TLS (Transport Layer Security)

Predecessor: SSL (Secure Socket Layer)

Most widely used protocol for encrypted data transmission

Same basic design, different crypto algorithms

Designed to provide secure communication over the insecure Internet

Authentication, confidentiality, and integrity

Used in many services and secure versions of protocols

HTTP, POP, IMAP, SMTP, OpenVPN, CalDAV, CardDAV, LDAP, NNTP, FTP, IRC, SIP, …

Separate port number: HTTPS: 443, FTPS: 990, IMAPS: 993, DoT: 853, …
**History**

SSL developed at Netscape

- **v1**: never released
- **v2 (1994)**: serious weaknesses
- **v3 (1995)**: re-design, basis of what we use today

**TLS working group was formed to migrate SSL to IETF**

- **TLS 1.0 (1999)**: minor differences but incompatible with SSL 3 (different crypto algorithms)
- **TLS 1.1 (2006)**: mostly security fixes, TLS extensions
- **TLS 1.2 (2008)**: authenticated encryption, more flexible
- **TLS 1.3 (2018)**: removal of legacy/weak algorithms, lower latency, perfect forward secrecy, …

**Endless cycle of vulnerabilities and improvements**

- Insecure renegotiation, RC4 weaknesses, compression side channels, padding oracle attacks, buggy implementations, PKI attacks, …

*BEAST, CRIME, TIME, Lucky 13, BREACH, POODLE, FREAK, Heartbleed, DROWN, …*
Handshake protocol
Negotiate public key crypto algorithms and establish shared secret keys
Authentication (server and optionally client)
Up to TLS 1.2, took 6–10 messages, depending on features used

Record Protocol
Uses the established secret keys to protect the transmitted data

*Message transport:* [header | data] records (16K)

*Encryption and integrity:* after handshake completion

*Compression:* before encryption… not a good idea
Side-channel attacks (e.g., CRIME)

*Subprotocols:* allow for extensibility
TLS defines four core subprotocols: *handshake, change cipher spec, application data, alert*
TLS 1.2 Handshake (Ephemeral DH)

1. ClientHello
2. ServerHello

3. Certificate
4. ServerKeyExchange
5. ServerHelloDone

3. ClientKeyExchange
4. ChangeCipherSpec
5. Finished

5. ChangeCipherSpec
6. Finished

GET /login HTTP/1.1

Server

Client
Cipher Suite Negotiation

ClientHello: *here are the cipher suites I support*

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
...  

ServerHello: *let’s use this one*

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

The server might not support the best of the client’s suites

Offers some other version hoping that the client will accept it
**Downgrade Attacks**

Force a weaker cipher suite selection through MitM

SSL 2: no handshake integrity
SSL 3: protocol rollback protection (still breakable)
TLS 1.0 and on: additional protections

Due to server bugs and interoperability issues, browsers responded by voluntarily downgrading the protocol upon handshake failure

Retrying connection with lower SSL/TLS version
Attackers can exploit this by blocking the initial handshake attempts, or alter the client’s list of supported suites
SSl 3.0, TLS 1.0, and TLS 1.1 are now completely removed by most browsers.

TLS 1.2 Session Resumption

Full handshake: 6-10 messages and two network round-trips
Along with CPU-intensive crypto operations, cert validation, …

Avoid re-negotiation by remembering security parameters
Server assigns and sends a unique *Session ID* as part of ServerHello
In future connections, the client sends the Session ID to resume the session

Alternative: *session tickets* (all state is kept at client)
TLS 1.3 Handshake (Ephemeral DH)

Latest draft supports even zero-RTT handshakes

Clients include encrypted data in the initial messages based on config. ID previously sent by server
Server (and Client) Authentication

After handshake completion, the client knows it can “trust” the information in the server’s certificate

Assuming it trusts the issuing certificate authority

SSL/TLS certs are based on the X.509 PKI standard

How is the certificate associated with the server?

Common Name (CN): server’s hostname

The same process is supported for authenticating clients

Highly-secure web services, some VPN services, cloud applications, …

Rarely used in practice for user authentication

Common alternative: username + password over TLS connection
Certificate Fields

Version: v1 (basic), v2 (additional fields), v3 (extensions)

Serial Number: high-entropy integer

Signature Algorithm: encryption and hash algorithm used to sign the cert

Issuer: contains the distinguished name (DN) of the certificate issuer

Validity: starting and ending date of validity period

Subject: DN of the entity associated with the certificate’s public key
  Deprecated in favor of the Subject Alternative Name (SAN) extension: DNS name, IP address, or URI (also supports binding to multiple identities)

Public Key: The subject’s public key

Signature
Certificate Information

This certificate is intended for the following purpose(s):
- Proves your identity to a remote computer
- Ensures the identity of a remote computer
- 2.23.140.1.2.1
- 1.3.6.1.4.1.44497.1.1.1

* Refer to the certification authority’s statement for details.

**Issued to:** www-prod.cs.stonybrook.edu

**Issued by:** R3

**Valid from** 1/26/2021 to 4/26/2021
### Certificate Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>V3</td>
</tr>
<tr>
<td>Serial number</td>
<td>047d50251aae74fe23c0e6ea452ed7f...</td>
</tr>
<tr>
<td>Signature algorithm</td>
<td>sha256RSA</td>
</tr>
<tr>
<td>Signature hash algorithm</td>
<td>sha256</td>
</tr>
<tr>
<td>Issuer</td>
<td>R3, Let's Encrypt, US</td>
</tr>
<tr>
<td>Valid from</td>
<td>Tuesday, January 26, 2021 07:02:10</td>
</tr>
<tr>
<td>Valid to</td>
<td>Monday, April 26, 2021 07:02:10</td>
</tr>
<tr>
<td>CN</td>
<td>R3</td>
</tr>
<tr>
<td>O</td>
<td>Let's Encrypt</td>
</tr>
<tr>
<td>C</td>
<td>US</td>
</tr>
</tbody>
</table>

---

The certificate issued by R3, Let's Encrypt, US, valid from Tuesday, January 26, 2021 07:02:10 to Monday, April 26, 2021 07:02:10, includes the following details:

- **Version**: V3
- **Serial number**: 047d50251aae74fe23c0e6ea452ed7f...
- **Signature algorithm**: sha256RSA
- **Signature hash algorithm**: sha256
- **Issuer**: R3, Let's Encrypt, US
- **CN**: R3
- **O**: Let's Encrypt
- **C**: US

---

The certificate can be edited or copied to a file by clicking the respective buttons at the bottom of the certificate window.

---

The certificate is trusted by the user, as indicated by the green check mark.
Certificate

General Details Certification Path

Show: <all> <all> <all>

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public key parameters</td>
<td>05 00</td>
</tr>
<tr>
<td>Enhanced Key Usage</td>
<td>Server Authentication (1.3.6.1.5.5...)</td>
</tr>
<tr>
<td>Subject Key Identifier</td>
<td>d1972f66c91f731b3f739956c6a80394702a496a64</td>
</tr>
<tr>
<td>Authority Key Identifier</td>
<td>KeyID = 1f42b317b759956c6a80394702a496a64</td>
</tr>
<tr>
<td>Authority Information ...</td>
<td>Authority Info Access: Access Me...</td>
</tr>
<tr>
<td>Subject:Alternative Name</td>
<td>DNS Name=www.prod.cs.stonybrook.edu</td>
</tr>
<tr>
<td>OITJIT</td>
<td>v1: 6a83f7f47770077194810399</td>
</tr>
</tbody>
</table>

DNS Name=www.prod.cs.stonybrook.edu
DNS Name=www.cs.stonybrook.edu
DNS Name=www.cs.sunysb.edu

[OK]
Certificate Chains

Trust anchors: systems are pre-configured with ~200 trusted root certificates

- System/public store: used by OS, browsers, …
- More can be added in the local/private cert store: vendor-specific certs, MitM certs for content inspection filters/AVs, …

Server provides a chain of certificates

- A certificate from an intermediate CA is trusted if there is a valid chain of trust all the way back to a trusted root CA

Any CA can issue and sign certificates for any subject

- The system is only as secure as the weakest certificate authority…

Certificate Authority Authorization (CAA): can be used to restrict which CAs can issue certificates for a particular domain
Certificate Revocation

Allow revocation of compromised or no longer needed certificates

Certificate revocation list (CRL)

Signed list of all revoked certificates that have not yet expired
Main problem: lists tend to be large, making real-time lookups slow
Can the attacker block connectivity to the CA’s server?

CRLSets (Chrome): revocation list pushed to the browser as a *software update*

Online Certificate Status Protocol (OCSP)

Obtain the revocation status of a *single* certificate ➔ faster
But the latency, security, and privacy issues still remain

OCSP stapling (Firefox): server embeds OCSP response directly into the TLS handshake (soft-fail issue remains: an adversary can suppress the OCSP response)
HTTPS

Most common use of TLS: *most web traffic is now encrypted*

Crypto is expensive, needs more CPU cycles
  - **Not** a big deal these days (native hardware support)

Mixed content: Ad networks, mashups, …
  - **Stop** using them! (easier said than done: lost revenue, increased development time)
  - Incentives: Google rewards HTTPS sites with higher ranking

Virtual Hosting: initially incompatible
  - **Not** anymore: solved as of TLS 1.1 through the *Server Name Indication (SNI)* extension

Needs expertise and certs cost $$$$ 
  - **Not** anymore: [letsencrypt.org](https://letsencrypt.org)
Firesheep In Wolves' Clothing: Extension Lets You Hack Into Twitter, Facebook Accounts Easily

It seems like every time Facebook amends its privacy policy, the web is up in arms. The truth is, Facebook's well publicized privacy fight is nothing compared to the vulnerability of all unsecured HTTP sites — that includes Facebook, Twitter and many of the web's most popular destinations.

Developer Eric Butler has exposed the soft underbelly of the web with his new Firefox extension, Firesheep, which will let you essentially eavesdrop on any open Wi-Fi network and capture users’ cookies.

As Butler explains in his post, “As soon as anyone on the network visits an insecure website known to Firesheep, their name and photo will be displayed” in the window. All you have to do is double click on
Browser Security Indicators

Convey information about the security of a page

Locks, shields, keys, green bars…

“This page was fetched using SSL”

Page content was not viewed or altered by a network adversary
Certificate is valid (e.g. not expired), issued by a CA trusted by the browser, and the subject name matches the URL’s domain

“This page uses an invalid certificate”

Parts of the page are not encrypted”

“The legal entity operating this web site is known”

Extended Validation (EV) certificates
Browser Security Indicators

Convey information about the security of a page

Locks, shields, keys, green bars...

“This page was fetched using SSL”
- Page content was not viewed or altered by a network adversary
- Certificate is valid (e.g. not expired, issued by a CA trusted by the browser, and the subject name matches the URL’s domain)

“This page uses an invalid certificate”

“Parts of the page are not encrypted”

“The legal entity operating this website is known”

Extended Validation (EV) certificates
Mixed Content Warning is Unnecessary

<table>
<thead>
<tr>
<th>Chrome 45</th>
<th>Chrome 46</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure HTTPS</td>
<td>Secure HTTPS</td>
</tr>
<tr>
<td><a href="https://www.google.com">https://www.google.com</a></td>
<td><a href="https://www.google.com">https://www.google.com</a></td>
</tr>
<tr>
<td>HTTP</td>
<td>HTTP</td>
</tr>
<tr>
<td><a href="http://www.example.com">www.example.com</a></td>
<td><a href="http://www.example.com">www.example.com</a></td>
</tr>
<tr>
<td>HTTPS with minor errors</td>
<td>HTTPS with minor errors</td>
</tr>
<tr>
<td>Broken HTTPS</td>
<td>Broken HTTPS</td>
</tr>
</tbody>
</table>

Basically the same in terms of security

Fewer security states for users to remember

Reflects better the security state of the page

Non-HTTPS traffic is a vulnerability! MitM/MotS attacks on the HTTP part are trivial

https://googleonlinesecurity.blogspot.com/2015/10/simplifying-page-security-icon-in-chrome.html
Marking HTTP as Not Secure

Phase 1: page is marked “Not secure” when
- The page contains a password field
- The user interacts with a credit card field

Treatment of HTTP pages with password or credit card form fields:

- Current (Chrome 53): login.example.com
- Jan. 2017 (Chrome 56): Not secure, login.example.com

Marking HTTP as Not Secure

Phase 2: page is marked “Not secure” when

- The page contains a password field
- The user interacts with *any* input field
- The user is browsing in *incognito mode*

Marking HTTP as Not Secure

Phase 3: all plain HTTP pages are marked “Not secure”

Marking HTTP as Not Secure

Current state: HTTPS pages are marked in a more neutral way, while HTTP pages are affirmatively marked “Not secure”
The information you’re about to submit is not secure

Because this form is being submitted using a connection that’s not secure, your information will be visible to others.

Send anyway  Go back
SSL stripping

Browsing sessions often start with a plain HTTP request

Web sites used to switch to HTTPS only for login or checkout

Example: Facebook in 2010 (optional full HTTPS in 2011, HTTPS by default in 2013)

Users type addresses without specifying https://

Browser connects over HTTP by default ➔ site may redirect to HTTPS

SSLstrip [Moxie Marlinspike, Black Hat DC 2009]

MitM attack to prevent redirection to HTTPS

Watch for HTTPS redirects and links, and map them to HTTP links

…or homograph-similar valid HTTPS links:

https://www.bank.com.attacker.com
SSL stripping

Victim — HTTP — Attacker

http://…

<form action="http://…">

Attacker — HTTPS — Server

https://…

<form action="https://…">

Location: http://…

Location: https://…

Missing lock icon or different domain, but who is going to notice?
**HSTS** (HTTP Strict Transport Security)

Defense against SSL stripping and other similar issues

- Force the use of HTTPS instead of HTTP before accessing a resource
- Treat all errors (e.g., invalid certificate, mixed content, plain HTTP) as fatal: do not allow users to access the web page

Servers implement HSTS policies by supplying an extra HTTP header

```
Strict-Transport-Security: max-age=31536000
```

“Use only HTTPS for future requests to this domain for the next year”

An instance of trust on first use (TOFU)

- Problem: the initial request remains unprotected because it is sent over HTTP
- **HSTS preloading:** browsers come preloaded with a list of known HSTS sites
transport_security_state_static.json

    // START OF LEGACY MANUAL CUSTOM ENTRIES

    780  { "name": "www.paypal.com", "policy": "custom", "mode": "force-https" },
    781  { "name": "www.yahoo.com", "policy": "custom", "mode": "force-https" },
    782  { "name": "www.cnn.com", "policy": "custom", "mode": "force-https" },
    783  { "name": "www.cnet.com", "policy": "custom", "mode": "force-https" },
    784  { "name": "www.elle.com", "policy": "custom", "mode": "force-https" },
    785  { "name": "www.linux.org", "policy": "custom", "mode": "force-https" },
    786  { "name": "www.ubuntu.com", "policy": "custom", "mode": "force-https" },
    787  { "name": "www.apache.org", "policy": "custom", "mode": "force-https" },
    788  { "name": "www.mozilla.org", "policy": "custom", "mode": "force-https" },
    789  { "name": "www.gmail.com", "policy": "custom", "mode": "force-https" },
    790  { "name": "www.facebook.com", "policy": "custom", "mode": "force-https" },
    791  { "name": "www.twitter.com", "policy": "custom", "mode": "force-https" },
    792  { "name": "www.instagram.com", "policy": "custom", "mode": "force-https" },
    793  { "name": "www.linkedin.com", "policy": "custom", "mode": "force-https" },
    794  { "name": "www.youtube.com", "policy": "custom", "mode": "force-https" },
    795  { "name": "www.reddit.com", "policy": "custom", "mode": "force-https" },
    796  { "name": "www.dropbox.com", "policy": "custom", "mode": "force-https" },
    797  { "name": "www.dropcam.com", "policy": "custom", "mode": "force-https" },
    798  { "name": "www.snapchat.com", "policy": "custom", "mode": "force-https" },
    799  { "name": "www.pivotal.io", "policy": "custom", "mode": "force-https" },
   800  { "name": "www.torproject.org", "policy": "custom", "mode": "force-https", "include_subdomains": true, "pinned": "tor" },
   801  { "name": "check.torproject.org", "policy": "custom", "mode": "force-https", "include_subdomains": true, "pinned": "tor" },
   802  { "name": "dist.torproject.org", "policy": "custom", "mode": "force-https", "include_subdomains": true, "pinned": "tor" },
   803  { "name": "ledgerscope.net", "policy": "custom", "mode": "force-https", "include_subdomains": true, "pinned": "tor" },
   804  { "name": "www.ledgerscope.net", "policy": "custom", "mode": "force-https", "include_subdomains": true, "pinned": "tor" },
   805  { "name": "www.greplin.com", "policy": "custom", "mode": "force-https" },
   806  { "name": "www.greplin.com", "policy": "custom", "mode": "force-https" },
   807  { "name": "www.greplin.com", "policy": "custom", "mode": "force-https" },

Firefox 83 introduces HTTPS-Only Mode

Christoph Kerschbaumer, Julian Gaibler, Arthur Edelstein and Thyla van der Merwe | November 17, 2020

Security on the web matters. Whenever you connect to a web page and enter a password, a credit card number, or other sensitive information, you want to be sure that this information is kept secure. Whether you are writing a personal email or reading a page on a medical condition, you don’t want that information leaked to eavesdroppers on the network who have no business prying into your personal communications.

That’s why Mozilla is pleased to introduce HTTPS-Only Mode, a brand-new security feature available in Firefox 83. When you enable HTTPS-Only Mode:

- Firefox attempts to establish fully secure connections to every website, and
- Firefox asks for your permission before connecting to a website that doesn’t support secure connections.

How HTTPS-Only Mode works

The Hypertext Transfer Protocol (HTTP) is a fundamental protocol through which web browsers and websites communicate. However, data transferred by the regular HTTP protocol is unprotected and transferred in cleartext, such that attackers are able to view, steal, or even tamper with the transmitted data. Hypertext Transfer Protocol over TLS (HTTPS) fixes this security shortcoming by creating a secure and encrypted...
HTTPS-Only Mode Alert

Secure Connection Not Available

You’ve enabled HTTPS-Only Mode for enhanced security, and a HTTPS version of example.com is not available.

Learn More...

What could be causing this?

- Most likely, the website simply does not support HTTPS.
- It’s also possible that an attacker is involved. If you decide to visit the website, you should not enter any sensitive information like passwords, emails, or credit card details.

If you continue, HTTPS-Only Mode will be turned off temporarily for this site.

Continue to HTTP Site  Go Back
Omnibox - Use HTTPS as the default protocol for navigations

Use HTTPS as the default protocol when the user types a URL without a protocol in the omnibox such as `example.com`. Presently, such an entry navigates to http://example.com. When this feature is enabled, it will navigate to https://example.com if the HTTPS URL is available. If Chrome can't determine the availability of the HTTPS URL within the timeout, it will fall back to the HTTP URL. – Mac, Windows, Linux, Chrome OS, Android

#omnibox-default-typed-navigations-to-https

Omnibox UI Sometimes Hide Steady-State URL Subdomains Beyond Registrable Domain

In the omnibox, occasionally hide subdomains as well as path, query and ref from steady state displayed URLs, depending on heuristics. Has no effect unless at least one of #omnibox-ui-reveal-steady-state-url-path-query-and-ref-on-hover or #omnibox-ui-hide-steady-state-url-path-query-and-ref-on-interaction is enabled. – Mac, Windows, Linux, Chrome OS

#omnibox-ui-sometimes-elide-to-registrable-domain

Omnibox UI Reveal Steady-State URL Path, Query, and Ref On Hover
A safer default for navigation: HTTPS
Tuesday, March 23, 2021

Starting in version 90, Chrome's address bar will use https:// by default, improving privacy and even loading speed for users visiting websites that support HTTPS. Chrome users who navigate to websites by manually typing a URL often don’t include “http://” or “https://”. For example, users often type “example.com” instead of “https://example.com” in the address bar. In this case, if it was a user’s first visit to a website, Chrome would previously choose http:// as the default protocol\(^1\). This was a practical default in the past, when much of the web did not support HTTPS.

Chrome will now default to HTTPS for most typed navigations that don’t specify a protocol\(^2\). HTTPS is the more secure and most widely used scheme in Chrome on all major platforms. In addition to being a clear security and privacy improvement, this change also better integrates Chrome with the modern web, which predominantly uses HTTPS.
MitM is Still Possible…

Rogue certificates
Most governments have a trusted root CA planted in our systems
Attackers may break into CAs and forge certificates

Pre-planted/generated certificates
Default static keys: Lenovo, Dell, anti-malware software, …
Low entropy during key generation: repeated or factorable keys

Self-signed certificates
If desperate… will trigger scary browser warning

Exploitation of certificate validation flaws
Programming errors while checking date, hostname, …
StartSSL suspends services after security breach

StartSSL has suspended issuance of digital certificates and related services following a security breach on 15 June. A trademark of Eddy Nigg’s StartCom, the StartSSL certificate authority is well known for offering free domain validated SSL certificates, but also sells organisation and extended validation certificates.

More than 25 thousand websites in Netcraft’s SSL survey use certificates issued by StartSSL. These are recognised by Internet Explorer, Firefox, Chrome and other mainstream browsers.

StartSSL is not alone in offering free certificates. AffirmTrust recently trumped StartSSL’s one-year certificates with its own offer of free three-year domain validated SSL certificates. Coincidently, AffirmTrust announced its launch on the same day as the StartSSL security breach.

StartSSL is also not the only certificate authority to come under attack this year. In March, Comodo came under attack through three of its resellers. By compromising a GlobalTrust website, the so-called ComodoHacker managed to fraudulently issue several valid certificates, including ones for the login pages of Yahoo and Skype. These certificates were subsequently revoked and browser software was updated to explicitly
Comodo hacker: I hacked DigiNotar too; other CAs breached
The hacker behind this year’s Comodo hack has claimed responsibility for the ...

by Peter Bright - Sep 6, 2011 5:30pm EDT

Hack mode is over!
My Office is your office!

Photograph by Augie Schvar
Trustwave to escape 'death penalty' for SSL skeleton key

Moz likely to spare certificate-confession biz same fate as DigiNotar

14 Feb 2012 at 09:28, John Leyden

Analysis Trustwave's admission that it issued a digital "skeleton key" that allowed an unnamed private biz to spy on SSL-encrypted connections within its corporate network has sparked a fiery debate about trust on the internet.

Trustwave, an SSL certificate authority, confessed to supplying a subordinate root certificate as part of an information security product that allowed a customer to monitor employees' web communications - even if the staffers relied on HTTPS. Trustwave said the man-in-the-middle (MITM) gear was designed both to be tamper-proof and to work only within its unnamed client's compound. Despite these precautions, Trustwave now admits that the whole approach was misconceived and would not be repeated. In addition, it revoked the offending certificate.

Trustwave came clean without the need for pressure beforehand. Even so its action have split security experts and prompted calls on Mozilla's Bugzilla security list to remove the Trustwave root certificate.
Self-signed Certificate Warning: One click away…
Self-signed Certificate Warning: Two clicks away…
Self-signed Certificate Warning: Two clicks away…

Your connection is not private

Attacks might be trying to steal your information from self-signed.badssl.com (for example, passwords, messages, or credit cards).

This server could not prove that it is self-signed.badssl.com; its security certificate is not trusted by your computer’s operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

Proceed to self-signed.badssl.com (unsafe)
iOS 7.0.6 signature verification error

Legitimate-looking TLS certificates with a mismatched private keys were unconditionally accepted...

```c
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
...
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
```

Check never executed
**HPKP** (HTTP Public Key Pinning)

Prevent *certificate forgery*: strong form of web site authentication

Browser knows the *valid* public keys of a particular website

   If a seemingly valid chain does not include at least one known pinned key, the cert is rejected
   Doesn’t apply for *private* root certificates (would break preconfigured proxies, anti-malware, content filters, …)

Many incidents involving rogue certificates were discovered after browsers started rolling out pinning

Similar deployment as HSTS

   TOFU: HTTP response header
   Built-in pins in browsers

**Must be used very carefully – things can go wrong**

   HPKP suicide: site can be bricked if keys are lost/stolen
   RansomPKP: compromise the server and push a malicious HPKP key
**HPKP** (HTTP Public Key Pinning)

Prevent *certificate forgery*: strong form of web site authentication

Browser knows the valid public keys of a particular website

If a seemingly valid certificate does not include at least one known pinned key, the cert is rejected.

Doesn’t apply for private root certificates (which impact preconfigured proxies, anti-malware, content filters, …)

Many incidents involving rogue certificates were discovered after browsers started rolling out pinning.

Similar deployment as HSTS

TOFU: HTTP response header

Built-in pins in browsers

Must be used very carefully – things can go wrong

HPKP suicide: site can be bricked if keys are lost/stolen

RansomPKP: compromise the server and push a malicious HPKP key

---

*Deprecated in favor of Certificate Transparency and the Expect-CT header*
Enhancing digital certificate security

January 3, 2013

Posted by Adam Langley, Software Engineer

Late on December 24, Chrome detected and blocked an unauthorized digital certificate for the "*.google.com" domain. We investigated immediately and found the certificate was issued by an intermediate certificate authority (CA) linking back to TURKTRUST, a Turkish certificate authority. Intermediate CA certificates carry the full authority of the CA, so anyone who has one can use it to create a certificate for any website they wish to impersonate.

In response, we updated Chrome’s certificate revocation metadata on
Certificate Transparency

Public monitoring and auditing of certificates

Identify mistakenly or maliciously issued certificates and rogue CAs

Certificate logs

Network services maintaining cryptographically assured, publicly auditable, append-only records of certificates

Monitors

Periodically contact all log servers and watch for suspicious certificates

Auditors

Verify that logs are behaving correctly and are cryptographically consistent

Check that a particular certificate appears in a log

https://certificate.transparency.dev/
HSTS/PKP

HSTS is HTTP Strict Transport Security, a way for sites to elect to always use HTTPS. See [https://www.chromium.org/hsts](https://www.chromium.org/hsts). PKP is Public Key Pinning: Chrome "pins" certain public keys for certain sites in official builds.

Add HSTS domain

Input a domain name to add it to the HSTS set:

- Domain: example.com
- Include subdomains for STS: [ ]
- Add

Query HSTS/PKP domain

Input a domain name to query the current HSTS/PKP set:

- Domain: stonybrook.edu
- Query

- Not found

Expect-CT


To protect against cross-site tracking, Expect-CT data will soon be keyed on the site of the main frame and innermost frame when an Expect-CT header is encountered. When that behavior is enabled, both adding and querying an Expect-CT domain use the cTLD+1 of the provided domain as the site for both frames. Deleting policies affects information stored for that domain in the context of all sites, however.

Add Expect-CT domain

Input a domain name to add it to the Expect-CT set. Leave Enforce unchecked to configure Expect-CT in report-only mode.

- Domain: example.com
- Report URI (optional): https://reporting.example.com
- Enforce: [ ]
- Add
HSTS/PKP

HSTS is HTTP Strict Transport Security: a way for sites to elect to always use HTTPS. See https://www.chromium.org/hsts. PKP is Public Key Pinning: Chrome "pins" certain public keys for certain sites in official builds.

Add HSTS domain

Input a domain name to add it to the HSTS set:

- Domain: [example.com]
- Include subdomains for STS: [ ]
  - Add

Query HSTS/PKP domain

Input a domain name to query the current HSTS/PKP set:

- Domain: [google.com]

  Found:
  - static_sts_domain: [value]
  - static_upgrade_mode: [value]
  - static_sts_include_subdomains: [value]
  - static_sts_observed: [value]
  - static_pkp_domain: [value]
  - static_pkp_include_subdomains: [value]
  - static_pkp_observed: [value]
  - static_spli_hashes: [value]
  - dynamic_sts_domain: [value]
  - dynamic_upgrade_mode: [value]
  - dynamic_sts_include_subdomains: [value]
  - dynamic_sts_observed: [value]
  - dynamic_sts_expiry: [value]

Expect-CT