Measuring small subgroup attacks against Diffie-Hellman

Luke Valenta*, David Adrian†, Antonio Sanso‡, Shaanan Cohney*,
Joshua Fried*, Marcella Hastings*, J. Alex Halderman†, Nadia Heninger*
*University of Pennsylvania
†University of Michigan
‡Adobe

***Abstract*—Several recent standards, including NIST SP 800-56A and RFC 5114, advocate the use of "DSA" parameters for Diffie-Hellman key exchange. While it is possible to use such parameters securely, additional validation checks are necessary to prevent well-known and potentially devastating attacks. In this paper, we observe that many Diffie-Hellman implementations do not properly validate key exchange inputs. Combined with other protocol properties and implementation choices, this can radically decrease security. We measure the prevalence of these parameter choices in the wild for HTTPS, POP3S, SMTP with STARTTLS, SSH, IKEv1, and IKEv2, finding millions of hosts using DSA and other non-"safe" primes for Diffie-Hellman key exchange, many of them in combination with potentially vulnerable behaviors. We examine over 20 open-source cryptographic libraries and applications and observe that until January 2016, not a single one validated subgroup orders by default. We found feasible full or partial key recovery vulnerabilities in OpenSSL, the Exim mail server, the Unbound DNS client, and Amazon's load balancer, as well as susceptibility to weaker attacks in many other applications.**

I. Introduction

Diffie-Hellman key exchange is one of the most common public-key cryptographic methods in use in the Internet. It is a fundamental building block for IPsec, SSH, and TLS. In the textbook presentation of finite field Diffie-Hellman, Alice and Bob agree on a large prime p and an integer g modulo p. Alice chooses a secret integer x_a and transmits a public value $g^{x_a} \bmod p$; Bob chooses a secret integer x_b and transmits his public value $g^{x_b} \bmod p$. Both Alice and Bob can reconstruct a shared secret $g^{x_a x_b} \bmod p$, but the best known way for a passive eavesdropper to reconstruct this secret is to compute the discrete log of either Alice or Bob's public value. Specifically, given g, p, and $g^x \bmod p$, an attacker must calculate x.

In order for the discrete log problem to be hard, Diffie-Hellman parameters must be chosen carefully. A typical recommendation is that p should be a "safe" prime, that is, that $p = 2q + 1$ for some prime q, and that g should generate the group of order q modulo p. For p that are not safe, the group order q can be much smaller than p. For security, q must still be large enough to thwart known attacks, which for prime q run in time $O(\sqrt{q})$. A common parameter choice is to use a 160-bit q with a 1024-bit p or a 224-bit q with a 2048-bit p, to match the security level under different cryptanalytic attacks. Diffie-Hellman parameters with p and q of these sizes were suggested for use and standardized in DSA signatures [50]. For brevity, we will refer to these non-safe primes as DSA primes, and to groups using DSA primes with smaller values of q as DSA groups.

A downside of using DSA primes instead of safe primes for Diffie-Hellman is that implementations must perform additional validation checks to ensure the key exchange values they receive from the other party are contained in the correct subgroup modulo p. The validation consists of performing an extra exponentiation step. If implementations fail to validate, a 1997 attack of Lim and Lee [54] can allow an attacker to recover a static exponent by repeatedly sending key exchange values that are in very small subgroups. We describe several variants of small subgroup confinement attacks that allow an attacker with access to authentication secrets to mount a much more efficient man-in-the-middle attack against clients and servers that do not validate group orders. Despite the risks posed by these well-known attacks on DSA groups, NIST SP 800-56A, "Recommendations for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography" [23] specifically recommends DSA group parameters for Diffie-Hellman, rather than recommending using safe primes. RFC 5114 [53] includes several DSA groups for use in IETF standards.

We observe that few Diffie-Hellman implementations actually validate subgroup orders, in spite of the fact that small subgroup attacks and countermeasures are well-known and specified in every standard suggesting the use of DSA groups for Diffie-Hellman, and DSA groups are commonly implemented and supported in popular protocols. For some protocols, including TLS and SSH, that enable the server to unilaterally specify the group used for key exchange, this validation step is not possible for clients to perform with DSA primes—there is no way for the server to communicate to the client the intended order of the group. Many standards involving DSA groups further suggest that the order of the subgroup should be matched to the length of the private exponent. Using shorter private exponents yields faster exponentiation times, and is a commonly implemented optimization. However, these standards provide no security justification for decreasing the size of the subgroup to match the size of the exponents, rather than using as large a subgroup as possible. We discuss possible motivations for these recommendations later in the paper.

NDSS'17, 26 February–1 March, 2017, San Diego, CA, USA
http://dx.doi.org/10.14722/ndss.2017.23171

We conclude that adopting the Diffie-Hellman group recommendations from RFC 5114 and NIST SP 800-56A may create vulnerabilities for organizations using existing cryptographic implementations, as many libraries allow user-configurable groups but have unsafe default behaviors. This highlights the need to consider developer usability and implementation fragility when designing or updating cryptographic standards.

Our Contributions. We study the implementation landscape of Diffie-Hellman from several perspectives and measure the security impact of the widespread failure of implementations to follow best security practices:

- We summarize the concrete impact of small-subgroup confinement attacks and small subgroup key recovery attacks on TLS, IKE, and SSH handshakes.
- We examined the code of a wide variety of cryptographic libraries to understand their implementation choices. We find feasible full private exponent recovery vulnerabilities in OpenSSL and the Unbound DNS resolver, and a partial private exponent recovery vulnerability for the parameters used by the Amazon Elastic Load Balancer. We observe that *no* implementation that we examined validated group order for subgroups of order larger than two by default prior to January 2016, leaving users potentially vulnerable to small subgroup confinement attacks.
- We performed Internet-wide scans of HTTPS, POP3S, SMTP with STARTTLS, SSH, IKEv1, and IKEv2, to provide a snapshot of the deployment of DSA groups and other non-"safe" primes for Diffie-Hellman, quantify the incidence of repeated public exponents in the wild, and quantify the lack of validation checks even for safe primes.
- We performed a best-effort attempt to factor $p-1$ for all non-safe primes that we found in the wild, using ~100,000 core-hours of computation. Group 23 from RFC 5114, a 2048-bit prime, is particularly vulnerable to small subgroup key recovery attacks; for TLS a full key recovery requires 2^{33} online work and 2^{47} offline work to recover a 224-bit exponent.

Disclosure and Mitigations. We reported the small subgroup key recovery vulnerability to OpenSSL in January 2016 [66]. OpenSSL issued a patch to add additional validation checks and generate single-use private exponents by default [11]. We reported the Amazon load balancer vulnerability in November 2015. Amazon responded to our report informing us that they have removed Diffie-Hellman from their recommmended ELB security policy, and have reached out to their customers to recommend that they use these latest policies. Based on scans performed in February and May 2016, 88% of the affected hosts appear to have corrected their exponent generation behavior. We found several libraries that had vulnerable combinations of behaviours, including Unbound DNS, GnuTLS, LibTomCrypt, and Exim. We disclosed to the developers of these libraries. Unbound issued a patch, GnuTLS acknowledged the report but did not patch, and LibTomCrypt did not respond. Exim responded to our bug report stating that they would use their own generated Diffie-Hellman groups by default, without specifying subgroup order for validation [10], [12]. We found products from Cisco, Microsoft, and VMWare lacking validation that key exchange values were in the range $(1, p-1)$. We informed these companies, and discuss their responses in Section III-D.

II. Background

A. Groups, orders, and generators

The two types of groups used for Diffie-Hellman key exchange in practice are multiplicative groups over finite fields ("mod p") and elliptic curve groups. We focus on the "mod p" case, so a group is typically specified by a prime p and a generator g, which generates a multiplicative subgroup modulo p. Optionally, the group order q can be specified; this is the smallest positive integer q satisfying $g^q \equiv 1 \bmod p$. Equivalently, it is the number of distinct elements of the subgroup $\{g, g^2, g^3, \ldots \bmod p\}$.

By Lagrange's theorem, the order q of the subgroup generated by g modulo p must be a divisor of $p-1$. Since p is prime, $p-1$ will be even, and there will always be a subgroup of order 2 generated by the element -1. For the other factors q_i of $p-1$, there are subgroups of order $q_i \bmod p$. One can find a generator g_i of a subgroup of order q_i using a randomized algorithm: try random integers h until $h^{(p-1)/q_i} \neq 1 \bmod p$; $g_i = h^{(p-1)/q_i} \bmod p$ is a generator of the subgroup. A random h will satisfy this property with probability $1 - 1/q_i$.

In theory, neither p nor q is required to be prime. Diffie-Hellman key exchange is possible with a composite modulus and with a composite group order. In such cases, the order of the full multiplicative group modulo p is $\phi(p)$ where ϕ is Euler's totient function, and the order of the subgroup generated by g must divide $\phi(p)$. Outside of implementation mistakes, Diffie-Hellman in practice is done modulo prime p.

B. Diffie-Hellman Key Exchange

Diffie-Hellman key exchange allows two parties to agree on a shared secret in the presence of an eavesdropper [29]. Alice and Bob begin by agreeing on shared parameters (prime p, generator g, and optionally group order q) for an algebraic group. Depending on the protocol, the group may be requested by the initiator (as in IKE), unilaterally chosen by the responder (as in TLS), or fixed by the protocol itself (SSH originally built in support for a single group).

Having agreed on a group, Alice chooses a secret $x_a < q$ and sends Bob $y_a = g^{x_a} \bmod p$. Likewise, Bob chooses a secret $x_b < q$ and sends Alice $y_b = g^{x_b} \bmod p$. Each participant then computes the shared secret key $g^{x_a x_b} \bmod p$.

Depending on the implementation, the public values y_a and y_b might be *ephemeral*—freshly generated for each connection—or *static* and reused for many connections.

C. Discrete log algorithms

The best known attack against Diffie-Hellman is for the eavesdropper to compute the the private exponent x by calculating the discrete log of one of Alice or Bob's public value y. With knowledge of the exponent, the attacker can trivially compute the shared secret. It is not known in general whether the hardness of computing the shared secret from the public values is equivalent to the hardness of discrete log.

The *computational Diffie-Hellman assumption* states that computing the shared secret $g^{x_a x_b}$ from g^{x_a} and g^{x_b} is hard for some choice of groups. A stronger assumption, the *decisional Diffie-Hellman problem*, states that given g^{x_a} and g^{x_b}, the shared secret $g^{x_a x_b}$ is computationally indistinguishable from random for some groups. This assumption is often not true for groups used in practice; even with safe primes as defined below, many implementations use a generator that generates the full group of order $p-1$, rather than the subgroup of order $(p-1)/2$. This means that a passive attacker can always learn the value of the secret exponent modulo 2. To avoid leaking this bit of information about the exponent, both sides could agree to compute the shared secret as $y^{2x} \bmod p$. We have not seen implementations with this behavior.

There are several families of discrete log algorithms, each of which apply to special types of groups and parameter choices. Implementations must take care to avoid choices vulnerable to any particular algorithm. These include:

Small-order groups. The Pollard rho [63] and Shanks' baby step-giant step algorithms [67] each can be used to compute discrete logs in groups of order q in time $O(\sqrt{q})$. To avoid being vulnerable, implementations must choose a group order with bit length at least twice the desired bit security of the key exchange. In practice, this means that group orders q should be at least 160 bits for an 80-bit security level.

Composite-order groups. If the group order q is a composite with prime factorization $q = \prod_i q_i^{e_i}$, then the attacker can use the Pohlig-Hellman algorithm [61] to compute a discrete log in time $O(\sum_i e_i \sqrt{q_i})$. The Pohlig-Hellman algorithm computes the discrete log in each subgroup of order $q_i^{e_i}$ and then uses the Chinese remainder theorem to reconstruct the log modulo q. Adrian et al. [18] found several thousand TLS hosts using primes with composite-order groups, and were able to compute discrete logs for several hundred Diffie-Hellman key exchanges using this algorithm. To avoid being vulnerable, implementations should choose g so that it generates a subgroup of large prime order modulo p.

Short exponents. If the secret exponent x_a is relatively small or lies within a known range of values of a relatively small size, m, then the Pollard lambda "kangaroo" algorithm [64] can be used to find x_a in time $O(\sqrt{m})$. To avoid this attack, implementations should choose secret exponents to have bit length at least twice the desired security level. For example, using a 256-bit exponent for for a 128-bit security level.

Small prime moduli. When the subgroup order is not small or composite, and the prime modulus p is relatively large, the fastest known algorithm is the number field sieve [40], which runs in subexponential time in the bit length of p, $\exp\left((1.923 + o(1))(\log p)^{1/3}(\log \log p)^{2/3}\right)$. Adrian et al. recently applied the number field sieve to attack 512-bit primes in about 90,000 core-hours [18], and they argue that attacking 1024-bit primes—which are widely used in practice—is within the resources of large governments. To avoid this attack, current recommendations call for p to be at least 2048 bits [21]. When selecting parameters, implementers should ensure all attacks take at least as long as the number field sieve for their parameter set.

D. *Diffie-Hellman group characteristics*

"Safe" primes. In order to maximize the size of the subgroup used for Diffie-Hellman, one can choose a p such that $p = 2q+1$ for some prime q. Such a p is called a "safe" prime, and q is a Sophie Germain prime. For sufficiently large safe primes, the best attack will be solving the discrete log using the number field sieve. Many standards explicitly specify the use of safe primes for Diffie-Hellman in practice. The Oakley protocol [59] specified five "well-known" groups for Diffie-Hellman in 1998. These included three safe primes of size 768, 1024, and 1536 bits, and was later expanded to include six more groups in 2003 [51]. The Oakley groups have been built into numerous other standards, including IKE [43] and SSH [71].

DSA groups. The DSA signature algorithm [50] is also based on the hardness of discrete log. DSA parameters have a subgroup order q of much smaller size than p. In this case $p-1 = qr$ where q is prime and r is a large composite, and g generates a group of order q. FIPS 186-4 [50] specifies 160-bit q for 1024-bit p and 224- or 256-bit q for 2048-bit p. The small size of the subgroup allows the signature to be much shorter than the size of p.

E. *DSA Group Standardization*

DSA-style parameters have also been recommended for use for Diffie-Hellman key exchange. NIST Special Publication 800-56A, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography" [23], first published in 2007, specifies that finite field Diffie-Hellman should be done over a prime-order subgroup q of size 160 bits for a 1024-bit prime p, and a 224- or 256-bit subgroup for a 2048-bit prime. While the order of the multiplicative subgroups is in line with the hardness of computing discrete logs in these subgroups, no explanation is given for recommending a subgroup of precisely this size rather than setting a minimum subgroup size or using a safe prime. Using a shorter exponent will make modular exponentiation more efficient, but the order of the subgroup q does not increase efficiency—on the contrary, the additional modular exponentiation required to validate that a received key exchange message is contained in the correct subgroup will render key exchange with DSA primes less efficient than using a "safe" prime for the same exponent length. Choosing a small subgroup order is not known to have much impact on other cryptanalytic attacks, although the number field sieve is somewhat (not asymptotically) easier as the linear algebra step is performed modulo the subgroup order q. [18]

RFC 5114, "Additional Diffie-Hellman Groups for Use with IETF Standards" [53], specifies three DSA groups with the above orders "for use in IKE, TLS, SSH, etc." These groups were taken from test data published by NIST [1]. They have been widely implemented in IPsec and TLS, as we will show below. We refer to these groups as Group 22 (1024-bit group with 160-bit subgroup), Group 23 (2048-bit group with 224-bit subgroup), and Group 24 (2048-bit group with 256-bit subgroup) throughout the remainder of the paper to be consistent with the group numbers assigned for IKE.

RFC 6989, "Additional Diffie-Hellman Tests for the Internet Key Exchange Protocol Version 2 (IKEv2)" [68], notes that "mod p" groups with small subgroups can be vulnerable to small subgroup attacks, and mandates that IKE implementations should validate that the received value is in the correct subgroup or never repeat exponents.

F. *Small subgroup attacks*

Since the security of Diffie-Hellman relies crucially on the group parameters, implementations can be vulnerable to an attacker who provides maliciously generated parameters that change the properties of the group. With the right parameters and implementation decisions, an attaker may be able to efficiently determine the Diffie-Hellman shared secret. In some cases, a passive attacker may be able to break a transcript offline.

Small subgroup confinement attacks. In a small subgroup confinement attack, an attacker (either a man-in-the-middle or a malicious client or server) provides a key-exchange value y that lies in a subgroup of small order. This forces the other party's view of the shared secret, y^x, to lie in the subgroup generated by the attacker. This type of attack was described by van Oorschot and Wiener [69] and ascribed to Vanstone and Anderson and Vaudenay [20]. Small subgroup confinement attacks are possible even when the server does not repeat exponents — the only requirement is that an implementation does not validate that received Diffie-Hellman key exchange values are in the correct subgroup.

When working $\bmod p$, there is always a subgroup of order 2, since $p-1$ is even. A malicious client Mallory could initiate a Diffie-Hellman key exchange value with Alice and send her the value $y_M = p - 1 \equiv -1 \bmod p$, which is is a generator of the group of order $2 \bmod p$. When Alice attempts to compute her view of the shared secret as $k_a = y_M^a \bmod p$, there are only two possible values, 1 and $-1 \bmod p$.

The same type of attack works if $p-1$ has other small factors q_i. Mallory can send a generator g_i of a group of order q_i as her Diffie-Hellman key exchange value. Alice's view of the shared secret will be an element of the subgroup of order q_i. Mallory then has a $1/q_i$ chance of blindly guessing Alice's shared secret in this invalid group. Given a message from Alice encrypted using Alice's view of the shared secret, Mallory can brute force Alice's shared secret in q_i guesses.

More recently, Bhargavan and Delignat-Lavaud [25] describe "key synchronization" attacks against IKEv2 where a man-in-the-middle connects to both the initiator and responder in different connections, uses a small subgroup confinement attack against both, and observes that there is a $1/q_i$ probability of the shared secrets being the same in both connections. Bhargavan and Leurent [26] describe several attacks that use subgroup confinement attacks to obtain a transcript collision and break protocol authentication.

To protect against subgroup confinement attacks, implementations should use prime-order subgroups with known subgroup order. Both parties must validate that the key exchange values they receive are in the proper subgroup. That is, for a known subgroup order q, a received Diffie-Hellman key exchange value y should satisfy $y^q \equiv 1 \bmod p$. For a safe prime, it suffices to check that y is strictly between 1 and $p-1$.

Small subgroup key recovery attacks. Lim and Lee [54] discovered a further attack that arises when an implementation fails to validate subgroup order and resues a static secret exponent for multiple key exchanges. A malicious party may be able to perform multiple subgroup confinement attacks for different prime factors q_i of $p-1$ and then use the Chinese remainder theorem to reconstruct the static secret exponent.

The attack works as follows. Let $p-1$ have many small factors $p-1 = q_1 q_2 \dots q_n$. Mallory, a malicious client, uses the procedure described in Section II-A to find a generator of the subgroup g_i of order $q_i \bmod p$. Then Mallory transmits g_i as her Diffie-Hellman key exchange value, and receives a message encrypted with Alice's view of the shared secret $g_i^{x_a}$, which Mallory can brute force to learn the value of $x_a \bmod q_i$. Once Mallory has repeated this process several times, she can use the Chinese remainder theorem to reconstruct $x_a \bmod \prod_i q_i$. The running time of this attack is $\sum_i q_i$, assuming that Mallory performs an offline brute-force search for each subgroup.

Application	Crypto Library	Short Exponent	Exponent Reuse
OpenSSH	OpenSSL	No	No
Cerberus	OpenSSL	No	Yes
GNU lsh	GnuTLS	No	No
Dropbear	LibTomCrypt	No	No
Lighttpd	OpenSSL	Yes	No
Unbound	OpenSSL	Yes	Yes
Exim	OpenSSL	Library dependent	Yes
Postfix	OpenSSL	No	No

TABLE I: **Common application behavior** — Applications make a diverse set of decisions on how to handle Diffie-Hellman exponents, likely due to the plethora of conflicting, confusing, and incorrect recommendations available.

A randomly chosen prime p is likely to have subgroups of large enough order that this attack is infeasible to carry out for all subgroups. However, if in addition Alice's secret exponent x_a is small, then Mallory only needs to carry out this attack for a subset of subgroups of orders $q_1, \dots, q_k$ satisfying $\prod_{i=0}^{k} q_i > x_a$, since the Chinese remainder theorem ensures that x_a will be uniquely defined. Mallory can also improve on the running time of the attack by taking advantage of the Pollard lambda algorithm. That is, she could use a small subgroup attack to learn the value of $x_a \bmod \prod_{i=1}^{k} q_i$ for a subset of subgroups $\prod_{i=1}^{k} q_i < x_a$, and then use the Pollard lambda algorithm to reconstruct the full value of a, as it has now been confined to a smaller interval.

In summary, an implementation is vulnerable to small subgroup key recovery attacks if it does not verify that received Diffie-Hellman key exchange values are in the correct subgroup; uses a prime p such that $p-1$ has small factors; and reuses Diffie-Hellman secret exponent values. The attack is made even more practical if the implementation uses small exponents.

A related attack exists for elliptic curve groups: an invalid curve attack. Similarly to the case we describe above, the attacker generates a series of elliptic curve points of small order and sends these points as key exchange messages to the victim. If the victim does not validate that the received point is on the intended curve, they return a response that reveals information about the secret key modulo different group orders. After enough queries, the attacker can learn the victim's entire secret. Jager, Schwenk, and Somorovsky [45] examined eight elliptic curve implementations and discovered two that failed to validate the received curve point. For elliptic curve groups, this attack can be much more devastating because the attacker has much more freedom in generating different curves, and can thus find many different small prime order subgroups. For the finite field Diffie-Hellman attack, the attacker is limited only to those subgroups whose orders are factors of $p-1$.

III. TLS

TLS (Transport Layer Security) is a transport layer protocol designed to provide confidentiality, integrity and (most commonly) one-side authentication for application sessions. It is widely used to protect HTTP and mail protocols.

A TLS client initiates a TLS handshake with the `ClientHello` message. This message includes a list of supported cipher suites, and a client random nonce r_c. The server responds with a `ServerHello` message containing the chosen cipher suite and server random nonce r_s, and a `Certificate` message that includes the server's X.509 certificate. If the server selects a cipher suite using ephemeral Diffie-Hellman key exchange, the server additionally sends a `ServerKeyExchange` message containing the server's choice of Diffie-Hellman parameters p and g, the server's Diffie-Hellman public value $y_s = g^{x_s} \bmod p$, a signature by the server's private key over both the client and server nonces (r_c and r_s), and the server's Diffie-Hellman parameters (p, g, and y_s). The client then verifies the signature using the public key from the server's certificate, and responds with a `ClientKeyExchange` message containing the client's Diffie-Hellman public value $y_c = g^{x_c} \bmod p$. The Diffie-Hellman shared secret $Y = g^{x_s x_c} \bmod p$ is used to derive encryption and MAC keys. The client then sends `ChangeCipherSpec` and `Finished` messages. The `Finished` message contains a hash of the handshake transcript, and is encrypted and authenticated using the derived encryption and MAC keys. Upon decrypting and authenticating this message, the server verifies that the hash of the transcript matches the expected hash. Provided the hash matches, the server then sends its own `ChangeCipherSpec` and `Finished` messages, which the client then verifies. If either side fails to decrypt or authenticate the `Finished` messages, or if the transcript hashes do not match, the connection fails immediately [28].

TLS also specifies a mode of using Diffie-Hellman with fixed parameters from the server's certificate [62]. This mode is not forward secret, was never widely adopted, and has been removed from all modern browsers due to dangerous protocol flaws [44]. The only widely used form of Diffie-Hellman in TLS today is ephemeral Diffie-Hellman, described above.

A. *Small Subgroup Attacks in TLS*

Small subgroup confinement attacks. A malicious TLS server can perform a variant of the small subgroup attack against a client by selecting group parameters g and p such that g generates an insecure group order. TLS versions prior to 1.3 give the server complete liberty to choose the group, and they do not include any method for the server to specify the desired group order q to the client. This means a client has no feasible way to validate that the group sent by the server has the desired level of security or that a server's key exchange value is in the correct group for a non-safe prime.

Similarly, a man in the middle with knowledge of the server's long-term private signing key can use a small subgroup confinement attack to more easily compromise perfect forward secrecy, without having to rewrite an entire connection. The attack is similar to the those described by Bhargavan and Delignat-Lavaud [25]. The attacker modifies the server key exchange message, leaving the prime unchanged, but substituting a generator g_i of a subgroup of small order q_i for the group generator and g_i for the server's key exchange value y_s. The attacker then forges a correct signature for the modified server key exchange message and passes it to the client. The client then responds with a client key exchange message $y_c = g_i^{x_c} \bmod p$, which the man-in-the-middle leaves unchanged. The server's view of the shared secret is then $g_i^{x_c x_s} \bmod p$, and the client's view of the shared secret is $g_i^{x_c} \bmod p$. These views are identical when $x_s \equiv 1 \bmod q_i$, so this connection will succeed with probability $1/q_i$. For small enough q_i, this enables a man in the middle to use a compromised server signing key to decrypt traffic from forward-secret ciphersuites with a reasonable probability of success, while only requiring tampering with a single handshake message, rather than having to actively rewrite the entire connection for the duration of the session.

Furthermore, if the server uses a static Diffie-Hellman key exchange value, then the attacker can perform a small subgroup key-recovery attack as the client in order to learn the server's static exponent $x_s \bmod q_i$ for the small subgroup. This enables the attacker to calculate a custom generator such that the client and server views of the shared secret are always identical, raising the above attack to a 100% probability of success.

Small subgroup key recovery attacks. In TLS, the client must authenticate the handshake before the server, by providing a valid `Finished` message. This forces a small subgroup key recovery attack against TLS to be primarily online. To perform a Lim-Lee small subgroup key recovery attack against a server static exponent, a malicious client initiates a TLS handshake and sends a generator g_i of a small subgroup of order q_i as its client key exchange message y_c. The server will calculate $Y_s = g_i^{x_s} \bmod p$ as the shared secret. The server's view of the shared secret is confined to the subgroup of order q_i. However, since g_i and g generate separate subgroups, the server's public value $y_s = g_s^x$ gives the attacker no information about the value of the shared secret Y_s. Instead, the attacker must guess a value for $x_s \bmod q_i$, and send the corresponding client `Finished` message. If the server continues the handshake, the attacker learns that the guess is correct. Therefore, assuming the server is reusing a static value for x_s, the attacker needs to perform at most q_i queries to learn the server's secret $x_s \bmod q_i$ [54]. This attack is feasible if q_i is small enough and the server reuses Diffie-Hellman exponents for sufficiently many requests.

The attacker repeats this process for many different primes q_i, and uses the Chinese remainder theorem to combine them modulo the product of the primes q_i. The attacker can also use the Pollard lambda algorithm to reconstruct any remaining bits of the exponent [54].

We note that the TLS False Start extension allows the server to send application data before receiving the client's authentication [52]. The specification only allows this behavior for abbreviated handshakes, which do not include a full key exchange. If a full key exchange were allowed, the fact that the server authenticates first would allow a malicious client to mount a mostly offline key recovery attack.

B. *OpenSSL*

Prior to early 2015, OpenSSL defaulted to using static-ephemeral Diffie-Hellman values. Server applications generate a fresh Diffie-Hellman secret exponent on startup, and reuse this exponent until they are restarted. A server would be vulnerable to small subgroup attacks if it chose a DSA prime, explicitly configured the `dh->length` parameter to generate a short exponent, and failed to set `SSL_OP_SINGLE_DH_USE` to prevent repeated exponents. OpenSSL provides some test code for key generation which configures DSA group parameters, sets an exponent length to the group order, and correctly sets the `SSL_OP_SINGLE_DH_USE` to generate new exponents on every connection. We found this test code widely used across many applications. We discovered that Unbound, a DNS

Implementation	RFC 5114 Support	Allows Short Exponents	Reuses Exponents	Validates Subgroup
Mozilla NSS	No	Yes, hardcoded	No	$g \leq 2$
OpenJDK	No	Yes, uses max of p_size / 2 and 384	No	$g \leq 2$
OpenSSL 1.0.2	Yes	Yes, if q set or if user sets a shorter length	Default until Jan '16	Yes, as of Jan '16
BouncyCastle	Yes	No	Application dependent	$g \leq 2$
Cryptlib	No	Yes, uses quadratic curve calculation	Application dependent	$g \leq 2$
libTomCrypt	No	Yes, hardcoded	Application dependent	No
CryptoPP	No	Yes, uses work factor calculation	Application dependent	No
Botan	Yes	Yes, uses work factor calculation	No	No
GnuTLS	Application dependent	Yes, restricts to q_size (max 256)	Application dependent	No

TABLE II: **TLS Library Behavior**—We examined popular TLS libraries to determine which weaknesses from Section II-F were present. Reuse of exponents often depends on the use of the library; the burden is on the application developer to appropriately regenerate exponents. Botan and libTomCrypt both hardcode their own custom groups, while GnuTLS allows users to specify their own parameters.

resolver, used the same parameters as the tests, but without setting `SSL_OP_SINGLE_DH_USE`, rendering them vulnerable to a key recovery attack. A number of other applications including Lighttpd used the same or similar code with non-safe primes, but correctly set `SSL_OP_SINGLE_DH_USE`.

In spring 2015, OpenSSL added explicit support for RFC 5114 groups [6], including the ability for servers to specify a subgroup order in a set of Diffie-Hellman group parameters. When the subgroup order is specified, the exponent length is automatically adjusted to match the subgroup size. However, the update did not contain code to validate subgroup order for key exchange values, leaving OpenSSL users vulnerable to precisely the key recovery attack outlined in Section III-A.

We disclosed this vulnerability to OpenSSL in January 2016. The vulnerability was patched by including code to validate subgroup order when a subgroup was specified in a set of Diffie-Hellman parameters and setting `SSL_OP_SINGLE_DH_USE` by default [15]. Prior to this patch, any code using OpenSSL for DSA-style Diffie-Hellman parameters was vulnerable to small subgroup attacks by default.

Exim [4], a popular mail server that uses OpenSSL, provides a clear example of the fragile situation created by this update. By default, Exim uses the RFC 5114 Group 23 parameters with OpenSSL, does not set an exponent length, and does not set `SSL_OP_SINGLE_DH_USE`. In a blog post, an Exim developer explains that because of "numerous issues with automatic generation of DH parameters", they added support for fixed groups specified in RFCs and picked Group 23 as the default [12]. Exim narrowly avoided being fully vulnerable to a key recovery attack by not including the size of the subgroup generated by q in the Diffie-Hellman parameters that it passes to OpenSSL. Had this been included, OpenSSL would have automatically shortened the exponent length, leaving the server fully vulnerable to a key recovery attack. For this group, an attacker can recover 130 bits of information about the secret exponent using 2^{33} online queries, but this does not allow the attacker to recover the server's 2048-bit exponent modulo the correct 224-bit group order q as the small subgroup orders q_i are all relatively prime to q.

We looked at several other applications as well, but did not find them to be vulnerable to key recovery attacks (Table I).

C. Other Implementations

We examined the source code of multiple TLS implementations (Table II). Prior to January 2016, no TLS implementations that we examined validated group order, even for the well-known DSA primes from RFC 5114, leaving them vulnerable to small subgroup confinement attacks.

Most of the implementations we examined attempt to match exponent length to the perceived strength of the prime. For example, Mozilla Network Security Services (NSS), the TLS library used in the Firefox browser and some versions of Chrome [7], [36], uses NIST's "comparable key strength" recommendations on key management [21], [22] to determine secret exponent lengths from the length of the prime. [2] Thus NSS uses 160-bit exponents with a 1024-bit prime, and 224-bit exponents with a 2048-bit prime. In fall 2015, NSS added an additional check to ensure that the shared secret $g^{x_a x_b} \not\equiv 1 \bmod p$ [5].

Several implementations go to elaborate lengths to match exponent length to perceived prime strength. The Cryptlib library fits a quadratic curve to the small exponent attack cost table in the original van Oorschot paper [69] and uses the fitted curve to determine safe key lengths [41]. The Crypto++ library uses an explicit "work factor" calculation, evaluating the function $2.4n^{1/3}(\log n)^{2/3}$ [46]. Subgroup order and exponent lengths are set to twice the calculated work factor. The work factor calculation is taken from a 1995 paper by Odlyzko on integer factorization [58]. Botan, a C++ cryptography and TLS library, uses a similar work factor calculation, derived from RFC 3766 [42], which describes best practices as of 2004 for selecting public key strengths when exchanging symmetric keys. RFC 3766 uses a similar work factor algorithm to Odlyzko, intended to model the running time of the number-field sieve. Botan then doubles the length of the work factor to obtain subgroup and exponent lengths [9].

D. Measurements

We used ZMap [32] to probe the public IPv4 address space for hosts serving three TLS-based protocols: HTTPS, SMTP+STARTTLS, and POP3S. To determine which primes servers were using, we sent a `ClientHello` message containing only ephemeral Diffie-Hellman cipher suites. We combined this data with scans from Censys [30] to determine the overall population. The results are summarized in Table III.

In August 2016, we conducted additional scans of a random 1% sample of HTTPS hosts on the Internet. First, we checked for nontrivial small subgroup attack vulnerability. For servers that sent us a prime p such that $p-1$ was divisible by 7, we attempted a handshake using a client key exchange value of

Protocol	Scan Date	Total Hosts	*Number of hosts that use…* Diffie-Hellman	Non-Safe Primes	Static Exponents	Static Exponents and Non-Safe Primes
HTTPS	2/2016	40,578,754	10,827,565	1,661,856	964,356	309,891
POP3S	10/2015	4,368,656	3,371,616	26,285	32,215	25
STARTTLS	10/2015	3,426,360	3,036,408	1,186,322	30,017	932
SSH	10/2015	15,226,362	10,730,527	281	1,147	0
IKEv1	2/2016	2,571,900	2,571,900	340,300	109	0
IKEv2	2/2016	1,265,800	1,265,800	177,000	52	0

TABLE III: **IPv4 non-safe prime and static exponent usage**—Although non-safe primes see widespread use across most protocols, only a small number of hosts reuse exponents and use non-safe primes; these hosts are prime candidates for a small subgroup key recovery attack.

$g_7 \bmod p$, where g_7 is a generator of a subgroup of order 7. (7 is the smallest prime factor of $p-1$ for Group 22.) When we send g_7, we expect to correctly guess the `PreMasterSecret` and complete the handshake with one seventh of hosts that do not validate subgroup order. In our scan, we were able to successfully complete a handshake with 1477 of 10714 hosts that offered a prime such that $p-1$ was divisible by 7, implying that approximately 96% of these hosts fail to validate subgroup order six months after OpenSSL pushed a patch adding group order validation for correctly configured groups.

Second, we measured how many hosts performed even the most basic validation of key exchange values. We attempted to connect to HTTPS hosts with the client key exchange values of $y_c = 0 \bmod p, 1 \bmod p, -1 \bmod p$. As Table IV shows, we found that over 5% of hosts that accepted DHE ciphersuites accepted the key exchange value of $-1 \bmod p$ and derived the `PreMasterSecret` from it. These implementations are vulnerable to a trivial version of the small subgroup confinement attacks described in Section III-A, for *any* prime modulus p. By examining the default web pages of many of these hosts, we identified products from several notable companies including Microsoft, Cisco, and VMWare. When we disclosed these findings, VMWare notified us that they had already applied the fix in the latest version of their products; Microsoft acknowledged the missing checks but chose not to include them since they only use safe primes, and adding the checks may break functionality for some clients that were sending unusual key exchange values; and Cisco informed us that they would investigate the issue.

Key Exchange Value	Support DHE	Accepted
$0 \bmod p$	143.5 K	87
$1 \bmod p$	142.2 K	4.9 K
$-1 \bmod p$	143.5 K	7.6 K
$g_7 \bmod p$	10.7 K	1.5 K

TABLE IV: **TLS key exchange validation**—We performed a 1% HTTPS scan in August 2016 to check if servers validated received client key exchange values, offering generators of subgroups of order 1, 2 and 7. Our baseline DHE support number counts hosts willing to negotiate a DHE key exchange, and in the case of g_7, if $p-1$ is divisible by 7. We count hosts as "Accepted" if they reply to the `ClientKeyExchange` message with a `Finished` message. For g_7, we expect this to happen with probability $1/7$, suggesting that nearly all of the hosts in our scan did not validate subgroup order.

Of 40.6 M total HTTPS hosts found in our scans, 10.8 M (27%) supported ephemeral Diffie-Hellman, of which 1.6 M (4%) used a non-safe prime, and 309 K (0.8%) used a non-safe prime and reused exponents across multiple connections, making them likely candidates for a small subgroup key recovery attack. We note that the numbers for hosts reusing exponents are an underestimate, since we only mark hosts as such if we found them using the same public Diffie-Hellman value across multiple connections, and some load balancers that cycle among multiple values might have evaded detection.

While 77% of POP3S hosts and 39% of SMTP servers used a non-safe prime, a much smaller number used a non-safe prime and reused exponents (<0.01% in both protocols), suggesting that the popular implementations (Postfix and Dovecot [31]) that use these primes follow recommendations to use ephemeral Diffie-Hellman values with DSA primes.

Table V shows nine groups that accounted for the majority of non-safe primes used by hosts in the wild. Over 1.17 M hosts across all of our HTTPS scans negotiated Group 22 in a key exchange. To get a better picture of which implementations provide support for this group, we examined the default web pages of these hosts to identify companies and products, which we show in Table VI.

Of the the 307 K HTTPS hosts that both use non-safe primes and reuse exponents, 277 K (90%) belong to hosts behind Amazon's Elastic Load Balancer [8]. These hosts use a 1024-bit prime with a 160-bit subgroup. We set up our own load balancer instance and found that the implementation failed to validate subgroup order. We were able to use a small-subgroup key recovery attack to compute 17 bits of our load balancer's private Diffie-Hellman exponent x_s in only 3813 queries. We responsibly disclosed this vulnerability to Amazon. Amazon informed us that they have removed Diffie-Hellman from their recommended ELB security policy, and are encouraging customers to use the latest policy. In May 2016, we performed additional scans and found that 88% of hosts using this prime no longer repeated exponents. We give a partial factorization for $p-1$ in Table XII; the next largest subgroups have 61 and 89 bits and an offline attack against the remaining bits of a 160-bit exponent would take 2^{71} time. For more details on the computation, see Section VI.

SSLeay [33], a predecessor for OpenSSL, includes several default Diffie-Hellman primes, including a 512-bit prime. We found that 717 SMTP servers used a version of the

Group			Host Counts			
Source	**Prime Size**	**Subgroup Size**	**HTTPS**	**SMTP**	**POP3S**	**SSH**
RFC 5114 Group 22	1024	160	1,173,147	145	86	0
Amazon Load Balancer	1024	160	277,858	0	1	0
JDK	768	160	146,491	671	16,515	0
JDK	1024	160	52,726	2,445	9,510	0
RFC 5114 Group 24	2048	256	3,543	5	0	6
JDK	2048	224	982	12	20	0
Epson Device	1024	< 948	372	0	0	0
RFC 5114 Group 23	2048	224	371	1,140,363	2	0
Mistyped OpenSSL 512	512	497	0	717	0	0
Other Non-Safe Primes	—	—	6,366	41,964	151	275
Safe Primes	—	—	9,165,709	1,850,086	3,345,331	10,730,246
Total			10,827,565	3,036,408	3,371,616	10,730,527

TABLE V: **IPv4 top non-safe primes**—Nine non-safe primes account for the majority of hosts using non-safe primes.

OpenSSL 512-bit prime with a single character difference in the hexadecimal representation. The resulting modulus that these servers use for their Diffie-Hellman key exchange is no longer prime. We include the factorization of this modulus along with the factors of the resulting group order in Table XII. The use of a composite modulus further decreases the work required to perform a small subgroup attack.

Although TLS also includes static Diffie-Hellman cipher suites that require a DSS certificate, we did not include them in our study; no browser supports static Diffie-Hellman [44], and Censys shows no hosts with DSS certificates, with only 652 total hosts with non-RSA or ECDSA certificates.

IV. IPsec

IPsec is a set of Layer-3 protocols which add confidentiality, data protection, sender authentication, and access control to IP traffic. IPsec is commonly used to implement VPNs. IPsec uses the Internet Key Exchange (IKE) protocol to determine the keys used to secure a session. IPsec may use IKEv1 [43] or IKEv2 [49]. While IKEv2 is not backwards-compatible with IKEv1, the two protocols are similar in message structure and purpose. Both versions use Diffie-Hellman to negotiate shared secrets. The groups used are limited to a fixed set of pre-determined choices, which include the DSA groups from RFC 5114, each assigned a number by IANA [49], [51], [53].

Company	Product(s)	Count
Ubiquiti Networks	airOS/EdgeOS	272,690
Cisco	DPC3848VM Gateway	65,026
WatchGuard	Fireware XTM	62,682
Supermicro	IPMI	42,973
ASUS	AiCloud	39,749
Electric Sheep Fencing	pfSense	14,218
Bouygues Telecom	Bbox	13,387
Other	—	135,432

TABLE VI: **HTTPS support for RFC5114 Group 22**—In a 100% HTTPS scan performed in October 2016, we found that of the 12,835,911 hosts that accepted Diffie-Hellman key exchange, 901,656 used Group 22. We were able to download default web pages for 646,157 of these hosts, which we examined to identify companies and products.

IKEv1. IKEv1 [43], [55], [60] has two basic methods for authenticated key exchange: Main Mode and Aggressive Mode. Main Mode requires six messages to establish the requisite state. The initiator sends a Security Association (`SA`) payload, containing a selection of cipher suites and Diffie-Hellman groups they are willing to negotiate. The responder selects a cipher and responds with its own `SA` payload. After the cipher suite is selected, the initiator and responder both transmit Key Exchange (`KE`) payloads containing public Diffie-Hellman values for the chosen group. At this point, both parties compute shared key materials, denoted `SKEYID`. When using signatures for authentication, `SKEYID` is computed $\texttt{SKEYID} = \mathrm{prf}(N_i|N_r, g^{x_i x_r})$. For the other two authentication modes, pre-shared key and public-key encryption, `SKEYID` is derived from the pre-shared key and session cookies, respectively, and does not depend on the negotiated Diffie-Hellman shared secret.

Each party then in turn sends an authentication message (`AUTH`) derived from a hash over `SKEYID` and the handshake. The authentication messages are encrypted and authenticated using keys derived from the Diffie-Hellman secret $g^{x_i x_r}$. The responder only sends her `AUTH` message after receiving and validating the initiator's `AUTH` message.

Aggressive Mode operates identically to Main Mode, but in order to reduce latency, the initiator sends `SA` and `KE` messages together, and the responder replies with its `SA`, `KE`, and `AUTH` messages together. In aggressive mode, the responder sends an authentication message first, and the authentication messages are not encrypted.

IKEv2. IKEv2 [48], [49] combines the `SA` and `KE` messages into a single message. The initiator provides a best guess ciphersuite for the `KE` message. If the responder accepts that proposal and chooses not to renegotiate, the responder replies with a single message containing both `SA` and `KE` payloads. Both parties then send and verify `AUTH` messages, starting with the initiator. The authentication messages are encrypted using session keys derived from the `SKEYSEED` value which is derived from the negotiated Diffie-Hellman shared secret. The standard authentication modes use public-key signatures over the handshake values.

A. Small Subgroup Attacks in IPsec

There are several variants of small subgroup attacks against IKEv1 and IKEv2. We describe the attacks against these protocols together in this section.

Small subgroup confinement attacks. First, consider attacks that can be carried out by an attacking initiator or responder. In IKEv1 Main Mode and in IKEv2, either peer can carry out a small subgroup confinement attack against the other by sending a generator of a small subgroup as its key exchange value. The attacking peer must then guess the other peer's view of the Diffie-Hellman shared secret to compute the session keys to encrypt its authentication message, leading to a mostly online attack. However, in IKEv1 Aggressive Mode, the responder sends its `AUTH` message before the initiator, and this value is not encrypted with a session key. If signature authentication is being used, the `SKEYID` and resulting hashes are derived from the Diffie-Hellman shared secret, so the initiator can perform an offline brute-force attack against the responder's authentication message to learn their exponent in the small subgroup.

Now, consider a man-in-the-middle attacker. Bhargavan, Delignat-Lavaud, and Pironti [25] describe a transcript synchronization attack against IKEv2 that relies on a small subgroup confinement attack. A man-in-the-middle attacker initiates simultaneous connections with an initiator and a responder using identical nonces, and sends a generator g_i for a subgroup of small order q_i to each as its `KE` message. The two sides have a $1/q_i$ chance of negotiating an identical shared secret, so an authentication method depending only on nonces and shared secrets could be forwarded, and the session keys would be identical.

If the attacker also has knowledge of the secrets used for authentication, more attacks are possible. Similar to the attack described for TLS, such an attacker can use a small subgroup confinement attack to force a connection to use weak encryption. The attacker only needs to rewrite a small number of handshake messages; any further encrypted communications can then be decrypted at leisure without requiring the man-in-the-middle attacker to continuously rewrite the connection. We consider a man-in-the-middle attacker who modifies the key exchange message from both the initiator and the responder to substitute a generator g_i of a subgroup of small order q_i. The attacker must then replace the handshake authentication messages, which would require knowledge of the long-term authentication secret. We describe this attack for each of pre-shared key, signatures, and public-key authentication.

For pre-shared key authentication in IKEv1 Main Mode, IKEv1 Aggressive Mode, and IKEv2, the man-in-the-middle attacker must only know the pre-shared key to construct the authentication hash; the authentication message does not depend on the negotiated Diffie-Hellman shared secret. With probability $1/q_i$, the two parties will agree on the Diffie-Hellman shared secret. The attacker can then brute force this value after viewing messages encrypted with keys derived from it.

For signature authentication in IKEv1 Main Mode and in IKEv2, the signed hash transmitted from each side is derived from the nonces and the negotiated shared secret, which is confined to one of q_i possible values. The attacker must know the private signing keys for both initiator and responder and brute force `SKEYID` from the received signature in order to forge the modified authentication signatures on each side. The communicating parties will have a q_i chance of agreeing on the same value for the shared secret to allow the attack to succeed. For IKEv1 Aggressive Mode, the attack can be made to succeed every time. The responder's key exchange message is sent together with their signature which depends on the negotiated shared secret, so the man-in-the-middle attacker can brute force the q_i possible values of the responders private key x_r and replace the responder's key exchange message with $q_i^{x_r}$, forging an appropriate signature with their knowledge of the signing key.

For public key authentication in IKEv1 Main Mode, IKEv1 Aggressive Mode, and IKEv2, the attacker must know the private keys corresponding to the public keys used to encrypt the ID and nonce values on both sides in order to forge a valid authentication hash. Since the authentication does not depend on the shared Diffie-Hellman negotiated value, a man-in-the-middle attacker must then brute force the negotiated shared key once they receives a message encrypted with the derived key. The two parties will agree on their view of the shared key with probability $1/q_i$, allowing the attack to succeed.

Small subgroup key recovery attacks. Similar to TLS, an IKE responder that reuses private exponents and does not verify that the initiator key exchange values are in the correct subgroup is vulnerable to a small subgroup key recovery attack. The most recent version of the IKEv2 specification has a section discussing reuse of Diffie-Hellman exponents, and states that "because computing Diffie-Hellman exponentials is computationally expensive, an endpoint may find it advantageous to reuse those exponentials for multiple connection setups" [49]. Following this recommendation could leave a host open to a key recovery attack, depending on how exponent reuse is implemented. A small subgroup key recovery attack on IKE would be primarily offline for IKEv1 with signature authentication and for IKEv2 against the initiator.

For each subgroup of order q_i, the attacker's goal is to obtain a responder `AUTH` message, which depends on the secret chosen by the responder. If an `AUTH` message can be obtained, the attacker can brute-force the responder's secret within the subgroup offline. This is possible if the server supports IKEv1 Aggressive Mode, since the server authenticates before the client, and signature authentication produces a value dependent on the negotiated secret. In all other IKE modes, the client authenticates first, leading to an online attack. The flow of the attack is identical to TLS; for more details see Section III.

Ferguson and Schneier [34] describe a hypothetical small-subgroup attack against the initiator where a man-in-the-middle attacker abuses undefined behavior with respect to UDP packet retransmissions. A malicious party could "retransmit" many key exchange messages to an initiator and potentially receive a different authentication message in response to each, allowing a mostly offline key recovery attack.

B. Implementations

We examined several open-source IKE implementations to understand server behavior. In particular, we looked for implementations that generate small Diffie-Hellman exponents, repeat exponents across multiple connections, or do not correctly validate subgroup order. Despite the suggestion in IKEv2 RFC 7296 to reuse exponents [49], none of the implementations that we examined reused secret exponents.

All implementations we reviewed are based on

FreeS/WAN [13], a reference implementation of IPSec. The final release of FreeS/Wan, version 2.06, was released in 2004. Version 2.04 was forked into Openswan [16] and strongSwan [17], with a further fork of Openswan into Libreswan [14] in 2012. The final release of FreeS/WAN used constant length 256-bit exponents but did not support RFC 5114 DSA groups, offering only the Oakley 1024-bit and 1536-bit groups that use safe primes.

Openswan does not generate keys with short exponents. By default, RFC 5114 groups are not supported, although there is a compile-time option that can be explicitly set to enable support for DSA groups. strongSwan both supports RFC 5114 groups and has explicit hard-coded exponent sizes for each group. The exponent size for each of the RFC 5114 DSA groups matches the subgroup size. However, these exponent sizes are only used if the `dh_exponent_ansi_x9_42` configuration option is set. It also includes a routine inside an `#ifdef` that validates subgroup order by checking that $g^q \equiv 1 \bmod p$, but validation is not enabled by default. Libreswan uses Mozilla Network Security Services (NSS) [7] to generate Diffie-Hellman keys. As discussed in Section III-C, NSS generates short exponents for Diffie-Hellman groups. Libreswan was forked from Openswan after support for RFC 5114 was added, and retains support for those groups if it is configured to use them.

Although none of the implementations we examined were configured to reuse Diffie-Hellman exponents across connections, the failure to validate subgroup orders even for the pre-specified groups renders these implementations fragile to future changes and vulnerable to subgroup confinement attacks.

Several closed source implementations also provide support for RFC 5114 Group 24. These include Cisco's IOS [27], Juniper's Junos [47], and Windows Server 2012 R2 [56]. We were unable to examine the source code for these implementations to determine whether or not they validate subgroup order.

C. Measurements

We performed a series of Internet scans using ZMap to identify IKE responders. In our analysis, we only consider hosts that respond to our ZMap scan probes. Many IKE hosts that filter their connections based on IP are excluded from our results. We further note that, depending on VPN server configurations, some responders may continue with a negotiation that uses weak parameters until they are able to identify a configuration for the connecting initiator. At that point, they might reject the connection. As an unauthenticated initiator, we have no way of distinguishing this behavior from the behaviour of a VPN server that legitimately accepts weak parameters. For a more detailed explanation of possible IKE responder behaviors in response to scanning probes, see Wouters [70].

In October 2016, we performed a series of scans offering the most common cipher suites and group parameters we found in implementations to establish a baseline population for IKEv1 and IKEv2 responses. For IKEv1, the baseline scan offered Oakley groups 2 and 14 and RFC 5114 groups 22, 23, and 24 for the group parameters; SHA1 or SHA256 for the hash function; pre-shared key or RSA signatures for the authentication method; and AES-CBC, 3DES, and DES for the encryption algorithm. Our IKEv2 baseline scan was similar, but also offered the 256-bit and 384-bit ECP groups and AES-GCM for authenticated encryption.

On top of the baseline scans, we performed additional scans to measure support for the non-safe RFC 5114 groups and for key exchange parameter validation. Table VII shows the results of the October IKE scans. For each RFC 5114 DSA group, we performed four handshakes with each host; the first tested for support by sending a valid client key exchange value, and the three others tested values that should be rejected by a properly-validating host. We did not scan using the key exchange value 0 because of a vulnerability present in unpatched Libreswan and Openswan implementations that causes the IKE daemon to restart when it receives such a value [3].

We considered a host to accept our key exchange value if after receiving the value, it continued the handshake without any indication of an error. We found that 33.2% of IKEv1 hosts and 17.7% of IKEv2 hosts that responded to our baseline scans supported using one of the RFC 5114 groups, and that a surprising number of hosts failed to validate key exchange values. 24.8% of IKEv1 hosts that accepted Group 23 with a valid key exchange value also accepted $1 \bmod p$ or $-1 \bmod p$ as a key exchange value, even though this is explicitly warned against in the RFC [59]. This behavior leaves these hosts open to a small subgroup confinement attack even for safe primes, as described in Section II-F.

For safe groups, a check that the key exchange value is strictly between 1 and $p-1$ is sufficient validation. However, when using non-safe DSA primes, it is also necessary to verify that the key exchange value lies within the correct subgroup (i.e., $y^q \equiv 1 \bmod p$). To test this case, we constructed a generator of a subgroup that was not the intended DSA subgroup, and offered that as our key exchange value. We did not find any IKEv1 hosts that rejected this key exchange value after previously accepting a valid key exchange value for the given group. For IKEv2, the results were similar with the exception of Group 24, where still over 93% of hosts accepted this key exchange value. This suggests that almost no hosts supporting DSA groups are correctly validating subgroup order.

We observed that across all of the IKE scans, 109 IKEv1 hosts and 52 IKEv2 hosts repeated a key exchange value. This may be due to entropy issues in key generation rather than static Diffie-Hellman exponents; we also found 15,891 repeated key exchange values across different IP addresses. We found no hosts that used both repeated key exchange values and non-safe groups. We summarize these results in Table III.

V. SSH

SSH contains three key agreement methods that make use of Diffie-Hellman. The "Group 1" and "Group 14" methods denote Oakley Group 2 and Oakley Group 14, respectively [71]. Both of these groups use safe primes. The third method, "Group Exchange", allows server to select a custom group [35]. The group exchange RFC specifies that all custom groups should use safe primes. Despite this, RFC 5114 notes that group exchange method allows for its DSA groups in SSH, and advocates for their immediate inclusion [53].

In all Diffie-Hellman key agreement methods, after negotiating cipher selection and group parameters, the SSH client generates a public key exchange value $y_c = g^{x_c} \bmod p$ and sends it to the server. The server computes its own Diffie-Hellman public value $y_s = g^{x_s} \bmod p$ and sends it to the client, along with a signature from its host key over the resulting shared secret $Y = g^{x_s x_c} \bmod p$ and the hash of the handshake so far. The client verifies the signature before continuing.

Protocol	Groups Offered	Support	*Client key exchange public values offered...* 1 mod p	-1 mod p	g_s mod p
IKEv1	Group 22	332.4 K	82.6 K	78.5 K	332.4 K
	Group 23	333.4 K	82.5 K	82.5 K	333.4 K
	Group 24	379.8 K	93.9 K	95.2 K	379.8 K
	Baseline (Groups 2, 14, 22, 23, 24)	1139.3 K	–	–	–
IKEv2	Group 22	182.1 K	553	553	181.9 K
	Group 23	181.9 K	542	550	180.1 K
	Group 24	213.0 K	2245	2173	200.0 K
	Baseline (Groups 2, 14, 19, 20, 22, 23, 24)	1203.7 K	–	–	–

TABLE VII: **IKE group support and validation.**—We measured support for RFC5114 DSA groups in IKEv1 and IKEv2 and test for key exchange validation by performing a series of 100% IPv4 scans in October 2016. For Group 23, g_s is a generator of a subgroup with order 3, and for Groups 22 and 24, g_s is a generator of a subgroup of order 7.

A. *Small Subgroup Attacks in SSH*

Small subgroup confinement attacks. An SSH client could execute a small subgroup confinement attack against an SSH server by sending a generator g_i for a subgroup of small order q_i as its client key exchange, and immediately receive the server's key exchange $g^{x_s} \bmod p$ together with a signature that depends on the server's view of the shared secret $Y_s = g_i^{x_s} \bmod p$. For small q_i, this allows the client to brute force the value of $x_s \bmod q_i$ offline and compare to the server's signed handshake to learn the correct value of $x_s \bmod q_i$. To avoid this, the SSH RFC specifically recommends using safe primes, and to use exponents at least twice the length of key material derived from the shared secret [35].

If client and server support Diffie-Hellman group exchange and the server uses a non-safe prime, a man in the middle with knowledge of the server's long-term private signing key can use a small subgroup confinement attack to man-in-the-middle the connection without having to rewrite every message. The attack is similar to the case of TLS: the man in the middle modifies the server group and key exchange messages, leaving the prime unchanged, but substituting a generator g_i of a subgroup of small order q_i for the group generator and g_i for the server's key exchange value y_s. The client then responds with a client key exchange message $y_c = g_i^{x_c} \bmod p$, which the man in the middle leaves unchanged. The attacker then forges a correct signature for the modified server group and key exchange messages and passes it to the client. The server's view of the shared secret is $g_i^{x_c x_s} \bmod p$, and the client's view of the shared secret is $g_i^{x_c} \bmod p$. As in the attack described for TLS, these views are identical when $x_s \equiv 1 \bmod q_i$, so this connection will succeed with probability $1/q_i$. For a small enough q_i, this enables a man in the middle to use a compromised server signing key to decrypt traffic with a reasonable probability of success, while only requiring tampering with the initial handshake messages, rather than having to actively rewrite the entire connection for the duration of the session.

Small subgroup key recovery attacks. Since the server immediately sends a signature over the public values and the Diffie-Hellman shared secret, an implementation using static exponents and non-safe primes that is vulnerable to a small subgroup confinement attack would also be vulnerable to a mostly offline key recovery attack, as a malicious client would only need to send a single key exchange message per subgroup.

B. *Implementations*

Censys [30] SSH banner scans show that the two most common SSH server implementations are Dropbear and OpenSSH. Dropbear group exchange uses hard-coded safe prime parameters from the Oakley groups and validates that client key exchange values are greater than 1 and less than $p-1$. While OpenSSH only includes safe primes by default, it does provide the ability to add additional primes and does not provide the ability to specify subgroup orders. Both OpenSSH and Dropbear generate fresh exponents per connection.

We find one SSH implementation, Cerberus SFTP server (FTP over SSH), repeating server exponents across connections. Cerberus uses OpenSSL, but fails to set `SSL_OP_SINGLE_DH_USE`, which was required to avoid exponent reuse prior to OpenSSL 1.0.2f.

C. *Measurements*

Of the 15.2 M SSH servers on Censys, of which 10.7 M support Diffie-Hellman group exchange, we found that 281 used a non-safe prime, and that 1.1 K reused Diffie-Hellman exponents. All but 26 of the hosts that reused exponents had banners identifying the Cerberus SFTP server. We encountered no servers that both reused exponents and used non-safe primes.

We performed a scan of 1% of SSH hosts in February 2016 offering the key exchange values of $y_c = 0 \bmod p, 1 \bmod p$ and $p-1 \bmod p$. As Table VIII shows, 33% of SSH hosts failed to validate group order when we sent the key exchange value $p-1 \bmod p$. Even when safe groups are used, this behaviour allows an attacker to learn a single bit of the private exponent, violating the decisional Diffie-Hellman assumption and leaving the implementation open to a small subgroup confinement

Key Exchange Value	Handshake Initiated	Accepted
$0 \bmod p$	175.6 K	5.7 K
$1 \bmod p$	175.0 K	43.9 K
$-1 \bmod p$	176.0 K	59.0 K

TABLE VIII: **SSH validation**—In a 1% SSH scan performed in February 2016, we sent the key exchange values $y_c = 0, 1$ and $p-1$. We count hosts as having initiated a handshake if they send a `SSH_MSG_KEX_DH_GEX_GROUP`, and we count hosts as "Accepted" if they reply to the client key exchange message with a `SSH_MSG_KEX_DH_GEX_REPLY`.

Prime	Exact Order Known							Exact Order Unknown	
lg(p)	160 bits	224 bits	256 bits	300 bits	lg(p) − 8	lg(p) − 32	lg(p) − 64	Unlikely DSA	Likely DSA
512	3	0	0	0	5	0	0	760	43
768	4	0	0	4	2,685	0	0	220	1,402
1024	29	0	0	0	323	944	176	1,559	26,881
2048	0	1	1	0	0	0	0	1,128	4,890
3072	0	0	0	0	0	5	0	9	152
4096	4	0	0	0	0	0	0	20	183
8192	0	0	0	0	0	0	0	0	1
Other	0	0	0	0	0	0	0	400	15

TABLE IX: **Distribution of orders for groups with non-safe primes**—For groups for which we were able to determine the subgroup order exactly, 160-bits subgroup orders are common. We classify other groups to be likely DSA groups if we know that the subgroup order is at least 8 bits smaller than the prime.

attack (Section III-A).

VI. Factoring Group Orders of Non-Safe Primes

Across all scans, we collected 41,847 unique groups with non-safe primes. To measure the extent to which each group would facilitate a small subgroup attack in a vulnerable implementation, we attempted to factor $(p-1)/2$. We used the GMP-ECM [39] implementation of the elliptic curve method for integer factorization on a local cluster with 288 cores over a several-week period to opportunistically find small factors of the group order for each of the primes.

Given a group with prime p and a generator g, we can check whether the generator generates the entire group or generates a subgroup by testing whether $g^{q_i} \equiv 1 \bmod p$ for each factor q_i of $(p-1)/2$. When $g^{q_i} \equiv 1 \bmod p$, then if q_i is prime, we know that q_i is the exact order of the subgroup generated by g; otherwise q_i is a multiple of the order of the subgroup. We show the distribution of group order for groups using non-safe primes in Table IX. We were able to completely factor $p-1$ for 4,701 primes. For the remaining primes, we did not obtain enough factors of $(p-1)/2$ to determine the group order.

Of the groups where we were able to deduce the exact subgroup orders, several thousand had a generator for a subgroup that was either 8, 32, or 64 bits shorter than the prime itself. Most of these were generated by the Xlight FTP server, a closed-source implementation supporting SFTP. It is not clear whether this behavior is intentional or a bug in an implementation intending to generate safe primes. Primes of this form would lead to a more limited subgroup confinement or key recovery attack.

Given the factorization of $(p-1)/2$, and a limit for the amount of online and offline work an attacker is willing to invest, we can estimate the vulnerability of a given group to a hypothetical small subgroup key recovery attack. For each subgroup of order q_i, where q_i is less than the online work limit, we can learn q_i bits of the secret key via an online brute-force attack over all elements of the subgroup. To recover the remaining bits of the secret key, an attacker could use the Pollard lambda algorithm, which runs in time proportional to the square root of the remaining search space. If this runtime is less than the offline work limit, we can recover the entire secret key. We give work estimates for the primes we were able to factor and the number of hosts that would be affected by such a hypothetical attack in Table X.

The DSA groups introduced in RFC 5114 [53] are of particular interest. We were able to completely factor $(p-1)/2$ for both Group 22 and Group 24, and found several factors for Group 23. We give these factorizations in Table XII. In Table XI, we show the amount of online and offline work required to recover a secret exponent for each of the RFC 5114 groups. In particular, an exponent of the recommended size used with Group 23 is fully recoverable via a small subgroup attack with 33 bits of online work and 47 bits of offline work.

VII. Discussion

The small subgroup attacks require a number of special conditions to go wrong in order to be feasible. For the case of small subgroup confinement attacks, a server must both use a non-safe group and fail to validate subgroup order; the widespread failure of implementations to implement or enable group order validation means that large numbers of hosts using non-“safe” primes are vulnerable to this type of attack.

For a full key recovery attack to be possible the server must additionally reuse a small static exponent. In one sense, it is surprising that any implementations might satisfy all of the requirements for a full key recovery attack at once. However, when considering all of the choices that cryptographic libraries leave to application developers when using Diffie-Hellman, it is surprising that any protocol implementations manage to use Diffie-Hellman securely at all.

We now use our results to draw lessons for the security and cryptographic communities, provide recommendations for future cryptographic protocols, and suggest further research.

RFC 5114 Design Rationale. Neither NIST SP 800-56A nor RFC 5114 give a technical justification for fixing a much smaller subgroup order than the prime size. Using a shorter private exponent comes with performance benefits. However, there are no known attacks that would render a short exponent used with a safe prime less secure than an equivalently-sized exponent used with in a subgroup with order matched to the exponent length. The cryptanalyses of both short exponents and small subgroups are decades old.

If anything, the need to perform an additional modular exponentiation to validate subgroup order makes Diffie-Hellman over DSA groups *more* expensive than the safe prime case, for identical exponent lengths. As a more minor effect, a number field sieve-based cryptanalytic attack against a DSA prime is computationally slightly easier than against a safe prime. The design rationale may have its roots in preferring to implicitly use the assumption that Diffie-Hellman is secure for a small

Exponent	Work (bits) Online	Work (bits) Offline	HTTPS Groups	HTTPS Hosts	MAIL Groups	MAIL Hosts	SSH Groups	SSH Hosts
160	20	30	3	2	3	7	0	0
160	30	45	517	1,996	1963	1,143,524	11	10
160	40	60	3,701	8,495	13,547	1,159,853	109	68
224	20	30	0	0	0	0	0	0
224	30	45	2	2	14	16	0	0
224	40	60	307	691	1039	1,141,840	3	1
256	20	30	0	0	0	0	0	0
256	30	45	0	0	1	1	0	0
256	40	60	42	478	180	1,140,668	0	0

TABLE X: **Full key recovery attack complexity**—We estimate the amount of work required to carry out a small subgroup key recovery attack, and show the prevalence of those groups in the wild. Hosts are vulnerable if they reuse exponents and fail to check subgroup order.

prime-order subgroup without conditions on exponent length, rather than assuming Diffie-Hellman with short exponents is secure inside a group of much larger order. Alternatively, this insistence may stem from the fact that the security of DSA digital signatures requires the secret exponent to be uniformly random, although no such analogous attacks are known for Diffie-Hellman key exchange. [57] Unfortunately, our empirical results show that the necessity to specify and validate subgroup order for Diffie-Hellman key exchange makes implementations more fragile in practice.

Cryptographic API design. Most cryptographic libraries are designed with a large number of potential options and knobs to be tuned, leaving too many security-critical choices to the developers, who may struggle to remain current with the diverse and ever-increasing array of cryptographic attacks. These exposed knobs are likely due to a prioritization of performance over security. These confusing options in cryptographic implementations are not confined to primitive design: Georgiev et al. [37] discovered that SSL certificate validation was broken in a large number of non-browser TLS applications due to developers misunderstanding and misusing library calls. In the case of the small subgroup attacks, activating most of the conditions required for the attack will provide slight performance gains for an application: using a small exponent decreases the work required for exponentiation, reusing Diffie-Hellman exponents saves time in key generation, and failing to validate subgroup order saves another exponentiation. It is not reasonable to assume that applications developers have enough understanding of algebraic groups to be able to make the appropriate choices to optimize performance while still providing sufficient security for their implementation.

Group	Exponent Size	Online Work	Offline Work
Group 22	160	8	72
Group 23	224	33	47
Group 24	256	32	94

TABLE XI: **Attacking RFC 5114 groups**—We show the log of the amount of work in bits required to perform a small subgroup key recovery attack against a server that both uses a static Diffie-Hellman exponent of the same size as the subgroup order and fails to check group order.

Cryptographic standards. Cryptographic recommendations from standards committees are often too weak or vague, and, if strayed from, provide little recourse. The purpose of standardized groups and standardized validation procedures is to help remove the onus from application developers to know and understand the details of the cryptographic attacks. A developer should not have to understand the inner workings of Pollard lambda and the number field sieve in order to size an exponent; this should be clearly and unambiguously defined in a standard. However, the tangle of RFCs and standards attempting to define current best practices in key generation and parameter sizing do not paint a clear picture, and instead describe complex combinations of approaches and parameters, exposing the fragility of the cryptographic ecosystem. As a result, developers often forget or ignore edge cases, leaving many implementations of Diffie-Hellman too close to vulnerable for comfort. Rather than provide the bare minimums for security, the cryptographic recommendations from standards bodies should be designed for defense-in-depth such that a single mistake on the part of a developer does not have disastrous consequences for security. The principle of defense-in-depth has been a staple of the systems security community; cryptographic standards should similarly be designed to avoid fragility.

Protocol design. The interactions between cryptographic primitives and the needs of protocol designs can be complex. The after-the-fact introduction of RFC 5114 primes illustrates some of the unexpected difficulties: both IKE and SSH specified group validation only for safe primes, and a further RFC specifying extra group validation checks needed to be defined for IKE. Designing protocols to encompass many unnecessary functions, options, and extensions leaves room for implementation errors and makes security analysis burdensome. IKE is a notorious example of a difficult-to-implement protocol with many edge cases. Just Fast Keying (JFK), a protocol created as a successor to IKEv1, was designed to be an exceedingly simple key exchange protocol without the unnecessarily complicated negotiations present in IKE [19]. However, the IETF instead standardized IKEv2, which is nearly as complicated as IKEv1. Protocols and cryptosystems should be designed with the developer in mind—easy to implement and verify, with limited edge cases. The worst possible outcome is a system that appears to work, but provides less security than expected.

To construct such cryptosystems, secure-by-default primitives are key. As we show in this paper, finite-field based

Source	Factored Completely?	Order Factorization
RFC 5114 Group 22	Yes	2^3 * 7 * df * 183a872bdc5f7a7e88170937189 * 228c5a311384c02e1f287c6b7b2d * 5a85 7d66c65a60728c353e32ece8be1 * f518aa8781a8df278aba4e7d64b7cb9d49462353 * 1a3adf8 d6a69682661ca6e590b447e66ebd1bbdeab5e6f3744f06f46cf2a8300622ed50011479f18143d471 a53d30113995663a447dcb8e81bc24d988edc41f21
RFC 5114 Group 23	No	3^2 * 5 * 2b * 49 * 9d * 5e9a5 * 93ee1 * 2c3f0539 * 136c58359 * 1a30b7358d * 335 a378eb0d * 801c0d34c58d93fe997177101f80535a4738cebcbf389a99b36371eb * 22bbe4b573 f6fc6dc24fef3f56e1c216523b3210d27b6c078b32b842aa48d35f230324e48f6dc2a10dd23d28d3 82843a78f264495542be4a95cb05e41f80b013f8b0e3ea26b84cd497b43cc932638530a068ecc44a f8ea3cc84139f0667100d426b60b9ab82b8de865b0cbd633f41366622011006632e0832e827febb7 066efe4ab4f1b2e99d96adfaf1721447b167cb49c372efcb82923b3731433cecb7ec3ebbc8d67ef4 41b5d11fb3328851084f74de823b5402f6b038172348a147b1ceac47722e31a72fe68b44ef4b
RFC 5114 Group 24	Yes	7 * d * 9f5 * 22acf * bd9f34b1 * 8cf83642a709a097b447997640129da299b1a47d1eb3750 ba308b0fe64f5fbd3 * 15adfe949ebb242e5cd0978fac1b43fdbd2e5b0c5f48924fbbd370195c0e b20596d98ad0a9e3fd98876413d926f41a8b918d2ec4b018a30efe5e336bf3c7ce60d515cf46af5f acf3bb389f68ad0c4ed2f0b1dbb970293741eb6509c64e731802259a639a7f57d4a9c0d9445241f5 bcdbdc50555b76d9c335c1fa4e11a8351f1bf4730dd67ffed877cc13e8ea40c7d51441c1f4e59155 ef1159eca75a2359f5e0284cd7f3b982c32e5c51dbf51b45f4603ef46bae528739315ca679703c1f fcf3b44fe3da5999daadf5606eb828fc57e46561be8c6a866361
Amazon Load Balancer	No	2 * 3 * 5 * edb * 181ac5dbfe5ce13b * 18aa349859e9e9de09b7d65 * 9414a18a7b575e8f4 2f6cb2dbc22eb1fc21d4929 * 2de9f1171a2493d46a31d508b63532cdf86d21db6f50f717736fc4 b0b722856a504ed4916e0484fe4ba5f5f4a9fff28a1233b728b3d043aec37c4f138ffd58fe7a8c3c 1e93cb52be527395e45db487b61daadded9c8ec35
Mistyped OpenSSL 512 "Prime" Factors	Yes	5 * b * a9b461e1636f4b51ef * 1851583cf5f9f731364e4aa6cdc2cac4f01* 3f0b39cacfc086 df4baf46c7fa7d1f4dfe184f9d22848325a91c519f79023a4526d8369e86b
Mistyped OpenSSL 512 Order Factors	Yes	2^13 * 3^3 * 5^2 * 11^2 * 269 * 295 * 4d5 * 97c3 * 9acfe7 * 8cdd0e128f * 385 b564eecd613536818f949 * 146d410923e999f8c291048dc6feffcebf8b9e99eec9a4d585f87422 e49b393256c23c9

TABLE XII: **Group order factorization for common non-safe primes**—We used the elliptic curve method to factor $(p-1)/2$ for each of the non-safe primes we found while scanning, as well as the mistyped OpenSSL "prime".

Diffie-Hellman has many edge cases that make its correct use difficult, and which occasionally arise as bugs at the protocol level. For example, SSH and TLS allow the server to generate arbitrary group parameters and send them to the client, but provide no mechanism for the server to specify the group order so that the client can validate the parameters. Diffie-Hellman key exchange over groups with different properties cannot be treated as a black-box primitive at the protocol level.

Recommendations. As a concrete recommendation, modern Diffie-Hellman implementations should prefer elliptic curve groups over safe curves with proper point validation [24]. These groups are much more efficient and have shorter key sizes than finite-field Diffie-Hellman at equivalent security levels. The TLS 1.3 draft includes a list of named curves designed to modern security standards [65]. If elliptic curve Diffie-Hellman is not an option, then implementations should follow the guidelines outlined in RFC 7919 for selecting finite field Diffie-Hellman primes [38]. Specifically, implementations should prefer "safe" primes of documented provenance of at least 2048 bits, validate that key exchange values are strictly between 1 and $p-1$, use ephemeral key exchange values for every connection, and use exponents of at least 224 bits.

Acknowledgments

We would like to thank the exceptional sysadmin Jose Antonio Insua Fernandez for his support. This material is based upon work supported by the U.S. National Science Foundation under Grants No. CNS-1345254, CNS-1408734, CNS-1409505, CNS-1505799, CNS-1513671, and CNS-1518888, an Alfred P. Sloan Foundation Research Fellowship, and a gift from Cisco.

References

[1] Finite field cryptography based samples. http://csrc.nist.gov/groups/ST/toolkit/documents/Examples/KS_FFC_All.pdf.

[2] NSS dh.c. https://hg.mozilla.org/projects/nss/file/tip/lib/freebl/dh.c.

[3] CVE-2015-3240. Available from MITRE, CVE-ID CVE-2015-3240., Aug. 2015. http://cve.mitre.org/cgi-bin/cvename.cgi?name=2015-3240.

[4] Exim Internet mailer, July 2015. http://www.exim.org/.

[5] Mozilla bug tracker, Nov. 2015. https://bugzilla.mozilla.org/show_bug.cgi?id=1160139.

[6] OpenSSL changes, Jan. 2015. https://www.openssl.org/news/cl102.txt.

[7] Overview of NSS, Sept. 2015. https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Overview.

[8] Amazon Elastic Load Balancer, 2016. https://aws.amazon.com/elasticloadbalancing/.

[9] Botan, 2016. https://github.com/randombit/botan.

[10] Bug 1837 - small subgroup attack, May 2016. https://bugs.exim.org/show_bug.cgi?id=1837.

[11] CVE-2016-0701. Available from MITRE, CVE-ID CVE-2016-0701., Jan. 2016. http://cve.mitre.org/cgi-bin/cvename.cgi?name=2016-0701.

[12] Exim TLS Security, DH and standard parameters, Oct. 2016. https://lists.exim.org/lurker/message/20161008.231103.c70b2da8.en.html.

[13] FreeS/WAN, 2016. http://www.freeswan.org/.

[14] Libreswan, 2016. https://libreswan.org/.

[15] OpenSSL security advisory [28th Jan 2016], Jan. 2016. https://www.openssl.org/news/secadv/20160128.txt.

[16] Openswan, 2016. https://www.openswan.org/.

[17] strongSwan, 2016. https://www.strongswan.org/.

[18] 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.

[19] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold. Just fast keying: Key agreement in a hostile internet. *ACM Transactions on Information and System Security (TISSEC)*, 7(2):242–273, 2004.

[20] R. Anderson and S. Vaudenay. Minding your p's and q's. In *ASIACRYPT*, 1996.

[21] E. Barker. NIST special publication 800-57 part 1 revision 4, Jan. 2014. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.

[22] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. NIST special publication 800-57 part 1 (revised), Jan. 2007. http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf.

[23] E. B. Barker, D. Johnson, and M. E. Smid. Sp 800-56a. recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised). 2007.

[24] D. J. Bernstein and T. Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography, Jan. 2014. https://safecurves.cr.yp.to/.

[25] K. Bhargavan, A. Delignat-Lavaud, and A. Pironti. Verified contributive channel bindings for compound authentication. In *Network and Distributed System Security Symposium*, 2015.

[26] K. Bhargavan and G. Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH. In *Network and Distributed System Security Symposium*, 2016.

[27] Cisco. Security for VPNs with IPsec configuration guide, 2016. http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_conn_vpnips/configuration/xe-3s/sec-sec-for-vpns-w-ipsec-xe-3s-book.html.

[28] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) protocol. IETF RFC RFC5246, 2008.

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

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

[31] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, K. Thomas, V. Eranti, N. Lidzborski, E. Bursztein, M. Bailey, and J. A. Halderman. The Matter of Heartbleed. In *15th ACM Internet Measurement Conference*, Oct. 2015.

[32] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *22nd USENIX Security Symposium*, Aug. 2013.

[33] Y. Eric. SSLeay, 1995. ftp://ftp.pl.vim.org/vol/rzm1/replay.old/libraries/SSL.eay/SSLeay-0.5.1a.tar.gz.

[34] N. Ferguson and B. Schneier. A cryptographic evaluation of IPsec. *Counterpane Internet Security, Inc*, 3031, 2000.

[35] M. Friedl, N. Provos, and W. Simpson. Diffie-Hellman group exchange for the Secure Shell (SSH) transport layer protocol. IETF RFC 4419, 2006.

[36] S. Gallagher. Google dumps plans for OpenSSL in Chrome, takes own Boring road, July 2014. http://arstechnica.com/information-technology/2014/07/google-dumps-plans-for-openssl-in-chrome-takes-own-boring-road/.

[37] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *2012 ACM Conference on Computer and Communications Security*, 2012.

[38] D. Gillmor. Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS). IETF RFC 7919, Aug. 2016.

[39] GMP-ECM Development Team. GMP-ECM, an implementation of the elliptic curve method for integer factorization, 2016. http://ecm.gforge.inria.fr/.

[40] D. M. Gordon. Discrete logarithms in $\mathrm{GF}(p)$ using the number field sieve. *SIAM Journal of Discrete Math*, 1993.

[41] P. Gutmann. Cryptlib, kg_dlp.c, 2010. http://www.cypherpunks.to/~peter/cl343_beta.zip.

[42] O. H., P. S. Dev., H. P., and V. Consortium. Determining strengths for public keys used for exchanging symmetric keys. IETF RFC 3766, Apr. 2004.

[43] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). Nov. 1998.

[44] C. Hlauschek, M. Gruber, F. Fankhauser, and C. Schanes. Prying open Pandora's box: KCI attacks against TLS. In *9th USENIX Workshop on Offensive Technologies (WOOT '15)*, Aug. 2015.

[45] T. Jager, J. Schwenk, and J. Somorovsky. Practical invalid curve attacks on TLS-ECDH. In *20th European Symposium on Research in Computer Security*, 2015.

[46] W. Jeffrey. Crypto++, Nov. 2015. https://github.com/weidai11/cryptopp/blob/48809d4e85c125814425c621d8d0d89f95405924/nbtheory.cpp#L1029.

[47] Juniper TechLibrary. VPN feature guide for security devices, 2016. http://www.juniper.net/documentation/en_US/junos15.1x49/topics/reference/configuration-statement/security-edit-dh-group.html.

[48] C. Kaufman et al. Internet Key Exchange (IKEv2) protocol. IETF RFC 4306, Dec. 2005.

[49] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. Internet Key Exchange protocol version 2 (IKEv2). IETF RFC 7296, Oct. 2014.

[50] C. F. Kerry. Digital Signature Standard (DSS). FIPS PUB 186-4, July 2013.

[51] T. Kivinen and M. Kojo. More modular exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). IETF RFC 3526, May 2003.

[52] A. Langley, N. Modadugu, and B. Moeller. Transport Layer Security (TLS) False Start. IETF RFC Draft, June 2014.

[53] M. Lepinski and S. Kent. Additional Diffie-Hellman groups for use with ietf standards. IETF RFC 5114, 2008.

[54] C. H. Lim and P. J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *17th International Cryptology Conference*, 1997.

[55] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet security association and key management protocol ISAKMP. Nov. 1998.

[56] Microsoft Windows Networking Team. VPN interoperability guide for Windows Server 2012 R2, 2014. https://blogs.technet.microsoft.com/networking/2014/12/26/vpn-interoperability-guide-for-windows-server-2012-r2/.

[57] Nguyen and Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *Journal of Cryptology*, 15(3):151–176, 2002.

[58] A. M. Odlyzko. The future of integer factorization, July 1995. http://www.dtc.umn.edu/~odlyzko/doc/future.of.factoring.pdf.

[59] H. Orman. The Oakley key determination protocol. IETF RFC 2412, Nov. 1998.

[60] D. Piper. The Internet IP security domain of interpretation for ISAKMP. IETF RFC 2407, Nov. 1998.

[61] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over $\mathrm{GF}(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1), 1978.

[62] W. Polk, R. Housley, and L. Bassham. Algorithms and identifiers for the internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. IETF RFC 3279, Apr. 2002.

[63] J. M. Pollard. A Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975.

[64] M. J. Pollard. Kangaroos, Monopoly and discrete logarithms. *Journal of Cryptology*, 2000.

[65] E. Rescorla. The Transport Layer Security (TLS) protocol version 1.3 draft 16. IETF RFC Draft, Sept. 2016.

[66] A. Sanso. OpenSSL Key Recovery Attack on DH small subgroups, Jan. 2016. http://blog.intothesymmetry.com/2016/01/openssl-key-recovery-attack-on-dh-small.html.

[67] D. Shanks. Class number, a theory of factorization, and genera. In *Symposia in Pure Math*, volume 20. 1969.

[68] Y. Sheffer and S. Fluhrer. Additional Diffie-Hellman tests for the Internet Key Exchange protocol version 2 (IKEv2). IETF RFC 6989, 2013.

[69] P. C. Van Oorschot and M. J. Wiener. On Diffie-Hellman key agreement with short exponents. In *EUROCRYPT*, 1996.

[70] P. Wouters. 66% of VPN's are not in fact broken, Oct. 2015. https://nohats.ca/wordpress/blog/2015/10/17/66-of-vpns-are-not-in-fact-broken/.

[71] T. Ylonen and C. Lonvick. The Secure Shell (SSH) transport layer protocol. IETF RFC 4253, 2006.