TLS/SSL and Certificates

Brian Dickinson

University of Rochester

Fall 2019
Securing the Internet

HTTP is fully functional as an application interface, but lacks
- Authentication
- Privacy
- Integrity

without which the modern internet could not function.
HTTPS | Authentication

Servers must provide proof that they are who they say.

1. Certificate authorities issue certificates upon proof of controlling a domain
2. Web Browsers have pre-defined lists of trusted certificate authorities
3. Modern web browsers warn users when the server they are talking to does not provide a certificate for its domain signed by a trusted certificate authority

This proves to clients that they are talking to the server identified in their URL and not an malicious imposter.
Under HTTPS the contents of a client’s requests and the server’s responses are encrypted.

- Protects username and password submission from servers relaying request
- Protects user information such as banking records in responses
- Underpins the ability to make purchases online
Encrypting request and response bodies also protects from man-in-the-middle attacks.
HTTP over TLS or SSL

- SSL Protocol version
- Session ID
- List of Cipher Suites
- CLIENT HELLO Extensions

- Pre-master secret
- Client Cert (If requested)

- Master secret is used to generate Symmetric session keys

- Server Authenticate the Client (Optional)
Key Exchange

Alice
- Bob's Public Key
- Alice's Private Key

Combine keys
- 751A696C24D97009

Alice and Bob's shared secret

Bob
- Alice's Public Key
- Bob's Private Key

Combine keys
- 751A696C24D97009

Alice and Bob's shared secret
Diffie-Hellman Key Exchange

1. Alice and Bob publicly agree to use a modulus $p = 23$ and base $g = 5$ (which is a primitive root modulo 23).\(^1\)

2. Alice chooses a secret integer $a = 4$, then sends Bob $A = g^a \mod p = 4$

3. Bob chooses a secret integer $b = 3$, then sends Alice $B = g^b \mod p = 10$

4. Alice computes $s = B^a \mod p = 18$

5. Bob computes $s = A^b \mod p = 18$

6. Alice and Bob now share a secret key (the number 18) which they can use to encrypt their further conversation.\(^2\)

---

\(^1\)This is much to small for practical use, but serves the example well.

\(^2\)Currently there are no known techniques to infer $s$ without $a$ or $b$ as long as $a$ and $b$ are randomly generated prime numbers of at least 2048 bits (It is widely believed the NSA can crack anything using 1024 bits or less in minutes). For forward secrecy I would advocate 4096 bit keys or more recent (and less proven) elliptic-curve techniques.
# This command installs a required Python package
# It only needs to be run once on your system
python3 -m pip install pyopenssl

# This is the updated flask run command to start your server
flask run --cert=adhoc
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run(ssl_context="adhoc")
Client Warning from Self-Signed Certificate

Your connection is not private

Attackers might be trying to steal your information from localhost (for example, passwords, messages, or credit cards). NET::ERR_CERT_AUTHORITY_INVALID

Automatically report details of possible security incidents to Google. Privacy policy

Back to safety
Creating a Static Self-Signed Certificate

# create a new certificate and private key using RSA with openssl this only needs to be done once and can be done on the department machines if you don’t have openssl
openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem \ -keyout key.pem -days 365

# you will then be prompted to enter some basic information for the certificate which will be saved in the file "cert.pem". The private key will be saved in "key.pem".
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run(ssl_context=("cert.pem", "key.pem"))
# you can now launch your flask server specifying these
# files as your certificate and private key
flask run --cert=cert.pem --key=key.pem
Certificates from certificate authorities are usually not free and require that you own and control a domain.

- LetsEncrypt is the most popular free certificate authority
- If you are using PythonAnywhere, they will manage certificates from LetsEncrypt for you
Bibliography I

Miguel Grinberg.
Running your flask application over https.

Internet Security Research Group.
Lets encrypt.
https://letsencrypt.org/.

PythonAnywhere.
How to set up an https/ssl certificate for a custom domain.
Various.
Diffie-hellman key exchange.

Various.
Https.
https://en.wikipedia.org/wiki/HTTPS.

Various.
Public-key cryptography.
Various.

Transport layer security.
