Perfect forward secrecy (PFS)

Akash Patel (SJSU)

What is Perfect Forward Secrecy?[1]

- Perfect forward secrecy (PFS) is a property of the key-agreement protocol that ensures that session key used to encrypt the data will not be compromised even if long term private key compromised in future.

- The idea is to not to use single key (e.g. private key) to generate all the session keys. Also, same material or method should not use to generate session key.

Why PFS is required?[2]

- Lets say Alice and Bob want to safely communicate over network using Symmetric-key cryptography.

- Suppose Alice generates Session key using her private key and encrypt data using session key. She sent the resulting cipher to Bob. Trudy wants to acquire this information. She can record all the messages communicate over network. She does not know Alice's private key so she cannot know how session key is derived and how data is encrypted.

- Later, Trudy breaks Alice computer and obtain her private key K. Trudy can now decrypt Session key and all the previously recorded messages.

- Above situation is the motivational factor for the Perfect forward secrecy.

- The main problem is, private key is used for two purposes: authentication and encryption. Authentication only matters while the communication is established, but encryption is expected to last for years

How PFS works?[3]

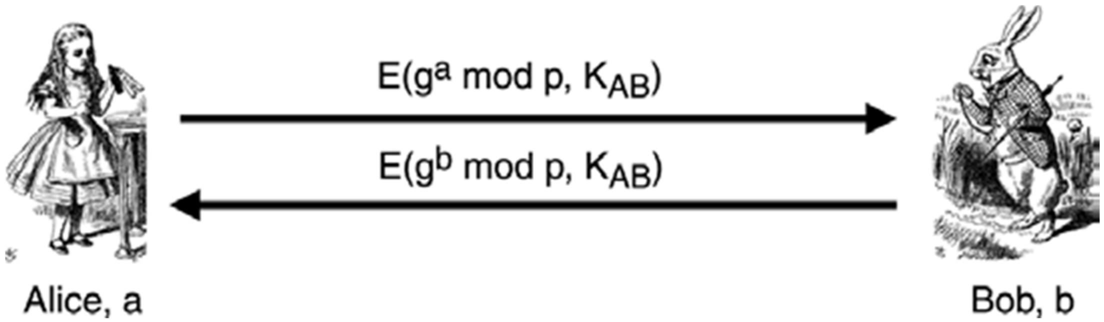
- PFS periodically creates a new session key value based on values supplied by both parties in the exchange. Because both parties contribute a random value known only to them, each new key generated is dissimilar to previously created keys.

- Even if a third party managed to intercept a private key, that party can only use the intercepted key for a short time.

- The newly created key is not used same method or material as used in the previously intercepted key, the third party must begin a new brute force calculation to guess the new key value

- Ephemeral Diffie-Hellman key exchange and Elliptic curve Diffie–Hellman are two most elegant approach for PFS.

How Diffie-Hellman Works? [4]



- In this protocol, g and p are public .Alice chooses a secret exponent a and Bob a secret exponent b.

- Then Alice sends g^a mod p to Bob and Bob sends g^b mod p to Alice. Alice and Bob can then each compute the shared secret g^{ab} mod p.

- In order to prevent the MiM attack, Alice and Bob can use their shared symmetric key K_{AB} to encrypt the Diffie-Hellman exchange

- Then to attain PFS, all that is required is that, once Alice has computed the shared session key $K_S = g^{ab}$ mod p, she must forget her secret exponent a and, similarly Bob must forget his secret exponent b.

How to set up PFS?[5]

- Session key is key factor to determine whether connection has perfect forward secrecy is determined by how the session key is derived. And how the session key is derived is determined by the cipher suite in use.

- In the beginning SSL handshake, the client sends a list of supported cipher suites .

- The server then picks one of the cipher suites, based on a ranking and inform client which cipher suite will be used from next onwards communication.

- Last step determines whether or not the further connection will have perfect forward secrecy

- Cipher suites that use ephemeral Diffie-Hellman (DHE) or the elliptic-curve variant (ECDHE) will have perfect forward secrecy

How to set up PFS on Apache Server?[6]

- Web server probably has a cipher suite configuration in its SSL configuration. There are two relevant options: first, the cipher suites that you want your server to use, and second, how the server picks the cipher suite. The order of cipher suite also matters

- For the Apache server Perfect Forward Secrecy requires Apache 2.3.3 or higher

- Here is one example configuration for mod_ssl that will work to enable Perfect Forward Secrecy:

```
SSLProtocol +TLSv1 +TLSv1.1 +TLSv1.2
SSLHonorCipherOrder On
SSLCipherSuite ECDHE-RSA-RC4-SHA:ECDHE-RSA-AES128-SHA:AES128-SHA:RC4-SHA
```

Challenges for PFS[7]

- DHE is significantly slower due to additional calculations.
- Web site operators tend to disable DHE suites in order to achieve better performance.
- Not all browsers support all the necessary suites.
- ECDHE too is slower, but not as much as DHE but ECDHE algorithms are relatively new and not as widely supported.

How to verify PFS?

There are many ways to verify PFS, I have listed below 3 ways to do that. I have used www.yioop.com as my test site.

1. OPENSSL Utility

2. SSL Scan Utility

3. https://www.ssllabs.com/ssltest/analyze.html

OPENSSL Utility

```
OpenSSL> s_client -connect  173.13.143.74:443
Server certificate
-----BEGIN CERTIFICATE-----
MIIFeDCCBGCgAwIBAgIHKw985qbAGjANBgkqhkiG9w0BAQUFADCByjELMAkGA1UE
BhMCVVMxEDAOBgNVBAgTB0FyaXpvbmExEzARBgNVBAcTClNjb3R0c2RhbGUxGjAY
BgNVBAoTEUdvRGFkZHkuY29tLCBJbmMuMTMwMQYDVQQLEypodHRwOi8vY2VydGlm
aWNhdGVzLmdvZGFkZHkuY29tL3JlcG9zaXRvcnkxMDAuBgNVBAMTJ0dvIERhZGR5
IFNlY3VyZSBDZXJ0aWZpY2F0aW9uIEF1dGhvcml0eTERMA8GA1UEBRMIMDc5Njky
ODcwHhcNMTMwOTA2MDczODE3WhcNMTUwOTEzMDAxOTM2WjA8MSEwHwYDVQQLExhE
b21haW4gQ29udHJvbCBWYWxpZGF0ZWQxFzAVBgNVBAMTDnNlZWtxdWFycnkuY29t
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0gMcw5vwD1wjBSFOtoJU
2X8KrxrWFwp2zAkWaLwUIWaBH9H7uk9q7i9DT6HljVkL6yJSsYOUmAoj+9nnn8tu
pFqDYb4yI1RiySiPWxq9GKsLo6PMN0Lk2/M+uZYXcuvYaMNpR6f8YWdi9VHzvvk/
0oOSPBzbCd0z0AvM8uyUdbF5cr9nmcCa9DLhuB9zEfvMLcfoYgmwWJUA1V9jgnV+
fE1IMYXehiYmpt1geX5WzPQtR5KdLbeyng65VS3/Ri/LyCaksOQSkjl7nMWyLDjw
Dn9Z27Vde66iec1E2Pom50U860NJjK0kQldakS3virVNv6z9GVFBdK7o98BESMol
3QIDAQABo4IB7jCCAeowDwYDVR0TAQH/BAUwAwEBADAdBgNVHSUEFjAUBggrBgEF
BQcDAQYIKwYBBQUHAwIwDgYDVR0PAQH/BAQDAgWgMDMGA1UdHwQsMCowKKAmoCSG
Imh0dHA6Ly9jcmwuZ29kYWRkeS5jb20vZ2RzMS050C5jcmwwUwYDVR0gBEwwSjBI
BgtghkgBhv1tAQcXATA5MDcGCCsGAQUFBwIBFitodHRwOi8vY2VydGlmaWNhdGVz
LmdvZGFkZHkuY29tL3JlcG9zaXRvcnkvMIGABggrBgEFBQcBAQR0MHIwJAYIKwYB
BQUHMAGGGGh0dHA6Ly9vY3NwLmdvZGFkZHkuY29tLzBKBggrBgEFBQcwAoY+aHR0
cDovL2NlcnRpZmljYXRlcy5nb2RhZGR5LmNvbS9yZXBvc2l0b3J5L2dkX2ludGVy
bWVkaWF0ZS5jcnQwHwYDVR0jBBgwFoAU/axhMpNsRdbi7oVfmrrndplozOcwWwYD
VR0RBFQwUoIOc2Vla3F1YXJyeS5jb22CEnd3dy5zZWVrcXVhcnJ5LmNvbYISd3d3
LnNlZWtxdWFycnkuY29tgg13d3cueWlvb3AuY29tggl5aW9vcC5jb20wHQYDVR0O
BBYEFEP+VC+oZx96um+YFW94JCjEG/g1MA0GCSqGSIb3DQEBBQUAA4IBAQAwkR3n
briuiJwRWXaFanisg4qNMao25MSB/VR8xYOHIagyMYy5Wn70O46sOiWHQlP6BNCc
J4OtWhD34Eh/Ig8BwXeKJ0I0BC0xUgkGt5HOpq2gZR61DhJcry6xHrkrG9+RQi0T
rB9EZJ1hgp0vLU1T5K2eKkcLrLTBssmTPWQXVlINFaJARHfB9rZiCTTY7AqydBQs
X5QG0B//tUzk5Q4fHgwD/reaiXUwfcCkY14308pSzlGW9uluAVHKdS6/hKnzV2es
v3H1NbN/YPazoN5YXKw0TfjXroplDkPU98nHRm7r7mq3sKUkpIvmzH4xKjplTLde
0q3vpk+zubtn4H+X
-----END CERTIFICATE-----
subject=/OU=Domain Control Validated/CN=seekquarry.com
issuer=/C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc./OU=http://certificates
godaddy.com/repository/CN=Go Daddy Secure Certification Authority/serialNumber=
7969287
---
No client certificate CA names sent
---
SSL handshake has read 5787 bytes and written 322 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 2048 bit
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1
    Cipher    : DHE-RSA-AES256-SHA
    Session-ID: FF4789A50225F6D4E8012DB8F4E0639939C6C7B09C114E9947D481B41BFDD45

    Session-ID-ctx:
    Master-Key: 9603E5AFE59932E4755E4721D944B70327DF11DBC47F0B8B3B80D08D05B4F1A
56557026DE0B7D416C294CBE7ABA6FEB
    Key-Arg   : None
    Start Time: 1381864537
    Timeout   : 300 (sec)
```

SSL Scan Utility

```
C:\Users\Akash\Desktop\SSLScan-1.8.2-win-r7>SSLScan --no-failed www.yioop.com

                  Version 1.8.2-win
             http://www.titania.co.uk
        Copyright Ian Ventura-Whiting 2009
    Compiled against OpenSSL 0.9.8m 25 Feb 2010

Testing SSL server www.yioop.com on port 443

  Supported Server Cipher(s):
    Accepted  SSLv3  256 bits  DHE-RSA-AES256-SHA
    Accepted  SSLv3  256 bits  AES256-SHA
    Accepted  SSLv3  128 bits  DHE-RSA-AES128-SHA
    Accepted  SSLv3  128 bits  AES128-SHA
    Accepted  SSLv3  168 bits  EDH-RSA-DES-CBC3-SHA
    Accepted  SSLv3  168 bits  DES-CBC3-SHA
    Accepted  SSLv3  128 bits  RC4-SHA
    Accepted  SSLv3  128 bits  RC4-MD5
    Accepted  TLSv1  256 bits  DHE-RSA-AES256-SHA
    Accepted  TLSv1  256 bits  AES256-SHA
    Accepted  TLSv1  128 bits  DHE-RSA-AES128-SHA
    Accepted  TLSv1  128 bits  AES128-SHA
    Accepted  TLSv1  168 bits  EDH-RSA-DES-CBC3-SHA
    Accepted  TLSv1  168 bits  DES-CBC3-SHA
    Accepted  TLSv1  128 bits  RC4-SHA
    Accepted  TLSv1  128 bits  RC4-MD5

  Prefered Server Cipher(s):
    SSLv3  256 bits  DHE-RSA-AES256-SHA
    TLSv1  256 bits  DHE-RSA-AES256-SHA
```

SSL Lab Test

QUALYS® SSL LABS

You are here: Home > Projects > SSL Server Test > yioop.com

SSL Report: yioop.com (173.13.143.74)

Assessed on: Tue Oct 15 19:43:19 UTC 2013 | Clear cache

Cipher Suites (sorted by strength; the server has no preference)

TLS_RSA_WITH_RC4_128_MD5 (0x4)	128
TLS_RSA_WITH_RC4_128_SHA (0x5)	128
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)	128
TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x33) DH 1024 bits (p: 128, g: 1, Ys: 128) FS	128
TLS_RSA_WITH_SEED_CBC_SHA (0x96)	128
TLS_DHE_RSA_WITH_SEED_CBC_SHA (0x9a) DH 1024 bits (p: 128, g: 1, Ys: 128) FS	128
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)	168
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x16) DH 1024 bits (p: 128, g: 1, Ys: 128) FS	168
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)	256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x39) DH 1024 bits (p: 128, g: 1, Ys: 128) FS	256

Forward Secrecy	With some browsers (more info)

References

1.http://en.wikipedia.org/wiki/Perfect_forward_secrecy

2.Section 9.3.4 of Information Security Principles and practice by Dr. Mark Stamp ,Published by JohnWiley & Sons, Inc

3.http://tinyurl.com/lamysw6

4.Section 9.3.4 of Information Security Principles and practice by Dr, Mark Stamp

5.http://crypto.stackexchange.com/questions/8933/how-can-i-use-ssl-tls-with-perfect-forward-secrecy

6.https://scottlinux.com/2013/06/26/how-to-enable-perfect-forward-secrecy-in-apache-on-linux/

7.https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy