

Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice

By 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, and Paul Zimmermann

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 Logjam, a novel flaw in TLS that lets a man-in-the-middle downgrade connections to “export-grade” Diffie-Hellman. To carry out this attack, we implement the number field sieve discrete logarithm algorithm. After a week-long precomputation for a specified 512-bit group, we can compute arbitrary discrete logarithms in that group in about a minute. We find that 82% of vulnerable servers use a single 512-bit group, and that 8.4% of Alexa Top Million HTTPS sites are vulnerable to the attack.^a In response, major browsers have changed to reject short groups.

We go on to consider Diffie-Hellman with 768- and 1024-bit groups. We estimate that even in the 1024-bit case, the computations are plausible given nation-state resources. A small number of fixed or standardized groups are used by millions of servers; performing precomputation for a single 1024-bit group would allow passive eavesdropping on 18% of popular HTTPS sites, and a second group would allow decryption of traffic to 66% of IPsec VPNs and 26% of SSH servers. 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 (DH) key exchange is a popular cryptographic algorithm that allows Internet protocols to agree on a shared key and negotiate a secure connection. It is fundamental to protocols such as Hypertext Transport Protocol Secure (HTTPS), Secure Shell (SSH), Internet Protocol Security (IPsec), Simple Mail Transfer Protocol Secure (SMTPS), and other protocols that rely on Transport Layer Security (TLS). Many protocols use Diffie-Hellman to achieve *perfect forward secrecy*, the property that a compromise of the long-term keys used for authentication does not compromise session keys for past connections. We examine how Diffie-Hellman is commonly implemented and deployed with common 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

^a Except where otherwise noted, the experimental data and network measurements for this article were obtained in early 2015.

support for obsolete 1990s-era “export-grade” cryptography. 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.

The current best technique for attacking Diffie-Hellman relies on compromising one of the private exponents (a , b) by computing the discrete logarithm 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 logarithm is more difficult than factoring a Rivest–Shamir–Adleman (RSA) modulus of the same size. However, an adversary who performs a large precomputation for a prime p can then quickly calculate arbitrary discrete logarithms in that group, amortizing the cost over all targets that share this parameter. 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 introduce Logjam, a new attack on TLS by which a man-in-the-middle attacker can downgrade a connection to export-grade cryptography. This attack is reminiscent of the FREAK attack¹ 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 logarithm algorithm and carried out precomputation for two 512-bit Diffie-Hellman groups used by more than 92% of the vulnerable servers. This allows us to compute individual discrete logarithms in about a minute. Using our discrete logarithm oracle, we can compromise connections to over 7% of Alexa Top Million HTTPS sites. Discrete logarithms over larger groups have been computed before,² but, as far as we are aware, this is the first time they have been exploited to expose concrete vulnerabilities in real-world systems.

Risks from common 1024-bit groups

We explore the implications of precomputation attacks for 768- and 1024-bit groups, which are widely used in practice

The full version of this paper was published in *Proceedings of the 22nd Conference on Computer and Communications Security (CCS), October 2015, ACM*. The full paper and additional materials are available at <https://weakdh.org/>.

and still considered secure. We estimate 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 nation-state adversaries. In both cases, individual logarithms can be quickly computed after the initial precomputation.

We then examine evidence from published Snowden documents that suggests that the National Security Agency (NSA) may already be exploiting 1024-bit Diffie-Hellman to decrypt Virtual Private Network (VPN) traffic. We perform measurements to understand the implications of such an attack for popular protocols, finding that an attacker who could perform precomputations for ten 1024-bit groups could passively decrypt traffic to about 66% of Internet Key Exchange (IKE) VPNs, 26% of SSH servers, and 24% of popular HTTPS sites.

Mitigations and lessons

In response to the Logjam attack, mainstream browsers have implemented a more restrictive policy on the size of Diffie-Hellman groups they accept, and Google Chrome has discontinued support for finite field key exchanges. We further 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 elliptic curve Diffie-Hellman.

2. DIFFIE-HELLMAN CRYPTANALYSIS

Diffie-Hellman key exchange was the first published public-key algorithm.⁵ 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 . Then each generates a random private exponent, a and b . Alice sends $g^a \bmod p$, Bob sends $g^b \bmod p$, and each computes a shared secret $g^{ab} \bmod p$. While there is also a Diffie-Hellman exchange over elliptic curve groups, we address only the “mod p ” case.

The security of Diffie-Hellman is not known to be equivalent to the discrete logarithm problem, but computing discrete logarithms remains the best known cryptanalytic attack. An attacker who can find the discrete logarithm x from $y = g^x \bmod p$ can easily find the shared secret.

Textbook descriptions of discrete logarithm algorithms can be misleading about the computational tradeoffs, for example by optimizing for computing a *single* discrete

logarithm. In fact, as illustrated in Figure 1, a single large precomputation on p can be used to efficiently break *all* Diffie-Hellman exchanges made with that prime.

Diffie-Hellman is typically implemented with prime fields and large group orders. In this case, the most efficient known algorithm for computing discrete logarithms is the Number Field Sieve (NFS).^{9, 11, 18} The algorithm 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. 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.

In the third stage, *linear algebra*, we construct a large, sparse matrix consisting of the coefficient vectors of prime factorizations we have found. This stage can be parallelized in a limited fashion, and produces a database of logarithms which are used as input to the final stage.

The final stage, *descent*, actually deduces the discrete logarithm of the target y . We re-sieve until we find a set of relations that allow us to write the logarithm of y in terms of the logarithms in the precomputed 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 logarithms of many targets.

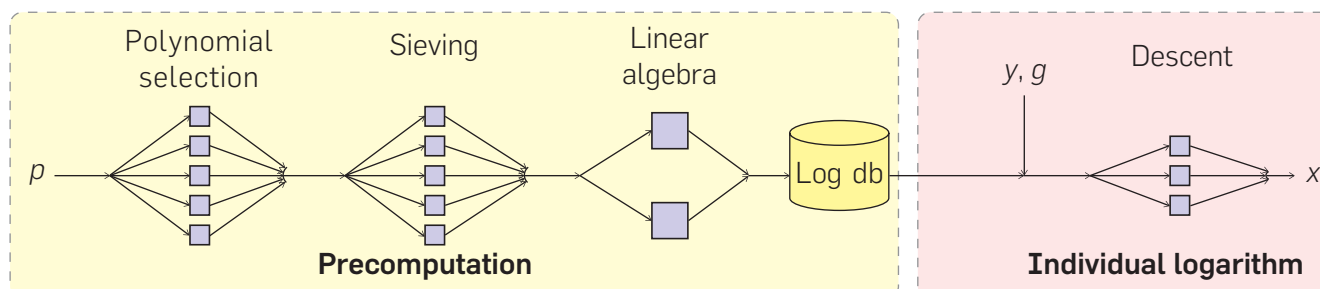
The numerous parameters of the algorithm allow some flexibility to reduce time on some computational steps at the expense of others. For example, sieving more will result in a smaller matrix, making linear algebra cheaper, and doing more work in the precomputation makes the final descent step easier.

Standard primes

Generating safe primes^b can be computationally burdensome, so many implementations use standardized

^b An odd prime p is safe when $(p - 1)/2$ is prime.

Figure 1. Number field sieve for discrete logarithms. This algorithm consists of a precomputation stage that depends only on the prime p and a descent stage that computes individual logarithms. With sufficient precomputation, an attacker can quickly break any Diffie-Hellman instances that use a particular p .



Diffie-Hellman parameters. A prominent example is the Oakley groups,¹⁷ 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 Off-the-Record Messaging (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 logarithm precomputation among vast numbers of potential targets.

3. ATTACKING TLS

TLS supports Diffie-Hellman as one of several possible key exchange methods, and prior to public disclosure of our attack, about two-thirds of popular HTTPS sites supported 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, which we call the Logjam attack. First, we perform NFS precomputations for the two most popular 512-bit primes on the web, so that we can quickly compute the discrete logarithm for any key exchange message that uses one of them. 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 our precomputations can compromise connections to about 8% of HTTPS servers among Alexa Top Million domains.

3.1. TLS and Diffie-Hellman

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

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_*`.^c 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 .

^c New ciphersuites that use elliptic curve Diffie-Hellman (ECDHE) are gaining in popularity, but we focus exclusively on the traditional prime field variety.

To ensure agreement on the negotiation messages, and to prevent downgrade attacks, each party computes the TLS master secret from g^{ab} and calculates a Message Authentication Code (MAC) of its view of the handshake transcript. These MACs are exchanged in a pair of Finished messages and verified by the recipients.

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 servers maintain support for backward compatibility.

To understand how HTTPS servers in the wild use Diffie-Hellman, we modified the ZMap⁶ toolchain to offer DHE and DHE_EXPORT ciphersuites and scanned TCP/443 on both the full public IPv4 address space and the Alexa Top Million domains. The scans took place in March 2015. Of 539,000 HTTPS sites among Top Million domains, we found that 68.3% supported DHE and 8.4% supported DHE_EXPORT. Of 14.3mn IPv4 HTTPS servers with browser-trusted certificates, 23.9% supported DHE and 4.9% DHE_EXPORT.

While the TLS protocol allows servers to generate their own Diffie-Hellman parameters, just two 512-bit primes account for 92.3% of Alexa Top Million domains that support DHE_EXPORT (Table 1), and 92.5% of all servers with browser-trusted certificates that support DHE_EXPORT. The most popular 512-bit prime was hard-coded into many versions of Apache; the second most popular is the `mod_ssl` default for DHE_EXPORT.

3.2. Active downgrade to export-grade DHE

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

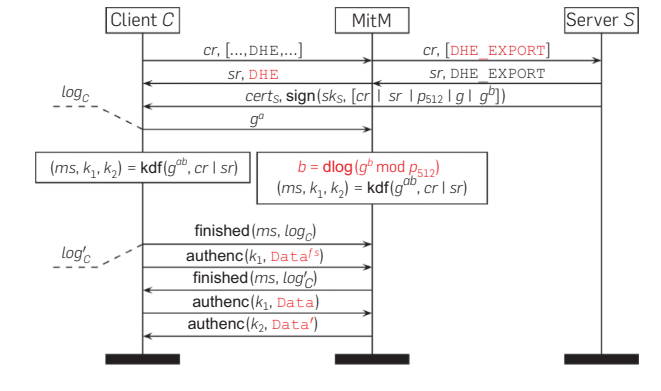
The attack, which we call Logjam, is depicted in Figure 2 and relies on a flaw in the way TLS composes DHE and

Table 1. Top 512-bit Diffie-Hellman primes for TLS^d.

Source	Popularity	Prime
Apache	82%	9fd8b8b8a004544f0045f1737d0ba2e0b274cdf1a9f588218fb435316a16e374171fd19d8d8f37c39bf863fd60e3e300680a3030c6e4c3757d08f70e6aa871033
mod_ssl	10%	d4bcd52406f69b35994b88de5db89682c8157f62d8f33633ee5772f11f05ab22d6b5145b9f241e5acc31ff090a4bc71148976f76795094e71e7903529f5a824b
(others)	8%	(463 distinct primes)

^d 8.4% of Alexa Top Million HTTPS domains allow DHE_EXPORT, of which 92.3% use one of the two most popular primes, shown here.

Figure 2. The Logjam attack. A man-in-the-middle can force TLS clients to use export-strength Diffie-Hellman 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. $Data^f$ refers to False Start application data that some TLS clients send before receiving the server's Finished message.



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 rewrite 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 rewrites 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 close to real time can then derive the master secret and connection keys to complete the handshake with the client.

There are two remaining challenges in implementing this active downgrade attack. The first is to compute individual discrete logarithms in close to real time, and the second is to delay handshake completion until the discrete logarithm computation has had time to finish.

3.3. 512-bit discrete logarithm computations

We modified CADO-NFS¹⁹ to implement the number field sieve discrete logarithm algorithm and applied it to the top two DHE_EXPORT primes shown in Table 1. Precomputation took seven days for each prime, after which computing individual logarithms requires a median of 70 seconds.

Precomputation. As illustrated 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 logarithms, 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 microprocessor cores in parallel. Polynomial selection ran for about 3hrs (7,600 core-hours). Sieving ran for 15hrs (21,400 core-hours). This sufficed to collect 40mn relations of which 28mn were unique, involving 15mn primes of at most 27 bits.

From this data set, we obtained a square matrix with 2.2mn rows and columns, with 113 nonzero coefficients per row on average. We solved the corresponding linear system on a 36-node cluster using the block Wiedemann algorithm.^{4,20} Using unoptimized code, the computation finished in 120hrs (60,000 core-hours).

The experiment above was done with CADO-NFS in early 2015. As of 2017, release 2.3 of CADO-NFS¹⁹ performs 20% faster for sieving, and drastically faster for linear algebra, since 9,000 core-hours suffice to solve the same linear system on the same hardware. In total, the wall-clock time for each precomputation was slightly over one week in 2015, and is reduced to about two days with current hardware and more recent software.

Descent. Once this precomputation was finished, we were able to run the final descent step to compute individual discrete logarithms in about a minute. We implemented the descent calculation in a mix of Python and C. On average, computing individual logarithms took about 70sec, but the time varied from 34sec to 206sec on a server with two 18-core Intel Xeon E5-2699 CPUs. For purposes of comparison, a single 512-bit RSA factorization using the CADO-NFS implementation takes about four days of wall-clock time on the computer used for the descent.¹⁹

3.4. Active attack implementation

The main challenge in performing this attack 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 70sec, but there are several ways an attacker can work around this delay:

Non-browser clients. Different TLS clients impose different time limits, after which they kill the connection. Command-line clients such as curl and git have long or no timeouts, and we can hijack their connections without difficulty.

TLS warning alerts. Web browsers tend to have shorter timeouts, but we can keep their connections alive by sending TLS warning alerts, which are ignored by the browser but reset the handshake timer. For example, this allows us to keep Firefox TLS connections alive indefinitely.

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. For example, F5 BIG-IP load balancers will reuse g^b by default. Microsoft Schannel caches g^b for two hours — this setting is hard-coded. For these servers, an attacker can compute the discrete logarithm of g^b from one connection and use it to attack later handshakes.

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 that use TLS False Start. Recent versions of Chrome, Internet Explorer, and Firefox implement False Start, but their policies on when to enable it vary. Firefox 35, Chrome 41, and Internet Explorer (Windows 10) send False Start data with DHE.

In these cases, a man-in-the-middle can record the handshake and decrypt the False Start payload at leisure.

4. NATION-STATE THREATS TO DIFFIE-HELLMAN

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 cryptography. 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 logarithms scales to 768- and 1024-bit groups. As we argue below, 768-bit groups in relatively widespread use are now within reach for academic computational resources. Additionally, performing precomputations for a small number of 1024-bit groups is plausibly within the resources of nation-state adversaries. The precomputation would likely require special-purpose hardware, but would not require any major algorithmic improvements. In light of these results, we examine several standard Internet security protocols — IKE, SSH, and TLS — to determine their vulnerability. Although the cost of the precomputation for a 1024-bit group is several times higher than for an RSA key of equal size, a one-time investment could be used to attack millions of hosts, due to widespread reuse of the most common Diffie-Hellman parameters. Finally, we apply this new understanding to a set of recently published documents to evaluate the hypothesis that the National Security Agency has *already* implemented such a capability.

4.1. Scaling NFS to 768- and 1024-bit Diffie-Hellman

Estimating the cost for discrete logarithm cryptanalysis at larger key sizes is far from straightforward due to the complexity of parameter tuning. We attempt estimates up to 1024-bit discrete logarithm based on the existing literature

and our own experiments but further work is needed for greater confidence. We summarize all the costs, measured or estimated in Table 2.

DH-768: done in 2016. When the ACM CCS version of this article was prepared, the latest discrete logarithm record was a 596-bit computation. Based on that work, and on prior experience with the 768-bit factorization record in 2009,¹² we made the conservative prediction that it was possible, as explained in Section 2, to put more computational effort into sieving for the discrete logarithm case than for factoring, so that the linear algebra step would run on a slightly smaller matrix. This led to a runtime estimate of around 37,000 core-years, most of which was spent on linear algebra.

This estimate turned out to be overly conservative, for several reasons. First, there have been significant improvements in our software implementation (Section 3.3). In addition, our estimate did not use the Joux-Lercier alternative polynomial selection method,¹¹ which is specific to discrete logarithms. For 768-bit discrete logarithms, this polynomial selection method leads to a significantly smaller computational cost.

In 2016, Kleinjung et al. completed a 768-bit discrete logarithm computation.¹³ While this is a massive computation on the academic scale, a computation of this size has likely been within reach of nation-states for more than a decade. This data is mentioned in Table 2.

DH-1024: Plausible with nation-state resources. Experimentally extrapolating sieving parameters to the 1024-bit case is difficult due to the trade-offs between the steps of the algorithm and their relative parallelism. The prior work proposing parameters for factoring a 1024-bit RSA key is thin, and we resort to extrapolating from asymptotic complexity. For the number field sieve, the complexity is $\exp((k + o(1))(\log N)^{1/3}(\log \log N)^{2/3})$, where N is the integer to factor or the prime modulus for discrete logarithm 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 logarithm and factorization, which are both dominated by sieving and linear algebra in the precomputation. 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.

Table 2. Estimating costs for factoring and discrete log^a.

	Sieving		Linear Algebra		Descent	
	Log ₂ B	Core-years	Rows	Core-years	Core-time	
RSA-512	29	0.3	4.2mn	0.03		Timings with default CADO-NFS parameters.
DH-512	27	2.5	2.2mn	1.1	10min	For the computations in this paper; may be suboptimal.
RSA-768	37	800	250mn	100		Est. based on Kleinjung and Aoki et al. ¹² with less sieving.
DH-768	36	4,000	24mn	920	43hrs	Data from, Kleinjung and Diem et al. ¹³ , Table 1.
RSA-1024	42	≈1,000,000	≈8.7bn	≈120,000		Crude estimate based on complexity formula.
DH-1024	40	≈5,000,000	≈0.8bn	≈1,100,000	30 days	Crude estimate based on formula and our experiments.

^a For sieving, we give one important parameter, which is the number of bits of the smoothness bound B . For linear algebra, all costs for DH are for safe primes; for Digital Signature Algorithm (DSA) primes with group order of 160 bits, this should be divided by 6.4 for 1024 bits, 4.8 for 768 bits, and 3.2 for 512 bits.

For 1024-bit precomputation, the total time complexity can be expected to increase by a factor of 1220 using the complexity formula, while space complexity increases by its square root, approximately 35. These ratios are relevant for both factorization and discrete logarithm since they have the same asymptotic behavior. For DH-1024, we get a total cost estimate for the precomputation of about 6mn core-years. In practice, it is not uncommon for estimates based merely on the complexity formula to be off by a factor of 10. Estimates of Table 2 must therefore be considered with due caution.

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. For a random target in Oakley Group 2, initialization took 22 core-days, and yielded 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 smoothness 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 several million core-years is a massive computational effort, it is not necessarily out of reach for a nation-state. At this scale, significant cost savings could be realized by developing application-specific hardware given that sieving is a natural target for hardware implementation. To our knowledge, the best prior description of an Application-Specific Integrated Circuit (ASIC) implementation of 1024-bit sieving is the 2007 work of Geiselmann and Steinwandt.⁸ Updating their estimates for modern techniques and adjusting parameters for discrete logarithm allows us to extrapolate the financial and time costs.

We increase their chip count by a factor of ten to sieve more and save on linear algebra as above, giving an estimate of 3mn chips to complete sieving in one year. Shrinking the dies from the 130 nanometer 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, it would cost about \$2 per chip to manufacture after fixed design and tape-out costs of roughly \$2mn.¹⁴ This suggests that an \$8mn investment would buy enough ASICs to complete the DH-1024 sieving precomputation in one year. Since a step of descent uses sieving, the same hardware could likely be reused to speed calculations of individual logarithms.

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 logarithm. To derive a rough estimate, we can begin with general purpose hardware and the core-year estimate from Table 2. Using the 300,000 CPU core Titan supercomputer it would take four years to complete the 1024-bit linear algebra stage (notwithstanding the fact that estimates from Table 2 are known to be extremely coarse, and could be optimistic by a factor of maybe 10). Titan was constructed in 2012 for \$94mn, suggesting a cost of under \$400mn 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 80.⁷ If we optimistically assume that a similar reduction can be achieved for discrete logarithm, the hardware cost to perform the linear algebra for DH-1024 in one year is plausibly on the order of \$5mn.

Combining these estimates, special-purpose hardware that can perform the precomputation for one 1024-bit group per year would cost roughly \$13mn. This is much less than the “hundreds of millions of dollars” that we conservatively estimated in 2015, making it even more likely that nation-state adversaries have implemented the attack.

To put this dollar figure in context, the FY 2012 budget for the U.S. Consolidated Cryptologic Program (which includes NSA) was \$10.5bn.²² The 2013 budget request, which prioritized investment in “groundbreaking cryptanalytic capabilities to defeat adversarial cryptography and exploit internet traffic” included notable \$100mn+ increases in two programs under Cryptanalysis & Exploitation Services: “Cryptanalytic IT Systems” (to \$247mn), and the cryptically named “PEO Program C” (to \$360mn).²²

4.2. Is NSA breaking 1024-bit Diffie-Hellman?

Our calculations suggest that it is plausibly within NSA’s resources to have performed number field sieve precomputations for 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?

Virtual private networks are widely used for tunneling business or personal traffic across potentially hostile networks. We focus on the IPsec VPN protocol using the IKE protocol for key establishment and parameter negotiation and the Encapsulating Security Payload (ESP) protocol for protecting packet contents.

IKE. There are two versions, IKEv1 and IKEv2, which differ in message structure but are conceptually similar. For the sake of brevity, we will use IKEv1 terminology.¹⁰

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. The shared secret is combined with other cleartext values transmitted by each side, such as nonces and cookies, to derive a value called SKEYID. In IKEv1, SKEYID also incorporates a Pre-Shared Key (PSK) used for authentication.

The resulting SKEYID is used to encrypt and authenticate a Phase 2 handshake. Phase 2 establishes the parameters and key material, KEYMAT, for protecting the subsequently tunneled traffic. Ultimately, KEYMAT is derived from SKEYID, additional nonces, and the result of an optional Phase 2 Diffie-Hellman exchange.

NSA’s VPN exploitation process. Documents published by Der Spiegel describe NSA’s ability to decrypt VPN traffic using passive eavesdropping and without message injection or man-in-the-middle attacks on IPsec or IKE. Figure 3 illustrates the flow of information required to decrypt the tunneled traffic.

When the IKE/ESP messages of a VPN of interest are collected, the IKE messages and a small amount of ESP

traffic are sent to the Cryptanalysis and Exploitation Services (CES).^{21, 23, 25} Within the CES enclave, a specialized “attack orchestrator” attempts to recover the ESP decryption key with assistance from high-performance computing resources as well as a database of known PSKs (“CORALREEF”).^{21, 23, 25} If the recovery was successful, the decryption key is returned from CES and used to decrypt the buffered ESP traffic such that the encapsulated content can be processed.^{21, 24}

Evidence for a discrete logarithm attack. The ability to decrypt VPN traffic does not necessarily indicate a defeat of Diffie-Hellman. There are, however, several features of the described exploitation process that support this hypothesis.

The IKE protocol has been extensively analyzed^{3,15} and is not believed to be exploitable in standard configurations under passive eavesdropping attacks. Absent a vulnerability in the key derivation function or transport encryption, the attacker must recover the decryption keys. This requires the attacker to calculate SKEYID generated from the Phase 1 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 (Section 4.3) show that the vast majority of IKE endpoints select one particular 1024-bit Diffie-Hellman group even when offered stronger groups. Conducting an expensive, but feasible, precomputation for this single 1024-bit group (Oakley Group 2) would allow the

efficient recovery of a large number of Diffie-Hellman shared secrets used to derive SKEYID and the subsequent KEYMAT.

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, and (2) any PSK used for deriving SKEYID in IKEv1. The available documents describe both of these as explicit prerequisites for the VPN exploitation process outlined above and provide the reader with internal resources available to meet these prerequisites.²³

Of course, this explanation is not dispositive and the possibility remains that NSA could defeat VPN encryption using alternative means. A published NSA document refers to the use of a router “implant” to allow decryption of IPsec traffic, indicating the use of targeted malware is possible. However, this implant “allows passive exploitation with just ESP”²³ without the prerequisite of collecting the IKE handshake messages. This indicates it is an alternative mechanism to the attack described above.

The most compelling argument for a pure cryptographic attack is the generality of NSA’s VPN exploitation process. This process appears to be applicable across a broad swath of VPNs without regard to endpoint’s identity or the ability to compromise individual endpoints.

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 IKE, SSH, and HTTPS. Our measurements, performed in early 2015, indicate that these protocols would be subject to widespread compromise by a nation-state attacker who had the resources to invest in precomputation for a small number of 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 initiate an IPsec VPN connection) in May 2015. We used the ZMap UDP probe module to measure support for Oakley Groups 1 and 2 (two popular 768- and 1024-bit, built-in groups) and which group servers prefer. Of the 80K hosts that responded with a valid IKE packet, 44.2% were willing to negotiate a connection using one of the two groups. We found that 31.8% of IKEv1 and 19.7% of IKEv2 servers supported 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

Figure 3. NSA’s VPN decryption infrastructure. This classified illustration published by Der Spiegel²⁵ 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.

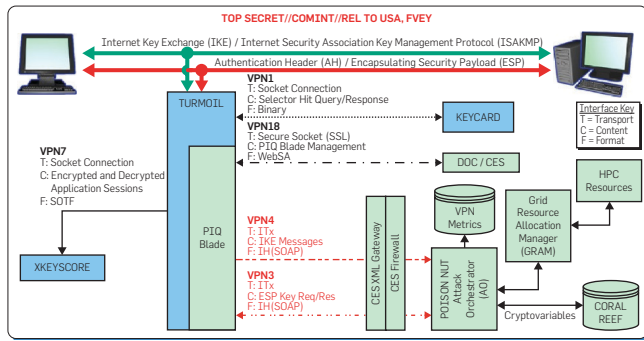


Table 3. Estimated impact of Diffie-Hellman attacks in early 2015^a.

	Vulnerable servers, if the attacker can precompute for...			
	All 512-bit groups	All 768-bit groups	One 1024-bit group	Ten 1024-bit groups
HTTPS Top Million w/ active downgrade	45,100 (8.4%)	45,100 (8.4%)	205,000 (37.1%)	309,000 (56.1%)
HTTPS Top Million	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%)

^a We used 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.

the 768-bit Oakley Group 1 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.

SSH. All SSH handshakes complete either a finite field or elliptic curve Diffie-Hellman exchange. The protocol explicitly defines support for Oakley Group 2 (1024-bit) and Oakley Group 14 (2048-bit) but also allows a server-defined group to be negotiated. We scanned 1% random samples of the public IPv4 address space in April 2015. We found that 98.9% of SSH servers supported the 1024-bit Oakley Group 2, 77.6% supported the 2048-bit Oakley Group 14, and 68.7% supported a server-defined group.

During the SSH handshake, the server selects the client's highest priority mutually supported key exchange algorithm. To estimate what servers will prefer in practice, 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, nearly all provided Oakley Group 2 rather than a custom group.

Combining these equivalent choices, we find that a nation-state adversary who performed NFS precomputations for the 1024-bit Oakley Group 2 could passively eavesdrop on connections to 3.6mn (25.7%) publicly accessible SSH servers.

HTTPS. Our 2015 scans found that DHE was commonly deployed on web servers. 68.3% of Alexa Top Million sites supported DHE, as did 23.9% of sites with browser-trusted certificates. Of the Top Million sites that supported DHE, 84% used 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 estimated the number of sites for which this would occur by offering the same sets of ciphersuites as Chrome, Firefox, and Safari. We found that browser connections to approximately 24% of browser connections with HTTPS-enabled Top Million sites (and 10% of all sites with browser-trusted sites certificates) would negotiate DHE using one of the ten most popular 1024-bit primes. After completing the NFS precomputation for only the most popular 1024-bit prime, an adversary could passive eavesdrop on browser connections to 17.9% of Top Million sites.

5. RECOMMENDATIONS

In this section, we present concrete recommendations to recover the expected security of Diffie-Hellman.

Transition to elliptic curves

Transitioning to Elliptic Curve Diffie-Hellman (ECDH) key exchange avoids all known feasible cryptanalytic attacks. Current elliptic curve discrete logarithm algorithms do not gain as much of an advantage from precomputation. In addition, ECDH keys are shorter and computations are faster. We recommend transitioning to elliptic curves; this is the most effective solution to the vulnerabilities in this paper. We note that in August 2015, NSA announced that it was planning to transition away from elliptic curve cryptography

for its Suite B cryptographic algorithms and would replace them with algorithms resistant to quantum computers.¹⁶ However, since no fully vetted and standardized quantum-resistant algorithms exist currently, elliptic curves remain the most secure choice for public key operations.

Increase minimum key strengths

To protect against the Logjam attack, server operators should disable `DHE_EXPORT` and configure DHE ciphersuites to use primes of 2048 bits or larger. Browsers and clients should raise the minimum accepted size for Diffie-Hellman groups to at least 1024 bits in order to avoid downgrade attacks.

Don't deliberately weaken cryptography


The Logjam attack illustrates the fragility of cryptographic "front doors." Although the key sizes originally used in `DHE_EXPORT` were intended to be tractable only to NSA, two decades of algorithmic and computational improvements have significantly lowered the bar to attacks on such key sizes. Despite the eventual relaxation of cryptography export restrictions and subsequent attempts to remove support for `DHE_EXPORT`, the technical debt induced by the additional complexity has left implementations vulnerable for decades. Like FREAK,¹ our results warn of the long-term debilitating effects of deliberately weakening cryptography.

6. CONCLUSION

We find that Diffie-Hellman key exchange, as used in practice, is often less secure than widely believed. The problems stem from the fact that the number field sieve for discrete logarithms allows an attacker to perform a single precomputation that depends only on the group, after which computing individual logarithms in that group has a far lower cost. Although this is well known to cryptographers, it apparently has not been widely understood by system builders. Likewise, many cryptographers did not appreciate that a large fraction of Internet communication depends on a few small, widely shared groups.

A key lesson is that cryptographers and creators of practical systems need to work together more effectively. System builders should take responsibility for being aware of applicable cryptanalytic attacks. Cryptographers should involve themselves in how cryptography is actually being applied, such as through engagement with standards efforts and software review. Bridging the perilous gap that separates these communities will be essential for keeping future systems secure.

Acknowledgments

The authors thank Michael Bailey, Daniel Bernstein, Ron Dreslinski, Tanja Lange, Adam Langley, Kenny Paterson, Andrei Popov, Ivan Ristic, Edward Snowden, Brian Smith, Martin Thomson, and Eric Rescorla. This work was supported by the U.S. National Science Foundation, the Office of Naval Research, the European Research Council, and the French National Research Agency, with additional support from the Mozilla Foundation, Supermicro, Google, Cisco, the Morris Wellman Professorship, and the Alfred P. Sloan Foundation. Some experiments used the Grid'5000 testbed, supported by INRIA, CNRS, RENATER, and others. 

References

1. Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.-Y., Zinzindohoue, J.K. A messy state of the union: Taming the composite state machines of TLS. In *IEEE Symposium on Security and Privacy* (2015).
2. Bouvier, C., Gaudry, P., Imbert, L., Jeljeli, H., Thomé, E. New record for discrete logarithm in a prime finite field of 180 decimal digits, 2014. <http://caramel.loria.fr/p180.txt>.
3. Canetti, R., Krawczyk, H. Security analysis of IKE's signature-based key-exchange protocol. In *Crypto* (2002).
4. Coppersmith, D. Solving linear equations over GF(2) via block Wiedemann algorithm. *Math. Comp.* 62, 205 (1994).
5. Diffie, W., Hellman, M.E. New directions in cryptography. *IEEE Trans. Inform. Theory* 22, 6 (1976), 644–654.
6. Durumeric, Z., Wustrow, E., Halderman, J.A. ZMap: Fast Internet-wide scanning and its security applications. In *Usenix Security* (2013).
7. Geiselmann, W., Kopfer, H., Steinwandt, R., Tromer, E. Improved routing-based linear algebra for the number field sieve. In *Information Technology: Coding and Computing* (2005).
8. Geiselmann, W., Steinwandt, R. Non-wafer-scale sieving hardware for the NFS: Another attempt to cope with 1024-bit. In *Eurocrypt* (2007).
9. Gordon, D.M. Discrete logarithms in GF(p) using the number field sieve. *SIAM J. Discrete Math.* 6, 1 (1993).
10. Harkins, D., Carrel, D. The Internet key exchange (IKE). RFC 2409 (Nov. 1998).
11. Joux, A., Lercier, R. Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method. *Math. Comp.* 72, 242 (2003), 953–967.
12. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., te Riele, H., Timofeev, A., Zimmermann, P. Factorization of a 768-bit RSA modulus. In *Crypto* (2010).
13. Kleinjung, T., Diem, C., Lenstra, A.K., Priplata, C., Stahlke, C. Computation of a 768-bit prime field discrete logarithm. In *EUROCRYPT* (2017).
14. Lipacis, M. Semiconductors: Moore stress = structural industry shift. Technical report, Jefferies, 2012.
15. Meadows, C. Analysis of the Internet key exchange protocol using the NRL protocol analyzer. In *IEEE Symposium on Security and Privacy* (1999).
16. National Security Agency. Cryptography today, August 2015. https://web.archive.org/web/20150905185709/https://www.nsa.gov/ia/programs/suiteb_cryptography/.
17. Orman, H. The Oakley key determination protocol. RFC 2412 (Nov. 1998).
18. Schirokauer, O. Virtual logarithms. *J. Algorithms* 57, 2 (2005), 140–147.
19. The CAD0-NFS Development Team. CAD0-NFS, an implementation of the number field sieve algorithm. <http://cado-nfs.gforge.inria.fr/>, 2017. Release 2.3.0.
20. Thomé, E. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *J. Symbolic Comput.* 33, 5 (2002), 757–775.
21. Fielded capability: End-to-end VPN SPIN 9 design review. Media leak. <http://www.spiegel.de/media/media-35529.pdf>.
22. FY 2013 congressional budget justification. Media leak. <https://cryptome.org/2013/08/spy-budget-fy13.pdf>.
23. Intro to the VPN exploitation process. Media leak, Sept. 2010. <http://www.spiegel.de/media/media-35515.pdf>.
24. SPIN 15 VPN story. Media leak. <http://www.spiegel.de/media/media-35522.pdf>.
25. TURMOIL VPN processing. Media leak, Oct. 2009. <http://www.spiegel.de/media/media-35526.pdf>.

David Adrian, Zakir Durumeric, J. Alex Halderman, Drew Springall, Benjamin VanderSloot, and Eric Wustrow, University of Michigan, Ann Arbor, MI, USA.

Karthikeyan Bhargavan, INRIA Paris-Rocquencourt, Paris, France.

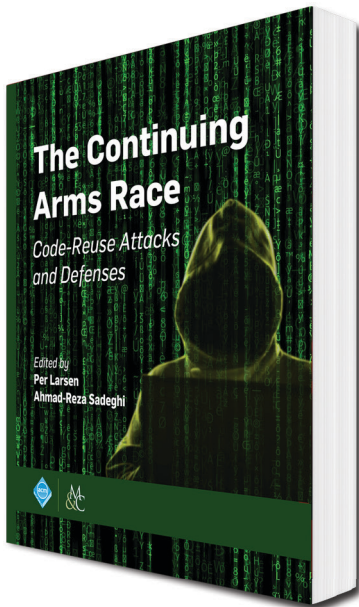
Pierrick Gaudry, Emmanuel Thomé, and Paul Zimmermann, INRIA Nancy-Grand Est, CNRS, and Université de Lorraine, France.

Matthew Green, Johns Hopkins University, Baltimore, MD, USA.

Nadia Heninger and Luke Valenta, University of Pennsylvania, Philadelphia, PA, USA.

Santiago Zanella-Béguelin, Microsoft Research, Cambridge, England, UK.

Copyright held by authors/owners.



There is no silver bullet...
there IS information you
can USE.

EDITED BY
Per Larsen, *Immunant, Inc.*
Ahmad-Reza Sadeghi, *Technische Universität Darmstadt*



ISBN: 978-1-970001-80-8 DOI: 10.1145/3129743
<http://books.acm.org>
<http://www.morganclaypoolpublishers.com/acm>