UNIVERSITY OF STUTTGART

Hauptseminar: Technical Details on the Logjam Attack

Li Yang Wu

June 21, 2017

Contents

Abstract

In the field of study of security for computer networks, [Adrian et al., 2015] developed a new attack called Logjam to break many cryptographic systems that use Diffie-Hellman key exchange, where TLS 1.2 was the current standard at that time to establish secure end-to-end connections. This work elucidates technical details on the background of Logjam, in particular Diffie-Hellman key exchange, Number Field Sieve algorithm and TLS handshake, and summarizes the overall results.

1 Introduction

If it comes to network security, Diffie-Hellman key exchange is the standard wildly used in practice and declared to be secure. However, [Adrian et al., 2015] could break the security on many popular and browser certified websites. They demonstrated this so called Logjam attack on the Transfer Level Security protocol that supports Diffe-Hellman as key exchange. They also implemented an attack exploiting the use of small exponents with unsafe primes using a combination of the Pohling-Hellman decomposition and Pollard's lambda method.

Section 2 provides a formal definition of the key exchange problem using asymmetric keys and describes the solution according to [Diffie and Hellman, 1976]. In order to break Diffie-Hellman, [Gordon, 1993] first proposed how the Number Field Sieve algorithm can be used to compute logarithms in prime fields. This method is rather complicated and covers a large part of this work in Section 3. While most methods presented here is about cryptography, Section 4 presents the Transport Layer Security, specified in [Dierks, 2008], which is a security protocol used in the transport layer from the OSI model. This model is the most commonly used one that provides a decomposition of a network system into layers, from physical to application level. Section 5 uses the methods from the previous sections to explain Logjam in short. Section 6 presents an attack proposed by [Van Oorschot and Wiener, 1996] that exploits a specific kind of bad parameter choices made by system designers when Diffie-Hellman key exchange is used. The second result presented in [Adrian et al., 2015] is an estimate on the cost of computing logs of group elements using 1024 bits primes. I will briefly summarize their assumptions for the estimate and their results in Section 7. A conclusion is drawn about what is necessary to make computer networks more secure in the future in Section 8.

2 Diffie-Hellman Key Exchange

[Diffie and Hellman, 1976] was the first paper presenting a practical solution to the privacy and the authentication problem that does not rely on secure communication channels. These kinds of channels are often not given or so inefficient to use that the benefit of telecommunication is nullified. Hence, it was a historical and great advance in computer networking and cryptography. They presented an asymmetric approach with trap door functions that have an exponential cipher-cryptanalyst ratio. Because of this, a security system can choose feasible large private keys that are at the same time infeasible to infer from public keys and encrypted messages w.r.t. computation time.

2.1 Problem Formulation

Given:

- A, B, the communication partners.
- Only insecure communication channels are available.
- $\{M\}$, a finite message space which contains all possible messages M to be exchanged.

Determine:

- $C = (\{E_K\}_{K\in\{K\}}, \{D_K\}_{K\in\{K\}})$, a *public key cryptosystem* with mutual inverse functions
  - $E_K : \{M\} \rightarrow \{M\}$
  - $D_K : \{M\} \rightarrow \{M\}$

  denoted as *encryption* and *decryption transformation* respectively.
- $\{K\}$, a suitable set of keys.
- Such that
  1. $\forall K \in \{K\} : M = D_K(E_K(M))$
  2. $\forall K \in \{K\}, M \in \{M\} : E_K(M), D_K(M)$ are easy to compute.
  3. $\forall^\infty K \in \{K\}, f : \{E_K\}_{K\in\{K\}} \rightarrow \{D_K\}_{K\in\{K\}} : D_K = f(E_K)$ is infeasible to compute.
  4. $\forall K \in \{K\} \exists g : \{K\} \rightarrow (\{E_K\}_{K\in\{K\}}, \{D_K\}_{K\in\{K\}}) : g(K) = (E_K, D_K)$ is feasible to compute.

The goal is therefore to find a key space $\{K\}$ and one transformation generator g to generate public and private keys (E_K, D_K) that fulfil these four conditions above.

The reason we call the public and private keys "encryption" and "decryption" transformations is that these keys can be used for secure message exchange directly. Everyone who wants to send a message M to, say, A just needs to send $E_A(M)$ and A can read $D_A(M)$. This however is slower than cipher algorithms using symmetric keys. For the sake of performance, the solution proposed uses this asymmetric mechanism to exchange a session key, that is then used to encrypt application data.

2.2 Solutions

Key Exchange The solution the authors propose is to draw public and private keys (E, D) from finite fields $\mathrm{GF}(q) \cong \mathbb{Z}/q\mathbb{Z}$, i.e.

$$E, D \in \mathrm{GF}(q) = \{x \mod q \mid x \in \mathbb{Z}\}. \tag{2.1}$$

By choosing the private key as the logarithm of the public key and an arbitrary basis $\alpha \in \mathrm{GF}(q)$,

$$E = \alpha^D \mod q, \tag{2.2}$$

$$D = \log_\alpha E \mod q, \tag{2.3}$$

$$1 \leq D, E \leq q - 1, \tag{2.4}$$

the requirements 3. and 4. are fulfilled. Calculating E from D has logarithmic time complexity in q. Inversely, calculating D from E has a complexity of $\sqrt{q}$. Therefore the cipher-cryptanalyst ratio is exponential. Note that this is not a proven bound. We use the best known algorithms for these problem to get this ratio. Better algorithms have not been found for decades, therefore this method is considered safe. Also, one need to find good primes q in order to assure worst or near worst case log-computation time.

Now if A and B want to exchange a session key, they first choose public and private keys E_A, E_B, D_A, D_B from a number field $\mathrm{GF}(q)$ with a sufficient size q and send each other their public keys over the insecure communication channel. Then, they can calculate the common session key

$$K_{A,B} = \alpha^{D_A D_B} \mod q \tag{2.5}$$

$$= (\alpha^{D_A})^{D_B} \mod q \tag{2.6}$$

$$= {E_A}^{D_B} \mod q \tag{2.7}$$

$$= (\alpha^{D_B})^{D_A} \mod q \tag{2.8}$$

$$= {E_B}^{D_A} \mod q. \tag{2.9}$$

While A calculates the right term in 2.7, B calculates the right term in 2.9 and both of them now have the same session key $K_{A,B}$.

Because of the trap-door property of this mechanism, it allows us to use it for one-way authentication. A signature s of some user A is $D_A(s)$. No one can forge it, since D_A is private. A cannot deny his or her signature, since everyone can confirm $E_A(D_A(s)) = s$.

Cryptanalysis The attacker can compute $K_{A,B}$ either by computing D_A from E_B or D_B from E_A, both are log-problems in finite fields. Thus, the attacker is successful iff he or she can afford exponentially more computation power than A and B.

3 Number Field Sieve Algorithm

As described above, the only known way to break Diffie-Hellman is to compute D from E, i.e. to compute discrete logs for prime fields for a fixed prime q and base α. [Gordon, 1993] first proposed how the *Number Field Sieve* (NFS) algorithm can be used to compute discrete logs. Before that, this technique was known to solve the factorization problem for large primes.

3.1 Approach

The idea is to factorize E and find logs for each of these factors. The logs of these factors can be constructed by the logs of some precomputed logs in the database. This step can be done very efficiently. The difficult problem is to build up such a database for the prime of interest q. But the good thing is that this precomputation only depends on q and not on a specific problem instance E. Once the precomputation is done, the database can be used to compute the log for every $E \in \mathrm{GF}(q)$.

The algorithm for NFS in general is technically difficult to understand. This means that there are many mathematical objects we must deal with and keep in mind at the same time. For simplicity I neglected some details for a better reading, while the drawback is that I cannot show all intermediate steps and instead need to refer to the original paper from time to time.

3.2 Precomputation

The precomputation consists of three steps. In Polynomial Selection, we find an appropriate polynomial representation for $\mathrm{GF}(q)$. Using this representation, we find equations that relate certain numbers with smaller primes in $\mathrm{GF}(q)$ in the Sieving step. These equations form matrices that can be used to find relations that lead to the logs of the base α in the Linear Algebra step. But first, one assumption must be made on α.

3.2.1 Smoothness Assumption

Define that an integer number x is B-smooth, if all prime factors of x are smaller than B. NFS assumes the base α to be B-smooth where B is an upper bound for the prime values in the factor base (which is the output of the precomputation). B is restricted by our computation capabilities. If α is not B-smooth, we can use a B-smooth α' instead. Then, we can compute $\log_{\alpha'} \alpha$ and formulate our original problem with

$$\log_\alpha E \equiv \frac{\log_{\alpha'} E}{\log_{\alpha'} \alpha} \mod (q-1) \,. \tag{3.1}$$

If α' is not a generator for $\mathrm{GF}(q)^*$, the logs on the right hand side might not exist. α' can be checked for feasibility by factoring $(q-1)$ with the NFS factoring algorithm and check

$$(a')^{\frac{q-1}{p}} \not\equiv 1 \mod q, \quad \forall p \parallel q \,, \tag{3.2}$$

where $p \parallel q$ denotes that p is a prime divisor of q. The reason why α or α' needs to be B-Smooth is that the factor base needs to have at least one inhomogeneous relation for the logs of the factor base, using the equation

$$\log_\alpha \alpha = 1 \equiv \sum_{p^t \| a} t \log_\alpha p \mod (q-1) \,, \tag{3.3}$$

which is not guaranteed to be the case if not all log factors are considered.

3.2.2 Polynomial Selection

The goal in the first step is to find a polynomial representation of $\mathrm{GF}(q)$. This has the advantage that we can express relations and encode informations in terms of linear equations to reason about them, which are the tasks that actually will be done in the next steps.

First, we represent $\mathrm{GF}(q)$ as $\mathbb{Z}/q\mathbb{Z}$ where elements are identified with their least non-negative residues. Then, choose an integer m and polynomial $f : \mathbb{Z} \to \mathbb{Z}$ of degree k such that f is monic, irreducible over $\mathbb{Q}$ and $f(m) \equiv 0 \mod q$. We can find such a polynomial f by choosing m of suitable size and find coefficients $a_1, ..., a_k$ such that

$$q = \sum_{i=0}^{k} a_i m^i \,. \tag{3.4}$$

It is required that $q \nmid \Delta_f$, where Δ_f is the discriminant of f. If the choice of f or m leads to $q \mid \Delta_f$, then try another m or alter f slightly, for example add m to a_i and subtract 1 from a_{i+1}.

Next, let $\gamma \in \mathbb{C}$ be the root of f, define $K = \mathbb{Q}(\gamma) = \mathbb{Q} \cup \{\gamma\}$ and O_K the ring of integers in K. Since $\gcd(q, \Delta_f) = 1$, $\hat{q} := (q, \gamma - m)$ is a first-degree prime factor of $q \in O_K$. It follows that $O_K/\hat{q} \cong \mathrm{GF}(q)$ and we can define a homomorphism $\phi : \mathbb{Z}[\alpha] \to \mathbb{Z}/q\mathbb{Z}$, providing us the representation we aimed for. We denote a prime ideal $s \in O_K$ as *good* if s does not divide the index $[O_K : \mathbb{Z}[\gamma]]$, otherwise we denote it as *bad*.

The actual factor base $\mathcal{B}$ for which we want to find the logarithms consists of two parts: $\mathcal{B}_\mathbb{Q}$ the set of rational primes and $\mathcal{B}_K$ the set of good prime ideals in O_K, where all of these elements are bounded in size, i.e. $p, \|p'\|_1 \leq B$, $\forall p \in \mathcal{B}_\mathbb{Q}, p' \in \mathcal{B}_K$. Note that B is the same bound for our base α (from our primary problem definition: Find D for $E = \alpha^D \mod q$). Also, let $\mathcal{B}' \subset \mathcal{B}_\mathbb{Q}$ denote the set of prime factors of α.

3.2.3 Sieving

In this step we try to find linear equations that are informative in the sense that they are related to the the logarithms in $\mathcal{B}$. For that, sieve through integer pairs (c, d) that are co-prime for which $c + dm$ and $c + d\gamma$ are S-smooth where $S = \prod_{s \in \mathcal{B}_\mathbb{Q}}$. For this smoothness criteria, we can formulate equations

$$c + dm = \prod_{s \in \mathcal{B}_\mathbb{Q}} s^{w_s(c,d)} \tag{3.5}$$

and

$$|N(c + d\gamma)| = \prod_{s \in \mathcal{B}_{\mathbb{Q}}} s^{v_s(c,d)} \ , \tag{3.6}$$

where $N(c+d\gamma) = (-d)^k f(-\frac{c}{d})$ (see original paper [Gordon, 1993] for explanation), for some $w_s, v_s \in \mathbb{Z}^+$.

From here we can derive factorizations for primes in $\mathcal{B}_K$ with the fact (not proven here) that there is an unique ideal $\hat{s} \in \mathcal{B}_K$ lying over s and dividing $c+d\gamma$. Let $v_{\hat{S}}(c, d) = v_s(c, d)$ for this ideal and be zero for other ideals in $\mathcal{B}_K$ of norm s. Then we have the following

$$c + dm = \prod_{s \in \mathcal{B}_{\mathbb{Q}}} s^{w_s(c,d)} \tag{3.7}$$

and

$$c + d\gamma = \prod_{\hat{s} \in \mathcal{B}_K} \hat{s}^{v_{\hat{s}}(c,d)} \ . \tag{3.8}$$

Repeat the steps described so far to find more pairs (c, d) to build equations 3.7 and 3.8 until we have at least $|\mathcal{B}|$ of these equations. For each equation, build a matrix, where its rows are the w_s's and $v_{\hat{s}}$'s, we call this set $\mathcal{A}$. We modify each $A \in \mathcal{A}$ by extracting only those columns where elements are in $\mathcal{B} - \mathcal{B}'$. This means, we are only interested in solutions for primes $s \in \mathcal{B}'$. $\mathcal{A}$ is now the input for the next step.

3.2.4 Linear Algebra

For each $A \in \mathcal{A},\ A = \mathbb{Z}^{S \times T},\ S > T$, where each row has at most X non-zero entries, find a linear relation over $\mathbb{Q}$ for the rows of A. This is accomplished in four steps, shown for the case $S = T+1$. This procedure was developed by Carl Pomerance (*1944) which was not separately published, instead only for helping out in [Gordon, 1993].

Step 1 Attempt to compute the rank $r = \mathrm{rk}(A)$. For that, choose a random prime $q_0 \leq XT \log T$. Later, we will check if that choice was feasible. If not, come back here and sample another one. Then, use Gaussian elimination mod q_0 to find the rank r_0 of $A \mod q_0$. Rearrange the rows so that the first r_0 rows are linearly independent mod q_0. Now the rearranged matrix is

$$A' = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{T+1} \end{pmatrix} = \begin{pmatrix} a'_{11} & a'_{12} & \cdots & a'_{1T} \\ a'_{21} & a'_{22} & \cdots & a'_{2T} \\ \vdots & \vdots & \ddots & \vdots \\ a'_{T+11} & a'_{T+12} & \cdots & a'_{T+1T} \end{pmatrix} . \tag{3.9}$$

The result of the Gaussian elimination provides a submatrix $\hat{A} = \mathbb{Z}^{r_0 \times r_0}$ of the first r_0 rows and columns that is nonsingular mod q_0,

$$\hat{A} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{r_0} \end{pmatrix} = \begin{pmatrix} a'_{11} & a'_{12} & \cdots & a'_{1r_0} \\ a'_{21} & a'_{22} & \cdots & a'_{2r_0} \\ \vdots & \vdots & \ddots & \vdots \\ a'_{r_01} & a'_{r_02} & \cdots & a'_{r_0r_0} \end{pmatrix} . \tag{3.10}$$

STEP 2 Attempt to express v_{r_0+1} as a linear combination of $v_1, ..., v_{r_0} \mod q$ for each $q \leq XT \log T$. For some q it will be successful, for others not. Denote the product of all successful primes as $\boldsymbol{P}$ and the product of all unsuccessful primes as $\boldsymbol{P'}$. If $\boldsymbol{P'} > XT \log T$, then repeat Step 1 (sample a new random prime number). Otherwise, continue with the next step.

STEP 3 Attempt to find the determinant $\det(\hat{A})$ of $\hat{A}$. For each prime $q|\boldsymbol{P}$, use the Wieldemann's probabilistic determinant algorithm [Wiedemann, 1986] to compute an integer $D_q \in \{0, ..., q-1\}$, which is the determinant of $A \mod q$ with probability of at least $1-(XT)^{-2}$, i.e.

$$\mathbb{P}(D_q = \det(A \mod q) \mid A, q) \geq 1-(XT)^{-2} . \tag{3.11}$$

For each D_q we can write an congruence

$$D_0 \equiv D_q \mod q, \quad \forall q|\boldsymbol{P} \tag{3.12}$$

and use the Chinese remainder theorem to compute D_0 fulfilling these congruences and being closest to zero. Repeat this step until we have found a value that lies within the bound

$$0 < |D_0| \leq (X^{\frac{1}{2}}T)^T . \tag{3.13}$$

STEP 4 In the final step we try to find the actual coefficients $c_1, ..., c_{r_0}$ for the linear relation between v_{r_0+1} and the rest of the rows. Again, use the Chinese remainder theorem to find these coefficients closest to zero such that

$$D_0 v_{r_0+1} \equiv \sum_{i=1}^{r_0} c_i v_i \mod \boldsymbol{P} . \tag{3.14}$$

In case one of these coefficients c_i exceeds $(X^{\frac{1}{2}}T)^T$ in absolute value, repeat Step 3 to find another D_0. Otherwise, we have found the relation

$$D_0 v_{r_0+1} = \sum_{i=1}^{r_0} c_i v_i . \tag{3.15}$$

It can also be shown that the expected runtime of this algorithm is

$$\mathcal{O}(X^2 T^3 \log^3 T) . \tag{3.16}$$

3.2.5 Results

Here I sum up the final result not showing details, as it needs more technical preparations that I neglected for simplicity. Feel free to look at the original paper for a complete derivation of the solution.

After applying the algorithm described in the Linear Algebra step on the matrices $\mathcal{A}$ resulted form the Sieving step, we have some new equations

$$\prod_{(c,d)} (c+dm)^{x(c,d)} = U \, , \tag{3.17}$$

where $x(c,d)$ is the solution for the matrix constructed from (c,d) and U is a unit in O_K. After some intermediate steps, we can express the logs of primes $s \in \mathcal{B}$ as z_s (which is a black box here) and state

$$\prod_{s\in\mathcal{B}'} s^{z_s} \equiv 1 \mod q \, . \tag{3.18}$$

Taking the log, we have

$$\sum_{s\in\mathcal{B}'} z_s \log_\alpha s \equiv 0 \mod (q-1) \, . \tag{3.19}$$

With more than $|\mathcal{B}'|$ such congruences, we can solve these with equation 3.3 and obtain the logs for all primes $s \in \mathcal{B}'$.

3.3 Individual Log Calculation

Now that we have our database ready, we can calculate a private key D given E. The first step is to find a sort of a decomposition of E by sampling random integers $l \in 1,..,q-1$ until we have found an l that satisfies

$$\alpha^l E \equiv p_1 p_2 \cdots p_t \mod q \, , \tag{3.20}$$

where each p_i should be "small", for example $\leq q^{\frac{1}{k}}$. If the discrete logs x_i of each q_i were known, we could compute D with

$$D = \left(\sum_{i=1}^{t} x_i \right) - l \mod q \, . \tag{3.21}$$

Finding the log for a certain p_i is a similar procedure as the precomputation, where we computed the logs of factors of α. We need to find a suitable polynomial that can be used to express relations between p_i and the factor base $\mathcal{B}'$ in form of linear equations. These equations can also be found by sieving through co-prime pairs (c,d) and cancelling out factors not in $\mathcal{B}'$. This gives us the desired result where p_i can be represented by factors in $\mathcal{B}'$, i.e.

$$p_i \equiv \prod_{s\in\mathcal{B}'} s^{z'_s} \mod q \tag{3.22}$$

and thus

$$\log_\alpha p_i = \sum_{s \in \mathcal{B}'} z'_s \log_\alpha s \mod (q-1) , \tag{3.23}$$

where z'_s is some term that comes from intermediate calculation steps that I neglected for simplicity. As described above, the private key D can now be calculated from the $\log_\alpha p_i$'s.

Note that improved versions of the NFS algorithm for logarithm calculation exist, such as the one proposed in [Joux and Lercier, 2003].

4 Transport Layer Security

The document [Dierks, 2008] presents the full Transport Layer Security (TLS) protocol specification. I summarize the handshake section from there because the Logjam attack exploits the protocol flaw exactly here. An abbreviated message flow diagram is shown in Figure 4.1.

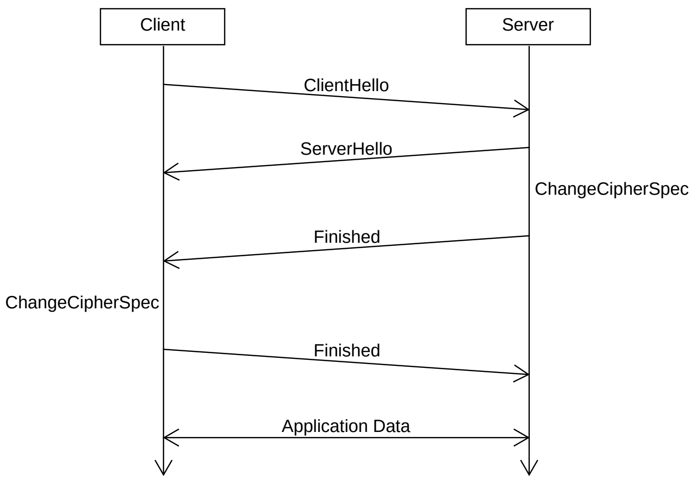

Figure 4.1: The message flow diagram of the TLS handshake, abbreviated.

4.1 Hello Messages

These messages initiate the handshake and exchange information about which cipher methods are supported on each side.

Hello Request This message is optional and can be sent by the server at any time. The meaning is that the server wishes to renegotiate the connection. It expects to receive the `ClientHello` message next.

CLIENT HELLO With this message, the client wants to establish or re-establish a connection to the server. Several information are sent that are used for negotiation. The `client_version` tells which TLS is used by the client. A random structure that consists of a time stamp and some random bytes are typically used for security checks, for example avoiding replay attacks. The client also offers a list of cipher and compression methods the server can choose from. It can use a session id to reuse security agreements from earlier or other session to abbreviate the handshake.

SERVER HELLO The server replies symmetrically by sending an `server_version` which is lower or equals the `client_version` and also a random structure that must not depend on the client's random structure. It selects the algorithms for cipher and compression. If none of these algorithms match, the handshake fails. If the client wished to reuse a session, the server sends the same session id back, otherwise it generates a new one.

SERVER HELLO DONE This message indicates the end of a `ServerHello` message, because its parameters do not have fixed lengths. The client should check all these parameters and also the server's certificate.

4.2 CERTIFICATION

CERTIFICATE REQUEST The server can request a certificate from the client if this is provided from the security method used. There are specific certificate types for the Diffie-Hellman key exchange method. The client needs to sign its certificate and make sure that the type matches the required one.

CLIENT CERTIFICATE After the client received the `ServerHelloDone` and `RequestCertificate` message from the server, the client sends this message. When there are no suitable certificates available, the content of this message should be empty. Then, the server decides to continue or to deny the handshake.

CERTIFICATE VERIFICATION When the server receives the certificate from the client, which turns out to be valid, the server sends this message to confirm the acknowledgement.

4.3 KEY EXCHANGE

TLS supports RSA and DH encryption. After the client has sent its certificate or, if it was necessary, after `ServerHelloDone` was received, the client sends its public key to the server. The procedure for exchanging the session key basically goes according to Section 2.

4.4 Finished

These messages are the last ones in the handshake phase and the first ones that are encrypted with the newly exchanged session key. One participant sends some encrypted content and the other answers with a proof that it can read the content, thus the handshake was successful. Now application data can be send supposedly secure.

5 The Logjam Attack

The previous Sections explained the background for the Logjam attack more in detail than in the original work [Adrian et al., 2015]. Now I summarize the attack in short to show how everything comes together.

For preparation, use NFS precomputation to build log databases for some commonly used 512 bit primes (Section 3.2). When the handshake (Section 4) we want to attack begins, catch the `ClientHello` message as man in the middle and rewrite the offered cipher suit such that it only contains all weak ($\leq$ 512 bit) DH methods supported by the server. Since the client expects the server to choose from a strong cipher suit, we need to rewrite the `ServerHello` message and replace the fake cipher suit with one that is strong but methodically matches the fake one. When keys are exchanged (Section 2), the client cannot detect any problems because the structure of these messages are the same regardless of which prime field is used. In the last step of the handshake, it is checked via `Finished` messages whether the communication partner knows the secret. The attacker computes the secret (Section 3.3) and replies accordingly. The client now beliefs that the attacker is the server and thereby lost its security.

The success of this attack depends on whether the servers being attacked support number fields of small primes or not. During the experiment it turned out that actually many servers do support those and some of them are even certificated being secure.

6 Exploiting Use of short Exponents

Some application designers optimize the computation speed of α^D by choosing small D's. This can be insecure, if the the prime q is not chosen safe, which means that the subgroup generated by α in $\mathrm{GF}(q)$ has small factors. The authors of the Logjam attack also implemented an attack proposed by [Van Oorschot and Wiener, 1996] and performed it on servers that have this weakness. This algorithm combines the Pollard's lambda method with the Pohlig-Hellman decomposition. In the following, these two methods are described, before presenting the attack.

6.1 Pollard lambda

This method is a randomized one that solves a special case of the log problem $E = \alpha^D \bmod q$, where the solution D is known to be in range $b < D < b + w$ for some $b, w \in \mathbb{N}$. Using some random function $f : \mathbb{N} \to R$, with $R \subsetneq \mathbb{N}$ being the range we want to randomize over, ideally tuned to maximize the probability of a success, two sequences of numbers are generated:

$$\begin{aligned} T &= (y'_0, ..., y'_N), \\ W &= (y_0, ..., y_M) . \end{aligned} \tag{6.1}$$

Sequence T starts with the value $y'_0 = g^{b+w}$ and generates the remaining elements as $y'_{i+1} = y'_i g^{f(y'_i)}$. For W, $y_0 = E$ and $y_{i+1} = y_i g^{f(y_i)}$. The log distance between y'_0 and y'_N that sets both in the following relation

$$\log_\alpha(y'_N) = \log_\alpha(y'_0) + d'_N \ , \tag{6.2}$$

is

$$d'_N = \sum_{i=0}^{N'-1} f(y'_i) \mod (q-1) \ . \tag{6.3}$$

and for W analogously. The T sequence stops at some N, while the W sequence stops if there is a hit at some point M such that $y_M = y'_N$ or if d_M exceeds $w + d'_N$. In the first case, the solution

$$D = b + w + d'_N - d_M \mod (q-1) \tag{6.4}$$

is found. Otherwise this iteration failed and we need to restart this procedure.

6.2 Pohlig-Hellman Decomposition

This method reduces the complexity of the log problem by reducing the size of the problem instance. The idea is to find a good way to infer the original logarithm from smaller logarithms efficiently. Then, the remaining problem is to use existing methods to compute these smaller logarithms. However, the requirement is that the prime power factorization

$$q - 1 = \prod_{i=1}^{v} p_i^{c_i} \tag{6.5}$$

is given. The procedure involves two steps. First, compute $E^{\frac{n}{p_i^{c_i}}}$, $i = 1..v$. Denote $E' := E^{\frac{n}{p_i^{c_i}}}$ and $\alpha' := \alpha^{\frac{n}{p_i^{c_i}}}$. Because

$$E' = \alpha'^D \ , \tag{6.6}$$

the reduced problem definition is now

$$E' = \alpha'^{x_i} \mod (p_i), \quad i = 1..v \ . \tag{6.7}$$

Compute, for example with Pollard lambda method, each x_i. With the Chinese remainder theorem we can compute D with the congruences

$$D \equiv x_i \mod p_i \ , \tag{6.8}$$

because

$$\prod_{i=1}^{v} p_i^{c_i} = q - 1 \ . \tag{6.9}$$

6.3 Improved Attack

Since $q-1$ does not always decompose in a way that every x_i is feasible to compute, we cannot use Pohlig-Hellman to its full extend. But we can still extract some information from those factors p_i that are B-smooth, where B is restricted by our computation capabilities. In fact, we can recover $D \mod B_r$ from those x_i that are feasible to compute, where

$$B_r = \prod_{i=1}^{r} x_i, \quad r \leq v,\ i < i+1\ . \tag{6.10}$$

If $B_r > D$, the problem is solved. Otherwise we have $k = \log B_r$ bits of information about D that can be used to reduce the computation time of finding D.

It can be shown that $k \approx \log B$. Because typically $B < D$ in practice, the expectation is that work is not finished after computing B_r. Let $z := B_r$ and $n := q-1$, rewrite $n = zQ$, where Q is the product of the remaining factors q_i that are not B-smooth. We can compute $V = D \mod z,\ 0 \leq V < z$ with a partial Pohlig-Hellman decomposition and represent D in the following way:

$$D = Az + V\ , \tag{6.11}$$

where A is now the unknown to be determined. The problem formulation for finding A can be derived as follows:

$$\begin{aligned} y = g^D = g^{Az+V} \quad &\Rightarrow \quad A \leq \tfrac{D}{z} \\ &\Rightarrow \quad A \in \{0, ..., 2^c\},\ c \approx u-k \\ &\Rightarrow \quad \tfrac{y}{g^V} = g^{Az} = (g^z)^A \\ &\Rightarrow \quad y^* = h^A, \quad y^* := \tfrac{y}{g^V}, h := g^z\ . \end{aligned} \tag{6.12}$$

Now we can use Pollard's lambda method to compute A, because A is known to lie in a bounded range $\{b, ..., b+w\} = \{0, ..., 2^c\}$, fulfilling the requirement. With A found we also found D.

Runtime Complexity Let u be the number of bits of the binary representation of D, i.e. $u = \log D$. Before this method was discovered, the best attack was known to run at $\mathcal{O}(2^{\frac{u}{2}})$ time. This method has an overall cost of $\mathcal{O}(2^{\frac{u}{4}})$, which made key exchange systems that expose these properties even more insecure.

7 Security of 1024 bit Groups

The second part of [Adrian et al., 2015] estimates the cost of attacking DH key exchange with strong 1024 bit groups using NFS, motivated by the leaks from Edward Snowden, a whistle blower and former employee of the *National Security Agency* (NSA). These leaked documents claim that NSA is already breaking cryptographic systems that are considered to be secure.

Based on the most recent records in log computation and factorization at that time, the authors extrapolated a cost of 45 million core-years to do the precomputation for a group using 1024 bit primes and 30 core-days for individual log computation. Assuming specialized hardware to do the job, they estimate a cost of \$8M to buy enough hardware to do the computation in one year just for Sieving and another \$11B to buy multiple Titan supercomputer clusters for the Linear Algebra step. This sum is indeed within the budget that the U.S. government planned for a cryptologic program, where the NSA is part of. Additionally, the leaked documents also contain an architecture of the decrypt infrastructure, which is perfectly consistent with NFS breaking Diffie-Hellman.

If this estimation is correct, then most servers on the world wide web can be attacked successfully by the NSA, as the documents claim.

8 Conclusion

The inventors of Logjam gave recommendations to system designers on how to improve the security of their systems due to the success of this attack. Software engineers and cryptography experts need to work together more closely to establish a better sensitivity for security overall. Technically, the most secure groups nowadays are elliptic curves. On top of that, it has a much better cipher-cryptanalyst ratio than "$\bmod q$"-DH or RSA. But it is mathematically hard to understand, which is why it is not wide spread. This obstacle must be overcome as it is really worth the effort. Insecure legacy systems that support weak groups must be replaced by up-to-date ones, which requires to renew security standards at a much higher rate, where collaboration between service providers and browser developers needs to be intensified. This also counteracts state-level threats that have high resource capacities always ready to perform the latest state-of-art attacks at large scales.

References

Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., and Zimmermann, P. (2015). Imperfect forward secrecy: How diffie-hellman fails in practice. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 5–17, New York, NY, USA. ACM.

Dierks, T. (2008). The transport layer security (tls) protocol version 1.2.

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654.

Gordon, D. M. (1993). Discrete logarithms in gf(p) using the number field sieve. *SIAM Journal on Discrete Mathematics*, 6(1):124–138.

Joux, A. and Lercier, R. (2003). Improvements to the general number field sieve for discrete logarithms in prime fields. a comparison with the gaussian integer method. *Mathematics of computation*, 72(242):953–967.

Van Oorschot, P. C. and Wiener, M. J. (1996). On diffie-hellman key agreement with short exponents. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 332–343. Springer.

Wiedemann, D. (1986). Solving sparse linear equations over finite fields. *IEEE transactions on information theory*, 32(1):54–62.

Erklärung:

Hiermit erkläre ich, dass ich die vorgelegte Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe.
Die Arbeit habe ich bisher keiner anderen Stelle zur Bewertung in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.

Stuttgart, 21.06.2017	[illegible]
Ort, Datum	Unterschrift

Declaration:

I hereby declare that the submitted work is my own unaided work. All direct or indirect sources used are acknowledged as references.
This work was not previously presented to another examiner in the same or in a comparable version and has not been published.

Stuttgart, June 21, 2017	[illegible]
Place, Date	Signature