Factoring as a Service

Luke Valenta, Shaanan Cohney, Alex Liao,
Joshua Fried, Satya Bodduluri, Nadia Heninger

University of Pennsylvania

Abstract The difficulty of integer factorization is fundamental to modern cryptographic security using RSA encryption and signatures. Although a 512-bit RSA modulus was first factored in 1999, 512-bit RSA remains surprisingly common in practice across many cryptographic protocols. Popular understanding of the difficulty of 512-bit factorization does not seem to have kept pace with developments in computing power. In this paper, we optimize the CADO-NFS and Msieve implementations of the number field sieve for use on the Amazon Elastic Compute Cloud platform, allowing a non-expert to factor 512-bit RSA public keys in under four hours for $75. We go on to survey the RSA key sizes used in popular protocols, finding hundreds or thousands of deployed 512-bit RSA keys in DNSSEC, HTTPS, IMAP, POP3, SMTP, DKIM, SSH, and PGP.

1 Introduction

A 512-bit RSA modulus was first factored by Cavallar et al. in 1999, which took about seven calendar months in a distributed computation using hundreds of computers and at least one supercomputer [8]. The current public factorization record, a 768-bit RSA modulus, was reported in 2009 by Kleinjung et al. and took about 2.5 calendar years and a large academic effort [22].

Despite these successes, 512-bit RSA keys are still regularly found in use. Several implementations of the number field sieve have been published, including CADO-NFS [34], Msieve [28], and ggnfs [26], allowing even enthusiastic amateurs to factor 512-bit or larger RSA moduli. In 2009, Benjamin Moody factored a 512-bit RSA code signing key used on the TI-83+ graphing calculator using 2.5 calendar months of time on a single computer, and a distributed effort then factored several more 512-bit TI-68k and TI-Z80 calculator signing keys [33]. The NFS@Home project has organized several large distributed factorizations since 2009. [9] In 2012, Zachary Harris factored the 512-bit DKIM RSA keys used by Google and several other major companies in 72 hours per key using CADO-NFS and Amazon's Elastic Compute Cloud (EC2) service [37].

The persistence of 512-bit RSA is likely due in part to the legacy of United States policies regarding cryptography. In the 1990s, international versions of cryptographic software designed to comply with United States export control regulations shipped with 40-bit symmetric keys and 512-bit asymmetric keys, and export-grade cipher suites with these key sizes were built into protocols like SSL. Restrictions were later raised or lifted on open-source and mass-market software

with cryptographic capabilities, but as of 2015, the United States Commerce Control List still includes systems "designed or modified to use 'cryptography' employing digital techniques performing any cryptographic function other than authentication, digital signature, or execution of copy-protected 'software' and having . . . an 'asymmetric algorithm' where the security of the algorithm is based on . . . factorization of integers in excess of 512 bits (e.g., RSA)". [7]

Factoring a 512-bit RSA key using the number field sieve is still perceived by many as a significant undertaking. In 2015, Beurdouche et al. [6] discovered the FREAK attack, a flaw in many TLS implementations that allows man-in-the-middle attacks to downgrade connections to 512-bit export-grade RSA cipher suites. In evaluating the prospect of a fully exploitable vulnerability, the paper states "we observe that 512-bit factorization is currently solvable at most in weeks." Subsequently, Bhargavan, Green, and Heninger developed a FREAK attack proof-of-concept in part by configuring CADO-NFS to run more efficiently on Amazon EC2. This setup was reported to factor a 512-bit key in approximately 7 hours on EC2, with a few additional hours for startup and shutdown [5].

In this paper, we present an improved implementation which is able to factor a 512-bit RSA key on Amazon EC2 in as little as four hours for $75. Our code is available at `https://github.com/eniac/faas`.

We gain these improvements by optimizing existing implementations for the case of factoring in the cloud. In particular, we rewrote the distributed portion of the number field sieve to use the Slurm job scheduler [35], allowing us to more effectively scale to greater amounts of computational resources. We describe our implementation and parallelizations in Section 3. We then performed extensive experiments on both CADO-NFS and Msieve to determine optimal parameter settings for the network interconnect speeds and resource limits achievable on Amazon EC2. Our experiments are detailed in Section 4.

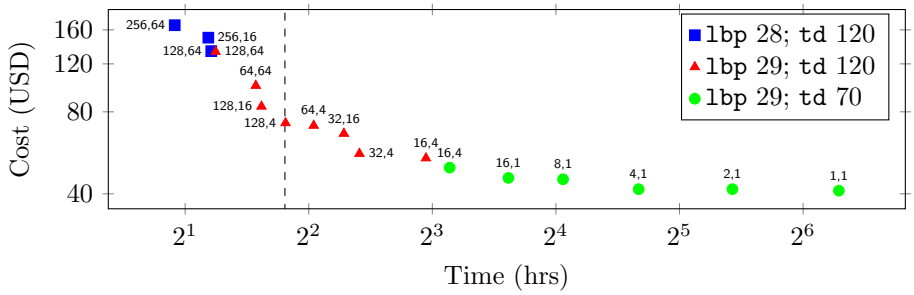


Figure 1: **A time/cost curve for 512-bit factorization.** Each point above is annotated with the instances used for sieving and linear algebra, respectively, and represents an experimental estimate. There are diminishing returns from imperfect parallelization in linear algebra. The dotted line shows the fastest time we were able to achieve; larger experiments usually encountered node instability.

Figure 1 summarizes the time and cost to factor a 512-bit RSA key using current optimal parameters with varying amounts of resources, and the average cost we paid between May and September 2015 for EC2 resources. By tuning the parameters for factoring, one can achieve different points in the trade-off between overall clock time and overall cost. Using more machines gives a faster overall factoring time, but has diminishing returns because of imperfect parallelism. Linear algebra time was measured empirically and sieving was measured once for each parameter set and extrapolated to different numbers of instances.

The order of magnitude of the costs we give lines up with previous reports and estimates of factoring on EC2, and we achieve a significant speedup in overall running time. Performing a computation of this magnitude reliably remains a challenging endeavor. Our paper can also be viewed as a case study on the successes and challenges in trying to replicate a high-performance computing environment in the Amazon EC2 cloud.

In order to measure the impact of fast 512-bit factorization, in Section 5 we analyze existing datasets and perform our own surveys to quantify 512-bit RSA key usage in modern cryptographic public key infrastructures. We find thousands of DNSSEC records signed with 512-bit keys, millions of HTTPS, SMTP, IMAPS, and POP3S servers still supporting `RSA_EXPORT` cipher suites for TLS, and a long tail of 768-bit, 512-bit, and shorter RSA keys in use across DKIM, SSH, IPsec VPNs, and PGP.

2 Background

In this paper, we focus on the impact of factoring on the security of RSA public keys [32], though integer factorization has many applications across mathematics. Factoring the modulus of an RSA public key allows an attacker to compute the corresponding private key, and thus to decrypt any messages encrypted to that key, or forge cryptographic signatures using the private key.

2.1 Number Field Sieve

The general number field sieve is the fastest known algorithm for factoring generic integers larger than a few hundred bits [25]. Its running time is described using L-notation as $L_N[1/3, 1.923] = \exp\left(1.923(\log N)^{1/3}(\log\log N)^{2/3}\right)$–sub-exponential, but super-polynomial [21] in the size of N, the integer to be factored. A gentle introduction to the big ideas behind sieving algorithms for integer factorization and can be found in Pomerance's 1996 survey [30], and more in-depth information on the number field sieve can be found in the books by Lenstra, Lenstra, Manasse, and Pollard [25] and Crandall and Pomerance [12].

In this section, we give a brief overview of the structure of the algorithm, in order to identify potential implementation optimizations and barriers to parallelization. The number field sieve has four main computational stages: *polynomial selection*, *sieving*, *linear algebra*, and *square root*.

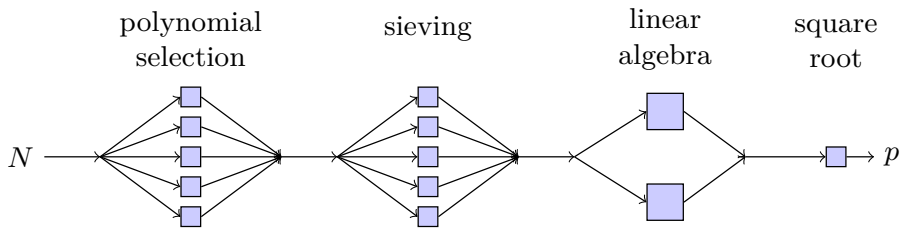


Figure 2: **The number field sieve.** The number field sieve factoring algorithm consists of several main stages. Sieving and linear algebra are the most computationally intensive stages. Sieving is embarrassingly parallel, while parallelizing linear algebra can encounter communication bottlenecks.

The first stage of the algorithm, *polynomial selection*, searches for a polynomial $f(x)$ and integer m satisfying $f(m) \equiv 0 \bmod N$, where N is the integer to factor. $f(x)$ defines the number field $\mathbb{Q}(x)/f(x)$ to be used in the rest of the algorithm. A good choice of polynomial in this stage can significantly speed up the rest of the computation, by generating smaller elements in the sieving phase. Several techniques exist for choosing the polynomial, but in general many different polynomials are tested and the best one is passed on to the next stage. The polynomial selection stage is embarrassingly parallel.

The next stage of the algorithm, *sieving*, factors ranges of integers and number field elements to find many relations of elements and saves those whose prime factors have size less than some size bound B, called the smoothness bound. CADO-NFS uses the *large prime variant* of sieving, and the large prime bound parameters `lbp` control the log of the smoothness bounds. Decreasing these bounds increases the difficulty of sieving, since relations are less likely to factor completely into smaller factors. The sieving stage is also embarrassingly parallel, since candidate relations can be evaluated independently in small batches.

In the third stage, *linear algebra*, the coefficient vectors of the relations are used to construct a large sparse matrix with entries over $\mathbb{F}_2$. Before beginning this stage, some preprocessing on the relations is used to decrease the dimension of the resulting matrix. In general, more relations collected during sieving will produce a smaller matrix and reduce the runtime for linear algebra. The goal of the linear algebra stage is to discover a linear dependency among the rows. This is accomplished via the Block Wiedemann [10] or Block Lanczos [27] algorithms, which are specialized for sparse linear algebra. This step can be parallelized, but the parallelization requires much more communication and synchronization.

The final stage involves computing the *square root* of a number field element corresponding to a dependency in the matrix. In practice, many dependencies will be tested since not all of them will lead to a nontrivial factor; the square roots can be computed and tested in parallel. This step takes only a few minutes.

Discrete log There is also a number field sieve algorithm for discrete logarithms with a nearly identical structure. Many of the implementation improvements that

we describe here also apply to discrete log. However a 512-bit prime-field discrete log is significantly more burdensome than a 512-bit factorization, in large part because the linear algebra stage involves arithmetic over a large-characteristic finite field. Adrian et al. [1] describe 512-bit discrete log computations in practice; we estimate that a single equivalent discrete log computation performed on Amazon EC2 would cost approximately $1400 and take 132 hours.

2.2 Amazon EC2

Amazon Elastic Compute Cloud (EC2) is a service that provides virtualized computing resources that can be rented by the hour. Several competitors exist, including Google Compute Engine. We specialize our results to Amazon largely out of convenience and because when we began this project some tools were specialized to Amazon's infrastructure.

Amazon EC2 bills for computing resources by the *instance-hour*. An *instance* is a single virtualized machine associated with resources including processing cores, memory, and disk storage. Amazon offers many different instance types. We chose the largest type of compute-optimized instance available as of August 2015, the `c4.8xlarge` instance. This instance type has two Intel Xeon E5-2666 v3 processor chips, with 36 vCPUs in a NUMA configuration with 60 GB of RAM.

There are multiple pricing structures available to purchase instance-hours. For our purposes, one can purchase fixed-rate *on-demand* instances, or bid a variable rate for *spot instances* which may be terminated depending on demand. The difference can be significant: for a `c4.8xlarge` instance, the on-demand price as of September 2015 is $1.763, while the average spot price we paid between May and September 2015 was $0.52. We used spot instances for our experiments. Amazon raised our account limit to allow us to launch up to 200 instances.

The `c4.8xlarge` instance type supports Enhanced Networking with 10 GbE interconnect between instances. Machines can be rented in different *availability zones* located around the world, and within an availability zone one can request machines to be co-located in a single *placement group* to minimize latency. We measured the interconnect bandwidth of instances in the same availability zone and placement group at 9.46 Gbit/s, and between instances not in the same placement group at 4–5 Gbit/s. We enabled enhanced networking and launched instances used for linear algebra in one placement group.

The networking environment of Amazon EC2 is distinct from a traditional HPC cluster. The connection was not saturated during our linear algebra optimization tests in Section 4 below. However, our measured interconnect latency, at 151 μs, is significantly greater than most HPC standards. For reference, InfiniBand FDR has latency requirements of 7 μs at 10 Gbit speeds.

Kleinjung, Lenstra, Page, and Smart [23] estimated in 2012 that factoring 512-bit RSA on Amazon EC2 would cost $107 for sieving and $30 for linear algebra. Their estimates were obtained from experiments on truncated sieving jobs and simplified linear algebra. In comparison, we focused on building a system to reliably perform full 512-bit factorizations as quickly as possible given the current state of the EC2 platform. Paterson, Poettering, and Schuldt [29] used EC2 to perform large-scale cryptanalytic experiments for the RC4 stream cipher.

3 Implementation

In order to speed up factoring, we wanted to maximize parallelism. In the polynomial selection and sieving stages, parallelization is straightforward, because the tasks can be split into arbitrarily small pieces to be executed independently, with only a relatively small amount of sequential work to process the results together at the end. Our improvements in these stages come from reliably distributing these tasks across cluster resources in a scalable way. Scaling the linear algebra stage is more complex, because the communication overhead results in diminishing returns from additional resources. We performed extensive experiments to characterize the trade-offs and guide parameter selection.

3.1 Managing Amazon EC2 resources with Ansible

We used Ansible [13], a cluster management tool, to set up and configure an EC2 cluster and to scale the cluster appropriately at each stage of factorization. After the sieving stage, we terminate nodes not required for linear algebra. Ansible can launch and configure a cluster of 50 on-demand instances in under 5 minutes, and 50 spot instances in 10–15 minutes.

3.2 Parallelizing polynomial selection and sieving with Slurm

The polynomial selection and sieving stages generate thousands of individual tasks to be distributed to cluster compute nodes. This requires a job distribution framework that is fast and scalable to many machines. The CADO-NFS implementation is distributed with a Python script to coordinate each stage, including a job distribution system over HTTP designed to require minimal setup from participating computers. Unfortunately this implementation did not scale well to simultaneously tracking thousands of tasks. We experimented with Apache Spark [36] to manage data flow, but Spark was not flexible enough for our needs, and our initial tests suggested that a Spark-based job distribution system was more than twice as slow as the system we were aiming to replace.

Ultimately we chose Slurm (Simple Linux Utility for Resource Management) [35] for job distribution and management during polynomial selection and sieving. Slurm can resubmit failed or timed-out tasks, monitors for and deals with failed nodes, has low startup overhead, and scales well to large clusters.

Our implementation uses a management thread to submit polynomial selection and sieving tasks asynchronously in batches to the Slurm controller, which then handles distribution and execution. This thread rate limits batch sizes in order to get around Slurm's job submission rate of a thousand jobs per second. [4] We found that scheduling two jobs per vCPU yielded faster sieving times than one job per vCPU, since the latter did not always fully saturate CPU usage.

3.3 Parallelizing linear algebra with MPI

After sieving has completed, the relations that have been produced are processed to generate a large, sparse matrix. The runtime of this linear algebra phase

depends on the dimension of the matrix and the number of nonzero entries per matrix row, called the *density*, so the preprocessing stage attempts to produce a matrix that is as small as possible by filtering and combining relations. The parameters that control the effectiveness of the dimension reduction are the number of relations collected and the allowed density of the matrix.

The parallelization of the linear algebra stage is more complex than sieving or polynomial selection. In general, the matrix is divided up into an $n \times n$ grid. In each iteration, each worker operates on its own grid element, gathers results from each of the other workers using the Message Passing Interface (MPI), and combines the results into its own grid element. We used OpenMPI 1.8.6. [18]

Comparing CADO-NFS and Msieve linear algebra We compared the linear algebra implementations of CADO-NFS, which implements the Block Wiedemann algorithm, and Msieve, which implements the Block Lanczos algorithm for linear algebra. Although Block Wiedemann is designed to parallelize well on independent resources, Msieve was significantly faster on our EC2 configuration. Both implementations support MPI out of the box. For a 512-bit factorization with an identical set of 53 million relations, we found that CADO-NFS without MPI completed the linear algebra stage in 350 minutes, while Msieve without MPI completed linear algebra in 140 minutes. When parallelized across multiple EC2 instances, CADO-NFS's runtime did not decrease significantly, whereas Msieve's did. We decided to use Msieve's implementation for linear algebra.

Unfortunately, the input and output formats used by CADO-NFS and Msieve are not compatible, so using Msieve's linear algebra meant we also needed to use Msieve's matrix preprocessing and final square root phases or rewrite these stages ourselves. We compromised by parallelizing Msieve's square root implementation to test multiple dependencies simultaneously, so that the square root phase finishes in approximately 10 minutes.

4 Experiments

We performed several experiments to explore the effects of different parameter settings on running time. All of the experiments in this section were carried out on the same arbitrarily chosen 512-bit RSA modulus. There will be some variation in running time across different moduli. In order to understand this variation, we measured the CPU time required to sieve 54.5 million relations for five different randomly generated RSA moduli with the parameters `lbp` 29 and target density 70 on a cluster with 432 CPUs. We observed a median of 2770 CPU hours with a standard deviation of 227 CPU hours in the sample set.

4.1 Large prime bounds

The large prime bounds `lbp` specify the log of the smoothness bound for relations collected in the sieving stage. Decreasing the large prime bound will decrease the dimension of the matrix and therefore decrease the linear algebra running time,

Table 1: **Large prime bounds.** Decreasing the large prime bound parameter increases the amount of work required for sieving, but decreases the work required for linear algebra. This is an advantageous choice when large amounts of resources can be devoted to sieving.

lbp	relations	matrix rows	matrix size	sieve CPU-hours	linalg instance-hours
28	28.2M	4.96M	1.48 GB	3271.1	5.4
29	44.8M	5.68M	1.71 GB	2369.2	8.5

but will increase sieving time because relations with smaller prime factors are less common. The `lbp` parameter provides the first step for tuning the trade-off between sieving and linear algebra time to optimize for different-sized clusters.

We experimented with `lbp` values 28 and 29. At `lbp` 27, CADO-NFS was unable to gather enough relations even after increasing the sieving area. At `lbp` 30, linear algebra will dominate the computation time even for small clusters.

Table 1 shows the effect of the changing the large prime bound for one experimental setup. Both of the runs used the minimum number of relations required to build a full matrix with target density 70 (see Section 4.2), and linear algebra was completed on a single machine with 36 vCPUs. Decreasing `lbp` from 29 to 28 causes the sieving CPU time to increase by 38% even though fewer relations are collected, but the linear algebra time decreases by 36%.

4.2 Target density

The target density parameter specifies the average number of sparse nonzero entries per matrix row that Msieve will aim for in matrix construction. Linear algebra time is dependent on the product of the density and dimension, and can be decreased by raising the target density to lower the dimension. Figure 3a shows how increasing the target density decreases linear algebra time for a fixed set of input relations on a cluster of 16 instances.

For a 512 bit number with 53 million relations (more than 20 million relations over the minimum), a matrix with target density 70 took 15 minutes to construct and 68 minutes for the linear algebra computation. For the same set of relations, a matrix with target density of 120 took 17 minutes to construct and 55 minutes for linear algebra, a 19% reduction in linear algebra time. However, there were diminishing returns to increases in target density: increasing the target density from 120 to 170 reduced the overall time by only 4%.

The drawback to increasing target density is that more relations are needed from the sieving stage to construct the matrix. Figure 3b shows how the minimum number of relations required increases sharply as target density is increased beyond a particular threshold. When large amounts of resources are available for sieving, the increased work required to collect additional relations can be compensated for by a larger decrease in linear algebra time. For a given cluster size, there is an optimal target density that takes into account these trade-offs.

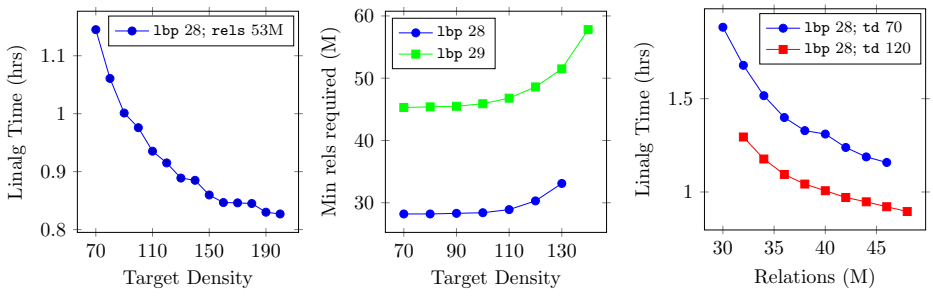


Figure 3: **Target density and oversieving.** Increasing the target density parameter decreases linear algebra time, but requires more relations to construct the matrix. Collecting additional relations beyond the minimum also produces a better matrix and decreases linear algebra time. This trade-off can be advantageous if more resources can be devoted to sieving, as sieving parallelizes well.

4.3 Oversieving

Oversieving means generating excess relations during the sieving phase. This can help to produce an easier matrix for the linear algebra phase, reducing linear algebra runtime. We ran experiments varying cluster configurations, target densities, and large prime bounds to determine an oversieving curve for each. Figure 3c shows two representative oversieving curves for a 16-node linear algebra cluster with `lbp` 28 and target densities 70 and 120, respectively. For the target density 70 curve, the linear algebra time for the minimum number of relations required to construct the matrix, 30 million, was 112 minutes. At 32 million relations, the linear algebra time was reduced to 101 minutes, an 11% improvement. However, as Figure 3c shows, there are diminishing returns to oversieving, while the work required to produce additional relations scales close to linearly. Optimal oversieving amounts are dependent on the cluster configuration.

4.4 MPI grid size

The grid size parameter directly controls the number of work units that MPI can assign to cluster resources. We experimented with both fine-grained grids matching the number of work units to the total number of vCPUs, and coarse-grained grids matching work units to instances. The optimum turned out to be somewhere in the middle: a single multithreaded work unit was not able to occupy all of the 36 vCPUs on a single instance, while the other extreme is likely to become limited by communication overhead since the Block Lanczos algorithm requires each node to gather results from every other node at each iteration.

In order to determine the optimal grid size, we tested a range of grid sizes for cluster sizes of 1, 4, 16, and 64 instances. The best performance for clusters with 1 and 4 instances was 4x4 and 8x8, respectively, where each cluster had 16 work

units in total. For the clusters with 16 and 64 instances, the optimal grid size was 8x8 and 16x16, where each cluster had 4 work units in total. The differences as cluster size grows are likely due to communication bottlenecks.

4.5 Processor affinity

The default parameters of OpenMPI dictate that each of the work units is bound to a specific machine, but when multiple work units are assigned to the same instance they compete for the same processor and memory resources, creating processor scheduling overhead and increased variance in the work unit iteration times. Each work unit must iterate together, so the time per iteration is dictated by the slowest work unit. Since the `c4.8xlarge` EC2 instances have two processor sockets and a NUMA memory layout, the distribution of the threads of a work unit across two processors means longer intra-process communication times and slower memory access times. We used the rankfile/process affinity parameter in OpenMPI to bind each of the work units on a single instance to its own subset of processor cores and saw an improvement of 1-2% in linear algebra time.

We also tested binding each thread of each of the work units to individual cores, but this did not improve running times.

4.6 Block size

The default block size in Msieve is 8192 bytes. Theoretically, matching the block size used in Msieve with the size of the L1 cache of the processor should yield better performance by decreasing cache and memory access times. However, for the parameters `lbp` 28 and target density 70, increasing the block size from 8K to 16K increased computation time from 67 minutes to 69 minutes, and increasing the block size from 8K to 32K increased computation time from 67 minutes to 73 minutes. We decided to leave the block size unchanged.

4.7 Putting it all together

To generate the data points in Figure 1, we individually timed each sieving job together with system overhead. For each set of parameters, we combined the linear algebra running time from the experiments in this section with the total measured running time to complete enough sieving jobs to generate the required number of relations. We then added a measured estimate of costs for the remaining steps of factoring to get our total running time estimates. We were able to reliably achieve running times under four hours for factoring, but in several attempts to verify lower overall times, we encountered issues where some EC2 instances in our cluster ran more slowly than others or became unresponsive. These issues become more pronounced with larger cluster sizes. Our sieving setup can deal gracefully with slow nodes, but linear algebra is more fragile and is currently limited by the slowest node.

5 512-bit keys still in use

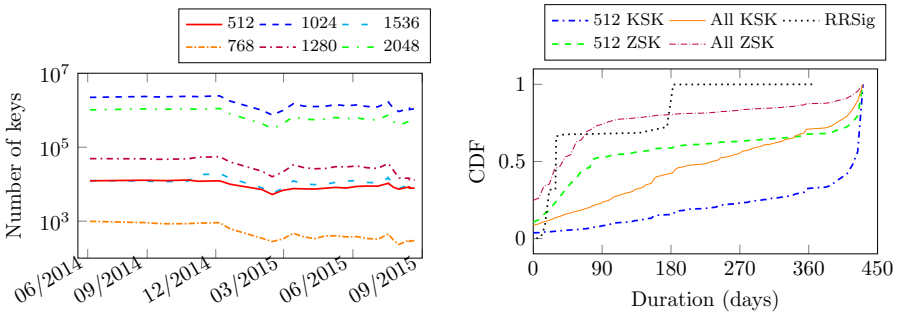
In this section, we survey RSA key lengths across public key infrastructures for a variety of protocols, finding that 512-bit RSA keys are surprisingly persistent.

5.1 DNSSEC

DNSSEC [3] is a DNS protocol extension that allows clients to cryptographically authenticate DNS records. DNS records protected by DNSSEC include a public key record (usually RSA) and a signature that can be chained up to a trusted root key. DNSKEY records can contain either a zone-signing key (ZSK), used to sign DNS records, or a key-signing key (KSK), used to sign DNSKEY records. RFC 4033 [3] specifies that zone-signing keys may have shorter validity periods, and key-signing keys should have longer validity periods. RFC 6781 [24], published by the IETF in 2012 on DNSSEC Operational Practices, states that "it is estimated that most zones can safely use 1024-bit keys for at least the next ten years."

An attacker who knows the private key to a zone-signing key or key-signing key could mount an active attack to forge DNS responses for any descendants below that location in the chain.

We analyzed several DNSSEC datasets. The most comprehensive is a collection of DNS records collected by Rapid7 which we downloaded from Scans.io. They performed biweekly DNS lookups on approximately 529 million domains starting in June 2014 and continuing to present. The number of lookups varies by as much as 61 million domains across scans, and the number of domains with valid DNSSEC records fluctuated between 3.7 million and 1.1 million and decreased over time compared to total domains. The relative fraction of DNSSEC key sizes did not change much over time. The distribution is shown in Figure 4a.



(a) Key sizes

(b) Key and signature durations

Figure 4: **DNSSEC key sizes and duration.** The ratios of RSA key lengths has remained relatively stable over time, although the total number of DNSSEC keys collected fluctuated across scans. The number of 512-bit keys remained around 10,000, or 0.35% of the total. Many DNSSEC keys are rotated infrequently, and 512-bit keys are rotated less frequently than longer keys.

Table 2: **HTTPS RSA common key lengths and export RSA support.**

Length	All Certificates	Distinct Keys	Trusted Certificates	Trusted and Valid
512	303,199 (0.9%)	32,870	0 (0.0%)	0 (0.0%)
768	26,582 (0.1%)	14,581	0 (0.0%)	0 (0.0%)
1024	12,541,661 (36.8%)	3,196,169	4,016 (0.0%)	4,012 (0.0%)
1536	2,537 (0.0%)	2,108	0 (0.0%)	0 (0.0%)
2048	20,782,686 (60.9%)	6,891,678	14,413,589 (42.2%)	14,411,618 (42.2%)
2432	2,685 (0.0%)	1,191	128 (0.0%)	128 (0.0%)
3072	65,765 (0.2%)	58,432	1,787 (0.0%)	1,787 (0.0%)
4096	391,123 (1.1%)	218,334	259,898 (0.8%)	259,830 (0.8%)
8192	2,172 (0.0%)	971	481 (0.0%)	481 (0.0%)
RSA Export	2,630,789 (7.7%)			
Total	34,121,474 (100.0%)		14,680,782 (43.0%)	14,678,739 (43.0%)

In order to measure the completeness of the Rapid7 dataset, we compared to a second dataset of anonymized 512-bit DNSSEC keys for all .com, .net, and .org domains between February 22, 2015 and September 3, 2015 from the SURFnet DNS measurement infrastructure of van Rijswijk-Deij, Jonker, Sperotto, and Pras [31] which was provided to us by the researchers. The SURFnet data contained 2,116 distinct public keys of which 1,839 (86%) were present in the Rapid7 scans from the same time period. To measure how many 512-bit keys are in active use, SURFnet provided a set of all 512-bit DNSkey records collected using their passive DNS monitoring system for a one-month period between September 12, 2015 and October 13, 2015. The set included 1,239 records covering 613 distinct domains and contained 705 distinct keys.

Finally, we performed DNS lookups on eleven thousand zones not contained in the Rapid7 dataset that were required for signature validation. 56% of domains with 512-bit keys failed signature verification, most commonly because the TLD signature was not present in the chain of trust.

Many keys were never rotated at all over the 431-day period spanned by the Rapid7 dataset, and signatures were renewed more frequently than keys were updated. Figure 4b illustrates signature validity periods and key lifetimes. Signature validity periods are clustered around a few common ranges: 33% of keys were signed for six months, 34% percent for one month, 25% for three weeks, and 6% for 14 days. 512-bit zone-signing keys and key-signing keys were less frequently rotated than other key sizes.

5.2 HTTPS

RSA public keys are used for both encryption and authentication in the TLS protocol. If the client and server negotiate an RSA cipher suite, the client encrypts the premaster secret used to derive the session keys to the RSA public key in the server's certificate. An adversary who compromises the private key

can passively decrypt session traffic from the past or future. However, since no 512-bit certificates have currently valid signatures from certificate authorities, these servers are also vulnerable to an active man-in-the-middle attack from an adversary who simply replaces the certificate.

If the client and server negotiate a Diffie-Hellman or elliptic curve Diffie-Hellman cipher suite, the server uses the public key in its certificate to sign its key exchange parameters. An adversary who knows the private key could carry out a man-in-the-middle attack by forging a correct signature on their desired parameters. Since again no 512-bit certificates are currently signed or trusted, such an active adversary could also merely replace the server certificate in the exchange along with the chosen Diffie-Hellman parameters.

Finally, connections to servers supporting `RSA_EXPORT` cipher suites may be vulnerable to an active downgrade attack if the clients have not been patched against the FREAK attack. [6] Successfully carrying out this attack requires the attacker to factor the server's ephemeral RSA key, which is typically generated when the server application launches and is reused as long as the server is up. "Ephemeral" RSA keys can persist for weeks and are almost always 512 bits.

We examined IPv4 scan results for HTTPS on port 443 performed using Zmap [17] by the University of Michigan which we accessed via Scans.io and the Censys scan data search interface developed by Durumeric et al. [14]. Table 2 summarizes scans from August 23 and September 1, 2015.

Durumeric, Kasten, Bailey, and Halderman [16] examined the HTTPS certificate infrastructure in 2013 using full IPv4 surveys and found 2,631 browser-trusted certificates with key lengths of 512 bits or smaller, of which 16 were valid. Heninger, Durumeric, Wustrow, and Halderman [20] performed a full IPv4 scan of HTTPS in October 2011 with responses from 12.8 million hosts, and found 123,038 certificates (trusted and non-trusted) containing 512-bit RSA keys. Similar to [20], we observe many repeated public keys.

5.3 Mail

Table 3 summarizes several Internet-wide scans targeting SMTP, IMAPS, and POP3S. The scans were performed by the University of Michigan using Zmap between August 23, 2015, and September 3, 2015.

We used the Censys scan database interface provided by [14] to analyze the data. While only a few hundred few mail servers served TLS certificates containing 512-bit RSA public keys, 13% of IMAPS and POP3S servers and 30% of SMTP

Table 3: **Mail protocol key lengths.**

	Port	Handshake	RSA_EXPORT	512-bit Certificate Key
SMTP	25	4,821,615	1,483,955 (30.8%)	64 (0%)
IMAPS	993	4,468,577	561,201 (12.6%)	102 (0%)
POP3S	995	4,281,494	558,012 (13.0%)	115 (0%)

servers supported RSA_EXPORT cipher suites with 512-bit ephemeral RSA, meaning that unpatched clients are vulnerable to the FREAK downgrade attack by an adversary with the ability to quickly factor a 512-bit RSA key.

We also examined DKIM public keys. DomainKeys Identified Mail [2] is a public key infrastructure intended to prevent email spoofing. Mail providers attach digital signatures to outgoing mail, which recipients can verify using public keys published in a DNS text record.

We gathered DKIM public keys from the Rapid7 DNS dataset. However, the published dataset had lowercased the base64-encoded key entries, so we performed DNS lookups on the 11,600 domains containing DKIM records ourselves on September 4, 2015. We made a best-effort attempt to parse the records, but 5% of the responses contained a key that was malformed or truncated and could not be parsed. Of the remainder, 124 domains used 512-bit keys or smaller, including one that used a 128-bit RSA public key. We were able to factor this key in less than a second on a laptop and verify that it is, in fact, a very short RSA public key. Table 4 summarizes the distribution.

Table 4: **DKIM key sizes.**

Length	Keys
4096	5 (0.0%)
2048	64 (0.5%)
1028	1 (0.0%)
1024	10,726 (92.2%)
768	126 (1.1%)
512	103 (0.9%)
384	20 (0.2%)
128	1 (0.0%)
Parse error	591 (5.1%)
Total	11,637

Durumeric et al. [15] surveyed cryptographic failures in email protocols using Internet-wide scans and data from Google. They examine DKIM use in April 2015 and discovered that 83% of mail received by Gmail contained a DKIM signature, but of these, 6% failed to validate. Of these failures, 15% were due to a key size of less than 1024 bits, and 63% were due to other errors.

Table 5: **IPsec VPN certificate keys**

Length	Keys
4096	37 (0.8%)
3072	1 (0.0%)
2048	2,257 (51.3%)
1024	1,804 (41.0%)
768	1 (0.0%)
512	69 (1.6%)
Parse error	234 (5.3%)
Total	4,403 (100%)

5.4 IPsec

We conducted two Zmap scans of the full IPv4 space to survey key sizes in use by IPsec VPN implementations that use RSA signatures for identity validation during server-client handshakes. An adversary who compromised the private keys for one of these certificates could mount a man-in-the-middle attack.

Our Zmap scans targeted IKEv1 aggressive mode [19], which allows the server to send a certificate after a only a single message is received. The messages we sent contained proposals for DES, 3DES, AES-128, and AES-256 each with both SHA1 and MD5. Our first scan offered a key exchange using Oakley group 2 (a 1024-bit Diffie-Hellman group) and elicited certificates from 4% of the servers that accepted

our message. Our second scan offered Oakley group 1 (a 768-bit Diffie-Hellman group) and received responses from 0.2% of hosts. Of the non-responses from both scans, 71% of the servers responded indicating that they did not support our combination of aggressive mode with our chosen parameters, 16% rejected our connection for being unauthorized (not on a whitelist), and the remaining 11% returned other errors.

5.5 SSH

SSH hosts authenticate themselves to the client by signing the protocol handshake with their public host key. Clients match the host key to a stored trusted fingerprint. An adversary who is able to compromise the private key for an SSH host key can perform an active man-in-the-middle attack.

Table 6: **SSH host key lengths.**

RSA Size	Hosts	Distinct
512	508 (0.0%)	316
768	2,972 (0.0%)	2,419
784	3,119 (0.0%)	223
1020	774 (0.0%)	572
1024	296,229 (4.4%)	91,788
1040	2,786,574 (41.3%)	1,407,922
1536	639 (0.0%)	536
2048	3,632,865 (53.9%)	1,752,406
2064	1,612 (0.0%)	957
4096	15,235 (0.2%)	1,269
RSA Total	6,741,352	3,258,742
DSA	692,011	421,944
ECDSA	2,192	2,192

Table 6 summarizes host key sizes collected by a Zmap scan of SSH hosts on port 22 mimicking OpenSSH 6.6.1p1. The data was collected in April 2015 by Adrian et al. [1], who provided it to us. A very large number of hosts used 1040-bit keys; these hosts had banners identifying them as using Dropbear, a lightweight SSH implementation aimed at embedded devices. Heninger, Durumeric, Wustrow, and Halderman [20] performed a full IPv4 scan of SSH public keys in February 2012 offering only `Diffie-Hellman Group 1` key exchange. Of 10 million responses, they reported that 8,459 used 512-bit RSA host keys and observed many repeated host keys.

Clients can also use public keys to authenticate themselves to a server. An adversary who is able to compromise the private key for a client SSH authentication key can access the server by logging in as the client. Ben Cox [11] collected 1,376,262 SSH public keys that had been uploaded to GitHub by users to authenticate themselves to the service between December 2014 and January 2015 by using GitHub's public API. He collected 1,205,330 RSA public keys, 27,683 DSA public keys, and 1,060 ECDSA public keys. Of the RSA public keys, 2 had 256-bit length, 3 had 512-bit length, and 28 had 768-bit length.

5.6 PGP

PGP implements encryption and digital signatures on email or files. RSA public keys can be used for both encryption and signatures. PGP uses a public "web of trust" model: users can distribute their public keys along with signatures attesting trust relationships via a public network of keyservers. An adversary

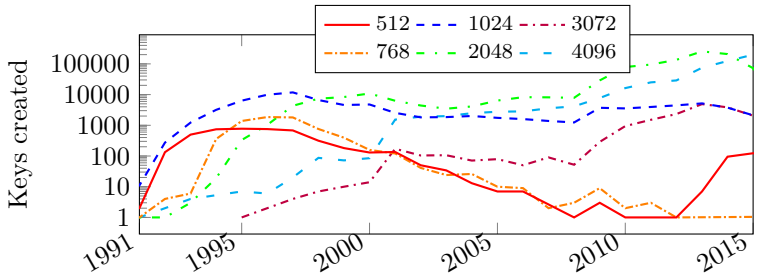


Figure 5: **PGP RSA public key lengths by reported creation date.**

who compromises a PGP public key could use it to impersonate a user with a digital signature or decrypt content encrypted to that user.

We downloaded a PGP keyserver bootstrap dataset from `keyserver.borgnet.us` on October 4, 2015. It contained 4.9 million public keys from 3 million users. Of these, 1.6 million were RSA, 1.7 million were DSA, 1.7 million were ElGamal, 398 were ECDH, 158 were EdDSA, and 513 were ECDSA. 4,688 512-bit RSA keys were present in the dataset; 123 of them listed a creation date in 2015. Figure 5 shows the shift to longer RSA key lengths over time.

6 Conclusions

512-bit RSA has been known to be insecure for at least fifteen years, but common knowledge of precisely *how* insecure has perhaps not kept pace with modern technology. We build a system capable of factoring a 512-bit RSA key in under four hours. We then measure the impact of such a system by surveying the incidence of 512-bit RSA in modern cryptographic infrastructure, and find a long tail of too-short public keys and export-grade cipher suites still in use in the wild. These numbers illustrate the challenges of keeping an aging Internet infrastructure up to date with even decades-old advances in cryptanalysis.

Acknowledgements

We thank Daniel Bernstein, Tanja Lange, Pierrick Gaudry, Emmanuel Thomé, and Paul Zimmermann for helpful comments and discussion. Nicole Limtiaco, Toma Pigli, Zachary Ives, and Sudarshan Muralidhar contributed to early versions of this project. We thank Osman Surkatty for help with Amazon services. We are grateful to Zakir Durumeric, Roland van Rijswijk-Deij, and Ryan Castellucci for providing data. We thank Ian Goldberg for suggesting additional references [21] and Lionel Debroux for a correction. This work is based upon work supported by the National Science Foundation under grant no. CNS-1408734, a gift from Cisco, and an AWS Research Education grant.

References

1. Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., Zimmermann, P.: Imperfect forward secrecy: How Diffie-Hellman fails in practice. In: 22nd ACM Conference on Computer and Communications Security (CCS '15) (2015)
2. Allman, E., Callas, J., Delany, M., Libbey, M., Fenton, J., Thomas, M.: DomainKeys identified mail (DKIM) signatures (2007), http://www.ietf.org/rfc/rfc6376.txt
3. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: DNS Security Introduction and Requirements. RFC 4033, Internet Society (March 2005), http://www.ietf.org/rfc/rfc4033.txt
4. Auble, D., Jette, M., et al.: Slurm documentation. http://slurm.schedmd.com/, accessed: 2015-09-19
5. Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.Y., Zinzindohoue, J.K.: FREAK: Factoring RSA export keys (2015), https://www.smacktls.com/#freak
6. 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)
7. Bureau of Industry and Security: Export administration regulations (2015), http://www.bis.doc.gov/index.php/regulations/export-administration-regulations-ear
8. Cavallar, S., Dodson, B., Lenstra, A.K., Lioen, W., Montgomery, P.L., Murphy, B., te Riele, H., Aardal, K., Gilchrist, J., Guillerm, G., Leyland, P., Marchand, J., Morain, F., Muffett, A., Putnam, C., Putnam, C., Zimmermann, P.: Factorization of a 512-bit RSA modulus. In: Preneel, B. (ed.) Advances in Cryptology - EUROCRYPT 2000, Lecture Notes in Computer Science, vol. 1807, pp. 1–18. Springer Berlin Heidelberg (2000)
9. Childers, G.: NFS@home, http://escatter11.fullerton.edu/nfs/
10. Coppersmith, D.: Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. Mathematics of Computation 62(205), 333–350 (1994)
11. Cox, B.: Auditing GitHub users SSH key quality, https://blog.benjojo.co.uk/post/auditing-github-users-keyscollected
12. Crandall, R., Pomerance, C.B.: Prime numbers: a computational perspective, vol. 182. Springer Science & Business Media (2006)
13. DeHaan, M.: Ansible, http://www.ansible.com
14. Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., Halderman, J.A.: A search engine backed by Internet-wide scanning. In: Proceedings of the 22nd ACM Conference on Computer and Communications Security (Oct 2015)
15. Durumeric, Z., Adrian, D., Mirian, A., Kasten, J., Bursztein, E., Lidzborski, N., Thomas, K., Eranti, V., Bailey, M., Halderman, J.A.: Neither snow nor rain nor MITM... an empirical analysis of email delivery security. In: Proceedings of Internet Measurement Conference (IMC) 2015 (2015)
16. Durumeric, Z., Kasten, J., Bailey, M., Halderman, J.A.: Analysis of the HTTPS certificate ecosystem. In: Proceedings of the 13th Internet Measurement Conference (Oct 2013)
17. Durumeric, Z., Wustrow, E., Halderman, J.A.: ZMap: Fast Internet-wide scanning and its security applications. In: Proceedings of the 22nd USENIX Security Symposium (Aug 2013)

18. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Proceedings, 11th European PVM/MPI Users' Group Meeting. pp. 97–104. Budapest, Hungary (September 2004)
19. Harkins, D., Carrel, D.: The Internet Key Exchange (IKE). RFC 2409, RFC Editor (November 1998), http://www.rfc-editor.org/rfc/rfc2409.txt
20. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your Ps and Qs: Detection of widespread weak keys in network devices. In: Proceedings of the 21st USENIX Security Symposium (Aug 2012)
21. Hughes, E.: How to give a math lecture at a party (2000), https://web.archive.org/web/20010222192642/http://www.xent.com/FoRK-archive/oct00/0429.html
22. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., et al.: Factorization of a 768-bit RSA modulus. In: Advances in Cryptology–CRYPTO 2010, pp. 333–350 (2010)
23. Kleinjung, T., Lenstra, A.K., Page, D., Smart, N.P.: Using the cloud to determine key strengths. In: Progress in Cryptology-INDOCRYPT 2012, pp. 17–39 (2012)
24. Kolkman, O.M., Mekking, W.M., Gieben, R.M.: DNSSEC Operational Practices, Version 2. RFC 6781, Internet Society (December 2012), http://www.ietf.org/rfc/rfc6781.txt
25. Lenstra, A.K., Lenstra Jr, H.W., Manasse, M.S., Pollard, J.M.: The number field sieve. Springer (1993)
26. Monico, C.: GGNFS, http://www.math.ttu.edu/~cmonico/software/ggnfs/
27. Montgomery, P.L.: A block Lanczos algorithm for finding dependencies over GF(2). In: Guillou, L.C., Quisquater, J.J. (eds.) Advances in Cryptology–EUROCRYPT '95, pp. 106–120 (1995)
28. Papadopoulos, J.: Msieve, http://www.boo.net/~jasonp/qs.html
29. Paterson, K.G., Poettering, B., Schuldt, J.C.: Big bias hunting in Amazonia: Large-scale computation and exploitation of RC4 biases (invited paper). In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology—ASIACRYPT 2014, pp. 398–419 (2014)
30. Pomerance, C.: A tale of two sieves. In: Notices Amer. Math. Soc (1996), http://www.ams.org/notices/199612/pomerance.pdf
31. van Rijswijk-Deij, R., Jonker, M., Sperotto, A., Pras, A.: The Internet of names: A DNS big dataset. SIGCOMM Comput. Commun. Rev. 45(5), 91–92 (Aug 2015)
32. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
33. Smith, D.: All TI signing keys factored (September 2009), http://www.ticalc.org/archives/news/articles/14/145/145273.html
34. Team, T.C.D.: CADO-NFS, an implementation of the number field sieve algorithm (2015), http://cado-nfs.gforge.inria.fr/
35. Yoo, A.B., Jette, M.A., Grondona, M.: Slurm: Simple Linux utility for resource management. In: Job Scheduling Strategies for Parallel Processing. pp. 44–60. Springer (2003)
36. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. vol. 10, p. 10 (2010)
37. Zetter, K.: How a Google headhunter's e-mail unraveled a massive net security hole, http://www.wired.com/2012/10/dkim-vulnerability-widespread/