computer emergency response team | CERT-EU for the EU institutions, bodies and agencies

CERT-EU Security Whitepaper 16-002

Weaknesses in Diffie-Hellman Key Exchange Protocol

Vicente REVUELTO, Krzysztof SOCHA

ver. **1.0**
July 7, 2016

TLP: WHITE

Summary

Recently, there have been some discussions about the minimum key length in public-key cryptography – more precisely in the Diffie-Hellman (DH) protocol – in order to be considered secure against state-level attackers [6].

DH is used often to negotiate session key over an insecure channel. DH relies on Discrete Logarithm Cryptography (DLC), which is comprised of both Finite Field Cryptography (FFC) and Elliptic Curve Cryptography (ECC). FFC is the public-key cryptographic methods using operations in a *multiplicative group of a finite field*. ECC is the public-key cryptographic methods using operations in an *elliptic curve group*. In order to use FFC, both peers have to share some domain parameters. Some of the elements that have to be shared by the peers as part of these domain parameters, is a multiplicative group in a finite field with prime `p` elements (called `GF(p)`), and one of its subgroups or at least the parameters to generate them [9]. It appears that often many applications tend to use standardized or hard-coded FFC domain parameters. An attack described recently allows to get the common secret from this information shared over an insecure channel [6]. This attack is based on prior knowledge of the domain parameters used to sequentialize the attack to an ongoing key establishment. Pre-computing the phases of the attack depends only on the FCC domain parameters.

There have been estimations that – with the resources available to an *academic team* – it is possible to do the pre-computation for a prime `p` with 768 bits of length, dramatically reducing the strength of the group used by FFC, also called DH group [10]. However, it is likely that the same can be possible for 1024-bit-long primes with the *nation-state-level* resources.

Diffie-Hellman Protocol

The DH protocol is a cornerstone of modern cryptography, and it is used for VPNs, HTTPS websites, email, and many other protocols (many based on SSL/TLS). For example, this protocol is used at the beginning of the connection by VPNs based on IPSEC in order to negotiate the session key used by block symmetric cyphers during the rest of the session.

The popularity of DH is due to the fact that the DH protocol allows two peers to securely agree on a common secret after a negotiation over an insecure channel. Only public information needs to be transmitted and complementary secret information is kept by the peers [3].

Weaknesses

Common Domain Parameters

The current best technique for attacking Diffie-Hellman relies on compromising one of the private keys by computing the discrete log of the corresponding public value in the DH group. As mentioned above, this group – or the parameters to generate it – are previously shared in clear as part of the FCC domain parameters. However, an adversary who performs a large pre-computation for a prime `p` can then quickly calculate arbitrary discrete logs in that group, amortizing the cost over all targets that share this parameter [6].

This attack takes advantage of the fact that *an adversary can perform a single enormous computation to crack a particular prime and then easily break any individual connection that uses that prime* [2]. The attacker can start with the phase of the attack which only depends on the prime. This can be done *in advance*, which leaves only the second phase to be done *on-the-fly* for any particular connection.

Websites that use one of a few commonly shared 1024-bit Diffie-Hellman groups may be susceptible to passive eavesdropping from an attacker with nation-state-level resources. The table below shows how various protocols would be affected if a single 1024-bit group was broken in each protocol, assuming a typical up-to-date client (e.g., most recent version of OpenSSH or up-to-date installation of Chrome) [4]:

Protocol	Vulnerable %
HTTPS – Top 1 Million Domains	17.9%
HTTPS – Browser Trusted Sites	6.6%
SSH – IPv4 Address Space	25.7%
IKEv1 (IPsec VPNs) – IPv4 Address Space	66.1%

Logjam Attacks

On the 9th of June, CERT-EU published an advisory concerning the Logjam attack [1]. It is a man-in-the-middle attack, which allows an attacker to force the negotiation of 512-bit-long keys in order to break encrypted communications. It concerns websites, mail servers, and other SSL/TLS-dependent services that support `DHE_EXPORT` ciphers.

Based on some Internet-wide scanning to measure who is vulnerable [4], the following estimates can be made:

Protocol	Vulnerable %
HTTPS – Top 1 Million Domains	8.4%
HTTPS – Browser Trusted Sites	3.4%
SMTP+StartTLS – IPv4 Address Space	14.8%
POP3S – IPv4 Address Space	8.9%
IMAPS – IPv4 Address Space	8.4%

Recommendations

Generally speaking, avoiding the use of common domain parameters or increasing the minimum key strengths to use primes of 2048 bits or larger in FCC can address this issue. The second approach relies on the expected complexity of the calculations for bigger groups, which makes the pre-computation phase out of the practical scope of the known algorithms. The second approach just avoids that pre-computation phases can be reused. So, in the short-term, it is suggested to use a 2048-bit DH group or larger. This group can be a standard DH group (as those standardized for the IETF [10, 23]) or a unique 2048-bit DH group generated following the specifications in `NISP SP-186-4`, and especially the Appendix A [11]. If – for compatibility – it is needed to use primes of 1024 bits, then the groups must be generated *ad-hoc* following the previous recommendation.

In the medium-term, transitioning to Elliptic Curve Diffie-Hellman (DH ECC) key exchange with appropriate parameters avoids all known feasible crypt-analytic attacks. Current elliptic curve discrete-log algorithms for strong curves do not gain as much of an advantage from pre-computation [6]. On the other hand, it is important to pay attention to the implementation of these protocols, because some of them have well known problems as well [12]. Some curves are undergoing scrutiny and new curves, such as `Curve25519`, are being standardized by the IETF for use in Internet protocols [6, 13]. Some comparisons between available curves can be found in the references [14].

It is also important to carefully consider other aspects affecting the security strength of an implementation [1]. For example, the security strength assumes that the random number generator has been provided with adequate entropy to support the desired security strength [15].

For System Administrators

Being consistent with the general recommendations listed above, we are collecting here some recommended configurations for some products that we think might be of interest for our constituency. These recommendations are mainly based on those listed on the `weakdh.org` site [5]. They are considering two steps:

- setting up a safe cipher suites (to increase the key strength), and
- providing the unique group you have previously created to your server, if possible.

The decision about using standard groups or generating new ones, depends on operational reasons, such as availability of platforms which allow to use them, and availability of procedures for secure generation and reliable distribution.

An easy way to securely generate a new 2048 group, in order to be used by different servers is to use OpenSSL:

```
openssl dhparam -out dhparams.pem 2048
```

[1] There have been concerns about weaknesses into at least one elliptic curve-based pseudo-random generator [7]. One analysis concluded that an adversary in possession of the algorithm's secret key could obtain encryption keys given only 32 bytes of ciphertext [8].

Apache HTTP Server (mod_ssl)

Disable SSL support enabling only TLS. To do that edit `httpd.conf`:

```
SSLProtocol             all -SSLv2 -SSLv3

SSLCipherSuite          ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
  AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128
  -GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-
  AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-
  RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-DSS-
  AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA:DHE-RSA-AES256-SHA:AES128-GCM-
  SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-
  CBC3-SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-SHA:!EDH-RSA-
  DES-CBC3-SHA:!KRB5-DES-CBC3-SHA

SSLHonorCipherOrder     on
```

In order to specify the new generated group, in newer versions of Apache (2.4.8 and newer) and OpenSSL 1.0.2 or later, you can directly specify your DH parameters in the previous file:

```
SSLOpenSSLConfCmd DHParameters "<path to dhparams.pem>"
```

Microsoft IIS

Microsoft relies on increasing the key strength and setting up standard DH groups from the IETF, more than generating unique groups. On the other hand, strong DH ECC is supported.

Follow these steps in order to set up a cipher suite in ISS:

- Open the **Group Policy Object Editor** (i.e. run `gpedit.msc` in the command prompt).
- Expand **Computer Configuration**, **Administrative Templates**, **Network**, and then click **SSL Configuration Settings**.
- Under **SSL Configuration Settings**, open the **SSL Cipher Suite Order** setting.
- Set up a strong cipher suite from those supported by Microsoft [16] following the previous recommendations.

From the generic Windows Server TLS/SSL implementation using the **Schannel Security Service Provider** (SSP) it worths to review [17]. Especially, for information about restricting algorithms and protocols in Schannel, see [18].

CISCO SYSTEMS

The generic document from Cisco providing cryptographic guidance is *Next Generation Encryption* [19]. As Microsoft, Cisco keeps relying on increasing the key strength and setting up standard DH groups from the IETF (as Group 15 or 19 [10]), more than generating unique groups. On the other hand, strong DH ECC is supported.

The following example shows a Cisco IOS Software IKE configuration that uses 128-bit AES for encryption, pre-shared key authentication, and 256-bit ECDH (Group 19):

```
crypto isakmp policy 10
encryption aes
authentication pre-share
group 19
```

The following example shows a Cisco IOS Software IKEv2 proposal configuration that uses 256-bit CBC-mode AES for encryption, SHA-256 for the hash, and 3072-bit DH (Group 15):

```
crypto ikev2 proposal my-ikev2-proposal
encryption aes-cbc-256
integrity sha256
group 15
```

OpenSSH

These recommendations are following [20] in addition to [5].

The following key exchange mechanisms are supported in the version (6.8) of OpenSSH:

- `curve25519-sha256@libssh.org`
- `ecdh-sha2-nistp256`
- `ecdh-sha2-nistp384`
- `ecdh-sha2-nistp521`
- `diffie-hellman-group1-sha1`
- `diffie-hellman-group14-sha1`
- `diffie-hellman-group-exchange-sha1`
- `diffie-hellman-group-exchange-sha256`

So, in the latest versions, strong cryptography based on DH ECC is supported but on the other hand, Group 1, which uses well known prime numbers is also supported. The first and easiest option is to force clients to use elliptic curve Diffie-Hellman. Specifically, `Curve 25519`. This can be accomplished by setting your by setting your `sshd_config` as follows:

```
KexAlgorithms curve25519-sha256@libssh.org
```

If you want to continue to support DH FFC, at the very least, you should disable Group 1 support, by removing the `diffie-hellman-group1-sha1` Key Exchange. It is fine to leave `diffie-hellman-group14-sha1`, which uses a 2048-bit prime.

The `diffie-hellman-group-exchange-sha1` and `diffie-hellman-group-exchange-sha256` mechanisms let the client and server negotiate a custom DH group. The client sends a tuple `(min, n, max)` to the server, indicating the client's *minimum*, *preferred*, and *maximum* group size, according to [21].

The OpenSSH server selects a suitable group from a pre-generated set of groups, installed system-wide in `/etc/ssh/moduli` (falling back to `/etc/ssh/primes`), using the `choose_dh` function in `dh.c`. In case no suitable group is found, the code defaults to Oakley Group 14, which is safe. A pre-generated set is distributed with the OpenSSH

source and many binary distributions and is infrequently changed. The group sizes distributed with OpenSSH are 1024, 1536, 2048, 3072, 4096, 6144, and 8192 bits, with about 30 groups per size. As mentioned, the OpenSSH-distributed 1024-bit groups are well-known and within the range of being a viable target for nation-state attackers, and as such should not be used.

It is also an option to generate new DH groups:

```
ssh-keygen -G moduli-2048.candidates -b 2048
ssh-keygen -T moduli-2048 -f moduli-2048.candidates
```

You then need to install `moduli-2048` to your system's `moduli` file. In Debian/Ubuntu, this is located at `/etc/ssh/moduli`. SSH chooses (practically randomly) from this file, so you should replace your existing moduli file with the new groups you generated instead of appending these new groups.

For Client Systems

Browsers should be patched and updated to the most recent version in order to support most recent cipher suites. It might be interesting taking a look the recommended configurations in [22] in other to check clients compatibility with strong cryptography.

For Developers

Make sure the cryptographic libraries you use are updated – especially, but not only, for SSL/TLS.

References

[1] https://cert.europa.eu/static/SecurityAdvisories/CERT-EU-SA2015-325.txt

[2] https://freedom-to-tinker.com/blog/haldermanheninger/how-is-nsa-breaking-so-much-crypto

[3] https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[4] https://weakdh.org/

[5] https://weakdh.org/sysadmin.html

[6] https://weakdh.org/imperfect-forward-secrecy.pdf

[7] https://www.schneier.com/essay-198.html

[8] http://rump2007.cr.yp.to/15-shumow.pdf

[9] http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf

[10] https://tools.ietf.org/html/rfc5114

[11] http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[12] https://cryptome.org/2013/11/ecc-practice.pdf

[13] https://tools.ietf.org/html/draft-ietf-tls-curve25519-01

[14] http://safecurves.cr.yp.to/

[15] http://csrc.nist.gov/publications/drafts/800-57/sp800-57p1r4_draft.pdf

[16] https://technet.microsoft.com/en-us/library/dn786419.aspx

[17] https://technet.microsoft.com/en-us/library/hh831381.aspx

[18] https://support.microsoft.com/en-us/kb/245030

[19] http://www.cisco.com/web/about/security/intelligence/nextgen_crypto.html

[20] https://jbeekman.nl/blog/2015/05/ssh-logjam/

[21] https://www.ietf.org/rfc/rfc4419.txt

[22] https://wiki.mozilla.org/Security/Server_Side_TLS#Recommended_configurations

[23] https://www.ietf.org/rfc/rfc3526.txt