HTTPS Demystified!

What is that green lock and why is it so important?

What is that green lock and why is it important?

Security

It protects* the information you send across the Internet.

Integrity

It guarantees* that you are talking to who you think you are.

* mostly

Agenda

- Overview of HTTP
- Overview of HTTPS
- Keywords
- 3 things to make it work
- Where to get a Cert
- Demo
- Conclusion

HTTPS wraps HTTP

It wraps and scrambles HTTP with complex math and big prime numbers.

Invented by wizards.

Remind me about HTTP?

It’s a way to move files from one place to another.

Let’s do it by pretending to be a web browser. ⇒

HTTP Demo

Open a console window: Powershell, Terminal, DOS, Bash, pick your poison.

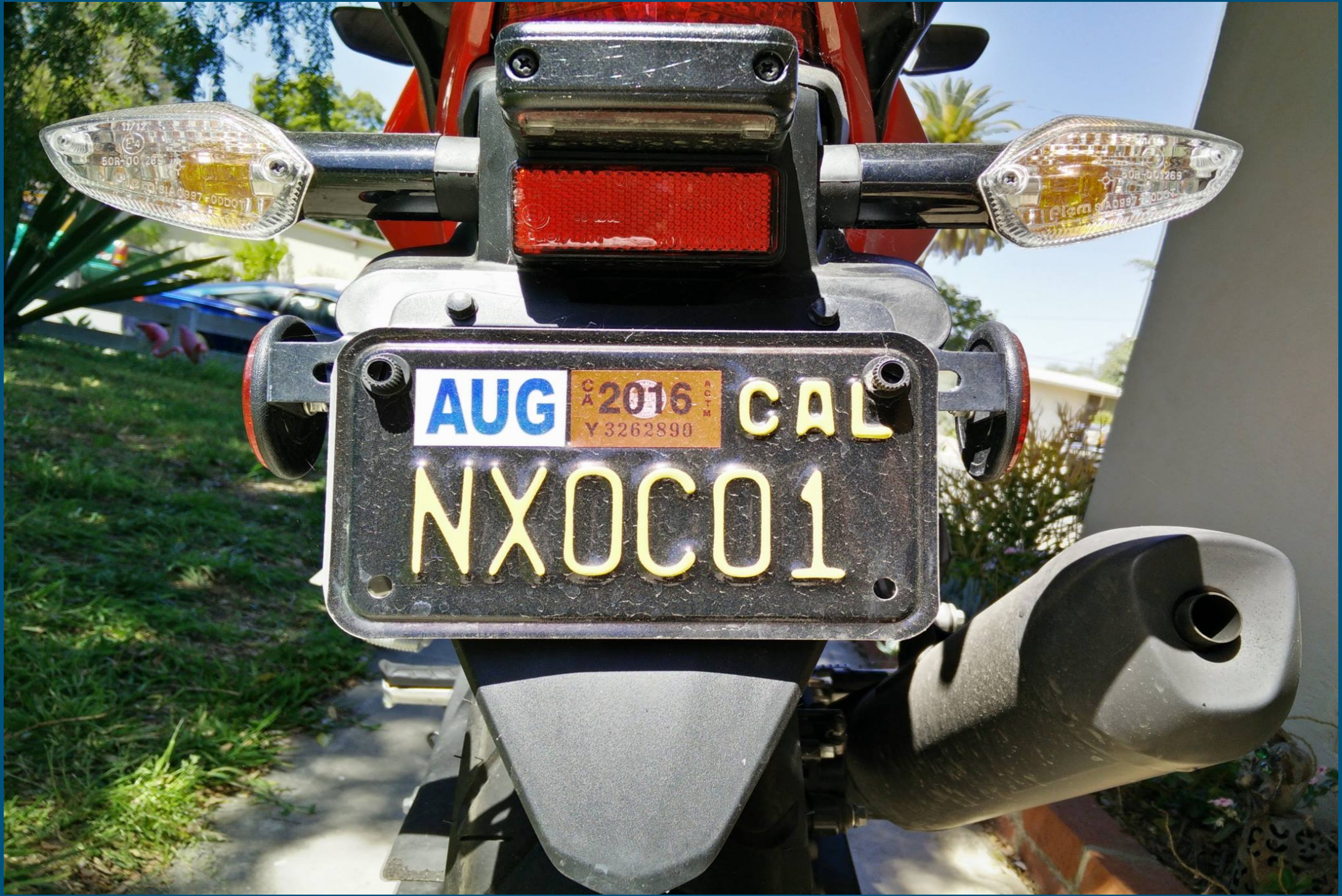
```
telnet www.ucsb.edu 80 {enter}
```

```
host: www.ucsb.edu {enter}
```

```
{enter again}
```

OMG, HTML, it’s a web page. I love HTML!

California
OMNOMNM
California
I ♥ HTML


AUG
CA 2016 ACTM
Y 3262890
CAL
NXOC01

Back to HTTPS

It encrypts (fancy word for scrambles) anything sent between your web browser and a web server.

It is scrambled in a manner that can only be unscrambled by the intended consumer of that information. This is called asymmetric (or public-key) cryptography and it is how all secure traffic on the web is handled.

Fun fact: Hashing as opposed to encryption cannot be decrypted (unscrambled). It’s good for password storage, you just hash incoming password and compare.

SSL and TLS

Secure Socket Layer

Version 1, 2 and 3. All broken, don’t use these.

Transport Layer Security, same thing, different name.

1.0 and 1.1 broken, don’t use these either.

TLS 1.2 is currently the only non-broken protocol.

TLS 1.3 being worked on now.

Some Key Terms

Private Key - Really just a password. This can unlock everything sent to you. Keep it safe.

Public Key - What you give out to others so they can send stuff that can only be opened by you.

Root Certificate - A self-signed cert that is trusted by web browsers. Take a look.

Intermediate Certificate - Certs used to sign your cert. They are signed by a Root.

Chain - All the Certs from a root to your Cert. Usually needed to make it all work.

Some key words and terms

Signed - A Process to put a stamp on a Cert saying it is trusted.

Self-Signed - You can sign a Cert you make yourself. You trust yourself, right?

Certificate Authority - An organization that signs Certs and gets their Root Certs installed and trusted by web browsers.

Algorithms and Ciphers - The math and methods behind all this magic. They have names like AES, DES, RSA, Diffie Hellman, Elliptic Curve. See a cool list here https://www.openssl.org/docs/manmaster/apps/ciphers.html

X.509

Standards! I love them, there are lots to choose from,

TL;DR: https://en.wikipedia.org/wiki/X.509

X.500 is LDAP. The format of certificates is related to LDAP.

CN or Common Name is the most important field. There are others you must supply like Region, Company, etc. These are in X.500 format. You will see more in the Demo.

Three Things To Make It All Work!

1. CN must match the domain name the browser is accessing exactly. There are wildcard certs and other methods but generally a cert issued to exmaple.com will show an error if server they ww.example.com.

2. The visitor’s computer’s date must be within the validity dates of the cert. The cert must not be expired. Update your certs months before they expire.

3. The visitor’s browser must trust the cert itself or an intermediate or root that signed the cert. Look at your browser’s Root cert list.

1. CN (Common Name)

When you create a cert you must do CN=example.com

Generally you need to decide on a single domain to serve HTTPS from and set that to the CN.

There are other things out there like wildcards and SAN that allow for multiple domains. You have options, I just never explored much into them.

https://en.wikipedia.org/wiki/SubjectAltName
https://en.wikipedia.org/wiki/Wildcard_certificate

2. Valid Date

Certificates have dates. There is an issue date and an expiration date.

This is done for two reasons. 1) to make more money. 2) if a certificate is stolen it will only be good for set amount of time.

Certs expiring is a problem when running websites. People forget to renew since this only happens once a year or so. Keep the expiration dates in the calendars of multiple people. I used to set a calendar event and send it to half of the operations team. That way if you leave, someone else will know about it.

If you create an internal certificate set it to something like 10 years (3650 days).

3. Trust

The Web of Trust https://en.wikipedia.org/wiki/Web_of_trust

<rave>I love computers, each subject seems to go in infinite directions.</rave>

Cert Authorities pay browser makers to install their Root and Intermediate certs. Then any cert, like yours, that is signed by these CAs will be trusted by all those browsers.

If you get your cert signed by a CA you should be OK. Self-signed certs are not trusted, of course, unless you go to the computer and tell it to trust it and even that does not seem to work sometimes.

EV (Extended Validation) Certs

They give you a bigger green lock.

I think is is a ploy to make more money.

But the CAs do actually check into the companies. When I worked at a bank, I convinced them to get EV certs. We had to jump thru some legal processes to make sure we were who we said we were.

Let’s go examine some. Any suggestions?

No Cost UCSB Certs

We can get a cert for any ucsb.edu domain through ETS at no cost to you.

http://www.ets.ucsb.edu/services/ssl-certificates

Let’s examine one: https://www.learningcenter.ucsb.edu/

Click the lock and look at the certificate properties.

Let’s Encrypt

Certs have always cost money, but recently a free service has launched.

https://letsencrypt.org/

First fully free real certificates.

A bit tricky to use.

Client application on your server creates and renews cert.

I have personally used it since the closed beta was launched.

Other Certificate Authorities

A zillion Certificate Authorities out there.

Costs vary greatly, but really it’s all the same.

I have used Network Solutions, Verisign, and GoDaddy. All worked just fine.

As far as I can tell there are 2 types: regular and EV. But look at features like wildcard certs that might help your business model.

But really if ucsb.edu get a cert thru ETS, if personal go Let’s Encrypt.

Creating a Certificate Signing Request

Make a CSR to end to the CA. This is a file with the info that a CA will sign to create a public certificate which they will send to you. This is all safe because those things are public info. You always keep the private key safe.

Each system will have its own method.
Microsoft IIS - Tool inside IIS
Apache HTTP and Nginx - OpenSSL
Java - Java Keytool

File Formats and File Extensions

PEM - Text, base64 encoded
DER, CER, CRT
PFX - PKCS
Look in text editor or rename .cer and open on Windows.

Use OpenSSL to examine.

I don't know what they all are, all I know is some systems need certain kinds and you can use OpenSSL to convert. Google it.

Creating a Cert in OpenSSL

Make a cert with openssl

Make Private Key, keep this safe!

```
openssl genrsa -out example.com.key 2048
```

Make Request

```
openssl req -new -sha256 -key
example.com.key -out example.csr
```

You will get prompted for values

US 2 digit country code

Full name of State

City

Org name, maybe full ucsb

unit, anything

Common Name, must match exactly with what your site will be visited. www.example.com and example.com are different. If you want both make 2 certs to look into wildcard or SAN certs.

Emails, password, etc optional

Creating a Cert in OpenSSL Continued

Review the Cert:

```
openssl req -noout -text -on example.csr
```

Make sure the subject is correct

Send this CSR file off to the CA. Never give out the private key.

The CA will sign it, and give you back the public certificate.

Install that cert on your server and you are good to go.

Test it!

Visit the site in your web browser.

Self-Signed Certificate

Self-signed can be used for testing.
For test systems I recommend getting a real certificate.
But for a quick and dirty example we can make a self-signed cert that will operate the same way except it will not be trusted.

```
openssl req -new -x509 -sha256 -newkey rsa:2048 -nodes -
keyout example.key -out example.crt -days 365
```

Answer questions when prompted, watch out for typos.
Review the cert.

Testing HTTPS Connection

Using openssl_client.

```
openssl s_client -connect www.google.com:443
```

You can see the “handshake”.

Now you can enter HTTP commands.

This is exactly how your web browser does it.

Testing HTTPS Vulnerabilities

https://www.ssllabs.com/

Free tool by Qualys, a security company.

Test your browser and test public websites.

Give a few sites a try.

Final Thoughts

HTTPS Everywhere

In recent years there have been the movement to use HTTPS by default.

HTTPS is now fast, cheap and fairly easy.

NO reason for anyone to snoop on your conversation with a 3rd party, even if there is not confidential information.

Google and Facebook are doing it. I’m doing it too.

Thank you

Gary Scott

gscott@arit.ucsb.edu / gary.scott@ucsb.edu

gary@garyscott.net

Let’s talk later!

https://garster.github.io/p/https/