Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice

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

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

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

ABSTRACT

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

We go on to consider Diffie-Hellman with 768- and 1024-bit groups. A small number of fixed or standardized groups are in use by millions of TLS, SSH, and VPN servers. Performing precomputations on a few of these groups would allow a passive eavesdropper to decrypt a large fraction of Internet traffic. In the 1024-bit case, we estimate that such computations are plausible given nation-state resources, and a close reading of published NSA leaks shows that the agency's attacks on VPNs are consistent with having achieved such a break. We conclude that moving to stronger key exchange methods should be a priority for the Internet community.

1. INTRODUCTION

Diffie-Hellman key exchange is widely used to establish session keys in Internet protocols. It is the main key exchange mechanism in SSH and IPsec and a popular option in TLS. We examine how Diffie-Hellman is commonly implemented and deployed with these protocols and find that, in practice, it frequently offers less security than widely believed.

There are two reasons for this. First, a surprising number of servers use weak Diffie-Hellman parameters or maintain support for obsolete 1990s-era export-grade crypto. More critically, the common practice of using standardized, hard-coded, or widely shared Diffie-Hellman parameters has the effect of dramatically reducing the cost of large-scale attacks, bringing some within range of feasibility today.

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

Active attacks on export ciphers in TLS. We identify a new attack on TLS, in which a man-in-the-middle attacker can downgrade a connection to export-grade cryptography. This attack is reminiscent of the FREAK attack [6], but applies to the ephemeral Diffie-Hellman ciphersuites and is a TLS protocol flaw rather than an implementation vulnerability. We present measurements that show that this attack applies to 8.4% of Alexa Top Million HTTPS sites and 3.4% of all HTTPS servers that have browser-trusted certificates. To exploit this attack, we implemented the number field sieve discrete log algorithm and carried out precomputation for a 512-bit Diffie-Hellman group used by 82% of the vulnerable servers. This allows us to compute individual discrete logs in minutes. Using our discrete log oracle, we can compromise connections to 7% of the Top Million sites. Discrete logs over larger groups have been computed before [7], but as far we are aware, this is the first time they have been exploited to expose concrete vulnerabilities in real-world systems.

We were also able to compromise Diffie-Hellman for many other servers because of design and implementation flaws and configuration mistakes. These include using a composite-order subgroup in combination with short exponents, which is vulnerable to a known attack of van Oorschot and Wiener [46], and the inability of clients to properly validate Diffie-Hellman parameters without knowing the subgroup order (which TLS has no provision to communicate). We implement these attacks and discover several vulnerable implementations.

Risks from common 1024-bit groups. We explore the implications of these attacks for 768- and 1024-bit groups, which are widely used in practice and still considered secure. We provide new estimates for the computational resources necessary to compute discrete logarithms in groups of these sizes, concluding that 768-bit groups are within range of academic teams, and 1024-bit groups may plausibly be within range of state-level attackers. In both cases, computing individual logs can be done efficiently after the initial precomputation. We then examine evidence from published Snowden documents suggesting that NSA may already be exploiting this capability to decrypt VPN traffic. We perform measurement studies to examine the implications of such an attack on the most commonly used groups in IKE, SSH, and TLS.

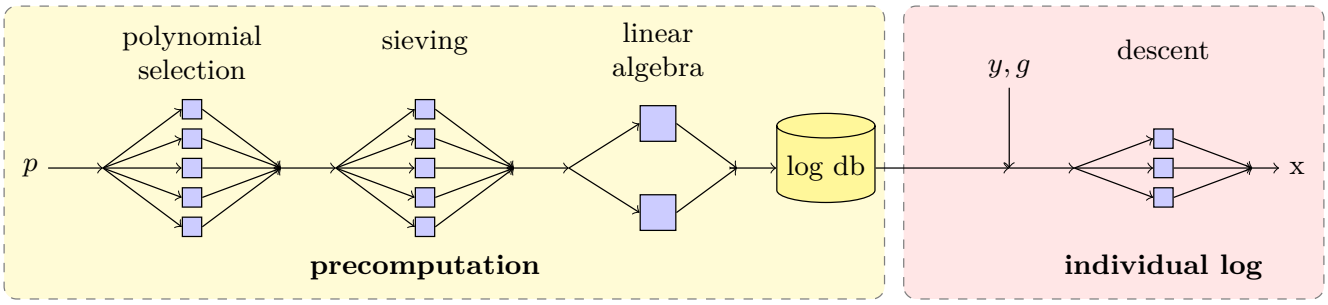

Figure 1: **The number field sieve algorithm for discrete log** consists of a precomputation stage that depends only on the prime p and a descent stage that computes individual logs. With sufficient precomputation, an attacker can quickly break any Diffie-Hellman instances using a particular p.

Mitigations and lessons. As a short-term countermeasure in response to our export-grade attacks on TLS, all mainstream browsers are implementing a more restrictive policy on the size of Diffie-Hellman groups they accept. We recommend that TLS servers disable export-grade cryptography and carefully vet the Diffie-Hellman groups they use. In the longer term, we advocate that protocols migrate to stronger Diffie-Hellman groups, such as those based on elliptic curves.

2. DIFFIE-HELLMAN CRYPTANALYSIS

Diffie-Hellman key exchange was the first published public-key algorithm [12]. In the simple case of prime groups, Alice and Bob agree on a prime p and a generator g of a multiplicative subgroup modulo p. Alice sends $g^a \bmod p$, Bob sends $g^b \bmod p$, and each computes a shared secret $g^{ab} \bmod p$.[1]

The security of Diffie-Hellman is not known to be equivalent to the discrete log problem (except in certain groups [11, 30, 31]), but computing discrete logs remains the best known cryptanalytic attack. An attacker who can find the discrete log x from $y = g^x \bmod p$ can easily find the shared secret.

Textbook descriptions of discrete log can be misleading about the computational tradeoffs, for example balancing parameters to minimize overall time to compute a single discrete log. In fact, as shown in Figure 1, a single large precomputation on p can be used to efficiently break a large number of different Diffie-Hellman exchanges made with that prime.

The typical case Diffie-Hellman is typically implemented with prime fields and large group orders. In this case, the most efficient discrete log algorithm is the number field sieve (NFS) [18, 21, 39].[2][3] The general technique is called index calculus, and has four stages with different computational properties. The first three steps are only dependent on the prime p, and comprise most of the computation.

First is *polynomial selection*, in which one finds a polynomial $f(z)$ defining a number field $\mathbb{Q}(z)/f(z)$ for the computation. (For our cases, $f(z)$ typically has degree 5 or 6.) This parallelizes well and is only a small portion of the runtime.

In the second stage, *sieving*, one factors ranges of integers and number field elements in batches to find many relations of elements, all of whose prime factors are less than some bound B (called B-smooth). Sieving parallelizes well, but is computationally expensive, because we must search through and attempt to factor many elements. The time for this step depends on heuristic estimates of the probability of encountering B-smooth numbers in this search.

In the third stage, *linear algebra*, we construct a large, sparse matrix consisting of the coefficient vectors of prime factorizations we have found. A non-zero kernel vector of the matrix modulo the order q of the group will give us logs of many small elements. This database of logs serves as input to the final stage. The difficulty depends on q and the matrix size and can be parallelized in a limited fashion.

The final stage, called *descent*, actually deduces the discrete log of the target y. We re-sieve until we can find a set of relations that allow us to write the log of y in terms of the logs in the precomputed database. This step is accomplished in three phases: an initialization phase, which sieves to write the target in terms of medium-sized primes, a middle phase, in which these medium-sized primes are further sieved until they can be represented by elements in the database of known logs, and a final phase that actually reconstructs the target using the log database. Crucially, descent is the only NFS stage that involves y (or g), so polynomial selection, sieving, and linear algebra can be done once for a prime p, and reused to compute the discrete logs of many targets.

The running time of this algorithm is $L_p(1/3, (64/9)^{1/3}) = \exp\left((1.923 + o(1))(\log p)^{1/3}(\log\log p)^{2/3}\right)$. This is obtained by carefully tuning the smoothness bound B and the sieving range. Early articles (e.g. [18]) encountered technical difficulties with descent and reported that the complexity of this step would equal the precomputation; this may have contributed to misconceptions about the performance of the NFS for discrete logs. More recent analysis has improved the complexity of descent to $L_p(1/3, 1.232)$ [2], much cheaper than the precomputation in practice.

The numerous parameters of the algorithm allow some flexibility to reduce time on some computational steps at the expense of others. For example, sieving more will result in a smaller matrix, making linear algebra cheaper, and doing more work in the precomputation makes the final descent step easier. In §3.3 we show how exploiting these trade-offs allows us to quickly compute 512-bit discrete logs in order to perform an effective man-in-the-middle attack on TLS.

[1] There is also a Diffie-Hellman exchange over elliptic curve groups; we address only the "mod p" case in this paper.

[2] Recent spectacular advances in discrete log algorithms have resulted in a quasi-polynomial algorithm for small-characteristic fields [3], but these advances are not known to apply to the prime fields used in practice.

[3] There is a closely related number field sieve algorithm for factoring [10, 28], and in fact many parts of the implementations can be shared.

Improperly generated groups A different family of algorithms runs in time exponential in group order, and they are practical even for large primes when the group order is small or has many small prime factors. To avoid this, most implementations use "safe" primes, which have the property that $p-1=2q$ for some prime q, so that the only possible subgroups have order 2 or q. However, as we show in §3.5, improperly generated groups are sometimes used in practice and susceptible to attack.

The baby-step giant-step [41] and Pollard rho [38] algorithms both take $\sqrt{q}$ time to compute a discrete log in any (sub)group of order q, while Pollard lambda [38] can find $x < t$ in time $\sqrt{t}$. These parallelize well [45], and precomputation can speed up individual log calculations. If the factorization of the subgroup order q is known, one can use any of the above algorithms to compute the discrete log in each subgroup of order $q_i^{e_i}$ dividing q, and then recover x using the Chinese remainder theorem. This is the Pohlig-Hellman algorithm [37], which costs $\sum_i e_i\sqrt{q_i}$ using baby-step giant-step or Pollard rho.

Standard primes Generating primes with special properties can be computationally burdensome, so many implementations use fixed or standardized Diffie-Hellman parameters. A prominent example is the Oakley groups [36], which give "safe" primes of length 768 (Oakley Group 1), 1024 (Oakley Group 2), and 1536 (Oakley Group 5). These groups were published in 1998 and have been used for many applications since, including IKE, SSH, Tor, and OTR.

When primes are of sufficient strength, there seems to be no disadvantage to reusing them. However, widespread reuse of Diffie-Hellman groups can convert attacks that are at the limits of an adversary's capabilities into devastating breaks, since it allows the attacker to amortize the cost of discrete log precomputation among vast numbers of potential targets.

3. ATTACKING TLS

TLS supports Diffie-Hellman as one of several possible key exchange methods, and about two-thirds of popular HTTPS sites allow it, most commonly using 1024-bit primes. However, a smaller number of servers also support legacy "export-grade" Diffie-Hellman using 512-bit primes that are well within reach of NFS-based cryptanalysis. Furthermore, for both normal and export-grade Diffie-Hellman, the vast majority of servers use a handful of common groups.

In this section, we exploit these facts to construct a novel attack against TLS. First, we perform NFS precomputations for the most popular 512-bit prime on the web, so that we can quickly compute the discrete log for any key-exchange message that uses it. Next, we show how a man-in-the-middle, so armed, can attack connections between popular browsers and any server that allows export-grade Diffie-Hellman, by using a TLS protocol flaw to downgrade the connection to export-strength and then recovering the session key. We find that this attack with a single precomputation can compromise about 6.9% of HTTPS servers among Alexa Top 1M domains.

3.1 TLS and Diffie-Hellman

The TLS handshake begins with a negotiation to determine the crypto algorithms used for the session. The client sends a list of supported ciphersuites (and a random nonce cr) within the ClientHello message, where each ciphersuite specifies a key exchange algorithm and other primitives. The server selects a ciphersuite from the client's list and signals its selection in a ServerHello message (containing a random nonce sr).

Source	Popularity	Prime
Apache	82 %	`9fdb8b8a004544f0045f1737d0ba2e0b 274cdf1a9f588218fb435316a16e3741 71fd19d8d8f37c39bf863fd60e3e3006 80a3030c6e4c3757d08f70e6aa871033`
mod_ssl	10%	`d4bcd52406f69b35994b88de5db89682 c8157f62d8f33633ee5772f11f05ab22 d6b5145b9f241e5acc31ff090a4bc711 48976f76795094e71e7903529f5a824b`
(*other*)	8%	(463 distinct primes)

Table 1: **Top 512-bit DH primes for TLS.** 8% of Alexa Top 1M HTTPS domains allow DHE_EXPORT, of which 92% use one of the two most popular primes, shown here.

TLS specifies ciphersuites supporting multiple varieties of Diffie-Hellman. Textbook Diffie-Hellman with unrestricted strength is called "ephemeral" Diffie-Hellman, or DHE, and is identified by ciphersuites that begin with `TLS_DHE_*`.[4] In DHE, the server is responsible for selecting the Diffie-Hellman parameters. It chooses a group (p, g), computes g^b, and sends a ServerKeyExchange message containing a signature over the tuple (cr, sr, p, g, g^b) using the long-term signing key from its certificate. The client verifies the signature and responds with a ClientKeyExchange message containing g^a.

To ensure agreement on the negotiation messages, and to prevent downgrade attacks [47], each party computes the TLS master secret from g^{ab} and calculates a MAC of its view of the handshake transcript. These MACs are exchanged in a pair of Finished messages and verified by the recipients. Thereafter, client and server start exchanging application data, protected by an authenticated encryption scheme with keys also derived from g^{ab}.

Export-grade Diffie-Hellman. To comply with 1990s-era U.S. export restrictions on cryptography, SSL 3.0 and TLS 1.0 supported reduced-strength DHE_EXPORT ciphersuites that were restricted to primes no longer than 512 bits. In all other respects, DHE_EXPORT protocol messages are identical to DHE. The relevant export restrictions are no longer in effect, but many libraries and servers maintain support for backwards compatibility. Many TLS servers are still configured with two groups: a strong 1024-bit group for regular DHE key exchanges and a 512-bit group for legacy DHE_EXPORT. This has been considered safe because most modern TLS clients do not offer or accept DHE_EXPORT ciphersuites.

To understand how HTTPS servers in the wild use Diffie-Hellman, we modified the ZMap [13] toolchain to offer DHE and DHE_EXPORT ciphersuites and scanned TCP/443 on both the full public IPv4 address space and the Alexa Top 1M domains. The scans took place in March 2015. Of 539,000 HTTPS sites among Top 1M domains, we found that 68.3% supported DHE and 8.38% supported DHE_EXPORT. Of 14.3 million IPv4 HTTPS servers with browser-trusted certificates, 23.9% supported DHE and 4.94% DHE_EXPORT.

[4]TLS also supports a "static" Diffie-Hellman format, where the server's key exchange value is fixed and contained in its certificate, but this is rarely used in practice. New ciphersuites that use elliptic curve Diffie-Hellman (ECDHE) are gaining in popularity, but in this paper we focus exclusively on the traditional prime-field ("mod p") variety.

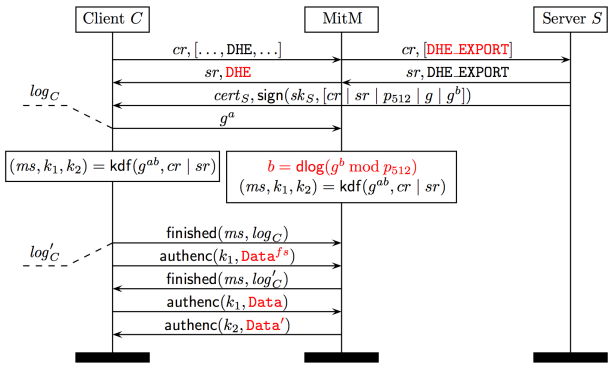

Figure 2: **DHE_EXPORT active downgrade attack.** A man-in-the-middle can force TLS clients to use export-strength DH with any server that allows DHE_EXPORT. Then, by finding the 512-bit discrete log, the attacker can learn the session key and arbitrarily read or modify the contents. $\mathtt{Data}^{fs}$ refers to False Start [27] application data that some TLS clients send before receiving the server's Finished.

While the TLS protocol allows servers to generate their own Diffie-Hellman parameters, the overwhelming majority use one of a handful of primes. As shown in Table 1, just two 512-bit primes account for 92% of Alexa Top 1M domains that support DHE_EXPORT, and 93% of all servers with browser-trusted certificates that support DHE_EXPORT. (Non-export DHE follows a similar distribution with longer primes.) The most popular 512-bit prime was hard-coded into many versions of Apache. Introduced in 2005 with Apache 2.1.5, it was used until 2.4.7, which disabled export ciphersuites. We found it in use by about 564,000 servers with browser-trusted certificates.

3.2 Active Downgrade to Export-Grade DHE

Given the widespread use of these primes, an attacker with the ability to compute discrete logs in 512-bit groups could efficiently break DHE_EXPORT handshakes for about 8% of Alexa Top 1M HTTPS sites, but modern browsers never negotiate export-grade ciphersuites. To circumvent this, we show how an attacker who can compute 512-bit discrete logs *in real time* can downgrade a regular DHE connection to use a DHE_EXPORT group, and thereby break both the confidentiality and integrity of application data.

The attack is depicted in Figure 2 and relies on a flaw in the way TLS composes DHE and DHE_EXPORT. When a server selects DHE_EXPORT for a handshake, it proceeds by issuing a signed ServerKeyExchange message containing a 512-bit p_{512}, but the structure of this message is identical to the message sent during standard DHE ciphersuites. Critically, the signed portion of the server's message fails to include any indication of the specific ciphersuite that the server has chosen. Provided that a client offers DHE, an active attacker can re-write the client's ClientHello to offer a corresponding DHE_EXPORT ciphersuite accepted by the server and remove other ciphersuites that could be chosen instead. The attacker re-writes the ServerHello response to replace the chosen DHE_EXPORT ciphersuite with a matching non-export ciphersuite and forwards the ServerKeyExchange message to the client as is. The client will interpret the export-grade tuple (p_{512}, g, g^b) as valid DHE parameters chosen by the server and proceed with the handshake. The client and server have different handshake transcripts at this stage, but an attacker who can compute b in real time can then derive the master secret and connection keys to complete the handshake with the client, and then freely read and write application data pretending to be the server.

There are two remaining challenges in implementing this active downgrade attack. The first is to compute individual discrete logs in close to real time, and the second is to delay handshake completion until the discrete log computation has had time to finish. We address these in the next subsections.

Comparison with FREAK. The attack is reminiscent of the recent FREAK [6] attack, in which an attacker downgrades a regular RSA key exchange to one that uses export-grade 512-bit ephemeral RSA keys, relying on a bug in several TLS client implementations. The attacker then factors the ephemeral key to hijack future connections that use the same key. The cryptanalysis takes several hours on commodity hardware and is usable until the server decides to regenerate a fresh ephemeral RSA key (typically when it restarts).

Our downgrade attack is due to a protocol flaw in TLS, not an implementation bug. From a client perspective, the only defense is to reject small primes in DHE handshakes. Prior to this work, most popular browsers accepted p of size ≥ 512 bits.[5] Requiring larger groups would prevent the downgrade attack. Our attack affects fewer HTTPS servers than FREAK, but, as we shall see, the cost per broken connection is far lower, since the precomputation for each 512-bit group can be used indefinitely against all servers that use the group, and since each individual discrete logarithm only takes a few minutes.

3.3 512-bit Discrete Log Computations

We modified CADO-NFS [1] to implement the number field sieve discrete log algorithm from §2 and applied it to two 512-bit primes, including the top DHE_EXPORT prime shown in Table 1. Precomputation took 7 days, for each prime, after which computing individual logs took a median time of 90 seconds. We list the runtime for each stage of the computation below. The times were about the same for both primes.

Precomputation As shown in Figure 1, the precomputation phase includes the polynomial selection, sieving, and linear algebra steps. For this precomputation, we deliberately sieved more than strictly necessary. This enabled two optimizations: first, with more relations obtained from sieving, we eventually obtain a larger database of known logs, which makes the descent faster. Second, more sieving relations also yield a smaller linear algebra step, which is desirable because sieving is much easier to parallelize than linear algebra.

For the polynomial selection and sieving steps, we used idle time on 2000–3000 CPU cores in parallel, of which most CPUs were Intel Sandy Bridge. Polynomial selection ran for about 3 hours, which in total corresponds to 7,600 core-hours. Sieving ran for 15 hours, corresponding to 21,400 core-hours. This sufficed to collect 40,003,519 relations of which 28,372,442 were unique, involving 15,207,865 large primes of at most 27 bits (hence bound B from §2 is 2^{27}).

From this data set, we obtained a square matrix with 2,157,378 rows and columns, with 113 non-zero coefficients

[5] In our experiments, Internet Explorer, Chrome, Firefox, Opera, all accepted 512-bit primes, whereas Safari allowed groups as small as 16 bits.

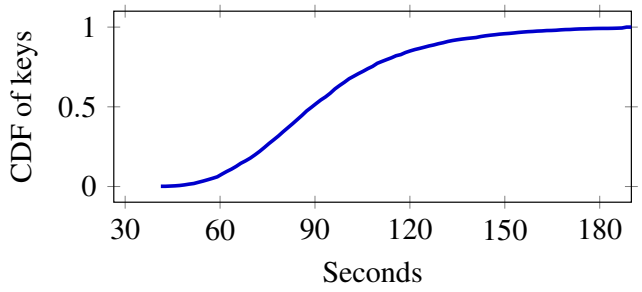

Figure 3: **Individual discrete log time for 512-bit DH.** After a week-long precomputation for the most common 512-bit prime used for DHE_EXPORT, we can quickly break TLS key exchanges that use it. Here we show the times for computing 3,500 individual logs; the median is 90 seconds.

per row on average. We solved the corresponding linear system on a 36-node cluster with two 8-core Intel Xeon E5-2650 CPUs per node, connected with Infiniband FDR. We used the block Wiedemann algorithm [9, 44] with parameters $m = 18$ and $n = 6$. Using the unoptimized implementation from CADO-NFS [1] for linear algebra over GF(p), the computation finished in 120 hours, corresponding to 60,000 core-hours. We expect that optimizations could bring this cost down by at least a factor of three.

In total, the wall-clock time for each precomputation was slightly over one week. The resulting database of known logs for the descent occupies about 2.5 GB in ASCII format.

Descent Once this precomputation was finished, we were able to run the final descent step to compute individual discrete logs in minutes for targets in each of these groups. In order to save time on individual computations, we implemented a client-server architecture using the ZeroMQ messaging library. The server maintains the precomputed data in RAM and returns logs for values passed to it by clients.

We implemented the descent calculation in a mix of Python and C. The first and second stages are parallelized and run sieving in C, and the final discrete log is deduced in Python. We ran the server on a machine with four 6-core Intel Xeon E7-8893 CPUs and 2 TB of RAM. (The memory is overkill for this application; 64 GB would be plenty.) On average, computing individual logs took about 90 seconds, but the time varied from 38–260 seconds (see Fig. 3). This is divided between about 20 seconds for descent initialization and the remainder on the middle phase, which is currently parallelized only in a limited fashion. Further optimizations—such as more effective parallelization or additional sieving—should bring the median time well below a minute.

For purposes of comparison, a single 512-bit RSA factorization using the CADO-NFS implementation takes about eight days of wall-clock time on the computer used for the descent, and about seven hours parallelized across 1,800 cores of Amazon ec2 `c4.8xlarge` instances.

3.4 Active Attack Implementation

We implemented a man-in-the-middle network attacker that sits between a TLS client (web browser) and any server that supports DHE_EXPORT and uses the most common 512-bit Apache group. Our implementation follows the message sequence in Figure 2: it downgrades the connection towards the server, computes the session keys, and takes over the connection towards the client by impersonating the server.

The main challenge is to compute the shared secret g^{ab} before the handshake completes in order to forge a Finished message from the server. With our descent implementation, the computation takes an average of 90 seconds, but there are several ways an attacker can work around this delay:

Non-browser clients Different TLS clients impose different time limits for the handshake, after which they kill the connection. Command-line clients such as `curl` and `git` often run unattended, so they have long or no timeouts, and we could hijack their connections without much difficulty.

TLS warning alerts Web browsers tend to have shorter timeouts, but we can keep browser connections alive by sending TLS warning alerts, which are ignored by the browser but reset the handshake timer. For example, this allowed us to keep Firefox's TLS connections alive indefinitely. (Other browsers closed the connection after a minute.) Although the victim connection still takes much longer than usual, the attacker might choose to compromise a request for a background resource that does not delay rendering the page.

Ephemeral key caching Many TLS servers do not use a fresh value b for each connection, but instead compute g^b once and reuse it for multiple negotiations, possibly until they are restarted. Without enabling the `SSL_OP_SINGLE_DH_USE` option, OpenSSL will reuse g^b for the lifetime of a TLS context. While both Apache and Nginx internally apply this option, certain load balancers, such as stud [43], do not. The F5 BIG-IP load balancers and hardware TLS frontends will reuse g^b unless the "Single DH" option is checked [48]. Microsoft Schannel caches g^b for two hours—this setting is hard-coded. For these servers, an attacker can compute the discrete log of g^b from one connection and use it to attack later handshakes, avoiding the need to complete the computation online. Based on a random sampling of IPv4 hosts serving browser-trusted certificates that support DHE, we found that 17% of TLS servers reused g^b at least once over the course of 20 handshakes, and that 15% only used one value. For DHE_EXPORT, only 0.1% reused g^b, likely because Microsoft IIS does not support 512-bit export ciphersuites.

TLS False Start Even when clients enforce shorter timeouts and servers do not reuse values for b, the attacker can still break the confidentiality of user requests if the client supports the TLS False Start extension [27]. This extension reduces connection latency by having the client send early application data without waiting for the server's Finished message to arrive. Recent versions of Chrome, Internet Explorer, and Firefox implement False Start, but their policies on when to enable this feature keeps changing between versions. Firefox 35, Chrome 41, and Internet Explorer (Windows 10) send False Start data with DHE.[6] In these cases, a man-in-the-middle can record the handshake and decrypt the False Start payload at leisure. We note that this initial data sent by a browser often contains sensitive user authentication information, such as passwords and cookies.

3.5 Other Weak and Misconfigured Groups

In our scans, we found several other exploitable security issues in the DHE configurations used by TLS servers.

[6] Firefox 36 disabled False Start for DHE, when Brian Smith raised concerns about weak Diffie-Hellman groups, similar to to those discussed in this paper: https://bugzilla.mozilla.org/show_bug.cgi?id=952863.

512-bit primes in non-export DHE We found 2,631 servers with browser-trusted certificates (and 118 in the Top 1M domains) that used 512-bit or weaker primes for non-export DHE. In these instances, active attacks may be unnecessary. If a browser negotiates a DHE ciphersuite with one of these servers, a *passive* eavesdropper can later compute the discrete log and obtain the TLS session keys for the connection. An active attack may still be necessary when the client's ordering of ciphersuites would result in the server not selecting DHE. In this case, as in the DHE_EXPORT downgrade attack, an active attacker can force the server to choose a vulnerable DHE ciphersuite.

As a proof-of-concept, we implemented a passive eavesdropper for regular DHE connections, and used it to decrypt test connections to www.fbi.gov. Until April 2015, this server used the default 512-bit DH group from OpenSSL, which was the second group for which we performed the NFS precomputation, enabling the attack. The website no longer supports DHE.

Attacks on Composite-Order Subgroups Failure to generate Diffie-Hellman primes according to known best practices can result in devastating attacks. Not every TLS server uses "safe" primes. Out of approximately 70,000 distinct primes seen across both export and non-export TLS scans, 4,800 were not safe, meaning that $(p-1)/2$ was composite. (Incidentally, we also found 9 composite p.) These groups are not necessarily vulnerable, as long as g generates a group with at least one sufficiently large subgroup order to rule out the Pohlig-Hellman algorithm as an attack.

In some real-life configurations however, choosing such primes can lead to an attack. For efficiency reasons, some implementations use ephemeral keys g^x with a short exponent x; common suggested sizes are as small as 160 or 224 bits, intended to match the estimated strength of a 1024 or 2048-bit group. For safe p, such exponent lengths are not known to decrease security, as the most efficient attack will be the Pollard lambda algorithm. But if the order of the subgroup generated by g has small factors, they can be used to recover information about exponents. From a subset of factors $\{q_1^{e_1} \dots q_k^{e_k}\}$ with $\prod_i q_i^{e_i} = z$, Pohlig-Hellman can recover $x \mod z$ in time $\sum_i e_i\sqrt{q_i}$. If $x \leq z$, this suffices to recover x. If not, Pollard lambda can use this information to recover x in time $\sqrt{x/z}$. This attack was first described as hypothetical by van Oorschot and Wiener [46].

To see if TLS servers in the wild were vulnerable to this attack, we tested various non-safe primes found in our scan. For each non-safe prime p, we opportunistically factored $p-1$ using Bernstein's batch method [4]. We then ran the GMP-ECM implementations of the Pollard $p-1$ algorithm and the ECM factoring methods [49] for 5 days parallelized across 28 cores and discovered 36,447 prime factors.

We then examined the generators g used with each prime p. We classified a tuple (p, g, y) sent by a server as interesting if the prime factorization of $p-1$ had revealed prime factors of the order of g, and ordered them by the estimated work required using Pohlig-Hellman and Pollard lambda to recover a target private exponent x of length ranging from 64 to 256 bits. There were 753 (p, g) pairs where we knew factors of the subgroup generated by g; these had been used for 40,903 connections across all of our scans.

We implemented the van Oorschot and Wiener algorithm in Sage, using a parallel Pollard rho implementation we wrote in C using the GMP library. We used the distinguished points method for collision detection; for a prime known in advance, this implementation can be arbitrarily sped up by precomputing a table of distinguished points.

We computed partial information about the server secret exponent used in 460 exchanges, and were able to recover the whole exponent used by 159 different hosts, 53 of which authenticated with valid browser-trusted certificates. In all cases, the vulnerable hosts used 512-bit prime moduli; three of them used 160-bit exponents whereas the rest used 128-bit exponents. The smallest-order subgroup had 46 bits (which Pollard rho handles in seconds) and the largest-order subgroup had 81 bits, which took 181260s–632012s in our implementation. The Pollard lambda calculations used interval width varying from 40 to 70 bits.

Our computations allowed us to hijack connections to a variety of vulnerable TLS servers, including web interfaces for VPN devices (48 hosts), communications software (21 hosts), web conferencing servers (27 hosts), and ftp servers (6 hosts). As a proof-of-concept, we modified our man-in-the-middle attacker of §3.3 to impersonate a vulnerable server and capture user credentials. Compared to an attack using NFS, we could compute the discrete log of the server ephemeral key, with a delay hardly noticeable for browser users.

Misconfigured groups The Digital Signature Algorithm (DSA) [34] uses primes p such that $p-1$ has a 160, 224, or 256-bit prime factor q and g generates only a subgroup of order q. When using properly generated DSA parameters, these groups are secure for use in Diffie-Hellman key exchanges. Notably, DSA groups are hard-coded in Java's `sun.security.provider` package, and are used by default in many Java-based TLS servers. However, some servers in our scans used Java's DSA primes as p, but mistakenly used the DSA group order q in the place of the generator g. We found 5,741 hosts misconfigured this way.

This substitution of q for g is likely due to a usability problem: the canonical ASN.1 representation of Diffie-Hellman key exchange parameters (coming from PKCS#3) is a sequence (p, g), while that of DSA parameters (coming from PKIX) is (p, q, g); we conjecture that the confusion between these formats led to a simple programming error.

In a DSA group, the subgroup generated by q is likely to have many small prime factors in its order, since for p generated according to [34], $(p-1)/q$ is a random integer. For Java's `sun.security.provider` 512-bit prime, using q as a generator leaks 290 bits of information about exponents at a cost of roughly 2^{40} operations. Luckily, since the provider generates exponents of length $\max(n/2, 384)$ for n-bit p, this does not suffice to recover a full exponent. Still, this misconfiguration bug results in a significant loss of security and serves as a cautionary tale for programmers.

4. STATE-LEVEL THREATS TO DH

The previous sections demonstrate the existence of practical attacks against Diffie-Hellman key exchange as currently used by TLS. However, these attacks rely on the ability to downgrade connections to export-grade crypto or on the use of unsafe parameters. In this section we address the following question: how secure is Diffie-Hellman in broader practice, as used in other protocols that do not suffer from downgrade, and when applied with stronger groups?

To answer this question we must first examine how the number field sieve for discrete log scales to 768- and 1024-bit

	Sieving			Linear Algebra		Descent	
	I	lpb	core-years	rows	core-years	core-time	
RSA-512	14	29	0.5	4.3M	0.33		Timings with default CADO-NFS parameters.
DH-512	15	27	2.5	2.1M	7.7	10 mins	For the computations in this paper; may be suboptimal.
RSA-768	16	37	800	250M	100		Est. based on [26] with less sieving.
DH-768	17	35	8,000	150M	28,500	2 days	Est. based on [7, 26] and own experiments.
RSA-1024	18	42	1,000,000	8.7B	120,000		Est. based on complexity formula.
DH-1024	19	40	10,000,000	5.2B	35,000,000	30 days	Est. based on complexity formula and our experiments.

Table 2: **Estimating costs for factoring and discrete log**. For sieving, we give two important parameters: the large prime bound `lpb` and a measure of how much sieving is happening per subprocess `I`. For linear algebra, all costs for DH are for safe primes; for DSA primes with q of 160 bits, this should be divided by 6.4 for 1024 bits, 4.8 for 768 bits, and 3.2 for 512 bits.

groups. As we argue below, 768-bit groups, which are still in relatively widespread use, are now within reach for academic computational resources, and performing precomputations for a small number of 1024-bit groups is plausibly within the resources of state-level attackers. The precomputation would likely require special-purpose hardware, but would not require any major algorithmic improvements beyond what is known in the academic literature. We further show that even in the 1024-bit case, the descent time — necessary to solve any specific discrete logarithm instance within a common group — would be fast enough to break individual key exchanges in close to real time.

In light of these results, we next examine several standard Internet security protocols — IKE, SSH, and TLS — to determine the vulnerability of these exchanges to attacks by resourceful attackers. Although the cost of the precomputation for a 1024-bit group is several times higher than for an RSA key of equal size, we observe that a one-time investment could be used to attack millions of hosts, due to widespread reuse of the most common Diffie-Hellman parameters. Unfortunately, our measurements also indicate that it may be very difficult to sunset the use of fixed 1024-bit Diffie-Hellman groups that have long been embedded in standards and implementations.

Finally, we apply this new understanding to a set of recently-published documents leaked by Edward Snowden [42], to evaluate the hypothesis that the National Security Agency has *already* implemented such a capability. We show that this hypothesis is consistent with the published details of the intelligence community's cryptanalytic capabilities, and indeed matches the known capabilities more closely than other proposed explanations, such as novel breaks on RC4 or AES. We believe that this analysis may help to shed light on unanswered questions about how NSA may be gaining access to VPN, SSH, and TLS traffic.

4.1 Scaling NFS to 768- and 1024-bit DH

Estimating the cost for discrete log cryptanalysis at longer key sizes is far from straightforward, due in part to the complexity of parameter tuning, and to tradeoffs between the sieving and linear algebra steps, which have very different computational characteristics. (Much more attention has gone to understanding 1024-bit factorization, but even there, many published estimates are crude extrapolations of the asymptotic complexity.) We attempt estimates for 768- and 1024-bit discrete log based on the existing literature and our own experiments, but further work is needed for greater confidence, particularly for the 1024-bit case. We summarize all the costs, measured or estimated, in Table 2.

DH-768: Feasible with academic power. For the 768-bit case, we base our estimates on the recent discrete log record at 596 bits [7] and the integer factorization record of 768 bits from 2009 [26]. While the algorithms for factorization and discrete log are similar, the discrete log linear algebra stage is many times more difficult, as the matrix entries are no longer boolean. We can reduce overall time by sieving more, thus generating a smaller input matrix to the linear algebra step. Since sieving parallelizes better than linear algebra, this tradeoff is desirable for large inputs.

A 596-bit factorization takes about 5 core-years, most of it spent on sieving. In comparison, the record 596-bit discrete log effort tuned parameters such that they spent 50 core-years on sieving. This reduced their linear algebra calculation to 80 core-years. We used this same strategy in our 512-bit experiments in §3.3.

Similarly, the 768-bit RSA factoring record spent more time in sieving in order to save time in the linear algebra step. The cost of sieving was around 1500 core-years, and the matrix that was produced had 200M rows and columns. As a result the linear algebra took 150 core-years, but taking algorithmic improvements since 2009 into account and optimizing for the total time[7], we estimate that factoring an RSA-768 integer would take 900 core-years in total.

For a 768-bit discrete log, we can expect that ten times as much sieving as the RSA case would reduce the matrix to around 150M rows. We extrapolate from experiments with existing software that this linear algebra would take 28,500 core-years, for a total of 36,500 core-years. This is within reach by computing power available to academics.

The descent step takes relatively little time. We experimented with both CADO-NFS and a new implementation with GMP-ECM based on the early-abort strategy described in [5]. Using these techniques, the initial descent phase took an average of around 1 core-day. The remaining phase uses sieving much as in the precomputation; extrapolating from experiments, the rest of the descent should take at most 1 core-day. In total, after precomputation, the cost of a single 768-bit discrete log computation is around 2 core-days and is easily parallelizable.

DH-1024: Plausible with state-level resources. Experimentally extrapolating sieving parameters to the 1024-bit case is difficult due to the tradeoffs between the steps of the algorithm and their relative parallelism. The prior work proposing parameters for factoring a 1024-bit RSA key is thin: [25] proposes large prime bounds of 42 bits, but the

[7]We would lower the large prime bounds and increase the sieving range compared to the parameters in [26].

proposed value of the sieving range `I` is clearly too small, giving too few smooth results per sieving subtask. Since no publicly available software can currently deal with values of `I` larger than those proposed, we could not experimentally update the estimates of this paper with more relevant parameter choices.

Without better parameter choices, we resort to extrapolating from asymptotic complexity. For the number field sieve, the complexity is $\exp\left((k + o(1))(\log N)^{1/3}(\log\log N)^{2/3}\right)$, where N is the integer to factor or the prime modulus for discrete log, and k is an algorithm-specific constant. This formula is inherently imprecise, since the $o(1)$ in the exponent can hide polynomial factors. This complexity formula, with $k = 1.923$, describes the overall time for both discrete log and factorization, which are both dominated by sieving and linear algebra in the precomputation. The space complexity (the size of the matrix in memory) is the square root of this function, *i.e.* the same function, taking $k = 0.9615$. Discrete log descent has a complexity of the same form as well; [2, Chapter 4] gives $k = 1.232$, using an early-abort strategy similar to the one in [5] mentioned above.

Evaluating the formula for 768- and 1024-bit N gives us estimated multiplicative factors by which time and space will increase from the 768- to the 1024-bit case. For precomputation, the total time complexity will increase by a factor of 1220, while space complexity will increase by a factor of 35. These are valid for both factorization and discrete log, since they have the same asymptotic behavior. Hence, for DH-1024, we get a total cost for the precomputation of about 45M core-years. The time complexity for each individual log after the precomputation should be multiplied by 95.

For 1024-bit descent, we experimented with our early-abort implementation to inform our estimates for descent initialization, which should dominate the individual discrete logarithm computation. Initialization for a random target in Oakley Group 2 took 22 core-days, yielding a few primes of at most 130 bits to be descended further. In twice this time, we reached primes of about 110 bits. At this point, we were certain to have bootstrapped the descent, and could continue down to the large prime bound in a few more core-days if proper sieving software were available. Thus we estimate that a 1024-bit descent would take about 30 core-days, once again easily parallelizable.

Costs in hardware Although 45M core-years is a huge computational effort, it is not necessarily out of reach for a nation state. Moreover, at this scale, significant cost savings could be realized by developing application-specific hardware.

Sieving is a natural target for hardware implementation. To our knowledge, the best prior description of an ASIC implementation of 1024-bit sieving is the 2007 work of Geiselmann and Steinwandt [16]. In the following, we update their estimates for modern techniques and adjust parameters for discrete log. We increase their chip count by a factor of ten to sieve more and save on linear algebra as above, giving an estimate of 3M chips to complete sieving in one year. Shrinking the dies from the 130 nm technology node used in the paper to a more modern size reduces costs, as transistors are cheaper at newer technologies. With standard transistor costs and utilization, this would cost about $2 per chip to manufacture, after fixed design and tape-out costs of roughly $2M [29]. This suggests that an $8M investment would buy enough ASICs to complete the DH-1024 sieving precomputation in one year.[8]

Estimating the financial cost for the linear algebra is more difficult, since there has been little work on designing chips that are suitable for the larger fields involved in discrete log. To derive a rough estimate, we can begin with general purpose hardware and the core-year estimate from Table 2. The Titan supercomputer [35] — at 300,000 CPU cores, currently the most powerful supercomputer in the U.S. — would take 117 years to complete the 1024-bit linear algebra stage. Titan was constructed in 2012 for $94M, suggesting a cost of $11B in supercomputers to finish this step in a year. In the context of factorization, moving linear algebra from general purpose CPUs to ASICs has been estimated to reduce costs by a factor of 80 [15]. If we optimistically assume that a similar reduction can be achieved for discrete log, the hardware cost to perform the linear algebra for DH-1024 in one year is plausibly on the order of hundreds of millions of dollars.

To put this dollar figure in context, the FY 2012 budget for the U.S. Consolidated Cryptologic Program (which includes the NSA) was $10.5 billion[9] [52]. The agency's classified 2013 budget request, which prioritized investment in "groundbreaking cryptanalytic capabilities to defeat adversarial cryptography and exploit internet traffic," included notable $100M increases in two programs [52]: "cryptanalytic IT services" (to $247M), and a cryptically named "cryptanalysis and exploitation services program C" (to $360M). NSA's leaked strategic plan for the period called for it to "continue to invest in the industrial base and drive the state of the art for high performance computing to maintain pre-eminent cryptanalytic capability for the nation" [58].

4.2 Is NSA Breaking 1024-bit DH?

Our calculations suggest that it is plausibly within NSA's resources to have performed number field sieve precomputations for at least a small number of 1024-bit Diffie-Hellman groups. This would allow them to break any key exchanges made with those groups in close to real time. If true, this would answer one of the major cryptographic questions raised by the Edward Snowden leaks: How is NSA defeating the encryption for widely used VPN protocols?

Classified documents published by Der Spiegel [42] indicate that NSA is passively decrypting IPsec connections at significant scale. The documents do not describe the cryptanalytic techniques used, but they do provide an overview of the attack system architecture. After reviewing how IPsec key establishment works, we will use the published information to evaluate the hypothesis that the NSA is leveraging precomputation to calculate discrete logs at scale.

IKE Internet Key Exchange (IKE) is the main key establishment protocol used for IPsec VPNs. There are two versions, IKEv1 [19] and IKEv2 [22], which differ in message structure but are conceptually similar. For the purpose of brevity, we will use IKEv1 terminology.

Each IKE session begins with a Phase 1 handshake, in which the client and server select a Diffie-Hellman group from a small set of standardized parameters and perform a key exchange to establish a shared secret, SKEYID. IKE provides several authentication mechanisms, including symmetric preshared keys (PSK). When IKEv1 is authenticated with a

[8] Since a step of descent uses sieving, the same hardware could likely be reused to speed calculations of individual logs.

[9] The National Science Foundation's budget was $7 billion.

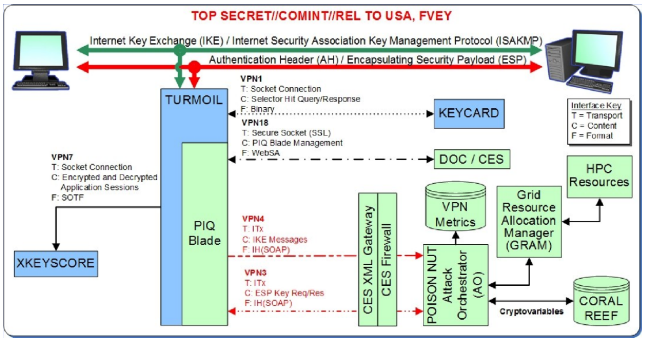

Figure 4: **NSA's VPN decryption infrastructure.** This classified illustration published by Der Spiegel [62] shows captured IKE handshake messages being passed to a high-performance computing system, which returns the symmetric keys for ESP session traffic. The details of this attack are consistent with an efficient break for 1024-bit Diffie-Hellman.

PSK, this value is incorporated into the derivation of SKEYID.

This shared secret is used to encrypt and authenticate a Phase 2 handshake. Phase 2 establishes the parameters and key material, KEYMAT, for a cryptographic transport protocol used to protect subsequent traffic, such as Encapsulating Security Payload (ESP) [24] or Authenticated Header (AH) [23]. In some circumstances, this phase includes an additional round of Diffie-Hellman. Ultimately, KEYMAT is derived from SKEYID, additional nonces, and the result of the optional Phase 2 Diffie-Hellman exchange.

NSA's VPN exploitation process The documents published by Der Spiegel describe a system named TURMOIL that is used to collect and decrypt VPN traffic. The evidence indicates that this decryption is performed using passive eavesdropping and does not require message injection or man-in-the-middle attacks on IPsec or IKE. Figure 4, an excerpt from one of the documents [62], illustrates the flow of information through the TURMOIL system

The initial phases of the attack involve collecting IKE and ESP payloads and determining whether the traffic matches any tasked selector [60]. If so, TURMOIL transmits the complete IKE handshake and may transmit a small amount of ESP ciphertext to NSA's Cryptanalysis and Exploitation Services (CES) [51, 60] via a secure tunnel. Within CES, a specialized VPN Attack Orchestrator (VAO) system manages a collection of high-performance grid computing resources located in the Tordella Supercomputer Building at NSA Headquarters and in a data center at Oak Ridge National Lab, which perform the computation required to generate the ESP session key [56, 57, 62]. VAO also maintains a database, CORALREEF, that stores cryptographic values, including a set of known PSKs and the resulting "recovered" ESP session keys [55, 56, 62].

The ESP traffic itself is buffered for up to 15 minutes [59], until CES can respond with the recovered ESP keys if they were generated correctly. Once keys have been returned, the ESP traffic is decrypted via hardware accelerators [54] or in software [63, 64]. From this point, decrypted VPN traffic is re-injected into TURMOIL processing infrastructure and passed to other systems for storage and analysis [64]. The documents indicate that NSA is recovering ESP keys at large scale, with a target of 100,000 per hour [59].

Evidence for a discrete log attack While the ability to decrypt VPN traffic does not by itself indicate a defeat of Diffie-Hellman, there are several features of IKE and the VAO's operation that support this hypothesis.

The IKE protocol has been extensively analyzed [8, 32], and is not believed to be exploitable in standard configurations under passive eavesdropping attacks. In order to recover the session keys for the ESP or AH protocols, the attacker must at minimum recover the SKEYID generated by the Phase 1 exchange. Absent a vulnerability in the key derivation function or transport encryption, this requires the attacker to recover a Diffie-Hellman shared secret after passively observing an IKE handshake.

While IKE is designed to support a range of Diffie-Hellman groups, our Internet-wide scans (§4.3) show that the vast majority of IKE systems select one particular 1024-bit DH group, Oakley Group 2, even when offered stronger groups.

Given an efficient oracle for solving the discrete logarithm problem, attacks on IKE are possible provided that the attacker can obtain the following: (1) a complete two-sided IKE transcript, including the Diffie-Hellman ephemeral keys g^a and g^b as well as the nonces and cookies transmitted by both sides of the connection, and (2) in IKEv1 only, the PSK used in deriving SKEYID.

Both of the above requirements are also present in the NSA's VPN attack system. As Figure 4 illustrates, a hard requirement of the VAO is the need to obtain the *complete* two-sided IKE transcript [55]. The published documents indicate that this requirement substantially increases the complexity of the attack execution, since IKE transcripts must be reassembled ("paired") whenever the interaction traverses multiple network paths [50, 51, 53, 61].

The attack system also seems to require knowledge of the PSK. Several documents describe techniques for analysts to locate a PSK, including using a database of router configurations [65, 66], the CORALREEF database of known PSKs [55], previously decrypted SSH traffic [55], or system administrator "chatter" [65]. Additionally, NSA is willing to "[r]un attacks to recover PSK" [55].

Of course, this explanation is not dispositive. The possibility remains that NSA could defeat IPsec using alternative means. Certain published NSA documents refer to software "implants" on VPN devices, indicating that the use of targeted malware is a piece of the collection strategy [55]; however, the same documents also note that decryption of the resulting traffic *does not* require IKE handshakes, and thus appears to be an alternative mechanism to the VAO attack described above. The most compelling argument for a pure cryptographic attack is the generality of the VAO approach, which appears to succeed across a broad swath of non-compromised devices.

4.3 Effects of a 1024-bit Break

In this section, we use Internet-wide scanning to assess the impact of a hypothetical DH-1024 break on three popular protocols: IKE, SSH, and HTTPS. Our measurements indicate that these protocols, as they are commonly used, would be subject to widespread compromise by a state-level attacker who had the resources to invest in precomputation for a small number of common 1024-bit groups.

IKE We measured how IPsec VPNs use Diffie-Hellman in practice by scanning a 1% random sample of the public IPv4 address space for IKEv1 and IKEv2 (the protocols used to

	If the attacker can precompute for . . .			
	all 512-bit groups	all 768-bit groups	one 1024-bit group	ten 1024-bit groups
HTTPS Top 1M w/ active downgrade	45,100 (8.4%)	45,100 (8.4%)	205,000 (37.1%)	309,000 (56.1%)
HTTPS Top 1M	118 (0.0%)	407 (0.1%)	98,500 (17.9%)	132,000 (24.0%)
HTTPS Trusted w/ active downgrade	489,000 (3.4%)	556,000 (3.9%)	1,840,000 (12.8%)	3,410,000 (23.8%)
HTTPS Trusted	1,000 (0.0%)	46,700 (0.3%)	939,000 (6.56%)	1,430,000 (10.0%)
IKEv1 IPv4	–	64,700 (2.6%)	1,690,000 (66.1%)	1,690,000 (66.1%)
IKEv2 IPv4	–	66,000 (5.8%)	726,000 (63.9%)	726,000 (63.9%)
SSH IPv4	–	–	3,600,000 (25.7%)	3,600,000 (25.7%)

Table 3: **Estimated impact of Diffie-Hellman attacks.** We use Internet-wide scanning to estimate the number of real-world servers for which typical connections could be compromised by attackers with various levels of computational resources. For HTTPS, we provide figures with and without downgrade attacks on the chosen ciphersuite. All others are passive attacks.

initiate an IPsec VPN connection) in May 2015. We used the ZMap UDP probe module to measure support for Oakley Groups 1 and 2 (the two popular 1024-bit or smaller, built-in groups), and which group servers prefer. To test support for individual groups, we offered only the single group in question. To detect default behavior, we offered servers a variety of DH groups, with the lowest priority groups being Oakley Groups 1 and 2. When measuring server preference, we scanned with the 3DES symmetric cipher—the most commonly supported symmetric cipher in our single group scans. Because of this, the percentages we present for IKEv1 and IKEv2 are a lower-bound for the number of servers that prefer Oakley Groups 1 and 2.

Of the 80 K hosts that responded with a valid IKE packet, 44.2% were willing to accept an offered proposal from at least one scan. The majority of the remaining hosts responded with a `NO-PROPOSAL-CHOSEN` message regardless of our proposal. Many of these may be site-to-site VPNs that reject our source address. We consider these hosts "unprofiled" and omit them from the results here.

We found that 31.8% of IKEv1 and 19.7% of IKEv2 servers support Oakley Group 1 (768-bit) while 86.1% and 91.0% respectively supported Oakley Group 2 (1024-bit). In our sample of IKEv1 servers, 2.6% of profiled servers preferred the 768-bit Oakley Group 1—which is within cryptanalytic reach today for moderately resourced attackers—and 66.1% preferred the 1024-bit Oakley Group 2. For IKEv2, 5.8% of profiled servers chose Oakley Group 1, and 63.9% chose Oakley Group 2. This coincides with our anecdotal findings that most VPN clients only offer Oakley Group 2 by default.

SSH All SSH handshakes complete either a finite field Diffie-Hellman or elliptic curve Diffie-Hellman exchange as part of the SSH key exchange. The SSH protocol explicitly defines support for Oakley Group 2 (1024-bit) and Oakley Group 14 (2048-bit), but also allows a server-defined group, which can be negotiated through an auxiliary Diffie-Hellman Group Exchange (DH GEX) handshake [14].

In order to measure how SSH uses DH in practice, we implemented the SSH protocol in the ZMap toolchain and scanned 1% random samples of the public IPv4 address space in April 2015. We find that 98.9% of SSH servers support the 1024-bit Oakley Group 2, 77.6% support the 2048-bit Oakley Group 14, and 68.7% support DH-GEX.

During the SSH handshake, the client and server select the client's highest priority mutually supported key exchange algorithm. Therefore, we cannot directly measure what algorithm servers will prefer in practice. In order to estimate this, we performed a scan in which we mimicked the algorithms offered by OpenSSH 6.6.1p1, the latest version of OpenSSH. In this scan, 21.8% of servers preferred the 1024-bit Oakley Group 2, and 37.4% preferred a server-defined group. 10% of the server-defined groups were 1024-bit, but, of those, near all provided Oakley Group 2 rather than a custom group.

Combining these equivalent choices, we find that a state-level attacker who performed NFS precomputations for the 1024-bit Oakley Group 2 (which has been in standards for almost two decades) could passively eavesdrop on connections to 3.6 M (25.7%) publicly accessible SSH servers.

HTTPS DHE is commonly deployed on web servers. 68.3% of Alexa Top 1M sites support DHE, as do 23.9% of sites with browser-trusted certificates. Of the Top 1M sites that support DHE, 84% use a 1024-bit or smaller group, with 94% of these using one of five groups.

Despite widespread support for DHE, a passive eavesdropper can only decrypt connections that organically agree to use Diffie-Hellman. We can estimate the number of sites for which this will occur by offering the same sets of ciphersuites as Chrome, Firefox, and Safari. While these the offered ciphers differ slightly between browsers, this turns out to result in negligible differences in whether DHE is chosen.

Approximately 24.7% of browser connections with HTTPS-enabled Top 1M sites (and 10% with browser-trusted sites) will negotiate DHE with one of the ten most popular 1024-bit primes; 17.9% of connections with Top 1M sites could be passively eavesdropped given the discrete log of a single 1024-bit prime. The most popular site that negotiates a DHE ciphersuite using one of the two most common 1024-bit primes is sohu.com (ranked 31st globally).

Mail TLS is also used to secure email transport. SMTP, the protocol used to relay messages between mail servers, allows a connection to be upgraded to TLS by issuing the `STARTTLS` command. POP3S and IMAPS, used by end users to fetch received mail, wrap the entire connection in TLS.

We studied 1% samples of the public IPv4 address space for IMAPS, POP3, and SMTP+StartTLS. We found that 50.7% of SMTP servers supported `STARTTLS`, 41.4% support DHE, and 14.8% supported DHE_EXPORT ciphers. 15.5% of SMTP servers used one of ten most common 1024-bit groups.

For IMAPS, 8.4% of servers supported DHE_EXPORT and 75% supported DHE. However, the ten most common 1024-bit primes account for only 5.4% of servers. POP3S deployment is similar, with 8.9% of servers supporting DHE_EXPORT and 74.9% supporting DHE, but with the ten most common 1024-bit primes accounting for only 4.8% of servers.

If each of the top ten 1024-bit primes used by each protocol were broken, this would affect approximately 1.7 M SMTP servers, 276 K IMAPS servers, and 245 K POP3S servers. Using our downgrade attack of §3.3, an attacker with modest resources can hijack connections to approximately 1.6 M SMTP servers, 429 K IMAPS servers, and 454 K POP3S.

5. RECOMMENDATIONS

Our findings indicate that one of the key recommendations from security experts in response to the threat of mass surveillance — promotion of DHE-based ciphersuites offering "perfect forward secrecy" for TLS over RSA-based ciphersuites — may have actually reduced security for many hosts. In this section, we present concrete recommendations to recover the expected security of Diffie-Hellman as it is used in mainstream Internet protocols.

Increase minimum key strengths As a short-term mitigation, server operators should disable DHE_EXPORT and configure DHE ciphersuites to use freshly-generated groups of at least 1024 bits or, preferably, 2048 bits or larger. Browsers and clients should raise the minimum accepted size for Diffie-Hellman groups to at least 1024 bits, to avoid downgrade attacks when communicating with servers that still support smaller groups.

Our analysis suggests that 1024-bit discrete log may be within reach of state-level actors. As such, 1024-bit DHE (and 1024-bit RSA) must be phased out in the near term. We recommend clients to raise the minimum DHE group size to 2048 bits as soon as server configurations allow. Server operators should move to 2048-bit or larger groups to facilitate this transition.

Avoid fixed-prime groups In the medium term, employing negotiated Diffie-Hellman groups can help mitigate some of the damage caused by NFS-style precomputation for very common fixed groups. A current IETF draft [17] proposes a negotiated group extension to TLS. However, we note that it is possible to create trapdoored primes [40] that are computationally difficult to detect. At the very least, primes should be checked to be safe primes, or groups should use a verifiable generation process such as the one proposed in FIPS 186 [34], and the process for generating primes within the TLS session should be fixed so as to thwart the risk of trapdoors.

Transition to elliptic curves In the long term, transitioning to elliptic curve Diffie-Hellman (ECDH) key exchange avoids all known feasible cryptanalytic attacks. Current elliptic curve discrete log algorithms for strong curves do not gain as strong an advantage from precomputation. Unfortunately, the most widely supported ECDH parameters, those specified by NIST, are now viewed with suspicion due to NSA influence on their design, despite no known or suspected weaknesses. These curves are undergoing scrutiny and new curves, such as Curve25519, are being standardized by the IRTF for use in Internet Protocols. We recommend transitioning to elliptic curves as a long-term solution. This is in line with the recommendation in Huang et al. [20].

Don't deliberately weaken crypto Our downgrade attack on export-grade 512-bit Diffie-Hellman groups in TLS illustrates the fragility of cryptographic "front doors". Although the key sizes originally used in DHE_EXPORT were intended to be tractable only to the NSA, two decades of algorithmic and computational improvements have significantly lowered the bar to attacks on such key sizes. Despite a policy change and attempts to remove support for DHE_EXPORT, the technical debt induced by the additional complexity has left implementations vulnerable for decades. In combination with FREAK [6], our attacks warn of the long-term debilitating effects of deliberately weakening cryptography.

Improve communication The NFS algorithm for discrete logarithms allows an attacker to perform a single precomputation, after which computing individual logs in that group has a much lower marginal cost. Although the cheaper cost of individual discrete logs was known to cryptographers, it appears to not have been as widely understood by implementers. Indeed, many implementations believed RSA key exchange to be inferior to Diffie-Hellman, which offered forward secrecy. Ironically, the opposite appears to be true: for a medium-value target, a fresh, well-generated 1024-bit RSA key would be significantly more expensive to factor than a 1024-bit discrete log in a group for which precomputation has already been done.

A key lesson from this state of affairs is that cryptographers and creators of practical systems need to communicate better. Systems builders should be aware of the difficulty of cryptographic attacks and tradeoffs, and cryptographers should be aware of how systems are actually being implemented and used in practice.

6. DISCLOSURE AND RESPONSE

We notified both client and server software developers of the vulnerabilities discussed in this work. As a result of our disclosure, Microsoft Internet Explorer [33], Mozilla Firefox, and Google Chrome have increased the minimum size of the groups they accept for DHE to 1024 bits, and OpenSSL and Apple Safari are expected to follow suit. On the server side, we notified Apache, Oracle, IBM, Cisco, and various hosting providers. Akamai has removed all support for export ciphersuites. In the medium-term, many TLS developers plan to support a new extension that allows clients and servers to negotiate a few well-known groups of size 2048-bits and higher, and to gracefully reject weak ones [17]. We will be able to report on the full vendor response in the final version of this paper.

7. CONCLUSION

The Diffie-Hellman key exchange is a cornerstone of many cryptographic protocols. Despite its relative simplicity and elegance, practical complications and technical debt over decades have left modern implementations vulnerable to attack from even low-resource adversaries. Additionally, due to a breakdown in communication between cryptographers and system implementers, there is evidence that suggests the way we are using Diffie-Hellman in today's protocols is insufficient to protect against state-level actors. As we move to using newer key exchanges, it is important to ensure that our implementations and protocols remain adaptable and can be easily updated to the relevant dynamic changes in the underlying cryptographic requirements.

Acknowledgments

The authors wish to thank Michael Bailey, Daniel Bernstein, Ron Dreslinski, Tanja Lange, Adam Langley, Andrei Popov, Edward Snowden, Brian Smith, Martin Thomson, and Eric Rescorla.

This material is based in part upon work supported by the U.S. National Science Foundation under contracts CNS-1345254, CNS-1410031, and EFRI-1441209, by the Office of Naval Research under contract N00014-11-1-0470, by the ERC Starting Grant 259639 (CRYSP), by the French ANR research grant ANR-12-BS02-001-01, by the NSF Graduate Research Fellowship Program under grant DGE-1256260, by the Mozilla Foundation, by the Google Ph.D. Fellowship in Computer Security, and by an Alfred P. Sloan Foundation Research Fellowship. Some experiments were conducted using the Grid'5000 testbed, which is supported by INRIA, CNRS, RENATER, and several other universities and organizations; additional experiments used UCS hardware donated by Cisco. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these sponsors.

8. REFERENCES

[1] S. Bai, C. Bouvier, A. Filbois, P. Gaudry, L. Imbert, A. Kruppa, F. Morain, E. Thomé, and P. Zimmermann. `cado-nfs`, an implementation of the number field sieve algorithm, 2014. Release 2.1.1.

[2] R. Barbulescu. *Algorithmes de logarithmes discrets dans les corps finis*. PhD thesis, Université de Lorraine, France, 2013.

[3] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In *Eurocrypt*, 2014.

[4] D. J. Bernstein. How to find smooth parts of integers, 2004. http://cr.yp.to/factorization/smoothparts-20040510.pdf.

[5] D. J. Bernstein and T. Lange. Batch NFS. In *Selected Areas in Cryptography (SAC)*, 2014.

[6] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *IEEE Symposium on Security and Privacy*, 2015.

[7] C. Bouvier, P. Gaudry, L. Imbert, H. Jeljeli, and E. Thomé. New record for discrete logarithm in a prime finite field of 180 decimal digits, 2014. http://caramel.loria.fr/p180.txt.

[8] R. Canetti and H. Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In *Crypto*, 2002.

[9] D. Coppersmith. Solving linear equations over GF(2) via block Wiedemann algorithm. *Math. Comp.*, 62(205), 1994.

[10] R. Crandall and C. B. Pomerance. *Prime numbers: a computational perspective*. Springer, 2001.

[11] B. den Boer. Diffie-Hellman is as strong as discrete log for certain primes. In *Crypto*, 1988.

[12] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, 22(6):644–654, 1976.

[13] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *Usenix Security*, 2013.

[14] M. Friedl, N. Provos, and W. Simpson. Diffie-Hellman group exchange for the secure shell (SSH) transport layer protocol. RFC 4419, Mar. 2006.

[15] W. Geiselmann, H. Kopfer, R. Steinwandt, and E. Tromer. Improved routing-based linear algebra for the number field sieve. In *Information Technology: Coding and Computing*, 2005.

[16] W. Geiselmann and R. Steinwandt. Non-wafer-scale sieving hardware for the NFS: Another attempt to cope with 1024-bit. In *Eurocrypt*, 2007.

[17] D. Gillmor. Negotiated finite field Diffie-Hellman ephemeral parameters for TLS. IETF Internet Draft, May 2015.

[18] D. M. Gordon. Discrete logarithms in GF(p) using the number field sieve. *SIAM J. Discrete Math.*, 6(1), 1993.

[19] D. Harkins and D. Carrel. The Internet key exchange (IKE). RFC 2409, Nov. 1998.

[20] L.-S. Huang, S. Adhikarla, D. Boneh, and C. Jackson. An experimental study of TLS forward secrecy deployments. *Internet Computing, IEEE*, 18(6):43–51, Nov 2014.

[21] A. Joux and R. Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method. *Math. Comp.*, 72(242):953–967, 2003.

[22] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. Internet key exchange protocol version 2 (IKEv2). RFC 7296, Oct. 2014.

[23] S. Kent. IP authentication header. RFC 4302, Dec. 2005.

[24] S. Kent. IP encapsulating security payload (ESP). RFC 4303, Dec. 2005.

[25] T. Kleinjung. Cofactorisation strategies for the number field sieve and an estimate for the sieving step for factoring 1024 bit integers, 2006. http://www.hyperelliptic.org/tanja/SHARCS/talks06/thorsten.pdf.

[26] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit RSA modulus. In *Crypto*, 2010.

[27] A. Langley, N. Modadugu, and B. Moeller. Transport layer security (TLS) false start. IETF Internet Draft, 2010.

[28] A. K. Lenstra and H. W. Lenstra, Jr., editors. *The Development of the Number Field Sieve*. Springer, 1993.

[29] M. Lipacis. Semiconductors: Moore stress = structural industry shift. Technical report, Jefferies, Sept. 2012.

[30] U. M. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In *Crypto*, 1994.

[31] U. M. Maurer and S. Wolf. Diffie-Hellman oracles. In *Crypto*, 1996.

[32] C. Meadows. Analysis of the Internet key exchange protocol using the NRL protocol analyzer. In *IEEE Symposium on Security and Privacy*, 1999.

[33] Microsoft Security Bulletin MS15-055. Vulnerability in Schannel could allow information disclosure, May 2015. https://technet.microsoft.com/en-us/library/security/ms15-055.aspx.

[34] NIST. FIPS PUB 186-4: Digital signature standard, 2013.

[35] Oak Ridge National Laboratory. Introducing Titan, 2012. https://www.olcf.ornl.gov/titan.

[36] H. Orman. The Oakley key determination protocol. RFC 2412, 1998.

[37] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (corresp.). *Trans. Inform. Theory*, 24(1), 1978.

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

[39] O. Schirokauer. Virtual logarithms. *J. Algorithms*, 57(2):140–147, 2005.

[40] I. A. Semaev. Special prime numbers and discrete logs in finite prime fields. *Math. Comp.*, 71(237):363–377, Jan. 2002.

[41] D. Shanks. Class number, a theory of factorization, and genera. In *Proc. Sympos. Pure Math.*, volume 20. 1971.

[42] Spiegel Staff. Prying eyes: Inside the NSA's war on Internet security. Der Spiegel, Dec 2014. http://www.spiegel.de/international/germany/inside-the-nsa-s-war-on-internet-security-a-1010361.html.

[43] stud: The scalable TLS unwrapping daemon, 2012. https://github.com/bumptech/stud/blob/19a7f19686bcdbd689c6fbea31f68a276e62d886/stud.c#L593.

[44] E. Thomé. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *J. Symbolic Comput.*, 33(5):757–775, July 2002.

[45] P. C. Van Oorschot and M. J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *CCS*, 1994.

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

[47] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *2nd Usenix Workshop on Electronic Commerce*, 1996.

[48] J. Wagnon. SSL profiles part 5: SSL options, June 2013. https://devcentral.f5.com/articles/ssl-profiles-part-5-ssl-options.

[49] P. Zimmermann et al. GMP-ECM, 2012. https://gforge.inria.fr/projects/ecm.

[50] APEX active/passive exfiltration. Media leak, Aug. 2009. http://www.spiegel.de/media/media-35671.pdf.

[51] Fielded capability: End-to-end VPN SPIN 9 design review. Media leak. http://www.spiegel.de/media/media-35529.pdf.

[52] FY 2013 congressional budget justification. Media leak. http://cryptome.org/2013/08/spy-budget-fy13.pdf.

[53] GALLANTWAVE@scale. Media leak. http://www.spiegel.de/media/media-35514.pdf.

[54] Innov8 experiment profile. Media leak. http://www.spiegel.de/media/media-35509.pdf.

[55] Intro to the VPN exploitation process. Media leak, Sept. 2010. http://www.spiegel.de/media/media-35515.pdf.

[56] LONGHAUL – WikiInfo. Media leak. http://www.spiegel.de/media/media-35533.pdf.

[57] POISONNUT – WikiInfo. Media leak. http://www.spiegel.de/media/media-35519.pdf.

[58] SIGINT strategy. Media leak. http://www.nytimes.com/interactive/2013/11/23/us/politics/23nsa-sigint-strategy-document.html.

[59] SPIN 15 VPN story. Media leak. http://www.spiegel.de/media/media-35522.pdf.

[60] TURMOIL/APEX/APEX high level description document. Media leak. http://www.spiegel.de/media/media-35513.pdf.

[61] TURMOIL IPsec VPN sessionization. Media leak, Aug. 2009. http://www.spiegel.de/media/media-35528.pdf.

[62] TURMOIL VPN processing. Media leak, Oct. 2009. http://www.spiegel.de/media/media-35526.pdf.

[63] VALIANTSURF (VS): Capability levels. Media leak. http://www.spiegel.de/media/media-35517.pdf.

[64] VALIANTSURF – WikiInfo. Media leak. http://www.spiegel.de/media/media-35527.pdf.

[65] VPN SigDev basics. Media leak. http://www.spiegel.de/media/media-35520.pdf.

[66] What your mother never told you about SIGDEV analysis. Media leak. http://www.spiegel.de/media/media-35551.pdf.