Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice

Authors: David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelink Paul Zimmermann

Presentation by Simran Tinani

Overview

- Logjam: **Man-in-the middle** downgrade attack which manipulates **the TLS handshake** (versions 1.2 and older) to use "export" parameters and then impersonate the server to finish it
- Exploits a server's willingness to accept a connection with 512-bit Diffie-Hellman keys
- Once the connection is downgraded, the attacker computes the server's secret key, and hence also the master secret and session keys, and can then impersonate the server completely by forging the **Finished message** in the handshake
- The server secret key is broken by computing 512-bit discrete logs using the **Number Field Sieve**
- The authors demonstrated compromise of a large number of websites
- The massive scale of the attack was caused by widespread reuse of a small set of DHE parameters across the internet and the ability to use precomputed values for the cryptanalysis

Contents of the talk

1. Conceptual Walkthrough
   a. Diffie-Hellman and discrete log
   b. Number field sieve
   c. The TLS Handshake
   d. Man-in-the-middle attacks
2. Logjam attack: working, time costs, consequences
3. State-level threats to 1024-bit DH, potential consequences
4. Author recommendations
5. Mitigations in TLS 1.3

1. Conceptual Walkthrough
    a. Diffie-Hellman and the discrete log problem
    b. Number field sieve
    c. Export grade cryptography
    d. The TLS Handshake
    e. Man-in-the-middle attacks
2. Logjam attack: working, time costs, consequences
3. State-level threats to 1024-bit DH, potential consequences
4. Mitigation: Author recommendations and in TLS 1.3

Diffie-Hellman Key Exchange

Let p be a large prime.

1. Alice and Bob agree on a large prime p and an integer g with large prime order modulo p.
2. Alice chooses a secret integer a, computes $A = g^a \pmod p$. She sends A to Bob. Her secret key is a, her public key is A.
3. Bob chooses a secret integer b and computes $B = g^b \pmod p$. He sends B to Alice. His secret key is b, his public key is B.
4. Alice computes her shared secret key, $K_A = B^a \pmod p$.
5. Bob computes his shared secret key, $K_B = A^b \pmod p$.

$K_A = g^{ab} = K_B$ is the shared secret key of Alice and Bob

Discrete logarithm problem

- **Discrete logarithm problem (DLP)**: Given group elements g *and* $h=g^x$, compute the number x (modulo the order of g).
- Shoup's theorem: any "generic" algorithm that solves the discrete logarithm problem in an arbitrary group G of size n must perform at least $\Omega(n^{1/2})$ group operations (exponential complexity)
- "Generic" means that the algorithm has access to the group structure only via to two oracles: one for performing group operations and one for testing for equality in the group.
- In some elliptic curve groups, the generic algorithm is the best we can do
- This is not so for finite field groups, where we can exploit the structure

1. Conceptual Discussion
    a. Diffie-Hellman and the discrete log problem
    b. Number field sieve
    c. Export grade cryptography
    d. The TLS Handshake
    e. Man-in-the-middle attacks
2. Logjam attack: working, time costs, consequences
3. State-level threats to 1024-bit DH, potential consequences
4. Mitigation: Author recommendations and in TLS 1.3.

Number field sieve (NFS) for DLP

- Instance of a more general method called **index calculus**
- Solves discrete logarithm problems in finite fields with heuristic subexponential, superpolynomial complexity
- Approach: Given $h=g^x \pmod p$, To compute this logarithm $x \pmod{(p-1)}$, we first find discrete logarithms modulo large prime divisors l of $p-1$, i.e., relations of the form $j = \log_g h \pmod l$ and use the Chinese remainder theorem to find the original logarithm
- The prime factorization of $p-1$ is typically known in most cryptographic contexts. If not, can use the factorization version of the NFS to find it, which runs in the same time as this algorithm

Index Calculus

Let $h=g^x \pmod p$. We want to calculate the integer x *modulo* $(p-1)$.

Fix a smoothness bound B.

- Pick integer i at random. Test divisibility of $h*g^i$ by all primes below B.
  If $h*g^i \pmod p$ is B-smooth, keep the relation:

$$h*g^i = p_1^{e_1} \times \cdots \times p_k^{e_k} \pmod p.$$

- Taking logs with base g gives

$$x + i = e_1 \log_g p_1 + \cdots + e_k \log_g p_k \pmod{p-1}$$

- Vary i to obtain multiple such relations, until there are enough linear equations
- Solve the linear system to find x (and the logarithms of all elements of the factor base that appear in at least one relation) modulo a factor of $p-1$.
- Combine the solutions of multiple systems to get a solution modulo $p-1$

Number field sieve

1. **Polynomial Selection:** select a suitable rational polynomial *f(z)* with a fixed degree defining a number field *Q(z)/f(z)*.
   - parallelizes well and is only a small portion of the runtime
2. **Sieving**: find special pairs of integers that satisfy the polynomial *f(z)*
   - factor integers in batches to find relations of elements whose prime factors modulo *p* are *B*-smooth
   - parallelizes well but is computationally expensive
3. **Matrix Step:** construct a large, sparse matrix consisting of the coefficient vectors of prime factorizations found. The null space of gives a database of the logs of many small elements
4. **Descent:** deduce the discrete log of the target *h*: re-sieve until we can find a set of relations that allow us to write the log of *h* in terms of the logs in the precomputed database. Three phases of sieving with decreasing prime size. The final phase reconstructs the target using the log database

Complexity

For a group of size n, the complexity is given by

$$e^{\sqrt[3]{\frac{64}{9}+o(1)}\ (\ln n)^{\frac{1}{3}}\ (\ln \ln n)^{\frac{2}{3}}}$$

This is obtained by tuning many parameters, including the degree of f, the sieving region parameter, and, most importantly, the smoothness bound B

1. Conceptual Discussion
    a. Diffie-Hellman and and the discrete log problem
    b. Number field sieve
    c. Export-grade cryptography
    d. The TLS Handshake
2. Logjam attack: working, time costs, consequences
3. State-level threats to 1024-bit DH, potential consequences
4. Mitigation: Author recommendations and in TLS 1.3.

Size of *p* and export grade cryptography

- As of 2023, |p|=2048 for appropriate “safe” primes *p* is considered sufficiently strong
- In the 1990s, the U.S. government did not approve export of cryptographic products unless the key size was strictly limited, therefore breakable. Attempt to control foreign countries usage of cryptography
- Cryptographic products were divided into two classes: products with “strong” cryptography and products with “weak” (exportable) cryptography.
- Weak cryptography: key sizes
  - ≤ 56 bits in symmetric algorithms,
  - ≤ 512 bits for RSA/Diffie-Hellman moduli
  - ≤ 112 bits for elliptic curve keys

Size of p and export grade cryptography

- To comply with 1990s-era U.S. export restrictions on cryptography, SSL 3.0 and TLS 1.0 supported reduced-strength DHE_EXPORT ciphersuites that were restricted to primes no longer than 512 bits.
- January 2000: restrictions on export regulations dramatically relaxed.
- Many libraries and servers retained support for backwards compatibility.
- Today: all cryptographic products are exportable without a license unless end-users are foreign governments or “embargoed destinations”. Export to governments may be approved under a license.

https://upload.wikimedia.org/wikipedia/commons/e/e0/Netscape_Navigator_1.1_for_Macintosh_Install_Disk.jpg

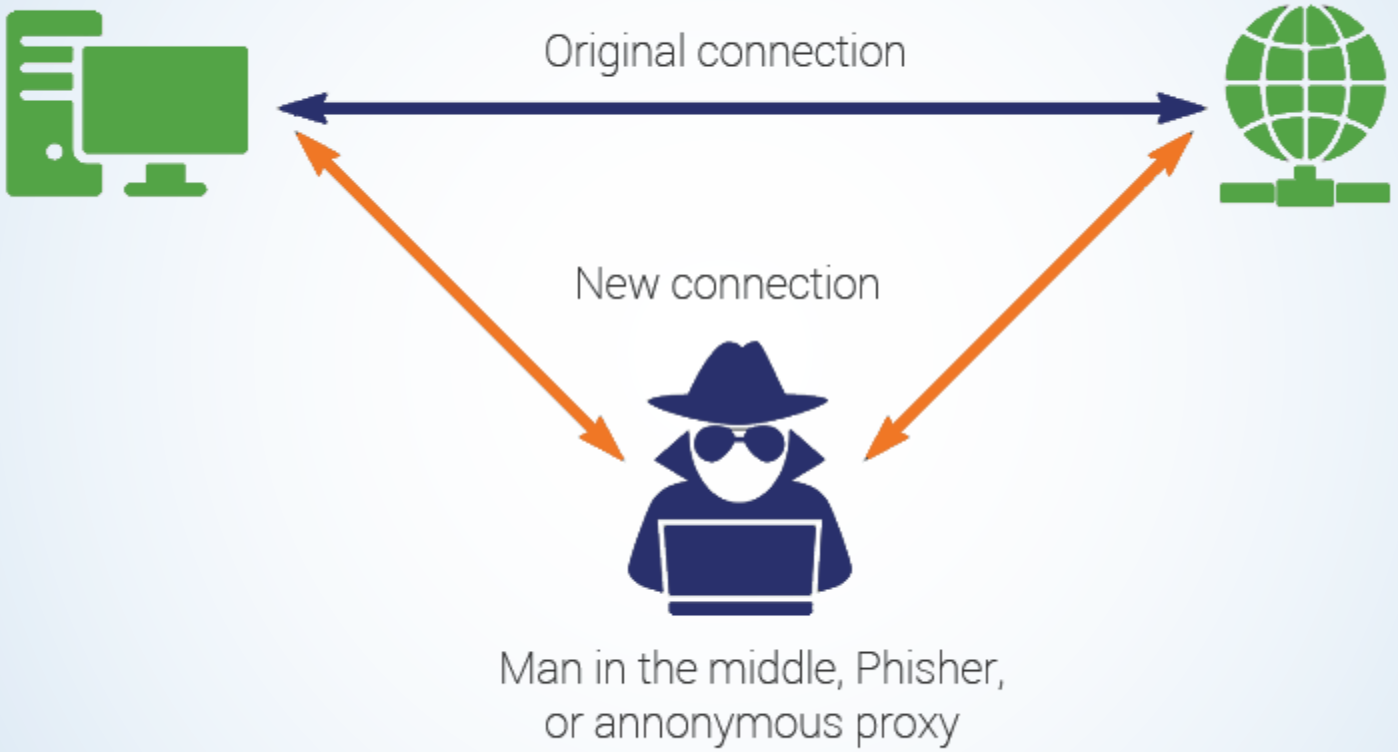
1. Conceptual Discussion
    a. Diffie-Hellman and and the discrete log problem
    b. Number field sieve
    c. Export-grade cryptography
    d. Man-in-the-middle attacks
    e. The TLS Handshake
2. Logjam attack: working, time costs, consequences
3. State-level threats to 1024-bit DH, potential consequences
4. Mitigation: Author recommendations and in TLS 1.3.

Man-in-the-middle (MITM) attacks

- An attacker intercepts and potentially alters the data exchanged between two communicating parties without their knowledge.
- **The two communicating parties do not suspect that their communication is being relayed.** From their perspective, it appears as if they are communicating directly with each other.
- For this, the MITM must transmit data between the parties so that the communication appears as usual
- Common targets are communication channels such as Wi-Fi networks, public hotspots, and unsecured websites.
- **Prevention: the best mitigation technique is mutual authentication and data encryption, both of which are achieved typically using TLS..**

Techniques

1. **Packet Sniffing:** Attacker intercepts and analyze data packets as they travel across a network. Tools: Wireshark, tcpdump
2. **ARP Spoofing/Poisoning:** Address Resolution Protocol (ARP) spoofing involves sending false ARP messages to link an attacker's MAC address with the IP address of a legitimate party
3. **DNS Spoofing:** Attackers can manipulate the DNS to redirect users to malicious websites.
4. **HTTP Session Hijacking:** capturing session cookies, session IDs, or other authentication tokens.
5. **SSL Stripping:** Forcing a connection to use HTTP instead of HTTPS where a website supports both
6. **Wi-Fi Eavesdropping:** Setup of rogue Wi-Fi hotspots with names similar to legitimate networks.
7. **Email Hijacking:** Compromising email accounts, Ex. by phishing
8. **Malicious Proxies:** Users unknowingly using the proxy think they are connecting directly to the intended website.
9. **Rogue Devices:** physical insertion of a rogue device (Ex. malicious router, switch) into a network to capture or manipulate data



https://www.thesslstore.com/blog/man-in-the-middle-attack-2/

1. Conceptual Discussion
    a. Diffie-Hellman and and the discrete log problem
    b. Number field sieve
    c. Export-grade cryptography
    d. Man-in-the-middle attacks
    e. TLS
2. Logjam attack: working, time costs, consequences
3. State-level threats to 1024-bit DH, potential consequences
4. Author recommendations
5. Mitigations in TLS 1.3

mperfect Forward Secrecy: Diffie-Hellman Fails in Practice

eyan Bhargavan* Zakir Durumeric¶ Pierrick Gaudry† Matthew Green§
J. Alex Halderman¶ Nadia Heninger‡ Drew Springall¶ Emmanuel Thomé† Luke Valenta‡
Benjamin VanderSloot¶ Eric Wustrow¶ Santiago Zanella-Béguelin‖ Paul Zimmermann†

*INRIA Paris-Rocquencourt †INRIA Nancy-Grand Est, CNRS, and Université de Lorraine
‖Microsoft Research ‡University of Pennsylvania §Johns Hopkins ¶University of Michigan

For additional materials and contact information, visit WeakDH.org.

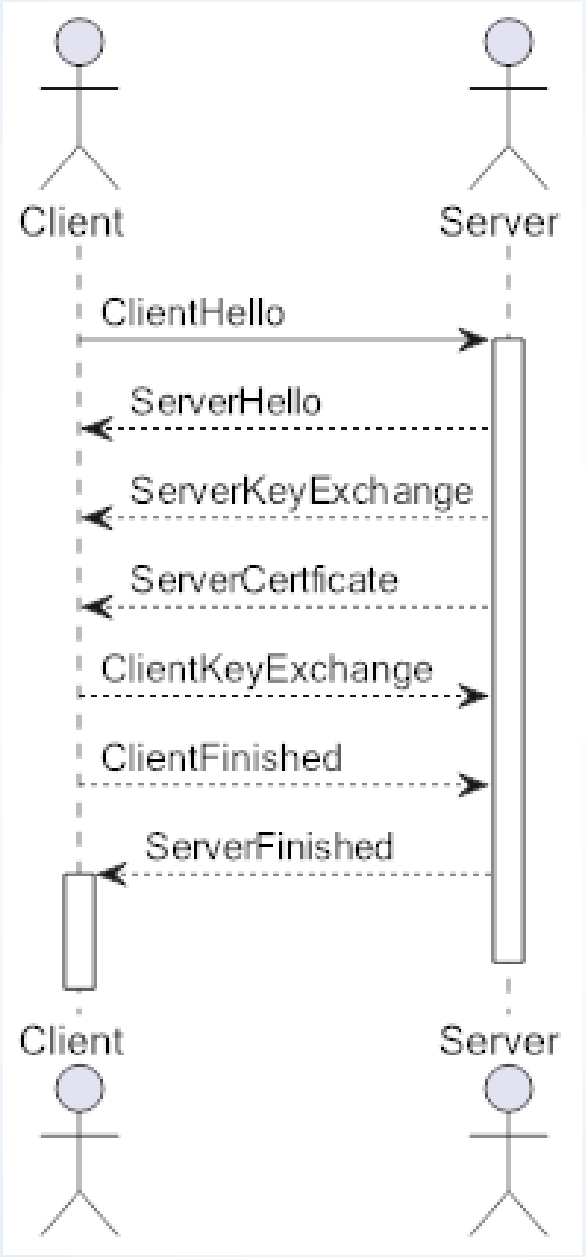
ABSTRACT

We investigate the security of Diffie-Hellman key exchange as used in popular Internet protocols and find it to be less secure than widely believed. First, we present Logjam, a novel flaw in TLS that lets a man-in-the-middle downgrade connections to "export-grade" Diffie-Hellman. To carry out this attack, we implement the number field sieve discrete log algorithm. After a week-long precomputation for a specified 512-bit group, we can compute arbitrary discrete logs in that group in about a minute. We find that 82% of vulnerable servers use a single 512-bit group, allowing us to compromise connections to 7% of Alexa Top Million HTTPS sites. In response, major browsers are being changed to reject short groups.

We go on to consider Diffie-Hellman with 768- and 1024-bit groups. We estimate that even in the 1024-bit case, the computations are plausible given nation-state resources. A small

coded, or widely shared Diffie-Hellman parameters has the effect of dramatically reducing the cost of large-scale attacks, bringing some within range of feasibility today.

The current best technique for attacking Diffie-Hellman relies on compromising one of the private exponents (a, b) by computing the discrete log of the corresponding public value $(g^a \bmod p, g^b \bmod p)$. With state-of-the-art number field sieve algorithms, computing a single discrete log is more difficult than factoring an RSA modulus of the same size. However, an adversary who performs a large precomputation for a prime p can then quickly calculate arbitrary discrete logs in that group, amortizing the cost over all targets that share this parameter. Although this fact is well known among mathematical cryptographers, it seems to have been lost among practitioners deploying cryptosystems. We exploit it to obtain the following results:

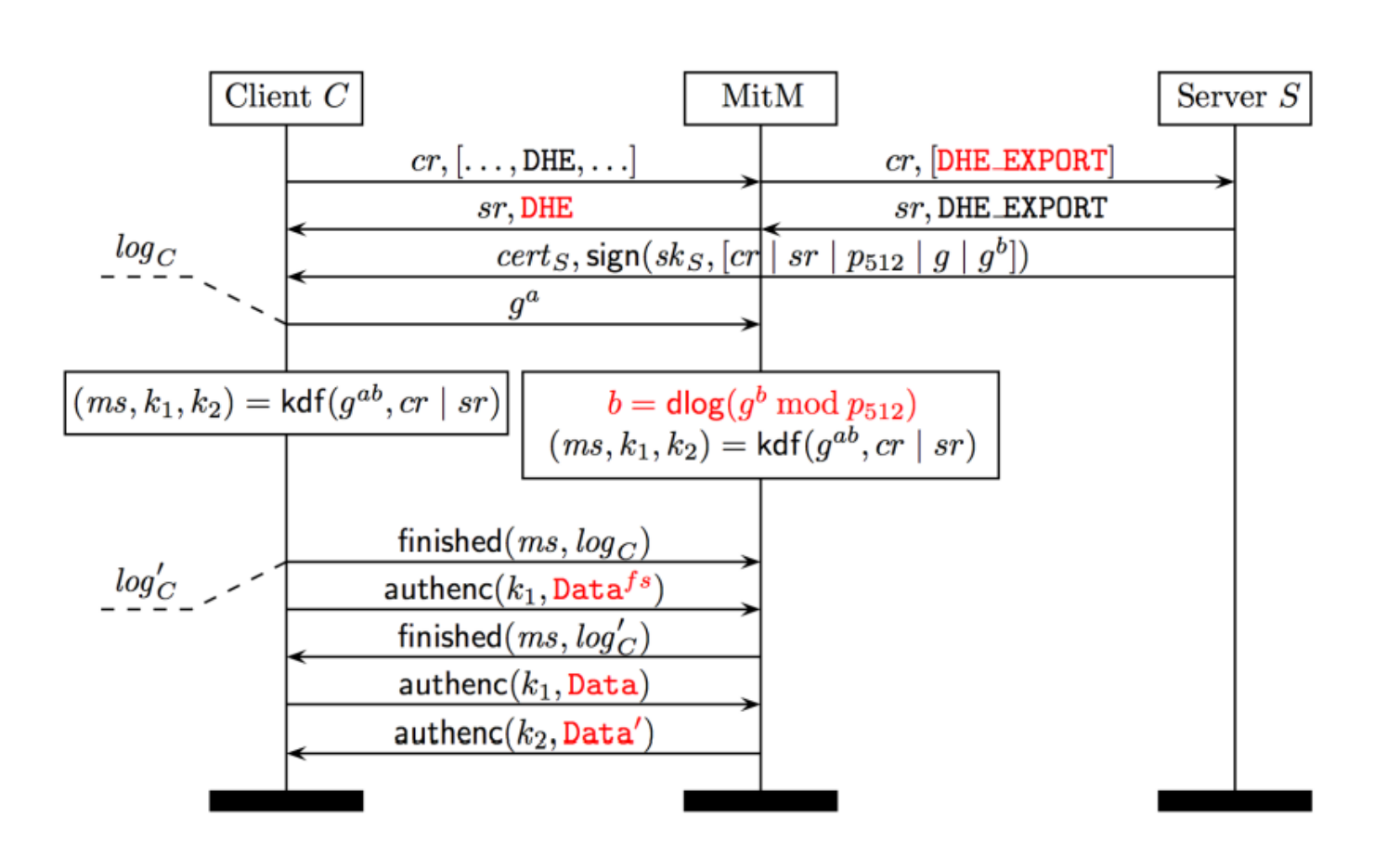

Client
Server
ClientHello
ServerHello
ServerKeyExchange
ServerCertficate
ClientKeyExchange
ClientFinished
ServerFinished
Client
Server

TLS Handshake messages

- **ClientHello**: protocol version, client random *cr*, optional session *id* to resume, a list of cipher suites, list of compression methods, list of extensions
- **ServerHello**: selected protocol version, server random *sr*, session *id*, selected cipher suite, selected compression method, list of extensions
- **Server certificate**
- **Server key generation and exchange:**
  - prime p, public key g^b, *signature*(p, g, g^b, sr, cr)
- **Sever Hello Done**
- **Client Key generation, exchange:** public key g^a
- **Client Key Calculation**: $(ms, k_1, k_2) = kdf(cr, sr, g^a, b)$
- **Server Key Calculation**: $(ms, k_1, k_2) = kdf(cr, sr, g^a, b)$
- Client and Server Handshake **Finished** messages: To verify that the handshake was not tampered with, the client and server calculate the verification data: a MAC of the handshake transcript, and send it with the Finished messages

1. Conceptual Discussion
    a. Diffie-Hellman and and the discrete log problem
    b. Number field sieve
    c. Export-grade cryptography
    d. Man-in-the-middle attacks
    e. TLS
2. Logjam attack: working, time costs, consequences
3. State-level threats to 1024-bit DH, potential consequences
4. Mitigations: Author recommendations and in TLS 1.3.

Logjam


Client C
MitM
Server S
$cr, [\ldots, \texttt{DHE}, \ldots]$
$cr, [\texttt{DHE\_EXPORT}]$
$sr, \texttt{DHE}$
$sr, \texttt{DHE\_EXPORT}$
log_C
$cert_S, \mathsf{sign}(sk_S, [cr \mid sr \mid p_{512} \mid g \mid g^b])$
g^a
$(ms, k_1, k_2) = \mathsf{kdf}(g^{ab}, cr \mid sr)$
$b = \mathsf{dlog}(g^b \bmod p_{512})$
$(ms, k_1, k_2) = \mathsf{kdf}(g^{ab}, cr \mid sr)$
$\mathsf{finished}(ms, log_C)$
log'_C
$\mathsf{authenc}(k_1, \texttt{Data}^{fs})$
$\mathsf{finished}(ms, log'_C)$
$\mathsf{authenc}(k_1, \texttt{Data})$
$\mathsf{authenc}(k_2, \texttt{Data}')$

The attack

- Logjam is reminiscent of the recent FREAK attack, which relied on an implementation bug in RSA key exchange
- Relies on a protocol flaw in TLS ≤1.2, namely its composition of the ephemeral Diffie-Hellman ciphersuites DHE and DHE_EXPORT
- **Downgrade attack**: forcing two participants to use the weakest cipher supported by both parties. so that the attacker can eventually calculate the key.
- Assumption: only the server can continue the session with the client due to its knowledge of the secret key b (therefore the ability to calculate the MAC and decrypt further messages)

The underlying vulnerability

- The structure of the signed **ServerKeyExchange** message containing a 512-bit p_{512} is identical to the message sent during standard DHE ciphersuites, i.e. *the signature of the server does not attest to the negotiated ciphersuite*

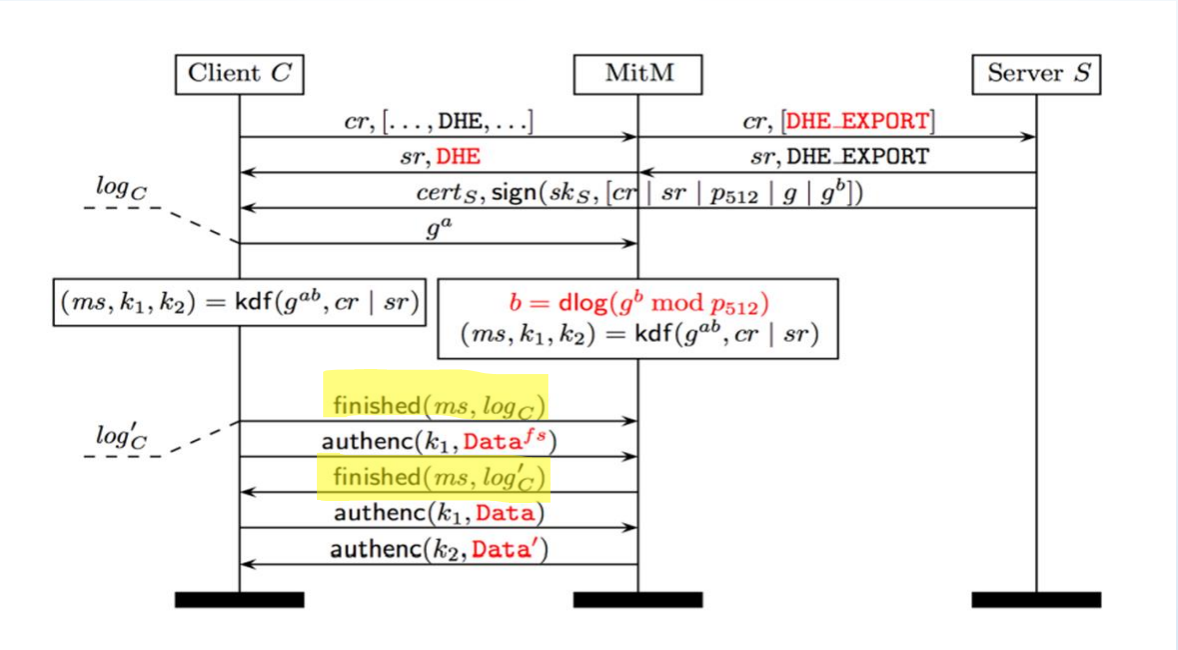
$$ServerKeyExchange = [prime\ p, public\ key\ g^b, signature(p, g, g^b, sr, cr)]$$

- An active MITM attacker can rewrite the client's **ClientHello** to DHE_EXPORT, remove other ciphersuites, and forward the **ServerKeyExchange** message to the client as is
- The client will verify the signature correctly and interpret the export-grade tuple (p_{512}, g, g^b) as valid DHE parameters chosen by the server and proceed with the handshake
- Note that this is possible because the *initial handshake messages are sent over HTTP* (before any encryption or authentication occurs)

Challenges

1. The client and server have different handshake transcripts at the end. The **Finished** messages include the MAC of the handshake to verify that nothing was tampered with

Solution: an attacker who can compute *b* in close to real time can derive the master secret and connection keys, and therefore can compute a valid MAC of their own version of the handshake with the client, thereby completing the handshake with the client and terminating its connection with the server.



Challenges

2. Computing individual discrete logs in close to real time

Solution:

- perform NFS precomputations in advance for the two most popular 512-bit primes
- Downgrade the protocol to use 512-bit keys to allows for practical real-time computation of the secret key using NFS

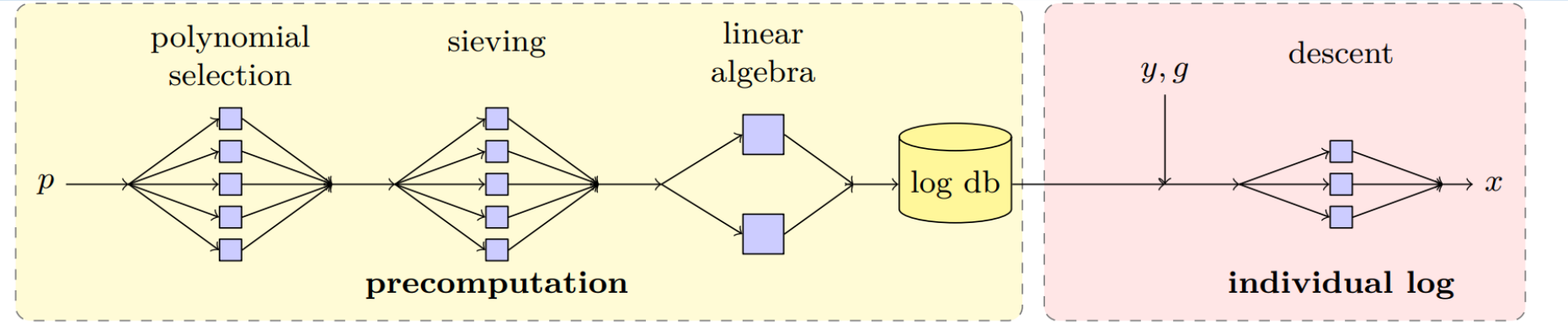


Figure 1: **The number field sieve algorithm for discrete log** consists of a precomputation stage that depends only on the prime p and a descent stage that computes individual logs. With sufficient precomputation, an attacker can quickly break any Diffie-Hellman instances that use a particular p.

https://weakdh.org/imperfect-forward-secrecy.pdf

Challenges

3. Delay handshake completion until the discrete log computation has had time to finish

- **Non-browser clients.** Different TLS clients impose different time limits handshake, before killing the connection. Command-line clients like curl and git often have long or no timeouts
- **TLS warning alerts**. Web browsers have shorter timeouts, we can keep their connections alive by sending *TLS warning alerts:* ignored by the browser but reset the handshake timer.
- **Ephemeral key caching**. Many TLS servers reuse their DH keys for multiple negotiations. An attacker can compute the discrete log of g^b from one connection and use it to attack later handshake. The authors found that 17% of IPv4 hosts reused g^b at least once over the course of 20 handshakes, and that 15% only used one value.
- **TLS False Start**. This extension reduces connection latency by having the client send early application data (such as an HTTP request) without waiting for the server's Finished message to arrive

Time costs: 512-bit cryptanalysis

- NFS Precomputations took a little over 7 days for each prime
- Each resulting database of known logs for the descent occupied about 2.5 GB in ASCII format
- Computing individual logs in real time took a median of 70 seconds.
- This time varied actually between 34 and 206 seconds.
- The times were about the same for each prime.

Consequences

- Scale of the attack: Since generating primes with special properties can be computationally burdensome, many implementations use fixed or standardized Diffie-Hellman parameters
- For both normal and export-grade Diffie-Hellman, the vast majority of servers use a handful of common groups
- The two 512-bit Diffie-Hellman groups they performed precomputations for were used by more than 92% of the vulnerable servers
- 82% of the vulnerable servers used a single 512-bit group, allowing compromised connections to 7% of Alexa Top Million HTTPS sites
- Using the Logjam attack, an attacker with modest resources can hijack connections to approximately 1.6M SMTP, 429K IMAPS, and 454K POP3S email servers

Other Weak and Misconfigured Groups

- **512-bit for non-export DHE**: 2,631 servers with browser-trusted certificates (and 118 in the Top 1M domains)
  - In these instances, active attacks may be unnecessary.
  - If a browser negotiates a DHE ciphersuite with one of these servers, a passive eavesdropper can later compute the discrete log and obtain the TLS session keys for the connection.
- **Non-safe primes**: 4,800 of 70,000 distinct primes scanned were not safe, i.e. $(p - 1)/2$ was composite.
  - not necessarily vulnerable, if g generates a group with at least one sufficiently large subgroup
- **Misconfigured groups:** The Digital Signature Algorithm uses primes p such that $p-1$ has a large prime factor q and g generates only a subgroup of order q. Some servers used Java's DSA primes as p but mistakenly used the DSA group order q in the place of the generator g

1. Conceptual Discussion
    a. Diffie-Hellman and and the discrete log problem
    b. Number field sieve
    c. Export-grade cryptography
    d. Man-in-the-middle attacks
    e. TLS
2. Logjam attack: working, time costs, consequences
3. State-level threats to 1024-bit DH, potential consequences
4. Mitigations: Author recommendations and in TLS 1.3

State-level threats to 1024-bit DH

- 768-bit groups are within reach for academic computational resources
- 1024-bit groups: performing precomputations for a small number of 1024-bit groups is plausibly within the resources of state-level attackers
- The authors offer calculations suggesting that it is plausibly within NSA's resources to perform NFS precomputations for at least a small number of 1024-bit Diffie-Hellman groups
- The precomputation would likely require special-purpose hardware but would not require any major algorithmic improvements beyond what is known in the academic literature
- Although the cost of the precomputation for a 1024-bit group is several times higher than for an RSA key of equal size, a one-time investment could be used to attack millions of hosts

Effects of a 1024-bit Break

- Small number of fixed/standardized groups are used by millions of servers.
- 68.3% of Alexa Top 1M sites support DHE, as do 23.9% of sites with browser-trusted certificates. Of these, 84% use a 1024-bit or smaller group, with 94% of these using one of five groups
- Performing precomputation for a single 1024-bit group would allow passive eavesdropping on 18% of popular HTTPS sites, and a second group would allow decryption of traffic to 66% of IPsec VPNs and 26% of SSH servers
- The authors hypothesize that the plausible break of a small number of 1024-bit groups explains the long-unanswered question raised by the Edward Snowden leaks: the decryption of VPN protocols by the NSA. They also provide speculative evidence for this explanation

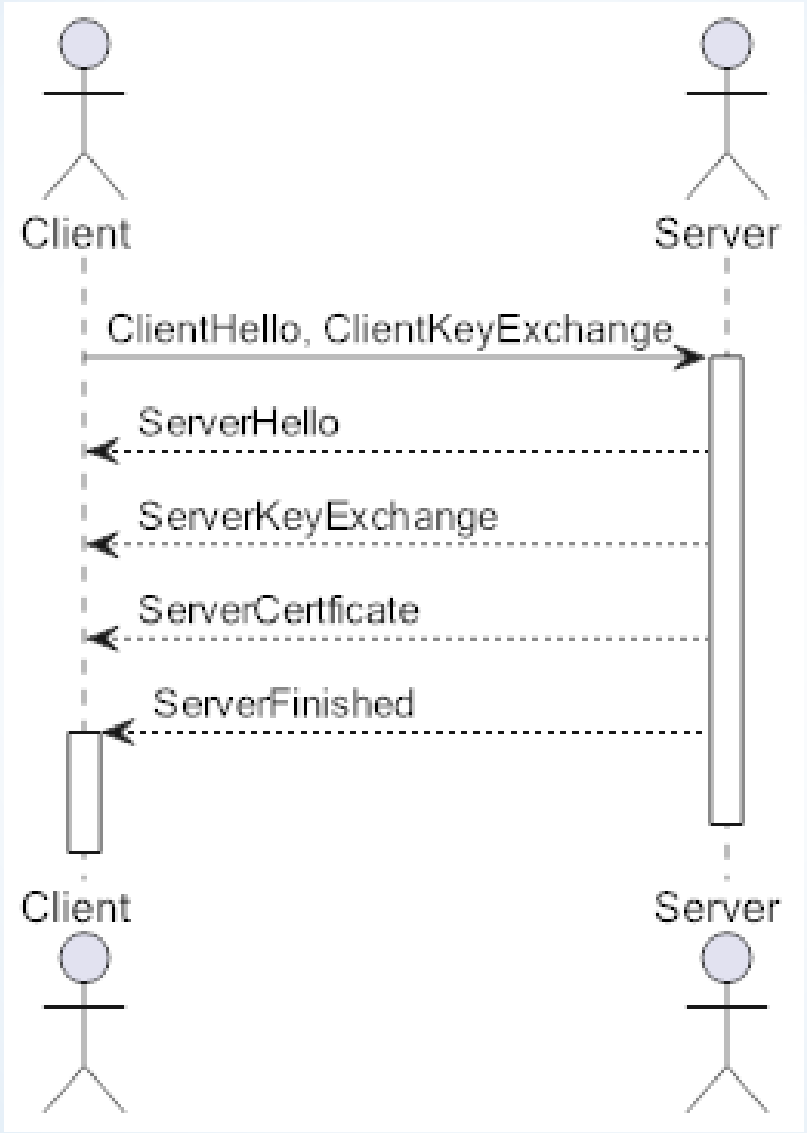
1. Conceptual Discussion
    a. Diffie-Hellman and and the discrete log problem
    b. Number field sieve
    c. Export-grade cryptography
    d. Man-in-the-middle attacks
    e. TLS
2. Logjam attack: working, time costs, consequences
3. State-level threats to 1024-bit DH, potential consequences
4. Mitigations: Author recommendations, and in TLS 1.3

Author Recommendations

1. Transition to elliptic curves
2. Increase minimum key strengths: disable DHE_EXPORT and configure DHE ciphersuites to use primes ≥2048 bits. Browsers and clients should raise the minimum accepted size for Diffie-Hellman groups to at least 1024 bits in order to avoid downgrade attacks
3. Phase out 1024-bit DHE (and 1024-bit RSA) in the near term. Clients should raise the minimum DHE group size to 2048 bits as soon as server configurations allow. Server operators should move to ≥2048-bit groups.
4. Avoid fixed-prime 1024-bit groups. When needed, generating fresh 1024-groups may help, but it is possible to create trapdoored primes that are computationally difficult to detect. At minimum, clients should check that servers' parameters use safe primes or a verifiable generation process. Ideally, the process for generating and validating parameters in TLS should be standardized to thwart the risk of trapdoors

Mitigations in TLS 1.3

- All legacy and dangerous functionality was removed from TLS version 1.3: SHA-1, RC4, DES, 3DES, AES-CBC, MD5, vulnerable DH groups, weak export ciphers
- Server signature is on the entire handshake transcript up to that point
- No more take-out menu: Colossal set of combinations was allowed for negotiation of cipher suites in TLS 1.2 is replaced by a much more compact set.


Client
Server
ClientHello, ClientKeyExchange
ServerHello
ServerKeyExchange
ServerCertficate
ServerFinished
Client
Server

Thanks for your attention!

Questions?