes	no	no
AWS ELB (Classic)	yes	yes	no	no	no	yes	no	no	no
AWS ELB (Application)	yes	yes	no	no	yes	yes	yes	no	no
AWS CloudFront	no	yes	yes	no	yes	yes	yes	no	no
BelugaCDN	yes	yes	yes	dynamic	yes	yes	yes	no	no
CDN77	yes	yes	yes	dynamic	yes	yes	yes	no	no
Cloudflare	yes	yes	yes	dynamic	yes	yes	yes	yes	yes

Best Practices

What about your stack?

Cipher Suites - TLS Server configuration guidance from Mozilla

Excellent all-in-one resource for operational teams.

Recommendations for Modern, Intermediate and Old classes of TLS

This is the Cliff's notes of TLS IMHO and includes a configuration generator: https://mozilla.github.io/server-side-tls/ssl-config-generator/

TLS 1.3

If it has to be fast, does that mean it has to be dangerous?

Significant upgrade from previous versions.

First major update since 2008.

Faster - handshake streamlined with opportunities for 0-RTT

- Resumption with data (from previous session) negating the need for full certificate exchange.
  - The server has been previously authenticated.
- Reduced latency

Fewer Choices = more secure

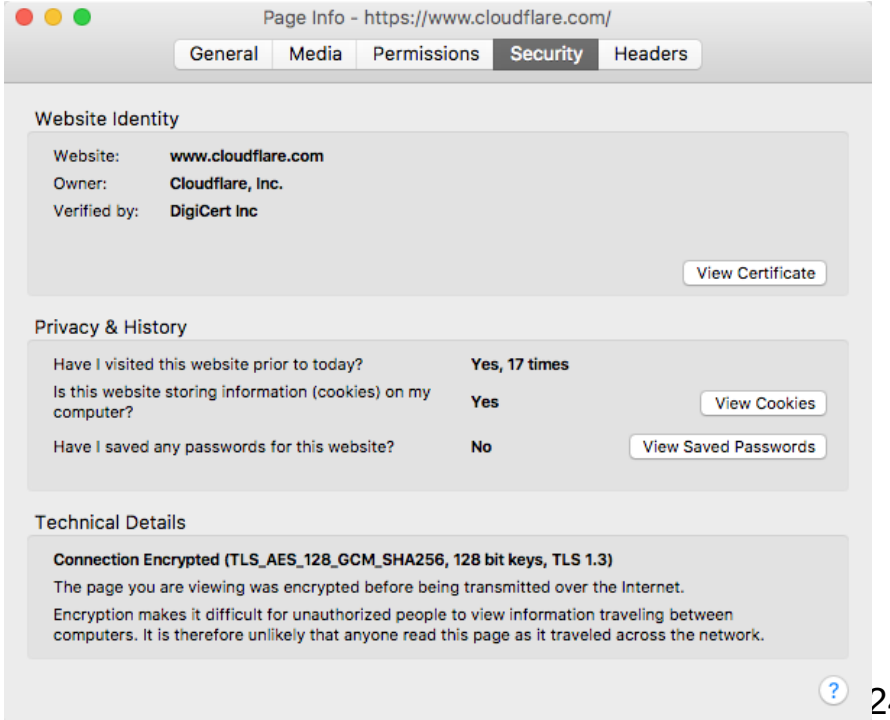
- No compression
- Removed features that are irresponsible (SHA1, MD5, RC4)
- AEAD Ciphers only
- Ephemeral key exchange = Forward Secrecy

TLS 1.3

Currently IETF Draft: https://tools.ietf.org/html/draft-ietf-tls-tls13-21

Supported in some browsers and some applications:

- Cloudflare
- Google Canary/Chrome
- Firefox



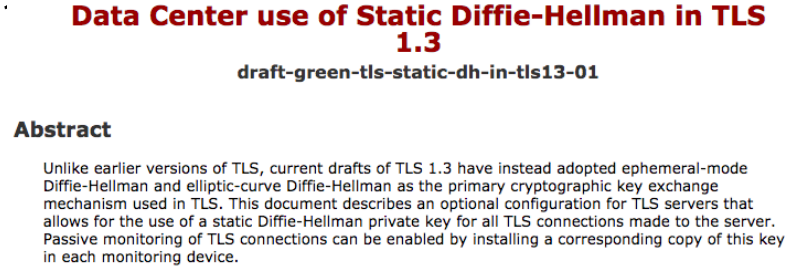
TLS 1.3

Forward Secrecy and MITM mitigation

Ephemeral key exchange provides a fantastic mitigation for MITM attacks.

- So fantastic, DLP and TLS decryption is broken.
- In Feb 2017, a school district in [1]Maryland enabled TLS 1.3 for Chromebooks (~120,000) and the Bluecoat appliances failed closed.

BITS (Financial Services Roundtable) became involved (late) in the IETF process, TLS 1.3 mode proposal for data center use. [2]Static Diffie-Hellman ke



Data Center use of Static Diffie-Hellman in TLS 1.3

draft-green-tls-static-dh-in-tls13-01

Abstract

Unlike earlier versions of TLS, current drafts of TLS 1.3 have instead adopted ephemeral-mode Diffie-Hellman and elliptic-curve Diffie-Hellman as the primary cryptographic key exchange mechanism used in TLS. This document describes an optional configuration for TLS servers that allows for the use of a static Diffie-Hellman private key for all TLS connections made to the server. Passive monitoring of TLS connections can be enabled by installing a corresponding copy of this key in each monitoring device.

[1] http://www.securityweek.com/tls-bug-blue-coat-proxy-breaks-chromebooks-pcs

Evaluation Criteria

As we purchase and consume services, consider the following:

- Does the application support and/or enforce Forward Secrecy?
- Does the application provide downgrade protection?
- Does the application enforce HTTP Strict Transport Security (HSTS)?
- Does the application support the use of AEAD Ciphers?
- Does the application use RSA >=2048 bit or ECC 256 bit keys?
- Does the application support secure renegotiation?
- Does the application support TLS >= 1.2?
- Does the application apply Content Security Policy (CSP)?*
- Does the application log TLS negotiations to assist customers?