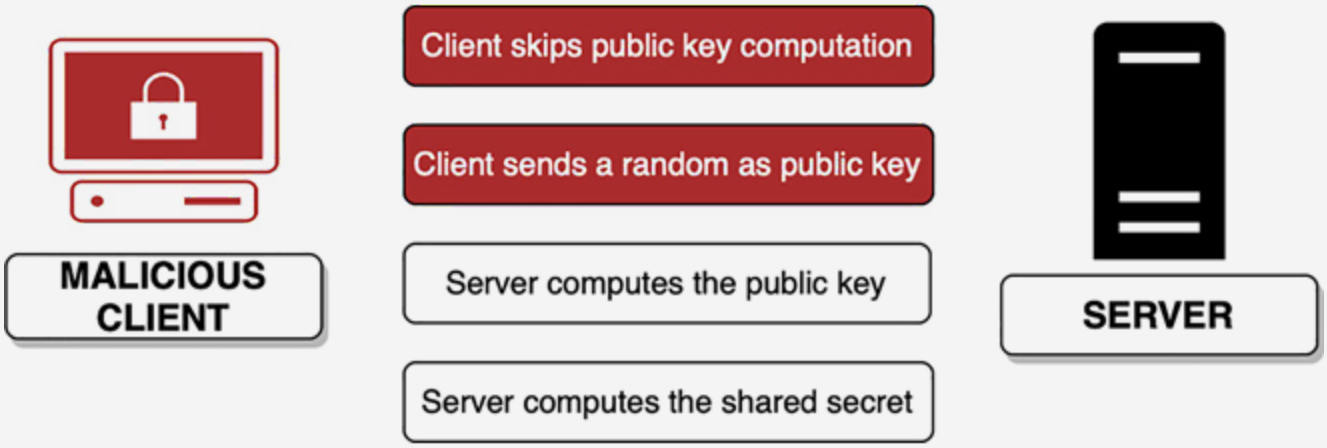
Technical Details

How does the attack work?

During a D(HE)at attack, the malicious client initiates cryptographic handshakes pretending to support only the ephemeral variant of the finite field Diffie-Hellman key agreement protocol (DHE), triggering key pair generation and shared secret calculation on the server side. Both operations require the server to perform compute-intensive modular exponentiation calculations, while the malicious client needs no significant computational resources.

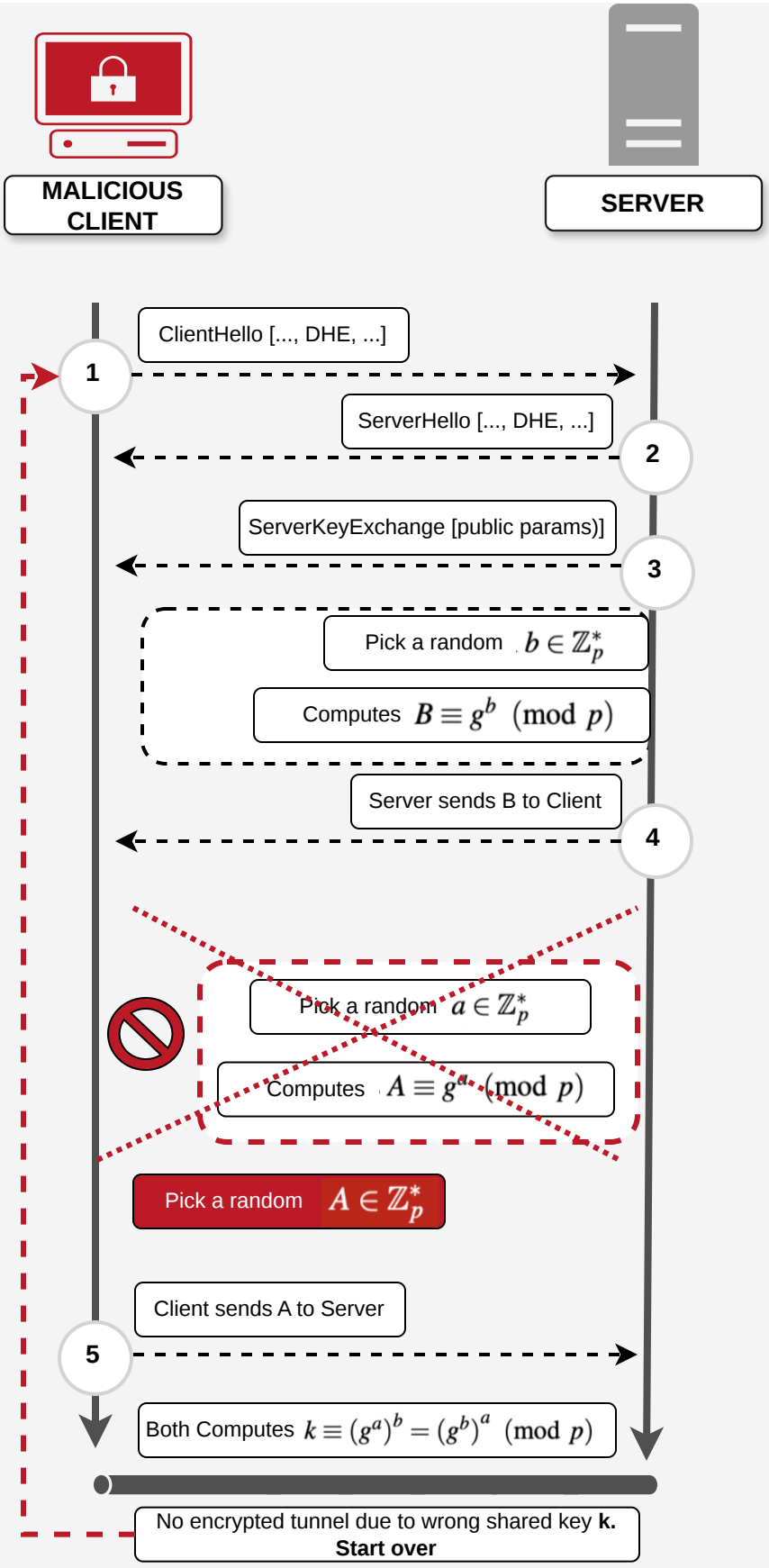


Under normal circumstances, both parties perform the same operations during the Diffie-Hellman key agreement (public key generation, shared secret calculation), meaning that theoretically the computational cost is the same for both parties. However, in practice, a malicious client can force a server to generate a public key by acting as if it would support only the Diffie-Hellman key agreement and would wait for the server's public key to ensure that the server has performed the expensive modular exponentiation, then it terminates the connection. In the case of certain protocols (e.g., TLS < 1.3), an initial cryptographic handshake message is sufficient to be sent without the client's public key.

Other protocols or protocol versions (e.g., TLS 1.3, SSH) require the client to send its public key before the server calculates its own. In such a scenario, a malicious client can send an arbitrary number as a public key without performing the modular exponentiation. In possession of that arbitrary number, the server performs the shared secret calculation, which means a second modular exponentiation on the server side. However, the key agreement will fail, and the server can only realize this after it has already performed the CPU-intensive operations.

How bad is it?

The D(HE)at attack is not a cryptographic library implementation issue that can be fixed by a software update, but a protocol flaw that exploits the peculiarity of the Diffie-Hellman key agreement, where a malicious client can force a server to perform CPU-intensive operation(s) without any significant resource (CPU) requirement. This is possible because servers cannot distinguish a randomly chosen ($A \in \mathbb{Z}_p^*$) number from the result of modular exponentiation ($A \equiv g^a \bmod p$), which requires CPU-intensive operations, meaning that the server has no proof-of-work from the client. Additionally, some implementation flaws can seriously affect the effectiveness of the D(HE)at attack.



Long exponent (CVE-2022-40735)

Some cryptographic libraries may use long exponents (a, b) leading to significantly more expensive calculation of the public keys ($B \equiv g^b \bmod p$) than they would use shorter exponents that provide the same security strength (CVE-2022-40735). As the resource requirement of modular exponentiation relates non-linearly to the size of the parameters (a/b, p), the larger key sizes that can be forced by a malicious client, the more significant the attack can be.

Affected Libraries

Library	Version	Private Key Size
OpenSSL	≤1.1.1	public key size
OpenSSL 3	≤3.0.5	public key size
Mbed TLS		public key size
OpenJDK	≤17.0.5	0.5 * public key size
Oracle Java	≤17.0.5	0.5 * public key size

Unnecessary validation (CVE-2024-41996)

Parties should validate the order of the peer's public key to avoid small subgroup confinement attack, according to NIST SP 800-56A Rev 3., by performing a modular exponentiation ($1 \equiv A^{(p-1)/2} \bmod p$). However, the special publication of NIST also states that when an approved safe-prime group is used, the validation can be skipped. Some cryptographic libraries may always perform the validation independently of the group value, allowing an attacker to trigger expensive modular exponentiation (CVE-2024-41996) which is as resource-intensive as public key generation using a large exponent.

Affected Libraries

Library	Version
OpenSSL 3	≤3.4.0

Large key size by default

Larger parameter sizes (e.g., ffdhe6144 or ffdhe8192) enabled by the default configuration of a cryptographic library or an application server can also seriously affect the potential of a D(HE)at attack, as the calculation of a larger public key size requires more resources. For instance, some popular cryptographic libraries enable the largest available parameter (ffdhe8192) size by default in TLS. This carries the risk of a successful DoS attack when application server implementations or configurations do not overwrite the cryptographic library defaults, such as OpenSSL has.

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

