

Frequently Asked Questions

Who is vulnerable?

In theory, all protocols that use the Diffie-Hellman algorithm for key exchange are affected. In practice, there are differences between particular cryptographic protocols and their implementations, making some more affected than others.

The proof-of-concept implementation of the attack supports the following protocols:

- Transport Layer Security (TLS)
- OpenVPN
- Secure Shell (SSH)

However, other protocols such as IPsec (IKEv1, IKEv2) must also be affected.

Is my server vulnerable?

It depends on your server configuration. If your server offers the Diffie-Hellman key exchange, then you are affected. However, the effectiveness of a potential attack largely depends on the specifics of the cryptographic protocol, the library implementation, and the key size used during the key exchange. You can use either free software (e.g., CryptoLyzer, SSLyze, ssh-audit, testssl.sh), or SaaS

solutions (e.g., Scanigma) to check whether your server offers Diffie-Hellman key exchange and if so, under what parameters.

How practical is the attack?

It depends on hardware capabilities and the server configuration. If your server offers Diffie-Hellman key exchange with larger key sizes and the cryptographic library you use is affected by the “long exponent” issue, an attacker needs to send an extremely low number of requests to cause 100% CPU load on the server. For the exact number of requests needed to overload a server, see the attack scenario section of the full technical paper about the attack. For testing *your* server, you can use the proof-of-concept implementation D(HE)ater.

Can I detect if someone has exploited this against me?

Yes, you can, though it may require some configuration changes. An application server can log instances where a client connection is interrupted during cryptographic handshakes. While this can happen under normal circumstances, it is also indicative of suspicious behavior. A high number of log messages about connections being lost during the handshake phase likely suggests an attack. For more details, see the mitigations page.

Should I disable Diffie-Hellman in my server configurations?

Not necessarily. However, you need to check both server configuration and cryptographic library implementation details. The former is simpler as the mentioned free software can be used to check whether your server configuration enables the Diffie-Hellman key exchange or not, and if so, what key sizes are offered. The latter is harder as there is no obvious way to check the performance of the Diffie-Hellman implementation and used exponent sizes in the case of a cryptographic library. For the details of the most popular libraries from this point of view, see the analysis section of the full technical paper about the attack.

Why is the attack called "D(HE)at"?

The name “D(HE)at” comes from the attack’s ability to overheat a server’s CPU by forcing it to repeatedly perform the CPU-intensive modular exponentiation operation used in the Diffie-Hellman key exchange across various cryptographic protocols (such as TLS, SSH, IPsec, OpenVPN, and others).

Is this vulnerability serious enough to deserve a web page?

D(HE)at is not an issue with client or server implementations, or a cryptographic library flaw, so it cannot be fixed by simply installing a software update. Instead, it exploits the Diffie-Hellman key agreement protocol, meaning that it may affect many cryptographic protocols (e.g., TLS, SSH, OpenVPN, IKEv1/IKEv2, …) that use the ephemeral version of the Diffie-Hellman for exchanging keys. Therefore, a comprehensive solution to this vulnerability would require modifying the cryptographic protocols that use DHE key agreement, as well as the client/server or library implementations.

How widespread is the Diffie-Hellman key exchange?

Very common. Diffie-Hellman is built into widely used cryptographic libraries such as Mbed TLS, GnuTLS, OpenSSL, and others. Most cryptographic protocol libraries support it, and many servers still enable it. However, adoption rates vary by protocol and application. Both the cryptographic protocol and the application protocol influence how often DHE is used. Even if elliptic-curve Diffie-Hellman (ECDHE) is preferred over traditional DHE, the attack remains possible as long as DHE is enabled in the server configuration—it does not have to be the default or preferred option.

DHE is mainly retained for backward compatibility with legacy clients that do not support ECDHE. It also provides a key advantage over RSA: forward secrecy. Forward secrecy ensures that even if the master secret is compromised in the future, previously intercepted encrypted communications cannot be decrypted.

DHE remains popular in Transport Layer Security (TLS), Secure Shell (SSH), Internet Protocol Security (IPsec), and OpenVPN because it offers both backward compatibility and forward secrecy. Outside of TLS, performance is less of a concern, since SSH and VPN servers (like IPsec and OpenVPN) typically handle fewer new connections. Because cryptographic handshakes are less frequent, using a slower key exchange algorithm like DHE is acceptable—especially when forward secrecy is important for remote access protocols such as SSH and IPsec.

Among the top 100 domain web servers, support for Diffie-Hellman key exchange is rare. In the top 10,000 domains, over 25% of servers support DHE, which is still a minority. For the top 1 million domains, the rate rises to 55% (measured in May 2023), which is significant—especially considering Shodan's estimate of over 25 million HTTPS servers. If similar adoption rates apply across the wider internet, more than 10 million servers could be accessible via Diffie-Hellman key exchange.

Are there cryptographic protocols more vulnerable than other?

Yes, some cryptographic protocols are more vulnerable than others. For instance, in TLS, an attacker can trigger a DHE key exchange by sending a small Client Hello message (about 100 bytes). Most other cryptographic protocols also require only two or three messages sent and one or two received to start the CPU-intensive part of the handshake. These messages are usually small and require almost no computation from the client. Once an attacker knows the server's configuration, they can further reduce message sizes by including only the necessary parameters in the initial client messages. Often, these messages can be pre-generated, so the attacker does not need to compute them for each connection, making the attack even more efficient.

Does this attack work just as well against opportunistic TLS?

No. In opportunistic TLS, upgrading a plain text connection to an encrypted one requires extra steps. This usually means one or two additional round trips before the TLS handshake can begin, resulting in more delay and extra data exchanged. Because of these extra steps, the attack is less effective in opportunistic TLS scenarios. However, the underlying vulnerability is still present.

Is this attack equally effective across all opportunistic TLS protocols?

No. The details of opportunistic TLS protocols differ. Some, like LDAP, PostgreSQL, and RDP, require only one extra round trip to start TLS. Others, such as FTP, IMAP, POP3, and SMTP, require two extra round trips. As a result, the delay and bandwidth overhead can vary by protocol. However, these differences do not fundamentally change the underlying vulnerability.

Are there implementations more vulnerable than others?

Yes, some implementation flaws can seriously affect the effectiveness of the D(HE)at attack.

Long exponent (CVE-2022-40735)

Implementations may use unnecessarily long exponents (CVE-2022-40735), which makes the modular exponentiation – the CPU-intensive operation of the Diffie-Hellman key exchange – significantly more expensive than it should be.

Library	Affected Versions
OpenSSL (1.x)	
OpenSSL (3.x)	
Mbed TLS	
OpenJDK	
Oracle JDK	

Unnecessary validation (CVE-2024-41996)

Implementations may perform public key validation against small subgroup confinement attack unnecessarily (CVE-2024-41996), where the validation is as expensive as public key generation using large exponents.

Library	Affected Versions
OpenSSL (1.x)	n/a
OpenSSL (3.x)	

* no public key validation

Large key size by default

The flaws mentioned earlier, coupled with large key sizes enabled by default, make applications using the library particularly vulnerable to the D(HE)at attack in the absence of mitigation techniques.

Library	Key Size
OpenSSL (1.x)	n/a
OpenSSL (3.x)	8192
OpenJDK	8192
Oracle JDK	8192

* no finite field Diffie-Hellman parameter negotiation support

Which key sizes are commonly used by web servers?

The vast majority of web servers use a 2048-bit key size, which is still vulnerable to the attack. The proportion of servers using a 1024-bit key size remains high because the Logjam attack is believed to be feasible for state-level attackers. Initially, TLS 1.2 supported only a single key size, but the introduction of an extension (RFC 7919) made it possible to negotiate the key size during the handshake. This extension is now part of the TLS 1.3 protocol definition.

However, unlike other cryptographic libraries (such as GnuTLS, OpenJDK, and WolfSSL), OpenSSL does not support this extension in TLS 1.2 and only supports Diffie-Hellman key exchange in TLS 1.3 starting from version 3.0.

How can I contact you?

You can reach us via Twitter or LinkedIn:

- Szilárd Pfeiffer: Twitter (X), LinkedIn
- Norbert Tihanyi: LinkedIn

