Indiscreet Logs: Diffie-Hellman Backdoors in TLS

Kristen Dorey
Western University, Canada
kdorey@uwo.ca

Nicholas Chang-Fong
Western University, Canada
nchangfo@uwo.ca

Aleksander Essex
Western University, Canada
aessex@uwo.ca

***Abstract*—Software implementations of discrete logarithm based cryptosystems over finite fields typically make the assumption that any domain parameters they encounter define cyclic groups for which the discrete logarithm problem is assumed to be hard. In this paper we explore this trust assumption and examine situations where it may not be justified. In particular we focus on groups for which the order is unknown and not easily determined, and explore the scenario in which the modulus is trapdoored to make computing discrete logarithms efficient for an entity with knowledge of the trapdoor, while simultaneously leaving its very existence as matter of speculation to everyone else.**

We conducted an investigation of discrete logarithm domain parameters in use across the Internet and discovered a multitude of instances of groups of unknown order in use in TLS and STARTTLS spanning numerous countries, organizations, and implementations. Although our disclosures resulted in a number of organizations taking down their suspicious parameters, none were able or willing to rule out the possibility that their parameters were trapdoors, and obtaining conclusive evidence in each case could be as hard as factoring an RSA modulus, highlighting a key feature of this attack method—deniability.

I. Introduction

Finite fields underlie a number of cryptographic primitives and protocols such as DH/DHE key exchange, DSA signatures, ElGamal encryption, and others. DHE in particular, though in steady decline over recent years, is still widely supported. In contrast to elliptic curve variants of these cryptosystems, it is common for finite field crypto implementations not only to support custom groups, but to accept almost *any* group parameters they are presented with. The potential consequences of working in weak groups are well known: the discrete logarithm problem is efficient when groups are of sufficiently low or smooth order. Being able to ascertain the size and primality of a group's order, therefore, would seem to be a critical functionality. It is not. All implementations of finite field cryptography we examined perform little or no validation whatsoever.

What this means is software implementations implicitly trust that the given parameters form a cyclic group of sufficiently large, non-smooth order. At first glance this seems a reasonable assumption, since the party with the private key typically chooses these parameters and communicates them to the other party via an authenticated channel. However, we examine a number of possible scenarios in which weakened parameters could be maliciously injected for the purposes of creating a persistent backdoor.

Significance. We argue backdoored parameters are interesting for a number of reasons:

- **Efficient.** Optimally weak parameters can be chosen to allow near instantaneous recovery of DH shared secrets;
- **Pervasive.** Attacks work on the examined implementations with no modification to existing source code—at either end point;
- **Surreptitious.** Implementations provide little or no means of validating parameters following their initial creation, and therefore detecting weak parameters is potentially difficult.
- **Deniable.** The very existence of a backdoor can be deniable. We uncovered a number of composite moduli online and organizations we contacted declined to comment on how the parameters came to be used.

Summary of Results. We found major implementations of finite field based discrete logarithms have a systematic vulnerability to weak groups and use bad parameter hygiene, and we examine the resulting security ramifications:

- We conducted an Internet wide survey of DHE support uncovering hundreds of TLS- and STARTTLS-enabled web and mail servers using composite moduli with no smooth factors. These potentially backdoored parameters were found across a range of protocols, including HTTPS, SMTP, SMTPS, IMAPS, and POP3S, and spanned over 30 countries and a diverse set of organizations. Some of these servers used generators of partially smooth order, which allowed us to recover large portions of the private key. We additionally found 1.6M servers offering non-safe prime groups of unknown order.
- We discuss how TLS 1.2 and earlier is vulnerable to DHE "downgrade" style attacks, in which an adversary that can exploit trapdoored parameters can force a DHE ciphersuite to be negotiated as long as both parties support it. We present several possible attack vectors to deliver these malicious parameters: directly attacking the server or TLS endpoint, or by attacking the software upstream.

NDSS '17, 26 February - 1 March 2017, San Diego, CA, USA
Copyright 2017 Internet Society, ISBN 1-891562-46-0
http://dx.doi.org/10.14722/ndss.2017.23006

- We disclosed the vulnerability to 17 companies, resulting in a security advisory (CVE-2016-5774). Interestingly, the organizations we spoke to declined to explain how composite moduli came to be used in their DHE configurations.

We cannot be certain that backdoored DHE parameters are in use on the Internet today. A backdoored modulus is deniable, therefore its existence cannot be known absolutely without external confirmation or complete factorization. However, our survey and subsequent analysis show that backdoor use cannot be ruled out.

The remainder of this paper is organized as follows. In § II we discuss related work. In § III we discuss the mechanics of creating TLS backdoors, and present a survey of composite DHE parameters in TLS and STARTTLS identifying potential backdoors. In § IV we demonstrate how weak parameters can be accepted by web clients and show how to exploit this scenario. In § V we discuss possible vectors of attack for injecting weakened parameters. In § VI we report on vulnerability disclosures we conducted with organizations found using composite moduli. Finally in § VII we discuss potential mitigation strategies.

II. Related Work

A. Inadequate DH Parameter Validation

The insecurity of working in small or smooth order subgroups has been known for decades [31], [7], [39]. Despite this many popular implementations of discrete logarithms do little or no parameter validation. Concurrent but independent work by Valenta et al. [38] presents a number of complementary findings regarding the prevalence of weak parameters on the Internet. Whereas our paper focuses on the possibility of trapdoors stemming from small subgroups of hidden order, their work focuses on how the lack of parameter checking can be exploited in the context of DSA style groups.

In other recent work exploiting improper parameter checking in this setting, Bhargavan et al. [13] demonstrated triple handshake attacks on TLS-DHE that relied on the use of non-prime groups which went unchecked on the client end. In a follow-up paper [12] the authors conduct small subgroup attacks on TLS, SSH, and IKEv2 that exploit the lack of public key validation. Mavrogiannopoulos et al. [32] defined a TLS attack used when a server supports explicit elliptic Diffie-Hellman (ECDH) curves. The attack is made possible through incorrect DH parameter validation, as the client views the ECDH parameters as DH parameters. Although recovery of the DH pre-master secret is possible, this attack is very limited as explicit ECDH curves are not supported in the majority of TLS implementations due to their open-source nature.

B. Trapdoors Based on Subgroups of Hidden Order

Henry and Goldberg [27] used trapdoor discrete logarithm groups as a component of privacy preserving reputation systems in anonymizing networks. Recent concurrent but independent work by Wong [41] found examples of composite DHE moduli over HTTPS in the wild. Our study, however, reports on considerably more specimens across a wider range of protocols. In addition, the exploitation by Wong required both the client and server to prefer a DHE ciphersuite, which limits the attack potential since current telemetry data indicates DHE key exchanges account for 1% of TLS handshakes.[1] In § IV-C we describe how an attacker can exploit backdoored parameters to force a DHE ciphersuite to be selected if both parties support it. Additionally we explain how one of the trapdoor constructions presented could be reversed in $O(2^{\frac{\ell}{2}})$ operations instead of the expected $O(2^{\ell})$. We also conducted a number of vulnerability disclosures and discuss vendor responses in § VI.

C. Trapdoors Based on Number Field Sieves

Lenstra [29] and Gordon [24] observed that even if it was established that a particular group had a sufficiently large prime order and that all relevant values were members of the group, it is not necessarily sufficient to ensure the hardness of the discrete logarithm problem if p was maliciously chosen to be "nice" in the context of the generalized number field sieve. Here, a trapdoored prime modulus could be constructed using a polynomial of low-degree and constrained coefficients for the purposes of greatly accelerating the generalized number field sieve (GNFS) sieving and descent steps. Given only p, a verifier would need to deduce this polynomial in order to establish the existence of a backdoor. This approach to building trapdoored DH parameters was previously considered too computationally intensive to perform in practice. Fried et al. [21], however, recently demonstrated the creation of a 1024-bit trapdoored prime modulus using the special number field sieve. Number field sieving can even be applied in some situations where the group was not attacker controlled. Adrian et al. [6] demonstrated a modified version of the GNFS in which an attacker could recover DH private keys from export strength groups.

III. Composite DH Moduli

In this section we discuss how trapdoored DH parameters could be used as a result of poor parameter validation, and outline the potentially trapdoored composite DH parameters found in TLS.

A. Preliminaries

We briefly review some important properties of finite-field based discrete logarithms. Let $\mathbb{G}_q$ be a cyclic group of order q. An element $1 < x < p$ has order q if q is the smallest number such that $x^q \bmod p = 1$. Let $p = qr + 1$ for p, q prime, and let $g = h^r \bmod p \neq 1$ for some $1 < g, h \leq p-1$. We say g generates $\mathbb{G}_q$. By *safe prime group* we denote the case where p is a safe prime, i.e., $r = 2$. Typically implementations choose $g \in \mathbb{Z}_p^*$, i.e., as having order $p-1$ as opposed to $(p-1)/2$. By *non-safe prime group* (also known as a Schnorr group, or DSA group) we denote the case where $r > 2$, i.e. the cyclic subgroup $\mathbb{G}_q$ of $\mathbb{Z}_p^*$. Typically $q \ll p$ allowing for more efficient operations.

Applications of the discrete logarithm problem (DLP) over finite fields, such as Diffie-Hellman, are defined by a set of *domain parameters* $\langle p, q, g \rangle$, where p is the prime modulus, q is the group order, and g is a generator of $\mathbb{G}_q$. Let $\mathbb{G}_q$

[1]https://telemetry.mozilla.org

be a cyclic group of large, prime order. Given two elements $g, y \in \mathbb{G}_q$ the DLP is the problem of finding the unique value $0 \leq x < q$ such that $g^x = y \bmod p$, and is believed to be computationally infeasible when q is sufficiently large and contains no small factors. The *computational Diffie-Hellman* assumption states that given the values $g, g^a, g^b \in \mathbb{G}_q$, it should be computationally infeasible to compute g^{ab}. Finally, the *decisional Diffie-Hellman* assumption states that given $g, g^a, g^b \in \mathbb{G}_q$, it is believed to be computationally infeasible to distinguish between g^{ab} and a random element in $\mathbb{G}_q$.

B. Validating Domain Parameters

Verifying the validity of the domain parameters is sufficient to detect the kinds of weakened or backdoored parameters considered by this paper. Most of the software implementations we examined, however, skip one or more of the following checks:

- **Length**: Check that $|p|$ and $|q|$ are sufficiently large (i.e. $|p| \geq$ 2048-bits, $|q| \geq$ 224-bits as per current NIST guidelines [9]);
- **Primality**: Check p and q are both prime;[2]
- **Group Order**: Check $q|(p-1)$. No mechanism is provided in TLS to communicate group order [19], [34];
- **Group Membership**: Check any asserted group element i.e. generator g, public key, etc. is a member of the group (i.e. $m \in \mathbb{G}_q$). Specifically, check $1 < m < p-1$ and $m^q \bmod p = 1$. Note $m = p-1$ is explicitly excluded by the associated NIST standard [8], since it always only has an order of 2, regardless of the choice of p. Safe prime groups working in $\mathbb{Z}_p^*$ can omit the exponentiation by the group size, since all elements $1 < m < p-1$ are part of this group.

C. Successful Connections with Valid-looking Moduli

A backdoored modulus may possibly remain undetected for longer if the weak modulus at least looks valid, e.g., does not end with an even digit. To demonstrate this, we investigated the visual similarity between a safe prime modulus and a deliberately weak modulus, and showed that lack of proper validation allows software implementations to connect with both moduli. As a demonstration, we modified the OpenSSH `\etc\moduli` file to use a deliberately weak modulus. The default OpenSSH moduli file consists of safe primes with short generators like 2 or 5. Although the software does not check group validity, an attack in the context of a version update should allow the parameters to pass casual inspection. The attacker's goal is then to create parameters that also have short generators (and thus are valid *looking*), but are still efficient to solve. Schnorr groups are unlikely to have short generators of small subgroups, and large generators (i.e. the same length as the modulus) would be overtly suspicious. Since OpenSSH does not verify the primality of the modulus, we can instead work with smooth composite moduli. Here discrete logarithms can be made to be efficiently solvable for any generator of any subgroup.

As an example, we set p as the product of all primes up to 1471, excluding 2 and 5 (so it is not obviously prime from inspection in base 2 or 10). This number is 2043 bits and has 231 factors. Multiplying it by 19 will bring the length to a standard 2048 bits. In this case, one of the factors will be 19^2. Table I shows an example of a safe prime modulus and our smooth composite modulus. The lack of proper validation described in § III-B means OpenSSH connects with both the safe prime modulus and our composite modulus designed to allow efficient DLs. The discrete logarithm of a number relative to an arbitrary base (*e.g.,* 2) can be computed individually across each of the factors of p and reassembled using the Chinese remainder theorem (CRT). The discrete log in each of the subgroups can be pre-computed. Computing a discrete log, therefore, can be reduced to 231 look-ups in this dictionary, followed by a single CRT of 231 congruences. Implementing this in Sage we were able to compute discrete logarithms in 4ms on a laptop.

D. Constructing Trapdoors in Groups of Hidden Order

Working in small subgroups is efficient from the attacker's perspective, but comes with two downsides: (1) others can also exploit the weak group, and perhaps more importantly (2) strong evidence exists that the parameters are compromised. A more interesting scenario is to trapdoor the modulus such that only the attacker can exploit it while making its very existence a matter of speculation. In this setting the attacker can use a composite (e.g., RSA) modulus to construct a trapdoor instance of the discrete logarithm problem. Let $n = pq$ for large primes p, q with $\phi = (p-1)(q-1)$. The idea is to work in small subgroups of hidden and smooth order, i.e., such that $(p–1)$ and $(q-1)$ contain smooth factors. A generator is then selected so as to have reasonably low order modulo p and q respectively, allowing the person knowing the factorization of n to solve several independent and efficient discrete logarithms.

Related Constructions. Concurrent and independent to us, Wong [41] also proposes using a hidden subgroup of a composite modulus in the context of trapdoored Diffie-Hellman key agreement. Let $p = 2p_1p_2 + 1$ and $q = 2q_1q_2 + 1$ where p_1, q_1 are sized small enough to allow efficient computation of the discrete log in subgroups of order p_1 and q_1, but large enough to prevent brute forcing the discrete logarithm in a subgroup of hidden order p_1q_1, while p_2, q_2 are large so as to prevent factorization attacks, such as Pollard's p-1 attack. Let $|p_1| = |q_1| = \ell$. A generator g is chosen of the unique subgroup $\mathbb{G} < \mathbb{Z}_n^*$ of order p_1q_1.

The order of g has length 2ℓ. The orders of g modulo p and q respectively are ℓ-bits in length each. Computing a discrete logarithm separately modulo p and q takes $2^{\frac{\ell}{2}}$ operations each using general discrete logarithm algorithms (e.g., Pollard's rho, etc.). With knowledge of the trapdoor, therefore, the attacker can compute a discrete logarithm in $2^{\frac{\ell}{2}+1}$ operations. Without knowledge of the group order, Wong argues an attacker would require 2^ℓ operations to compute a discrete logarithm. As an example, Wong suggests that if g had an order of 200 bits in length (i.e., where $\ell = |p_1| = |q_1| = 100$), then an observer would require 2^{200} operations to compute a discrete logarithm, while an attacker could solve the discrete logarithms separately modulo p and q, requiring $2 \cdot 2^{\frac{100}{2}} = 2^{51}$ operations.

[2]Technically q only must contain a sufficiently large factor.

```
# Time Type Tests Tries Size Generator Modulus

20160522030737 2 6 100 2047 2 DB36277B45EA5615C782C08BF6A290A3D61E6B9690E4A147042113FC1BFC0AE
EC5FB0FF82FC1FEA86E273F667EC387FEF3421FFFC617A70C34B1987986C6B35C715713914AB75932A3D1942ECC0F
324D81BF00D59916B3BFDC7BA432AF5C5DFCF30BF4A2C80B8CA52A9B80E989D3A852BD81A8BD3ADC97497F43C6F0A
90882D9CFA165CF1F735C96428BF9BC32A58B71CF1D4FD48A6D2C616E91BB6E07C5CB0DF0C59DAF79D659C6E53007
843497BBEE5B341D27DE2E2543B8DFEB4DDAE6328EAD441C3F36509C1FA689FE494B0426ADCAF9E567A1C5A330168
9C5CCC55EC4002FAA5D254C2F3C0F8636BEA7019D1CD212B74EE4F273E0B9997720E8AEC5D76B

20160522030739 2 6 100 2047 2 8A4F17035FD10C065879FCC6C6632C15F18E15B6F88CAE2BA8C40D23E3DC2FD
68E8897E12F9FD6C3447B72C1595B2EF56C103162BB6C15AA64761C4258E56D47FE156832F6BB4273A106D2E6310A
9D5E54C497517A928A988A359FB0032BED2FEF690487F6AC6F0B3659A43643A316F601DE73E563F7BC2C37A67E751
DE1916B08FBE92FB9E32E35DC5FD051E9EBC4B2256BC4021DACD2CA816F46C7A5C5D1B298A259C925AB0DC404BCF7
2FDAF04C849DCA4C2F6576FCC586A5B942188312787D971D9BE6D70896A8E8458F3D75D6C8F97CE289688A175F699
B938DBFFC7A349D4130558794936E67C349EF96B83517CB647BADBF012E9BF1B4890E72B70849
```

TABLE I. **OpenSSH moduli file**. One modulus is a valid safe prime (ostensibly) generated by developers. The other is a smooth composite allowing efficient discrete logarithms. OpenSSH will successfully connect with either.

This expectation, as it turns out, is false as shown by Coron et al. [17] in the context of the cryptosystem due to Groth [25], which works in small RSA subgroups of hidden order. Groth's construction is effectively identical to Wong's trapdoor discrete logarithm construction, except is being applied in the context of an encryption scheme. Once again let $n = pq$ for $p = 2p_1p_2 + 1$ and $q = 2q_1q_2 + 1$ for p, q, p_1, p_2, q_1, q_2 prime, and let g generate the unique subgroup $\mathbb{G} < \mathbb{Z}_n^*$ of order p_1q_1. Let h generate a subgroup of order $p_1p_2q_1q_2$. The values (n, g, h) form the public key. The values (p_1, q_1) form the private key. A message m is encrypted as follows:

$$\mathsf{Enc}(m) = g^r h^m \bmod n.$$

for random r. Decryption is accomplished as follows:

$$\mathsf{Dec}(c) = c^{p_1q_1} = (g^r)^{p_1q_1}(h^m)^{p_1q_1} = (h^{p_1q_1})^m \bmod n.$$

The discrete log of $(h^{p_1q_1})^m$ is computed to recover m. This can be efficient if m is small, although Groth also proposed a variant in which p_2 and q_2 are smooth, allowing for the discrete logarithm to be efficiently computed using Pohlig-Hellman. The best attack proposed by Groth [25] factorizes n in time $\mathcal{O}(2^\ell)$, and works as follows. Recall g has order p_1q_1 and that $g^{p_1} \equiv 1 \bmod p$ and that $g^{q_1} \equiv 1 \bmod q$. This gives

$$\gcd(g^{p_1} - 1, n) = q$$

and

$$\gcd(g^{q_1} - 1, n) = p.$$

Thus n can be factorized by computing $\gcd(g^i - 1, n)$ starting at $i = 2^\ell$ and incrementing until a factor is found, requiring at total of $\min(p_1, q_1) - 2^\ell$ operations. Note this approach is independent of the size of factors p_2 and q_2,

Similar to Wong, Groth proposed $\ell = |p_1| = |q_1| = 100$ as a trade-off between security and efficiency. Coron et al., however, demonstrated an attack on Groth's scheme recovering the factors of n in time $O(2^{\frac{\ell}{2}})$ instead of the expected $O(2^\ell)$. Notice here that g in Groth's scheme has the same order as g in Wong's scheme, and thus any attack on Groth's scheme that can recover the factors of n based on g can be directly applied to Wong's scheme revealing the trapdoor. Coron et al. proposes Groth's scheme use $\ell \geq 160$. This is problematic if applied to our trapdoor setting, since it would require the trapdoor owner to compute two discrete logarithms on the order of 2^{80} operations.

Our Trapdoor Construction. Like Groth's attack, Coron et al.'s attack exploits the overall order of g, but cannot directly exploit the order's factorization (since it is unknown). Our strategy, therefore, makes the overall order of g large enough to make factorization attacks infeasible, while smooth enough to still allow efficient computation of DLs by the trapdoor owner.

Let $p = 2p_1 \ldots p_k r_p + 1$ and $q = 2q_1 \ldots q_k r_q + 1$ for prime p, q. Let each p_i, q_i be distinct, randomly chosen primes of length ℓ. Let r_p, r_q be distinct randomly chosen primes. We choose g to generate a group $\mathbb{G} < \mathbb{Z}_n^*$ of order $p_1 \ldots p_k q_1 \ldots q_k$, which gives g an overall order of $2k\ell$ bits.

We size ℓ to be large enough to preclude factorization of n using Pollard's $p-1$, while small enough that solving discrete logarithm instances in subgroups of order approximately 2^ℓ is efficiently computable. Using Pollard's $p-1$ factorization method, n can be factored as follows. Choose some $a \in \mathbb{Z}_n^*$. Let ρ_i be the i-th prime. For each $\rho_i < 2^\ell$:

1) Set $a \leftarrow a^{\rho_i} \bmod n$
2) If $gcd(a-1, n) \neq 1$ and $\neq n$, output factor, otherwise continue.

Factorization is guaranteed after all primes $\rho_i < \ell_b$ have been exponentiated in, corresponding to approximately $\mathsf{li}(2^{\ell_b})$ modular exponentiations, where $\mathsf{li}(\cdot)$ is the logarithmic integral. Henry and Goldberg [27] studied solving discrete logarithms in smooth-order groups using optimized GPU implementations, and suggest $\ell_b = 55$ as sufficient, requiring 1500 years of (non-parallelizable) wall-clock time to factor n, while requiring less than two minutes to compute the discrete logarithm with knowledge of the trapdoor.

We size k to be large enough to preclude factorization of n based on the order of g (as in Coron et al.'s attack), i.e., $2^{\frac{k\ell}{2}}$ operations is computationally infeasible. Following Coron et al.'s suggestion we have $k\ell \geq 160$. As a concrete parameter choice, let p, q each be 1024-bit primes where $p = 2p_1p_2p_3r_p + 1$ and $q = 2q_1q_2q_3r_q + 1$ where p_1, p_2, p_3, q_1, q_2 and q_3, are distinct, random 55-bit primes and r_p, r_q are distinct, random primes of a length sufficient for p, q respectively to be 1024 bits. A generator g is chosen of order $p_1p_2p_3q_1q_2q_3$. Given a DH public value $g^x \bmod n$, recovering private key x requires 6 separate discrete logarithms to be computed in subgroups of order 2^{55}, for a total of approximately $6 \cdot 2^{\frac{55}{2}} \approx 2^{30}$ operations.

Plausible Deniability. One of the most desirable aspects of this attack paradigm is the ability for an attacker to construct a discrete-log trapdoor while maintaining plausible deniability. It is easy to tell that a modulus is composite (when you're looking), but determining group structure without knowledge of the factorization, and hence the likelihood of the existence of a trapdoor, can be made to be computationally infeasible. As we explain in § IV, none of the vendors we contacted about the composite moduli we discovered were able or willing to either confirm or deny the existence of a trapdoor—precisely as an attacker might hope!

One possible explanation for the origin of a composite modulus is that it was simply a random number chosen by accident, or perhaps began as a prime and had a digit or two flipped in an editor. In this case we would expect the resulting value to have a distribution of factors similar to that of a random composite number. We discussed setting $n = pq$ for large primes p, q, but this might arouse suspicion, beyond simply being composite, because it would contain no small factors. Small factors up to some bound b may be recoverable using elliptic curve factorization, and the probability that a random composite number is b-rough (i.e., contains no factors smaller than b) could be used as evidence toward the determination of the existence of a backdoor. One option would be for an attacker to use an RSA modulus as before but multiply in a sequence of naturally increasing factors up to bound b. We leave a heuristic for creating convincing random-looking but trapdoored moduli for future work.

E. Overview of Affected Protocols and Countries

Methodology. In order to find potential backdoors in discrete logarithm implementations, we collected Diffie-Hellman data from two sources. For HTTPS, we downloaded Censys IPv4 scans [20] where only DHE ciphersuites were offered by the client. For DHE-only scans in SMTP/S, POP3/S, and IMAP/S, we ran our own zgrab[3] scans. We investigated both non-safe and composite DH moduli in HTTPS, and focused on composite moduli only in SMTP/S, POP3/S, and IMAP/S. This section focuses on composite moduli; non-safe prime moduli are discussed in § IV-E.

Affected Protocols. Overall, there were over 500 IP addresses in 31 countries using potentially backdoored composite moduli. A summary of moduli properties and the affected protocols are seen in Table II. Out of the seven protocols investigated, composite moduli were found in five: HTTPS, IMAPS, POP3S, SMTP, and SMTPS. Almost all of the moduli were one of two numbers: a 512-bit modulus used in SMTP or a 2048-bit modulus used in HTTPS. This recycling of parameters is common practice; while it does not directly suggest backdoor use, having the same backdoor in hundreds of IP addresses is advantageous for an attacker. At the very least, this moduli reuse proves that weak DH parameters are used in the wild due to lack of DH parameter validation. Table II also shows three moduli with nonstandard lengths of 4255-, 1102-, and 904-bits, indicating further carelessness in parameter choice.

Affected Countries. To see the impact of these composite moduli, we determined each IP address' location using WHOIS queries. The results are seen in Table III. Nearly all the composite moduli were used in HTTPS or SMTP, but

Number	Number of IPs	Modulus Size (Bits)	Affected Protocols	Modulus
1	265	512	SMTP	`da583c16...4774e833`
2	242	2048	HTTPS	`c28992c5...d4681697`
3	28	4255	HTTPS	`4d494942...41674543`
4	5	1102	POP3S	`30818702...47020105`
5	2	1024	HTTPS	`a7790db6...288a9773`
6	2	1024	HTTPS	`cc17f2dc...8e073c6d`
7	2	2048	HTTPS	`8dd38f77...a8fdca8f`
8	1	904	HTTPS	`9ce85640...2220dc53`
9	1	1024	IMAPS, SMTP	`98ea99db...ab2b1b33`
10	1	1024	HTTPS	`d67de440...24218eb3`
11	1	2048	HTTPS	`f5a3da75...f564c113`
12	1	2048	SMTP, SMTPS	`ad85473c...3b2d764b`
13	1	4096	HTTPS	`9152ba0b...85fab358`

TABLE II. THE FREQUENCY, AFFECTED PROTOCOLS, AND OTHER PROPERTIES OF THE COMPOSITE DH MODULI USED IN THE WILD.

Affected Protocol	Number of IPs	Nationality
HTTPS	280	Austria, Bahrain, Bolivia, Canada, Chile, Czech Republic, France, Germany, India, Iraq, Israel, Italy, Japan, Lebanon, Malaysia, Mexico, Netherlands, Nicaragua, Pakistan, Poland, Romania, Saudi Arabia, Singapore, South Korea, Spain, Sweden, Taiwan, United States
IMAPS	1	Japan
POP3S	5	Ukraine
SMTP	267	China
SMTPS	1	Russia

TABLE III. COMPOSITE DHE MODULI BY PROTOCOL AND COUNTRY.

the HTTPS moduli were spread around the world while the SMTP moduli were only located in China. In HTTPS, North American and European countries were most heavily seen. The location spread in HTTPS and the relative moduli abundance in SMTP increases the likelihood that these moduli are backdoors rather than random composites.

F. Composite Moduli Used By Web Servers

We first downloaded a Censys IPv4 scan to investigate DH moduli in HTTPS. In April 2016, there were approximately 43M IP addresses in the HTTPS space, of which approximately 11M supported DH. Over 300,000 distinct DH moduli were observed across these 11M. We observed 5,783 unique non-safe prime moduli across 1.6M IPs, which will be further discussed in § IV-E. We observed 9 unique composite moduli across 280 IPs. We did a comparison to ECDHE and found that of 32 million IPs, all used a standard SECP curve, and that the server public key was a valid point on the curve. This, of course, is consistent with expectation. Discovering composite DHE moduli, on the other hand, was not.

None of the composite moduli observed in HTTPS were export-grade; all were at least 904-bits in length. In May 2016, 46% of these IP addresses chose a Diffie-Hellman ciphersuite

[3]https://github.com/zmap/zgrab

Server	Number of Uses
Apache	95
Apache-Coyote/1.1	3
Apache/2.2.9 (Debian)	3
Apache/2.2.12 (Linux/SUSE)	1
Apache/2.2.15 (CentOS)	3
Apache/2.2.15 (Red Hat)	3
Apache/2.2.16 (Debian)	1
Apache/2.2.22 (Debian)	1
Apache/2.2.22 (Red Hat)	2
Apache/2.2.22 (Ubuntu)	2
Apache/2.4.3 (Unix)	3
httpd/1.00	8
Microsoft-IIS/7.5	2
Microsoft-IIS/8.0	1
Microsoft-IIS/8.5	6
Oracle Application Server 10g	1
Lighttpd	1
Nginx	24
Nginx/1.6.3	1
Nginx/1.9.10	1
Others	16
Not Specified	103

TABLE IV. TYPES OF WEB SERVERS USING COMPOSITE DHE MODULI.

by default, meaning forcing DHE (as described in § IV-C) is not needed in those cases.

To determine if these composite moduli were the result of a specific server implementation, we looked at the types of web servers using these moduli. The breakdown of these servers can be seen in Table IV. Apache servers were used by 125 IP addresses, which accounted for 45% of the IP addresses using composite moduli in HTTPS. Almost the same percentage of IP addresses (37%) did not specify a server. The remaining 21% of servers were spread over Microsoft, Oracle, Lighttpd, Nginx, and other servers specified by their company name. Although Apache accounted for almost half the servers, the version numbers varied or did not exist. This trend was also seen in the other servers specified. Therefore the variety of servers and versions indicate that no one server implementation was responsible for the composite moduli.

The existence of composite moduli cannot be explained by poor entropy during generation, although poor entropy could potentially explain a systematic prime modulus. While it is possible that these composite moduli are pseudoprimes, enabling them to erroneously pass a probabilistic primality test, pseudoprimes occur so infrequently that they would not be a result of poor entropy. This fact coupled with the variety of server implementations means these moduli were potentially generated on purpose.

We then examined the public ownership information of the affected IPs in public databases and in the content of any public web pages. When the IP address owners and webpage content differed, both companies were considered identifiers for the IP address. For example, if one organization was supplied software by another, the second organization could have a logo displayed on the webpage. We decided to focus on companies associated with multiple IP addresses or with at least one active webpage. This left us with 21 companies: A1 Telekom Austria (A1), Amazon Web Services (AWS), Banco de Crédito (BCP), Bloomberg, Blue Coat Systems, Centre national de la recherche scientifique (CNRS), Deutsche Reisebüro (DER) Touristik, ELITE, Expedia, Eyou.net, FTSE Russell, JAMF Software, KDS, KPN, Nederlandse Spoorwegen (NS), NH Hotel Group, Nordea Bank, Santa Clara University (SCU), TravelTainment Germany, United Parcel Service (UPS), Universal Sompo General Insurance, and Universidad Nacional de Educación a Distancia (UNED).

We completed vulnerability disclosures to companies with at least one active webpage in HTTPS and which provided appropriate contact information; these disclosures are discussed in § VI. We also contacted the company with multiple affected IP addresses in SMTP. Companies in the tourism industry, such as TravelTainment and DER Touristik, accounted for about 50% of the IP addresses. The remaining companies were in various industries like education and finance. Most companies, noticeably those with more affected IP addresses, had an active webpage.

To determine the longevity of composite moduli, we tested the 280 IP addresses three times during the course of writing to see if composite moduli were still used. In May 2016, 88% of the IP addresses still used the same composite modulus as before. Of the remaining 12% of IP addresses, about half switched to a prime modulus and half no longer connected under Diffie-Hellman. In June 2016, these statistics remained approximately constant. However, by August 2016, only 39% still used the same composite modulus and 53% used a prime modulus. The remaining 8% no longer connected under Diffie-Hellman, almost the same amount from May and June 2016. The decrease in composite moduli used could be attributed to our vulnerability disclosures and, independently, Wong's [41]. This assumption seemed to coincide with company responses, as many companies changed from composite moduli to prime as their primary response. Despite this, many composite moduli remained in use over months, indicating backdoored DH parameters could go unnoticed for long periods of time.

G. Composite Moduli Used By Mail Servers

Since Censys did not have DH scans for mail servers, we ran zgrab scans in July 2016 on SMTP/S, POP3/S, and IMAP/S in TLS and STARTTLS looking for composite DH moduli. We found 272 IP addresses with composite DH moduli spread throughout IMAPS, POP3S, SMTPS, and SMTP. These results doubled the total number of composite moduli found, showing the problem extends beyond HTTPS.

IMAPS. Although there was only one IP address in IMAPS with a composite modulus, this IP address used the same modulus in SMTP. This modulus is number 9 in Table II. The address is linked to a transportation company in Japan, which supports the trend of HTTPS companies that are not related to security and thus provide an advantageous attack target.

POP3S. There were five IP addresses in POP3S that all used the same composite modulus. This modulus is number 4 in Table II. Although the company could not be determined accurately, the range of IP addresses suggested that only one Ukrainian company was involved.

SMTPS. Although there was only one IP address in SMTPS with a composite modulus, this IP address used the same modulus in SMTP. This modulus is number 13 in Table II. This address is linked to a real estate company in Russia, which is also an industry that provides an advantageous attack target.

SMTP. Almost all the composite moduli in mail protocols were seen in SMTP. Out of 267 IP addresses with composite moduli, 265 used the same composite modulus (number 1 in Table II). The remaining two were the IP addresses seen already in IMAPS and SMTPS. The 265 IP addresses were spread out across China, but all connected to an email service provider called Eyou.net [1]. This company was also contacted in the vulnerability disclosures described in § VI.

H. DH Moduli Factorization

While a well-implemented DHE trapdoor would *not* be exploitable, we set about conducting what partial factorizations of composite moduli we could. We used CADO-NFS and our own custom implementation of Pohlig-Hellman/Pollard's P-1 to recover, in many cases, numerous bits of a private key. We factored the 512-bit composite SMTP modulus (number 1 in Table II) revealing 5 factors:

```
114356381100738840153121389513746326020580788713898181372 \\
757840692493482634612304277048270052450717458185043187444 \\
98415461673127855611205755830392736507955
```

```
= 5 * 11 * 3130497666273667404271 * 132398438917079824212 \\
370893794766672908033 * 501650748974370233413468006002745 \\
013076943662195591458981539797641214671553476408791132267
```

We then factored $(f-1)$ of each factor f revealing the overall underlying group structure. The largest factor has a 280-bit subgroup, which prevented us from performing a complete discrete logarithm as the generator had order close to $p-1$. We were, however, able to recover 129 bits of the private key using Pohlig-Hellman. The servers we examined appeared not to be using short exponents. If, however, a server did use a short exponent such as 160-bits, this SMTP prime *would* make an efficient trapdoor: the first 129-bits could be recovered as described, and the remaining bits could be recovered from the 280-bit subgroup using Pollard's P-1 method in time approximately $2^{\frac{160-129}{2}} \approx 2^{16}$.

We conducted a partial factorization of the 904-bit composite modulus (number 8 in Table II) and found a number of suspiciously smooth factors:

```
5 * 23 * 474289 * 726101 * 72240863 * 48794510505931
* 70980749229449041 * 5093965413985867 * 2763354329179
* 1711955530550801 * 71015949150893819 * ...
```

This site used an improper generator of 4, which allowed us similarly to recover 372 bits of the private key. With either short exponents or knowledge of complete factorization, greater and more efficient recovery is possible.

IV. DH PARAMETER TESTING

In this section, we discuss the poor parameter validation of web clients, describe a man-in-the-middle attack in TLS that allows an attacker to take advantage of this, and outline the non-safe prime DH parameters found in TLS.

A. Parameter Hygiene in Discrete Logarithm Implementations

Most finite-field based implementations of the discrete logarithm cryptosystems we examined inherently treat domain parameters as trusted. Many of the necessary checks (e.g., primality, group membership, etc.) are done when the parameters are generated, but at no point thereafter. For example, the OpenSSL implementation of DSA does not check parameters during key generation, signing, or verification and we were able to construct accepting universal forgeries with maliciously constructed parameters. This wouldn't pose a problem in most cases since usually the expectation is that the signer would generate their own parameters, but this strategy does not always work out. One related example arose in OpenSSL when using non safe-prime groups (i.e., X9.42 groups) in Diffie-Hellman key exchanges where the server's private key was reused e.g., in static DH modes, or simply when, for efficiency sake, exponents were reused across more than one connection. By not checking the received client public value was in the intended group (i.e., $\mathbb{G}_q$), a malicious client could partially or fully recover the server's private key. This resulted in CVE-2016-0701, and now OpenSSL performs a group membership test of client public keys on the server side—but only when an X9.42 group is ostensibly in use. In the case of maliciously injected parameters, OpenSSL will still successfully proceed with DH key agreements using composite moduli, small groups, etc.

Many of the finite-field discrete logarithm implementations we examined work in $\mathbb{Z}_p^*$, as opposed to a prime order subgroup. The trend seems to have begun with the Handbook of Applied Cryptography (cf. Section 4.6.1 of [33]), and many implementations explicitly cite it. OpenSSL's default DH parameters and parameters generation utilities, for example, intentionally work in $\mathbb{Z}_p^*$, noting in a code comment that "actually there is no reason to insist that 'generator' be a generator.[4] It's just as OK (and in some sense better) to use a generator of the order-q subgroup." One reason that working in $\mathbb{G}_q$ is better than working in $\mathbb{Z}_p^*$ with a generator of order 2q is that the latter needlessly leaks a bit of the private key: it is easy for anyone to check if the private key was even or odd by checking respectively whether the public key is a quadratic residue or not.

Nominally there is little risk to the CDH assumption if $p-1$ contains a sufficiently large factor and full length exponents are used, i.e., the private exponent is also sampled from $\mathbb{Z}_p^*$, although Boneh et al. suggest related attacks in this setting [14]. A major risk comes about when developers, in the interest of performance, use short exponents (e.g., 160, 224, or 256 bits), and the Pohlig-Hellman attack may become applicable depending on the subgroup structure.

But we argue working in $\mathbb{Z}_p^*$ with a generator of order 2q is simply bad parameter hygiene (why leak anything when you don't have to?), and it sets a bad precedent for developers who might be tempted to apply this thinking to seemingly similar but subtly different situations. For example, we found the libgcrypt, pycrypto and bouncycastle implementations of ElGamal all by default work in $\mathbb{Z}_p^*$ with a generator of order 2q, which is conspicuous since it breaks the DDH assumption and hence semantic security.

[4]i.e., a generator of $\mathbb{Z}_p^*$

GPG, for example, uses libgcrypt and the authors confirmed their GPG public ElGamal encryption keys all leak one bit of their respective private keys. Although this does not lead directly to an attack because the plaintext in this setting is (largely) a random value, it is both unnecessary and potentially a sign of additional crypto issues. For example, GPG makes curious parameter choices and an ElGamal keypair at the 2048-bit level consists of a prime in which $p-1$ consists of a 340-bit private key in a 235-bit subgroup. Although many of the applications using these libraries seem not to require DDH, focusing instead on things like encrypting random nonces, neither do the libraries come with the caveat that the implementations are not semantically secure as one might nominally expect of an ElGamal implementation. This is probably fine when encrypting a session key, but is not as fine if the library were to be used as part of an implementation of a cryptographic voting system encrypting ballot choices. For example, Chang-Fong and Essex recently exploited small subgroups in Helios [16], an Internet voting system that provides end-to-end cryptographic verification. Finally we note the use of $\mathbb{Z}_p^*$ with a generator of order 2q is not universal. In contrast to the more ad hoc approach to parameter generation of many implementations, standardized parameters such as the MODP and Oakley safe-prime groups use generators that do not leak a bit. We consider working with safe prime groups with short exponents to be a good balance between security and efficiency.

B. DHE Support by Browsers

We determined DHE support by browsers, then tested their parameter validation by serving them weak DHE parameters. Many major web clients still support DHE, although Safari has removed DHE support. Chrome is in the process of removing support [10], but in the interest of interoperability connects with DHE if it is the only key exchange mode offered by the server. First it sends the `ClientHello` without DHE ciphersuites, and if that fails it will re-attempt with DHE ciphersuites added back in. This is largely in response to the difficulty in guaranteeing large moduli bit lengths following the results of Logjam, which we discuss further in § VI. Additional factors include the slower performance relative to ECDHE, although this gap is exacerbated by the predominance of safe-prime implementations using full-length exponents. Based on the current market share DHE is still supported in approximately 87% of browsers,[5] though this will drop steeply to about 22% once Chrome removes support. Based on our own survey approximately 26% of servers support DHE over HTTPS (see § IV for more).

We tested major web browsers to see to what extent they would accept weak DHE parameters. We configured OpenSSL's `s_server` to accept only DHE ciphersuites and serve custom generated Diffie-Hellman parameters. We wrote a program to generate malicious DH parameters and encode them in OpenSSL's ASN.1 / pem format. We tested a number of different composite moduli as well as non-safe prime groups of low order.

Tested browsers include Chrome, Safari, Firefox, Internet Explorer, and Microsoft Edge. At the time of testing all browsers still supported DHE ciphersuites. In each of the browser cases, the connection was successfully established with weak parameters or composite moduli, and no warnings were shown except in certain special cases. For example, Chrome generated an error when served moduli below 512-bits, even prior to the Logjam disclosure.

Interestingly browsers do perform a kind of limited primality test on the modulus and will reject even numbers. When presented with an even modulus, most browsers would generate an error, then switch to RSA for key exchange and proceed with the connection. In all cases the browsers would not accept obviously trivial values such as public keys or generators equaling 1 or $p-1$, meaning they *do* defend against working in the trivial group $\mathbb{G}_2$. The next smallest possible subgroup is one of order 3, in which the server public key can be either 1, g or g^2. Working in this group will generate a browser error approximately one third of the time (i.e., when $g = 1$), but in the interest of reliability many browsers would attempt the connection several more times and would succeed with high probability, and no errors would be displayed to the user. A 2-bit key is obviously an extreme example, and a real attacker can make failure extremely unlikely by selecting a slightly larger subgroup while still keeping discrete logarithms computable in real-time.

As a concrete example we used the following parameters in our browser test:

$$\begin{aligned} p &= 2^{2048} - 1557 \\ g_3 &= 2^{(p-1)/3} \bmod p \end{aligned}$$

Here p represents the largest 2048-bit prime and g_3 is a generator of a subgroup of order 3, i.e., the smallest possible non-trivial subgroup a browser would need to perform validation. As an illustration in Figure 1 we show a successful connection in Chrome with the server presenting the parameters $(p, g_3, y = g_3)$. In the developer tools Chrome warns that DHE is deprecated, but does not notice the weak group. This result is expected, as TLS contains no explicit field for communicating a group's order.

In summary, the browsers we tested were unable to defend against a variety of weak parameters (small or smooth order), as well as trapdoored groups involving composite moduli. The limited forms of checking that are performed are interesting from our perspective, as they constitute a kind of tacit acknowledgment that parameter validation is important—just so long as it is efficient.

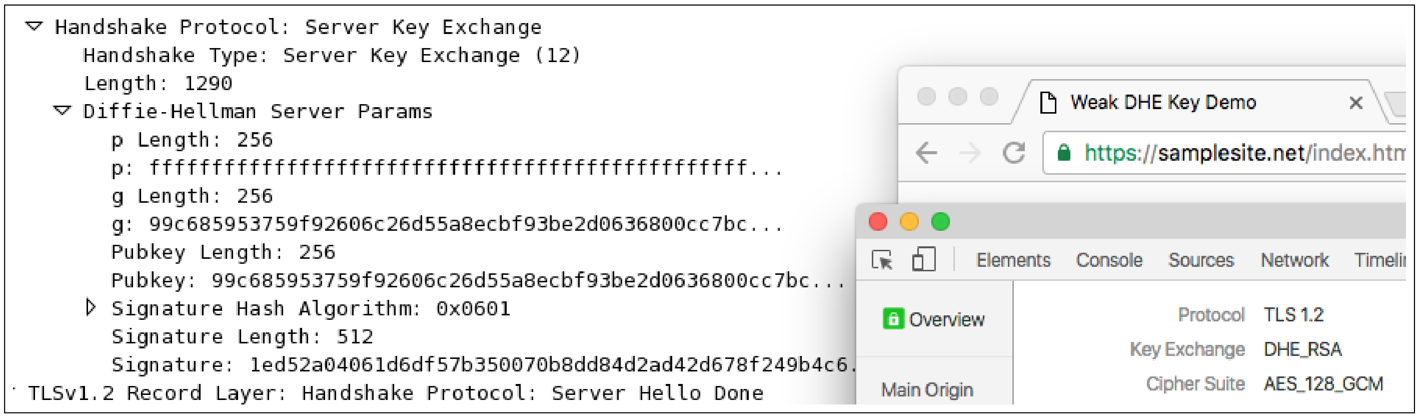
C. Forcing DHE in TLS

Based on current telemetry data, ciphersuites using DHE for key exchanges currently account for approximately 1% of TLS handshakes[6], limiting the potential for the attacker to exploit weak groups passively. Fortunately for the attacker, the message sequence of TLS makes it possible for someone knowing the master secret to actively modify the handshake to force DHE to be chosen if both parties support it. This is in contrast to SSH, which is not vulnerable to an active attack of this kind due to a differing message order (see § IV-D).

The client initiates a TLS handshake providing a list of supported ciphersuites. The man-in-the-middle modifies the

[5]https://www.w3counter.com/trends

[6]https://telemetry.mozilla.org



```
▽ Handshake Protocol: Server Key Exchange
    Handshake Type: Server Key Exchange (12)
    Length: 1290
  ▽ Diffie-Hellman Server Params
      p Length: 256
      p: ffffffffffffffffffffffffffffffffffffffffffffffff...
      g Length: 256
      g: 99c685953759f92606c26d55a8ecbf93be2d0636800cc7bc...
      Pubkey Length: 256
      Pubkey: 99c685953759f92606c26d55a8ecbf93be2d0636800cc7bc...
    ▷ Signature Hash Algorithm: 0x0601
      Signature Length: 512
      Signature: 1ed52a04061d6df57b350070b8dd84d2ad42d678f249b4c6.
· TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
```

Protocol TLS 1.2
Key Exchange DHE_RSA
Cipher Suite AES_128_GCM

Fig. 1. **Two-bit Security in TLS**. A successful DHE connection in Chrome using a generator of order 3. During this run the generator happened to equal the public key, indicating the private key was congruent to 1 mod 3.

client hello removing all but DHE ciphersuites. The client and server exchange keys as normal, except the attacker is able to exploit the weak or trapdoored parameters to compute the discrete logarithm of the client or server public values and compute the pre-master secret g^{ab}, from which they can compute the master secret. With a careful choice of parameters the attacker can compute the discrete log in real-time. Finally using the master secret, the attacker forges fake client- and server-finished messages tricking the respective parties into believing the other party only supported DHE ciphersuites, and thus there was no other choice but to connect under DHE. Furthermore, because the master secret is only a function of the pre-master secret and the client- and server-random values, both endpoints will derive the same master secrets, allowing the attacker to continue *passively* eavesdropping the connection from this point forward. This attack is illustrated in Figure 2.

D. Attack Limitations in SSH

The SSH protocol [43] specifies two fixed groups for Diffie-Hellman exchange: the 1024-bit Oakley group 2 [26] and the 2048-bit Oakley group 14 [28]. In major implementations of SSH, such as OpenSSH, these groups are included directly in the source code, although an extension of SSH does provide the option for a server to maintain its own list of group parameters [22]. Although the SSH standard calls specifically for the use of safe prime groups [22], older OpenSSH versions explicitly name Schnorr primes as an option[7].

However in addition to SSH version restriction, an attacker would also have to force DHE during the connection. OpenSSH now prefers ECDHE for key exchange, so an attacker wishing for the parties to use DHE instead would need to man-in-the-middle the handshake. Owing to the message sequence in SSH, being able to recover a DHE shared secret is not sufficient for this attack.

In SSH, the client chooses its preferred key-exchange method based on the server's indicated support [43]. An attacker could attempt to modify this initial server message, but then the attack would fail at the end of the handshake when the server provides a signed hash of the protocol messages. At this stage the client would detect that it saw a different sequence of messages than the server and would abort the connection, and the attacker could not forge this message without the server's signing key. This is outside our threat model. If either party does not support ECDHE, but both parties support Diffie-Hellman group-exchange, then they will connect under DH.

[7]http://man.openbsd.org/OpenBSD-4.3/cat5/moduli.0

Popularity	Modulus (bits)	Subgroup (bits)	Source
76.9%	1024	160	MODP (RFC5114) [30]
11.3%	1024	160	Amazon Web Services
7.5%	768	160	`sun.security.provider`
3.2%	1024	160	`sun.security.provider`
0.3%	2048	224	MODP (RFC5114) [30]
0.1%	2048	224	`sun.security.provider`
~1%	–	–	(others)

TABLE V. THE DISTRIBUTION AND SOURCES OF NON-SAFE DHE MODULI.

E. Non-Safe Prime Moduli Used By Web Servers

In addition to the composite moduli found in the HTTPS scan, we also found non-safe prime moduli used by 1.6M IPs. Of the 5,783 distinct non-safe primes we found, 5,409 were unique to a single IP. Six primes accounted for approximately 99% of sites. The distribution of non-safe primes is seen in Table V. MODP groups were seen in 77% of IP addresses using non-safe primes. Parameters used in the `sun.security.provider` package by Java were seen in 11% of IPs using non-safe primes. This package has had previous instances of misconfigured DH groups [6]. At the time of writing AWS load balancers no longer offer DHE ciphersuites following a security policy update.

Safe-prime groups have the property that all values in the range $1 < g < p-1$ are generators of groups of large order (either q or $2q$), and that an arbitrary value in this range is an element of $\mathbb{Z}_p^*$ with probability approaching $P = \frac{1}{2}$, meaning implementors are free to pick just about any generator they wish, and often opt for the smallest possible value (e.g., 2, 3, etc). Non safe-prime groups, on the other hand, generally should be more select in their choice of generator, especially when the order of $\mathbb{Z}_p^*$ contains smooth factors. If a group element has an order containing smooth factors, partial

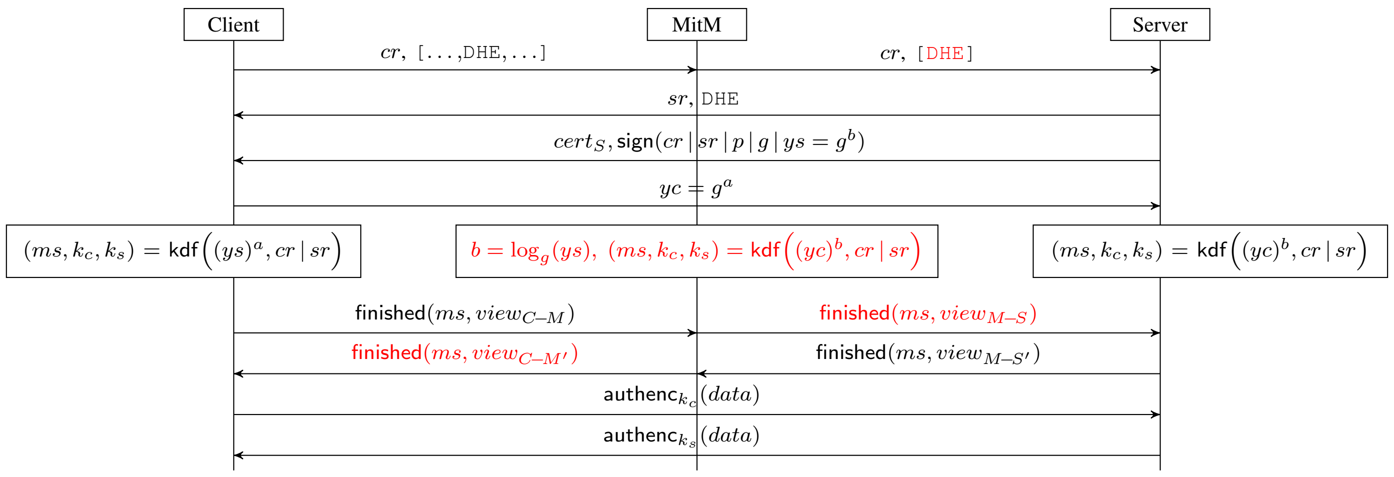

Client
MitM
Server
cr, [. . . ,DHE, . . .]
cr, [DHE]
sr, DHE
$cert_S$, sign$(cr \mid sr \mid p \mid g \mid ys = g^b)$
$yc = g^a$
$(ms, k_c, k_s) = \mathsf{kdf}\big((ys)^a, cr \mid sr\big)$
$b = \log_g(ys)$, $(ms, k_c, k_s) = \mathsf{kdf}\big((yc)^b, cr \mid sr\big)$
$(ms, k_c, k_s) = \mathsf{kdf}\big((yc)^b, cr \mid sr\big)$
finished$(ms, view_{C\text{–}M})$
finished$(ms, view_{M\text{–}S})$
finished$(ms, view_{C\text{–}M'})$
finished$(ms, view_{M\text{–}S'})$
authenc$_{k_c}(data)$
authenc$_{k_s}(data)$

Fig. 2. **Forcing DHE in TLS.** A man-in-the-middle with the ability to exploit weak or trapdoored parameters can force the parties to select a DHE cipher suite against their natural preferences.

recovery of the private key is possible. For a random non safe-prime group with an n-bit modulus and m-bit prime order subgroup $\mathbb{G}_q$, the probability an arbitrary value is a generator of $\mathbb{G}_q$ is approximately 2^{n-m}. Thus we shouldn't generally expect to see generators like 2 or 3 used in non safe-prime groups. We can expect such groups to leak more information about the exponent than the one bit of some safe prime groups.

Of the 1.6M IPs offering non safe-prime groups, we found 1,270 IPs using small generators. Generator values of 2 and 5 were most common but we also found cases of all prime numbers up to 31, as well as even values like 4 and 6. This doesn't directly break DHE so long as (a) the order of the generator contains a large prime factor and (b) full-length exponents are used. This is a precarious situation, since the typical reason for using non safe-prime groups is precisely for the purpose of using short exponents (e.g., X9.42 groups). It also speaks to the notion of parameter hygiene in which choices appropriate for one setting i.e., small generators of safe prime groups, is misapplied to another setting.

Like with composite moduli, we also were able to conduct partial key recoveries in non safe-prime groups with improper generators. In one improper export-grade non safe-prime group we were able to recover a full half of the private key (assuming a full-length exponent), though obviously for export moduli, Logjam would be a more efficient general attack strategy.

F. Survey of Open-source Projects

To determine if open-source projects use any weak moduli, we surveyed the default moduli of over a hundred open-source projects on GitHub. We used search terms based on common DH byte array names (e.g., `dh1024_p`, etc.). Out of the 95 projects supporting export grade 512-bit moduli, we found 16 distinct moduli, of which one was found in 44 projects. The most common modulus observed in Logjam was found in 9 projects. All were safe primes. Across 120 projects supporting 1024-bit moduli, there were 32 unique moduli. All the moduli were safe primes except for two: one reused from OpenSSL,[8] and a MODP group with 160-bit subgroup [30]. For 2048-bit moduli, there were 43 projects with 23 unique moduli. Similar to 1024-bit moduli, the only 2048-bit modulus that was not a safe prime was a MODP group with 256-bit subgroup [30]. For 3072-bit moduli, there were 3 unique safe primes spread over 4 projects. For 4096-bit moduli, there were 8 unique safe primes spread over 28 projects. Overall no weak moduli were found to be used, but parameter injection through an open-source project remains a possible attack vector for backdoors (see § V).

[8]https://github.com/openssl/openssl/blob/master/test/ssltest_old.c

V. Attack Vectors

The previous sections provided examples of potentially backdoored DH moduli in the wild and discussed the subsequent implications. We now propose three scenarios that enable an attacker to position weak parameters for use as a backdoor. If the target uses these parameters to perform cryptographic operations (i.e. key generations, signatures, key agreements, encryptions, etc.), the associated security guarantees no longer hold. Since Diffie-Hellman group parameters are infrequently modified, attacking them can lead to *persistent* backdoors, even if the keys themselves are ephemeral. The proposed threat vectors include dropping the parameters onto a server, incorporating the parameters in an open-source project, and installing the parameters on a network appliance that ships to customers.

A. Attacking the Server

The most intuitive way to get backdoored parameters in use is to install them at the source. First, the attacker creates the weak parameters and chooses a target that supports Diffie-Hellman ciphersuites. Second, the attacker injects these parameters as a backdoor payload onto the desired server. This step does require root access to the server, presumably in the context of a broader exploit. Having root access enables other attacks, such as stealing the server's private RSA signing key. This RSA attack would produce a similar outcome as the backdoored moduli, as efficient man-in-the-middle attacks are also possible for an attacker with the server's RSA signing key. However, obtaining and using the private RSA key has two disadvantages. In many enterprise situations, the private

RSA key is stored on a hardware security module (HSM) [2] attached to the server [15]. Since HSMs are designed to provide additional security to cryptographic keys, it would be difficult for an attacker to steal a key stored on an HSM even with root access to the server. The second disadvantage to using the private RSA key is that it requires an active man-in-the-middle attack. An active attack is also necessary to force DHE ciphersuites when not preferred, but only during the handshake. However, as seen in § III-F, *half* the IP addresses that use composite moduli in HTTPS prefer DHE ciphersuites. Therefore an attacker could choose attack targets that prefer DHE ciphersuites, allowing for passive eavesdropping. This type of passive attack is only possible with backdoored moduli; using the RSA signing key always requires an active attack.

Dropping the weak parameters onto the server requires no source code modification and creates a persistent backdoor; because of this, the backdoor may persist source code updates. The lack of parameter validation explained in § III-B and the examples of persistent composite moduli in § III-F mean that backdoored DH moduli could remain undetected for some time.

B. Attacking the Application

The second threat scenario involves submitting the backdoored parameters to an open-source project rather than attacking the server directly. First, the attacker creates the weak parameters and finds an open-source project that supports Diffie-Hellman. Second, the parameters are submitted as a patch to that repository. Once the repository accepts the change, the persistent backdoor would then be installed for users of that project. Conversely, the attacker could create a new project that already contains the backdoored parameters. Since the Logjam disclosure, many GitHub projects have been updating their Diffie-Hellman parameters to remove 512-bit moduli and modify 1024-bit moduli. This widespread change could ironically provide a reason for an attacker to submit a patch.

Socat, an open-source data transfer relay, recently published a security advisory [36] that outlines a similar scenario, and was one of the motivations behind Wong's recent paper [41]. Here a hard-coded 1024-bit composite DH modulus was discovered in the OpenSSL implementation. The Socat commit logs show that the composite modulus was introduced in January 2015 [35], and the security advisory was published more than a year later in February 2016, and the origin of the modulus remains unclear. Interestingly we also found this modulus twice in the HTTPS space (see modulus 6 in Table II). This gap between implementation and detection indicates backdoored moduli could remain undetected for a long time. The individual associated with the commit deleted much of his Internet presence on the day the advisory was published [42]. Attempts to factor the modulus suggest that there are large factors, which could indicate a backdoor configuration like those suggested in § III-D. Although we didn't find any suspicious parameters in the GitHub projects mentioned in § IV-F, the Socat example suggests that starting a malicious open-source project is one potential delivery vector, and that the ad hoc nature of parameter checking would hinder detection.

C. Attacking the Network

The final threat scenario involves installing backdoored parameters onto a network appliance that is shipped to customers. Network appliances such as load balancers and traffic shapers are often used by companies to optimize application or network performance. Load balancers optimize application performance by distributing traffic across many servers, which decreases the load on individual servers. This traffic can be application or network traffic. Balancers also provide SSL termination so that servers do not have to perform encryption and decryption [5]. Although this invites man-in-the-middle attacks, the servers and balancer are often located on the same internal network which decreases this possibility. Another network appliance is traffic or packet shapers, which optimize network performance by delaying less important network packets. Various applications can be shaped differently, a process called application-based traffic shaping or deep packet inspection (DPI). Since DPI allows users to look at layers 2 through 7 of the OSI model, it is possible to view the ServerKeyExchange message [37]. DPI also provides the possibility of packet payload tampering [40].

This threat scenario requires the attacker to be a company employee who creates the weak parameters. The employee then installs the backdoored parameters onto the load balancing network appliance sold by his company. Blue Coat's PacketShaper S-Series, a traffic shaping network appliance, can be connected with another PacketShaper to provide load balancing capability [3]. The load balancer equipped with backdoored parameters is then sold to a customer. The balancer sends decrypted traffic to the chosen server, then encrypts the server's response and sends it to the client as usual. Therefore the success of this scenario depends mostly on the trust placed in the load balancer to securely encrypt and decrypt traffic.

VI. Vulnerability Disclosures

As mentioned in § III-F, we issued vulnerability disclosures to companies that were using composite moduli in HTTPS. Security contact information for each company was searched for in the HackerOne directory,[9] although only one company had such information. Only companies with at least one active webpage were contacted, since webpage identifiers were important in determining the company associated with the IP address. Out of the 21 companies listed in § III-F, only 17 were contacted. Only 47% of the contacted companies responded to our disclosure.

Blue Coat Systems was the first company contacted, and we communicated on several occasions with a number of high-ranking employees within the company on the matter. A patch for the affected product, PacketShaper S-Series 11.5, was released in June 2016. A few weeks later on July 12, 2016, a CVE was released for this vulnerability under the label CVE-2016-5774 [4]. This CVE has a high severity score in CVSS v3 but only a medium score in CVSS v2, as v2 emphasizes percentage of impacted systems rather than level of impact like v3. Therefore although composite DH moduli are not abundant in the wild, these moduli have a high degree of impact on affected systems. An interesting side effect of our disclosure was that it inadvertently uncovered a number of

[9]https://hackerone.com/directory

improperly configured web-facing administrator login pages, which allowed Blue Coat to follow up with affected customers.

After disclosure, the other 16 companies were split into three groups depending on the status of the vulnerability fix: completed, partially completed, or not started. The vulnerability was fixed by 56% of these companies, although not all responded to us and three had implemented fixes prior to our disclosure. These independent solutions could have been a result of Wong's disclosures [41]. The solution implemented by most companies involved changing the composite moduli to prime, although one company simply removed its DHE ciphersuites altogether. Of the 19% of companies who partially completed the vulnerability fix, all are progressively changing composite moduli to prime. The remaining 25% of companies did not respond to our disclosure and have not modified their Diffie-Hellman parameters. One of these companies had the highest number of affected IP addresses by far. A language barrier existed for some companies, which could have contributed to this result.

We spoke to senior management at Blue Coat and technical staff at many other companies. Despite this, all companies we had discussions with declined to provide us with information on the source of the potentially backdoored parameters. Blue Coat more specifically stated that the information could not be provided due to security reasons. Another company explained that its composite modulus was attributed to cipher modifications made by the company, but no specifics were given. Two others provided broad information on their load balancing, but not in the context of the specific vulnerability. As we were unable to receive external confirmation that these moduli were backdoored and could not completely factor the moduli to prove it, we cannot say unequivocally that these moduli are backdoored. We have discovered everything possible about each company's vulnerability using publicly available information. Without additional information from the companies themselves, we cannot speculate further on topics such as the cause of the vulnerability.

VII. Discussion

There is a growing consensus that Diffie-Hellman negotiations are less secure than previously thought. Safari has removed DHE ciphersuites altogether, and Chrome plans to remove them in upcoming versions [10]. However, during the time of writing Chrome continued to offer DHE ciphersuites if all other ciphersuites offered were not accepted by the server. The current TLS 1.3 draft [34] proposes using named DHE groups [23], similar to the named ECDHE groups currently used. These named DHE groups are used in the `supported_groups` and `key_share` extensions, and would not be susceptible to the kinds of attacks described in this paper.

Information on using Diffie-Hellman properly has been extensively discussed by Adrian et al. [6], who suggest using at least 2048-bit DH groups with safe prime moduli. Therefore we restrict our discussion to mitigation strategies for the outlined vulnerability.

A. Mitigation Strategies

We propose four strategies for mitigating regrouping attacks: deprecating Diffie-Hellman ciphersuites, verifying Diffie-Hellman parameters correctly, using named DH groups, or modifying the ServerKeyExchange message to sign all previously seen messages.

Deprecate DHE. One option is to follow the example of Safari and Chrome and deprecate finite field Diffie-Hellman altogether. In our opinion, this option makes sense in certain situations, but not as a general solution. As we saw with Dual_EC_DRGB, there is a trade-off between trust and convenience through standardization. With that in mind, Bernstein et al. [18] added a new name to the standards of Alice and Bob: *Jerry*, an authority who generates curve parameters such that his attack cost is decreased. With the deprecation of RSA key exchange coming in TLS 1.3, DHE ciphersuites represent the only alternative key exchange method.

Verify parameters properly. Our preferred option would be to simply implement the necessary domain parameter validation to begin with. The first issue, however, is computational cost. In order to verify that a generator or DHE public key has the intended order, modular exponentiation must be performed at runtime for *each* connection. Similarly p must be tested for primality, and, importantly, if general Schnorr groups are to be permitted, the TLS and SSH protocols must provide an explicit means to communicate group order q. As we discussed in § II-A, basic checking is not sufficient to prevent all attacks.

Use named parameters. A third solution is to develop standardized, named parameters like those in an ECC setting. The RFC proposed by Gillmor [23] and supported in the TLS 1.3 draft [34] involves standardizing parameters in the FFC setting to augment the MODP groups. As we see in ECC, named parameters are a feasible mitigation strategy used in the real world. One issue of restricting moduli to only safe primes is performance: private key lengths are 10 times larger than NIST recommended minimum standards. One performance optimization Gillmor suggests is to compromise by using safe prime groups with short, DSA-like exponents.

Change TLS. The last solution is to modify the ServerKeyExchange message so that all previously exchanged messages are also signed. The MitM attack from § IV-C works because the ServerKeyExchange message only signs the DH parameters, ServerHello.random, and ClientHello.random. If the list of ciphersuites suggested in ClientHello and the chosen ciphersuite in ServerHello were also signed, then the ciphersuite tampering would be discovered upon receiving the ServerKeyExchange message. This solution was also proposed by Mavrogiannopoulos et al. [32] to prevent their cross-protocol attack.

Finally, a recent proposal by Bhargavan et al. [11] proposes an elegant method for downgrade resilience in TLS 1.3, and was incorporated into the draft as of Version 11. In their strategy, the server puts the highest version of TLS supported by the client into the ServerHello.Random, which will be incorporated into the signed ServerKeyExchange message. If a client supports TLS 1.3, but is being man-in-the-middled in the context of a downgrade attack such the one described in § V, the man in the middle will be unable to modify the signed ServerKeyExchange message, and the client will see that the server believes the client does not support TLS 1.3, which is false so the handshake is aborted. This method, combined with the use of named safe-prime DHE groups in TLS 1.3, would solve the issue of trapdoored groups.

VIII. Conclusion

In this paper we demonstrated a serious, systematic problem with real-world discrete logarithm implementations. A lack of parameter validation allows attackers to use weak or trapdoored groups to create persistent DHE backdoors in TLS. Hundreds of IP addresses in the wild were found to use potentially backdoored moduli, and both web and mail servers were equally affected, leading us in some cases to recover significant portions of the private key even without knowledge of the trapdoor. We proposed several threat scenarios that would enable an attacker to inject backdoored parameters. Vulnerability disclosures were completed to over 15 companies worldwide resulting in CVE-2016-5774. This study found evidence to suggest trapdoored DHE parameters are in use on the Internet today. Minding our Ps and Qs, it would seem, has proven more elusive that previously thought.

Acknowledgements

The authors wish to thank Jeremy Clark, Jakub Dalek, Matt Green, Ian Goldberg, Adam Senft, and Greg Zaverucha for their helpful input. This work was supported by the National Science and Engineering Research Council of Canada (NSERC) Discovery Grant and Canada Graduate Scholarship programs.

References

[1] "Company Overview of Eyou.net," 2016. [Online]. Available: http://www.bloomberg.com/research/stocks/private/snapshot.asp?privcapId=113374953

[2] "Hardware security module," 2016, https://www.ibm.com/support/knowledgecenter/SS9H2Y_7.5.0/com.ibm.dp.doc/hsm2.html.

[3] "Standby Feature with High Availability Clusters," 2016. [Online]. Available: https://bto.bluecoat.com/packetguide/11.6/Content/PDFs/standby.pdf

[4] "Vulnerability Summary for CVE-2016-5774," 2016. [Online]. Available: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-5774

[5] "What is an SSL Load Balancer?" 2016. [Online]. Available: https://www.nginx.com/resources/glossary/ssl-load-balancer/

[6] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," in *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[7] R. Anderson and S. Vaudenay, "Minding Your P's and Q's," in *ASIACRYPT*, 1996, pp. 26–35.

[8] E. Barker, L. Chen, A. Roginsky, and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography," Tech. Rep., 2013.

[9] E. Barker and A. Roginsky, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths," Tech. Rep., 2015.

[10] D. Benjamin, "Intent to Remove: DHE-based ciphers," 2016, https://groups.google.com/a/chromium.org/forum/#!topic/security-dev/sVq6r0i-CZM.

[11] K. Bhargavan, C. Brzuska, C. Fournet, M. Green, , M. Kohlweiss, and S. Zanella-Béguelin, "Downgrade resilience in key-exchange protocols," in *IEEE Symposium on Security and Privacy*, 2016.

[12] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P.-Y. Strub, "Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS," in *IEEE Symposium on Security and Privacy*, 2014.

[13] K. Bhargavan, A. Delignat-Lavaud, and A. Pironti, "Verified contributive channel bindings for compound authentication," in *NDSS*, 2015.

[14] D. Boneh, A. Joux, and P. Nguyen, "Breaking plain elgamal and plain rsa encryption," in *Asiacrypt*, 2000.

[15] K. Cairns, J. Mattsson, R. Skog, and D. Migault, "Session Key Interface (SKI) for TLS and DTLS," 2015. [Online]. Available: https://tools.ietf.org/html/draft-cairns-tls-session-key-interface-01

[16] N. Chang-Fong and A. Essex, "The cloudier side of cryptographic end-to-end verifiable voting: A security analysis of helios," in *ACSAC*, 2016.

[17] J.-S. Coron, A. Joux, A. Mandal, D. Naccache, and M. Tibouchi, *Cryptanalysis of the RSA Subgroup Assumption from TCC 2005*, 2011, pp. 147–155.

[18] D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, , E. Lambooij, T. Lange, R. Niederhagen, and C. van Vredendaal, "How to manipulate curve standards: a white paper for the black hat," 2014, http://bada55.cr.yp.to/bada55-20150927.pdf.

[19] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," Jan. 1999.

[20] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by Internet-wide scanning," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[21] J. Fried, P. Gaudry, N. Heninger, and E. Thomé, "A kilobit hidden snfs discrete logarithm computation," Cryptology ePrint Archive, Report 2016/961, 2016, http://eprint.iacr.org/2016/961.

[22] M. Friedl, N. Provos, and W. Simpson, "Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol," 2006. [Online]. Available: https://tools.ietf.org/html/rfc4419

[23] D. Gillmor, "Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for TLS," 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-negotiated-ff-dhe-10

[24] D. M. Gordon, "Designing and detecting trapdoors for discrete log cryptosystems," in *ADVANCES IN CRYPTOLOGY– CRYPTO '92*. Springer-Verlag, 1993, pp. 66–75.

[25] J. Groth, "Cryptography in subgroups of z*n," in *TCC*, 2005.

[26] D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)," 1998. [Online]. Available: https://tools.ietf.org/html/rfc2409

[27] R. Henry and I. Goldberg, "Solving discrete logarithms in smooth-order groups with CUDA," in *SHARCS*, 2012.

[28] T. Kivinen and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)," 2003. [Online]. Available: https://tools.ietf.org/html/rfc3526

[29] A. Lenstra, "Constructing trapdoor primes for the proposed dss," École polytechnique fédérale de Lausanne, Tech. Rep. EPFL-REPORT-164559, 1991.

[30] M. Lepinski and S. Kent, "Additional Diffie-Hellman Groups for Use with IETF Standards," 2008. [Online]. Available: https://tools.ietf.org/html/rfc5114

[31] C. H. Lim and P. J. Lee, "A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroup," *Crypto*, vol. 1294, pp. 249–263, 1997.

[32] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel, "A cross-protocol attack on the TLS protocol," *ACM Conference on Computer and Communications Security*, pp. 62–72, 2012.

[33] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press.

[34] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13-14

[35] G. Rieger, "FIPS requires 1024 bit DH prime," 2015. [Online]. Available: http://repo.or.cz/socat.git/commitdiff/281d1bd6515c2f0f8984fc168fb3d3b91c20bdc0

[36] ——, "Socat security advisory 7 - Created new 2048bit DH modulus," 2016. [Online]. Available: http://www.openwall.com/lists/oss-security/2016/02/01/4

[37] W. G. Sanchez, "SLOTH Downgrades TLS 1.2 Encrypted Channels," 2016. [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/sloth-downgrades-tls-1-2-encrypted-channels/

[38] L. Valenta, D. Adrian, A. Sanso, S. Cohney, J. Fried, M. Hastings,

J. A. Halderman, and N. Heninger, “Measuring small subgroup attacks against diffie-hellman,” in *NDSS*, 2017.

[39] P. C. van Oorschot and M. J. Wiener, “On diffie-hellman key agreement with short exponents,” in *EUROCRYPT*, 1996.

[40] N. Vratonjic, J. Freudiger, J.-P. Hubaux, and M. Felegyhazi, “Securing Online Advertising,” Tech. Rep., 2008.

[41] D. Wong, “How to backdoor diffie-hellman,” Cryptology ePrint Archive, Report 2016/644, 2016, http://eprint.iacr.org/2016/644.

[42] ——, “Socat? What? (timeline of events),” 2016, https://github.com/mimoo/Diffie-Hellman_Backdoor/tree/master/socat_reverse.

[43] T. Ylonen, “The Secure Shell (SSH) Transport Layer Protocol,” 2006. [Online]. Available: https://tools.ietf.org/html/rfc4253