Indiscreet Logs: Persistent 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 are presented with are trustworthy, i.e., the parameters implement cyclic groups where the discrete logarithm problem is assumed to be hard. An informal and widespread justification for this seemingly exists that says validating parameters at run time is too computationally expensive relative to the perceived risk of a server sabotaging the privacy of its own connection. In this paper we explore this trust assumption and examine situations where it may not always be justified.

We conducted an investigation of discrete logarithm domain parameters in use across the Internet and discovered evidence of a multitude of potentially backdoored moduli of unknown order in TLS and STARTTLS spanning numerous countries, organizations, and protocols. Although our disclosures resulted in a number of organizations taking down suspicious parameters, we argue the potential for TLS backdoors is systematic and will persist until either until better parameter hygiene is taken up by the community, or finite field based cryptography is eliminated altogether.

I. INTRODUCTION

Finite fields underly 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 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 it should be a critical functionality. It is not. Checking each value in each interaction at run-time is regarded as being too computationally expensive, and 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: the party with the private key typically chooses these parameters, which are typically communicated to the other party via an authenticated channel. We examine a number of possible scenarios in which weakened parameters could be maliciously injected for the purposes of creating a persistent backdoor. There are a number of possible way for these malicious parameters to be delivered, and we consider two main possibilities: directly attacking the server or TLS endpoint, or by attacking the software upstream. The first scenario involves the attacker achieving some kind of privileged access in the context of a potentially broader exploit, which places weak DH parameters as payload on a server or TLS endpoint (e.g., cloud load balancer). The second scenario involves the attacker compromising the software itself, for example by starting a malicious open-source software project, compromising software update, or by slipping the modified parameters into a closed-source commercial software product/appliance. Although the adversarial assumptions are strictly stronger than an in-protocol attack on DH like Logjam [6], they exhibit a number of desirable properties, and we find evidence in the wild of their deployment.

Significance. We argue parameter injection is 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 detecting weak parameters is a potentially very computationally expensive process. Community efforts to locate and draw attention to suspicious parameters have been largely ad hoc.

- **Deniable.** The very existence of a backdoor can be deniable. We uncovered a number of composite moduli online and none of the organizations we contacted were willing or able to prove the parameters did not contain a backdoor, nor explain how the parameters came to be used.

Summary of Results. We examined major implementations of the finite field based discrete logarithms showing a systematic vulnerability to weak groups and systematically bad parameter hygiene. Some of these issues are well known. What may not be as well known is the exact extent to which they occur.

Although DHE use is in steep decline, many servers and clients still support it, and we discuss how TLS in particular is vulnerable to DHE "downgrade" attacks, in which an adversary that can exploit trapdoored parameters can force a DHE ciphersuite to be selected if both parties support it. 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, spanning over 30 countries and a diverse set of organizations. We disclosed the vulnerability to 17 companies, resulting in a security advisory (CVE-2016-5774). We found thousands more sites offering non-safe prime groups of unknown order, and others using generators of partially smooth order, which allowed us in some cases to recover large portions of the private key. Interestingly, none of the organizations we spoke to were willing or able to explain how composite moduli came to be used in their DHE configurations, but we present several possible attack vectors. We finish with a discussion of potential mitigation strategies.

At worst, the conclusion of this study is that backdoored DHE parameters are in use on the Internet today. More troubling is that at best we cannot be certain, nor is there a straightforward way to robustly prevent it.

The remainder of this paper is organized as follows. In § II we discuss background and related work. In § III we examine software implementations and discuss the mechanics of creating TLS backdoors, as well as how they can be exploited in TLS and SSH. In § IV we present a survey of DHE parameters in TLS and STARTTLS. 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-A we discuss potential mitigation strategies.

II. BACKGROUND

A. Related Work

RSA Weaknesses. Although RSA does not use public domain parameters like DL implementations, RSA is susceptible to poor entropy in key generation. Both DL parameters validation and RSA key generation are mature areas, but maturity does not prevent weak implementations from occurring. Heninger et al. [24] and Lenstra et al. [30] recently conducted independent Internet-wide surveys of RSA keys. Both studies found instances of insufficiently random RSA keys, leading to private key recovery in some cases. The current TLS 1.3 draft [36] has recommended removing RSA for key exchange due to its security drawbacks, emphasizing the importance of proper protocol implementations.

Elliptic Curve Backdoors. For elliptic curves, there has been speculation for years that some NIST standard curves contained backdoors [12]. This speculation has been renewed recently, as revelations by Edward Snowden suggested that certain elliptic curve cryptography standards like the Dual_EC_DRBG random number generator contained an exploitable backdoor. This type of backdoor mathematically exists[1], and is practically exploitable based on recent work by Checkoway et al. [14].

General DL Attacks. General algorithms such as Baby-step gGiant-step and Pollard's P-1 can be applied to the discrete logarithm problem. Given a generator g of prime order q and

an element $y = g^x \in \mathbb{G}_q$, the value x can be recovered in time $O(\sqrt{q})$ and space $O(\sqrt{q})$. An algorithm by Pohlig and Hellman [35] provides better results if q is composite. Given $q = q_1 q_2 \ldots q_k$, the Pohlig-Hellman algorithm can compute the discrete logarithm of $y = g^x$, and therefore recover x, in time

$$O\left(\sum_{i=1}^{k} \sqrt{q_i} \right).$$

A variation of working in a group of smooth composite order is to simply work in a group of small order. This *small subgroup* attack is by Lim and Lee [32], with additional insight provided by Anderson and Vaudenay [7] and Oorschot and Weiner [41]. In § III, we show how small subgroup attacks can be applied toward shared-secret recovery in TLS.

Number Field Sieve Attacks. In ephemeral DH key agreement, forward secrecy is the process of generating independent, per-connection keys so that an attacker would theoretically have to solve a new instance of the discrete logarithm problem each time. Recently, Adrian et al. [6] showed that this assumption does not always hold in finite field cryptography. They demonstrated that an attacker could perform intensive precomputation once for a specific group, and later perform efficient discrete logarithms of independent instances in that group. Their attack, called Logjam, used a modified version of the generalized number field sieve (GNFS) to recover ephemeral DH keys where 512-bit groups, or export-grade parameters, were used.

Factoring 512-bit RSA moduli is becoming routine [40], but computing discrete logarithms remains at least as hard as integer factorization. Sieving in 2048-bit groups is currently considered well outside the range of computational feasibility, and DH moduli have been required by NIST [9] to be at least 2048 bits since 2013, (although this theory seems to differ from the current practice in many cases).

Although Diffie-Hellman ciphersuites are still widely supported, they are chosen with increasing rarity which makes NFS-based attacks largely infeasible. These attacks cannot recover DH shared secrets that were never generated, and they cannot force DH ciphersuites to be chosen if the DL computation takes considerably longer than a typical TLS handshake. For example, at 512-bit levels the implementation of Adrian et al. took 70 seconds on average, compared to a typical TLS handshake that takes less than one second. Therefore despite the stronger adversarial assumptions required by weakened or trapdoored moduli relative to GNFS-based attacks, they can be parameterized to allow real-time recovery of shared secrets—at any key length.

GNFS-Backdoored Moduli. As pointed out by Lensta [29], Gordon [21], and others, basic validation would not be sufficient to ensure the hardness of the discrete logarithm problem if p was maliciously chosen to be "nice" in the context of a GNFS attack. Here, a prime number could be constructed using a polynomial of low-degree and constrained coefficients for the purposes of greatly accelerating the 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 process, however, is much more costly than the steps outlined above, and is unrealistic at run-time.

---

[1] See United States Patent US8396213.

However, there fundamentally exists an inverse relationship between detectability and exploitability in this setting: the lower degree the polynomial, the easier to compute the GNFS, but also the more likely a verifier could discover the underlying polynomial and vice-versa. Although this relationship deserves further study, there are much more efficient options available unless covertness is critical to the attacker. Therefore we restrict our discussion to the basic validation techniques described in § III-B.

Diffie-Hellman Parameter Validation. The lack of DH parameter validation in TLS and SSH has been previously noted. Bhargavan *et al.* [26] 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 [27] the authors conduct small subgroup attacks on TLS, SSH, and IKEv2 that exploit the lack of public key validation. Mavrogiannopoulos et al. [33] 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.

Finally, recent concurrent but independent work by Wong [43] also 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.[2] In § III-G 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 he presents 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.

III. DISCRETE LOGARITHM BACKDOORS

In this section we discuss aspects of creating and exploiting weak or trapdoored DH parameters in TLS and OpenSSH.

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 $q << p$ allowing for more efficient operations.

Applications of the 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 $\mathbb{G}_q$ be a cyclic group of large, prime order. Given two elements $g, y \in \mathbb{G}_q$ the *discrete logarithm problem* (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 of 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 [10]);

- **Primality**: Check p and q are both prime;[3]

- **Group Order**: Check $q|(p-1)$. No mechanism is provided in TLS to communicate group order [17], [36];

- **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. 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 (and, of course, DSA has gone the way of the dinosaur anyway). 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 indented 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 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 [34]), 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 $\mathbb{Z}_p^*$ 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 [13]. 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 $\mathbb{Z}_p^*$ 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 $\mathbb{Z}_p^*$, which is conspicuous since it breaks the DDH assumption and hence semantic security.

GPG, for example, uses libcrypt 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 wonkiness. 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 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 ok when encrypting a session key. This is not so ok if the library were to be used as part of an implementation of a cryptographic voting system encrypting ballot choices. Finally we note the use of $\mathbb{Z}_p^*$ 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.

D. Trapdoors with Composite Moduli

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 a use composite (e.g., RSA style) 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.

Wong [43] proposes $p = 2p_1p_2+1$ and $q = 2q_1q_2+1$ where $p1, q1$ are both small (e.g., 40–55 bits), while p_2, q_2 are large so as to generally prevent factorization. Let $|p_1| = |q_1| = \ell$. A generator g is chosen with an order of bit length 2ℓ i.e., where g has an ℓ-bit order mod p and q respectively, and thus computing a discrete logarithm modulo p and q takes on the order of $2^{\frac{\ell}{2}}$ operations each, using standard DL techniques. With knowledge of the trapdoor, therefore, the attacker can compute a discrete logarithm in $2^{\frac{\ell}{2}+1}$ operations. Another person without knowledge of the factorization, meanwhile, would require 2^ℓ operations. This expectation, as it turns out, is false as Corono et al. [15] showed in the context of Groth's cryptosystem [22]. Groth proposed a cryptosystem in which ℓ-bit primes p_1, q_1 form the randomizer space, and are chosen to be small for efficiency. Groth proposed $\ell = 100$, and hence the order of the generator of the randomizer space is 2ℓ bits. Given this public value, Corono et al. demonstrated a clever attack that recovers the factors of n in time $O(2^{\frac{\ell}{2}})$ instead of the expected $O(2^\ell)$. Since Wong's proposed trapdoor can be viewed essentially as a version of Groth's cryptsytem, the trapdoor is similarly recoverable in $O(2^{\frac{\ell}{2}})$.

Thus for the trapdoor to be resistant to recovery, a generator of larger order is needed. Let $p = 2p_1p_2\ldots p_k + 1$ and $q = 2q_1q_2\ldots q_k + 1$ with p_i, q_i being randomly chosen ℓ-bit primes. ℓ is sized small enough that solving $2k$ discrete logarithm instances in subgroups of approximately 2^ℓ is efficiently computable, while sized large enough to prevent factorization of n, and in turn, discovery of the trapdoor. Using Pollard's $p-1$ factorization method in this setting, 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 $\text{li}(2^{\ell_b})$ modular exponentiations, where $\text{li}(\cdot)$ is the logarithmic interval. Using an optimized GPU implementation, Henry and

---

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

Goldberg [25] suggest $\ell_b = 55$ would be sufficient, requiring 1500 years of (non-paralellizable) wall-clock time to factor n, while requiring less than two minutes to compute the discrete logarithm with knowledge of the trapdoor. Of course a real attacker must weigh the trade-off between their efficiency in computing a discrete log, and the computational feasibility of someone else recovering the trapdoor.

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. Browsers Tests

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 supported DHE ciphersuites, although at the time of writing Safari had discontinued support for DHE, and Chrome is planning to as well [11]. 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. In each of the remaining browser cases, the connection was successfully established with weak parameters, or composite moduli and no warnings were shown except in certain special cases. In particular Chrome will generate an error when served short moduli (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. But how could it? 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.

F. Current Support for DHE

Many major web clients still support DHE, although Safari has removed DHE support. Chrome is in the process of removing support but still supports it in the interest of interoperability, connecting 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. The move away from DHE can be seen as largely a response to Logjam and the complexity of falling back to other ciphersuites when a client rejects the server modulus on account of a lower than acceptable bit length. 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 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).
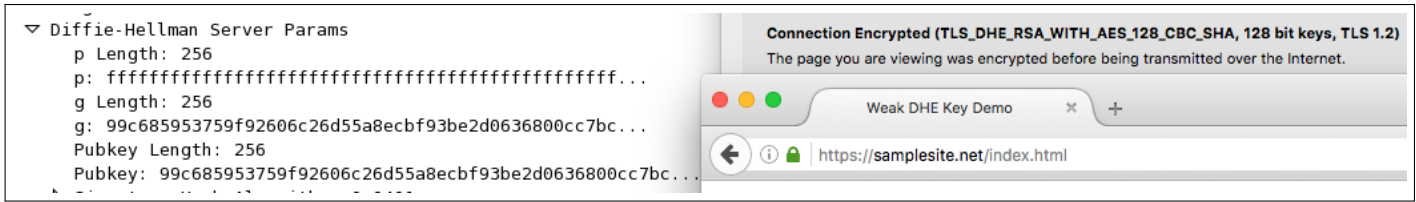

Fig. 1. **Two-bit Security in TLS**. A successful DHE connection in Chrome using a generator of order 3. During this run the generator happened to equal the public key, indicating the private key was congruent to 1 mod 3.

G. Forcing DHE in TLS

Based on current telemetry data, ciphersuites using DHE for key exchanges currently account for approximately 1% of TLS handshakes, 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 § III-H).

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

H. Attack Limitations in SSH

The SSH protocol [45] specifies two fixed groups for Diffie-Hellman exchange: the 1024-bit Oakley group 2 [23] 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, which would require a code modification to exploit. An extension of SSH does provide the option for a server to maintain its own list of group parameters [19] and exchange them with the client at connection time. Although the SSH standard calls specifically for the use of safe prime groups [19], older OpenSSH versions explicitly name Schnorr primes as an option[6].

However in addition to SSH version restriction, an attacker would also have to force DHE during the connection. OpenSSH now prefers elliptic curve Diffie-Hellman key exchange; any two clients supporting ECDHE will connect with that method. An attacker wishing for the parties to use DHE instead would need to man-in-the-middle the handshake.

However, owing to the message sequence in SSH, being able to recover a DHE shared secret is not sufficient to man-in-the-middle the connection and force DHE to be chosen.

In SSH, the client chooses its preferred key-exchange method based on the server's indicated support [45]. 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 it saw during the handshake. 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, which 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. DH is still widely supported, but support for ECDHE was implemented as of OpenSSH 6.2 (ca. 2013).

I. Creating Valid-looking Moduli

A backdoored modulus may possibly remain undetected for longer if the weak modulus at least looks valid, e.g., didn't end with an even digit. As a demonstration we played with the OpenSSH \etc\moduli file and created a 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's 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 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.

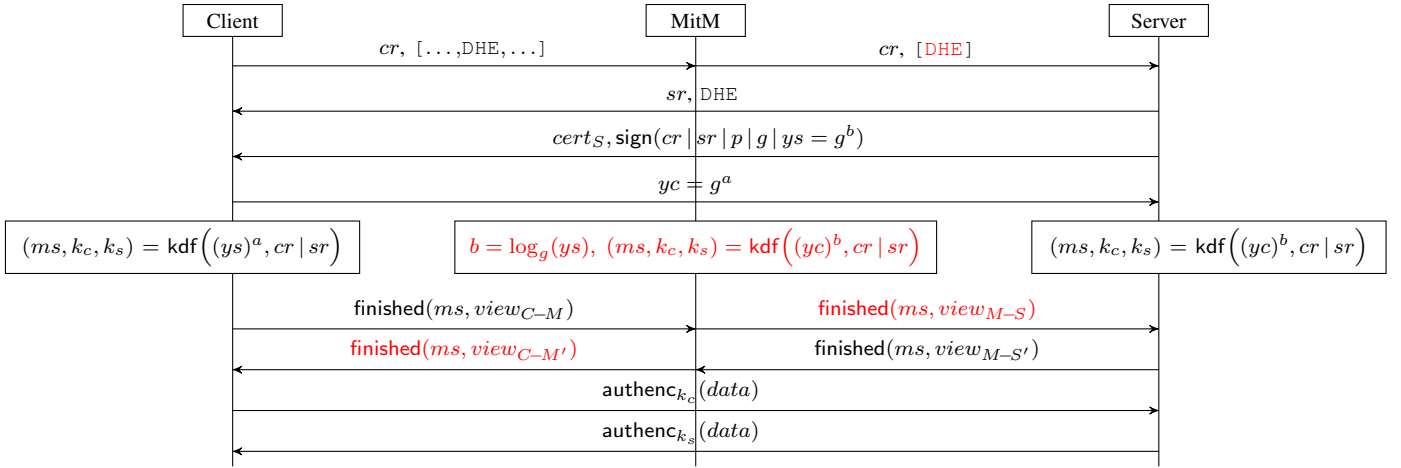[6]http://man.openbsd.org/OpenBSD-4.3/cat5/moduli.0

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.

```
# Time Type Tests Tries Size Generator Modulus

20160522030737 2 6 100 2047 2 DB36277B45EA5615C782C08BF6A290A3D61E6B9690E4A147042113FC1BFC0AE
EC5FB0FF82FC1FEA86E273F667EC387FEF3421FFFC617A70C34B1987986C6B35C715713914AB75932A3D1942ECC0F
324D81BF00D59916B3BFDC7BA432AF5C5DFCF30BF4A2C80B0DA52A9B80E989D3A852BD81A8BD3ADC97497F43C6F0A
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.

Implementing this in Sage we were able to compute discrete logarithms in 4ms on a laptop.

IV. SURVEY OF DH PARAMETERS IN TLS

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 [18] 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[7] 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.

A. Affected Protocols and Countries

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. 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 suggests backdoor use, as 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

---

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

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.

To see the impact of these composite moduli, we investigated each IP address' location as seen in Table III. Nearly all the composite moduli were used in HTTPS or SMTP, but 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.

B. Web Server Backdoors

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 some. We observed 5,783 unique non-safe prime moduli across 1.6M IPs. We observed 9 unique composite moduli across 289 IPs. We did a comparison to ECDHE and found that of 32 million IPs, all used a standard SECP curve, and that the sever 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

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.

not.

Of the 5,783 distinct non-safe primes we found, 5,409 were unique to a single IP. 6 primes accounted for approximately 99% of sites. The distribution of non-safe primes is seen in Table IV. 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.

Non Safe-Prime Groups With Safe-Prime Generators. 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 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 grousp. We can expect such groups to leak more information about the exponent the 1 bit of some safe prime groups.

Of the 1.5M 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. Although again 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 mis-applied to another setting.

Composite DH Moduli. In HTTPS we found nine composite moduli across 280 IP addresses. None of these were export-grade; all were at least 904-bits in length. In May 2016, 46% of these IP addresses chose a Diffie-Hellman ciphersuite by default, meaning forcing DHE (as described in § III-G) is not needed in those cases.

We then examined the public ownership information of the affected IPs in public databases and by examining 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

Popularity	Modulus (bits)	Subgroup (bits)	Source
76.9%	1024	160	MODP (RFC5114) [31]
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) [31]
0.1%	2048	224	sun.security.provider
~1%	–	–	(others)

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

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 [43]. 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.

C. Mail Server Backdoors

Since Censys did not have DH scans for mail servers, we run 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.

D. 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:

```
11435638110073884015312138951374632602058078871389818 1372 \\
7578406924934826346123042770482700524507174581850431874 44 \\
9841546167312785561120575583039273650 7955

= 5 * 11 * 313049766 6273667404271 * 132398438917079824212 \\
370893794766672908033 * 50165074897437023341346800600 2745 \\
0130769436621955914589815397976412146715534764 08791132267
```

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, though knowing the full factorization would have allowed for greater recovery. 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. With either short exponents or knowledge of complete factorization, however, even more efficient recovery is possible.

V. ATTACK VECTORS

The previous sections discussed the implications of a regrouping attack and provided examples of potentially backdoored DH moduli in the wild. 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) attached to the server [2]. 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 § IV-B, *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 § IV-B 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 [38] that outlines a similar scenario, and was one of the motivations behind Wong's recent paper [43]. 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 [37], 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 can 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 [44]. Attempts to factor the modulus suggest that there are large factors, which could indicate a backdoor configuration like those suggested in § III-D.

Survey of Open-source Projects. We surveyed the default moduli of over a hundred open-source projects on Github using 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 uniques. All the moduli were safe primes except for two: one reused from OpenSSL,[8] and a MODP group with 160-bit subgroup [31]. 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 [31]. 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. Although we didn't find any suspicious parameters in any of the Github projects, the Socat example suggests both 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 [39]. DPI also provides the possibility of packet payload tampering [42].

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 Packet-Shaper S-Series, a traffic shaping network appliance, can be connected with another PacketShaper to provide load balancing

---

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

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 § IV-B, 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 blank webpages are less relevant attack targets. Out of the 21 companies listed in § IV-B, 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, 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 improperly configured web-facing admin 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, and 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 [43]. 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.

None of the contacted companies were willing to provide us with information on the source of the potentially backdoored parameters. One 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.

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 [11]. 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 [36] proposes using named DHE groups [20], similar to the named ECDHE groups currently used. These named DHE groups are used in the `supported_groups` and `key_share` extensions. Although named groups would not be susceptible to the kinds of attacks described in this paper, they would not elimiate the threat entirely: adapting the attack in § III-G a MitM could modify the ClientHello message to force a downgrade from TLS 1.3 to 1.2 and proceed with a weak or trapdoored group.

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. [16] 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 [20] and supported in the TLS 1.3 draft [36] 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.

Modify ServerKeyExchange message. The final solution is to modify the ServerKeyExchange message so that all previously exchanged messages are also signed. The MitM attack

from § III-G works because the ServerKeyExchange message only signs the DH parameters, server random, and client 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. [33] to prevent their cross-protocol attack.

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.

IX. ACKNOWLEDGEMENTS

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., 2011.

[10] ——, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths," Tech. Rep., 2015.

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

[12] D. J. Bernstein and T. Lange, "Security dangers of the NIST curves," 2013. [Online]. Available: http://www.hyperelliptic.org/tanja/vortraege/20130531.pdf

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

[14] S. Checkoway, M. Fredrikson, R. Niederhagen, A. Everspaugh, M. Green, T. Lange, T. Ristenpart, D. J. Bernstein, J. Maskiewicz, and H. Shacham, "On the Practical Exploitability of Dual EC in TLS Implementations," *USENIX Security Symposium*, 2014.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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

[20] 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

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

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

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

[24] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining your Ps and Qs: Detection of widespread weak keys in network devices," in *Proceedings of the 21st USENIX Security Symposium*, Aug. 2012.

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

[26] A. P. Karthikeyan Bhargavan, Antoine Delignat-Lavaud, "Verified contributive channel bindings for compound authentication," in *NDSS*, 2015.

[27] C. F. A. P. P.-Y. S. Karthikeyan Bhargavan, Antoine Delignat-Lavaud, "Triple handshakes and cookie cutters: Breaking and fixing authentication over tls," in *IEEE Symposium on Security and Privacy*, 2014.

[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] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter, "Ron was wrong, whit is right," Cryptology ePrint Archive, Report 2012/064, 2012, http://eprint.iacr.org/2012/064.

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

[32] 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.

[33] 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.

[34] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[35] S. C. Pohlig and M. E. Hellman, "An Improved Algorithm for Computing Logarithms over GF(p) and Its Cryptographic Significance," *IEEE Transactions on Information Theory*, vol. 24, no. 1, pp. 106–110, 1978.

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

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

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

[39] 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/

[40] L. Valenta, S. Cohney, A. Liao, J. Fried, S. Bodduluri, and N. Heninger, "Factoring as a service," Cryptology ePrint Archive, Report 2015/1000, 2015, http://eprint.iacr.org/2015/1000.

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

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

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

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

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