# A Brute Force Search of DES Keyspace

Matt Curtin*
interhack.net
cmcurtin@interhack.net

Justin Dolske
ANS Communications
dolske@reston.ans.net

*This article to appear in the May 1998 issue of **;login:**.*

*"When in doubt, use brute force."* —*Ken Thompson*

**Abstract**

The Data Encryption Standard (DES) has been the workhorse of cryptography for some 20 years. Its wide deployment and small (by today's standards) key size make it an interesting target for attackers. This paper discusses the first public "crack" of a DES-encrypted message using brute force, and shows how the sort of power necessary to reproduce this can be mustered by individuals and very small organizations with little or no funding.

We originally suggested that this work is repeatable, and have been proven correct, as DES has fallen again, and RC5-32/12/7 has been defeated by brute force. We strongly advise systems based on DES to be replaced with systems that use longer keys.

## 1 Introduction

On January 28, 1997 RSA Laboratories launched a series of cryptographic challenges [5]. The goal was to find secret messages which had been encrypted with keys of varying length. One of the most tantalizing of these challenges was based on DES, a widely used encryption algorithm with a 56-bit key. Soon after two easier challenges had been broken, attention turned to the DES challenge.

Led by Rocke Verser, Matt Curtin, and Justin Dolske, the DESCHALL effort [8] sought to crack RSA's DES Challenge [5] by means of a large-scale, distributed computing project on the Internet. We simply endeavored to try each of the $2^{56}$ (over 72 quadrillion[1]) keys that might have been used to encrypt the secret message—a brute force attack. Brute force attacks like this are naturally suited to distributed or parallel computing efforts, since they essentially consist of a large number of independent problems—the testing of each key.

Although not a new attack by any means, brute force key search has been a metric by which the security of cryptosystems are judged. If an algorithm is believed to be "safe", that typically means that the best known attack against the system is infeasible. Often, a "safe" algorithm's security is measured in terms of the cost of a brute force attack. The number of possible keys determines the feasibility of this attack.

While there has been relatively widespread belief that government intelligence agencies have had the technology and resources available to efficiently perform brute force attacks against DES, no one had ever accomplished the feat in public before this project's success. DES is still widely deployed in a variety of environments, including financial circles. DES is, therefore, a real target, and because of its relatively small key size (by today's standards), it's an attractive one.

*This work completed while Matt was at Megasoft Online and Justin was at The Ohio State University.
[1]Unfortunately, names of large numbers are ambiguous. We'll use the US convention for these, and also provide an unambiguous representation of the number where necessary throughout this article.

## 2 Architecture

Our approach centered around a single “key server” which kept track of which blocks of keys had been tested. Clients would then contact the server, via the Internet, to request work and report the results, as illustrated in Figure 1.

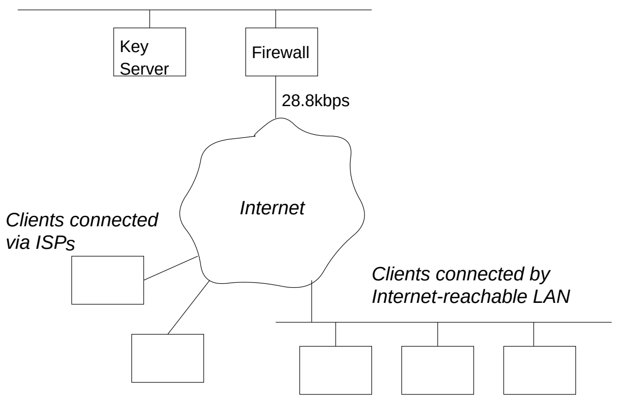
Key Server
Firewall
28.8kbps
Internet
Clients connected via ISPs
Clients connected by Internet-reachable LAN

Figure 1: DESCHALL Architecture

### 2.1 Protocol

All communication between a client and the server was done through the UDP protocol, a standard part of any IP stack. UDP is a low-overhead, connectionless protocol that was sufficient for our needs. The protocol used is an extension of the one designed and used by Germano Caronni in the crack of RSA’s RC5-32/12/6 contest [4]. It consisted of just a few simple messages:

**“Initial” request** provided the server with the client type/version, and requested an initial block of keys to check.

**“Not found” request** reported a range of previously assigned keys which were found to not contain the key, and requested a new block of keys to check.

**“Answer” reply** sent by the server in reply to a client’s request for more work (via either of the above two messages).

**“Message” reply** could be sent by the server to cause a text message to be displayed by the client, in order to convey important information.

**“Kill” reply** could be sent by the server to cause a client to terminate.

Clients would automatically increase the size of the key blocks they requested so that they would gradually reach the point where a block of keys took about 30 minutes to test. Blocks were always $2^N$ keys in size, where $N$ was generally between 22 and 30.

Additionally, all messages dealing with key blocks included checksums for message integrity, since UDP (as a low-overhead protocol) does not make any such checks itself. To help prevent sabotage, the client’s “Not found” message contained additional data, calculated during its search, to allow the server to verify that the client had actually searched the assigned keyspace.

### 2.2 Server and Clients

For most of the challenge, the keyserver was an IBM PS/2 Server (a relatively slow 486 based system) with 56 MB of RAM, connected to the Internet via a dedicated 28.8 kbps PPP connection. This server was able to easily handle the load from approximately 10,000 clients, although a Pentium based backup server was occasionally used during periods of unusually high load.

The clients that used this protocol were designed to run on a wide variety of systems. By the end of the contest, we had 40 different clients available for a variety of different hardware and operating system combinations. All of the clients running on Intel (or compatible) and Macintosh PowerPC hardware contained hand-optimized assembly code, while the remainder of the clients were entirely done in C. Java was briefly considered, but it was quickly dropped—we already had clients for a large variety of systems, and it was generally agreed that the speed of a generic Java version would be unacceptable.

The clients were highly optimized for decrypting DES messages, using a variety of methods to optimize the DES process and detect non-winning keys as early as possible. Using these methods, a 200 MHz Pentium system was able to test approximately 1 million keys/second, and a 250 MHz PowerPC 604e based system reached 1.5 million keys/second. Towards the end of the contest, we introduced a "bitslice" client inspired by Biham [3] which was extremely fast on 64-bit systems, as well as slightly faster on most other systems.

With this new client, a 500 MHz Alpha was able to test 5.3 million keys/second, and a 167 MHz UltraSPARC was able to test 2.4 million keys/second. In the end, Intel-compatible systems accounted for 53.8% of the keys searched, SPARC based systems for 21.3 %, PowerPC systems for 8.1%, and a mix of other systems for the remaining 16.8% of the keys.

All of the clients would, by default, run with low priority, so that only "idle cycles" would be used, rather than interfering with the work that the host would normally perform. An interesting side-effect of the "only use idle cycles" approach of DESCHALL, shown in Figure 2, is that weekends would show significant peaks in average host speed, as there were more idle cycles generally available. Also, performance improvements in the clients contributed toward a steady increase in average host speed.

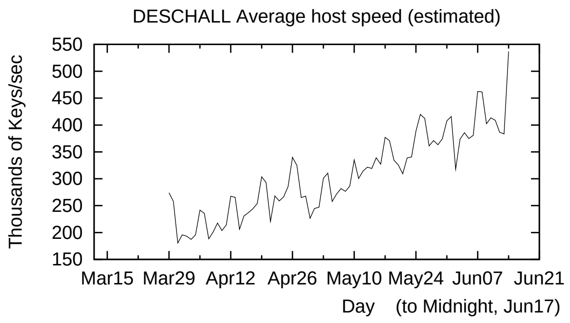

Figure 2: Estimated Average Host Speed

## 2.3 Gateways and Proxies

Soon after DESCHALL began to become popular, we found that firewalls at some sites would block the UDP messages the client and server were trying to exchange. To circumvent the problem, we developed a pair of "gateways" or "proxies" that would tunnel the UDP messages through TCP connections, illustrated in Figure 3. One of these proxies would sit inside the user's network, and the other was maintained by the DESCHALL organizers. Clients running behind a firewall would use the "U2T" gateway as a keyserver. The U2T gateway would receive the client's datagram, and send the data through a TCP connection to the "T2U" gateway. The data was also reformatted to simulate an HTTP (Web) request, to allow passage through firewalls that disallowed arbitrary TCP connections but allowed Web access. For sites with application-layer firewalls, the client's U2T gateway could use the site's Web proxy, which would forward the request to our T2U gateway. Via either method, the T2U gateway would then convert the received data back into a UDP datagram, and send to to the keyserver.

The DESCHALL gateways allowed a large number of people to participate, who would have been otherwise unable. For example, the entirety of Sun Microsystems' contribution (ranked 5th in total keys tested) was conducted through the gateways.

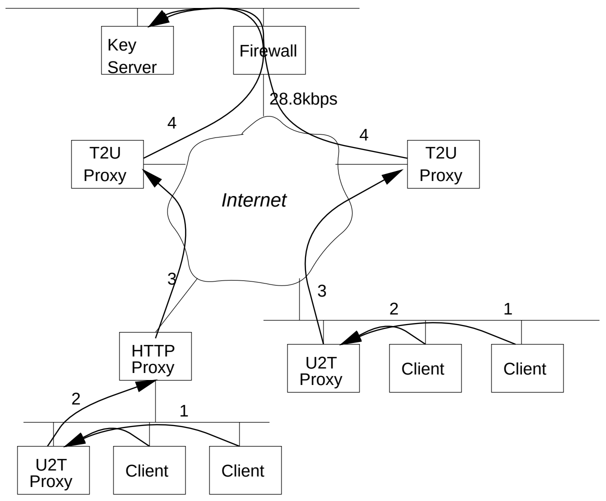

Figure 3: DESCHALL Proxy Architecture

## 3 Results

We have demonstrated that a brute-force search of DES keyspace is not only possible, but is also becoming practical for even modestly funded groups. RSA's prize for the find was US$10,000; it is safe to say that DES is inadequate for protecting data of any greater value.

### 3.1 Small Keys are Bad Keys

With the increasing amount of computing power available at lower and lower costs, today's cryptosystems must be able to withstand brute-force attacks that would have been unthinkable in the relatively recent past. Simply put, DES and other small-key cryptosystems are weak, and vulnerable to attack by groups with even minimal funding.

What we have done is something that the security community has known to be possible for some time. To our knowledge, however, this is the first time that a 56-bit key for DES (or any cryptosystem) has been successfully found in a brute force search.

### 3.2 Wide Availbility of Massive Computing Power

At the same time that the cost of computing is going down, the availability of massive computational power is increasing. Given the ubiquity of the Internet and the fact that key search is easily parallelizable, it's relatively easy to harness the power of many thousands of computers of all types.

During the course of the DESCHALL project, more than 78,000 unique IP addresses were recorded by the keyserver as having participated to some extent. We had a peak of about 14,000 unique hosts[2] within a single 24-hour period. All participants were volunteers; a one-time prize of US$4,000 was awarded to the person whose machine found the winning key.

While we did get a relatively large number of hosts to participate, it's very easy to imagine getting a much larger number of hosts involved. Three major considerations influenced the number of hosts available:

1. Because of concern about government restrictions on the export of cryptographic products from the US, our client distribution site would only allow hosts within the US or Canada to download our clients. This certainly restricted many of our friends abroad from participating.

[2]We actually mean IP addresses, which we assume to be hosts. The actual number is inflated by the number of participating hosts whose IP addresses are dynamically allocated and deflated by the number of hosts that participate from behind proxy gateways.

2. Everyone downloading and installing our client had to go get it, know what it did, and usually had to put forth some effort to keep it running. Spending time to make the client a neater "package", automatically making it start at system boot time, or implementing it as a screen saver, might have allowed more of the hosts that participated only briefly (i.e., until the system was rebooted) to remain active in the effort through its duration. If we were interested in accomplishing our objective without regard to ethical considerations, we could have built our client's functionality into viruses, worms, trojan horses, ActiveX controls, and other pieces of software that would do our bidding less conspicuously, likely even without the knowledge of the machine's owner.

3. The project ran for a relatively brief time: about three months. During this time, our effort gained in participating hosts and computing power. If we required more time, the number of participating hosts would have been greater. At some point, we would certainly have "topped-out", but that point was nowhere in sight at the time we found the key.

Often, when performing risk analysis, one will consider a threat model in terms of varying degrees of an attacker's resources ("rogue individual", "moderate", "well-equipped", etc.)

The dropping cost of computing technology and the easy access to large numbers of machines tends to blur the distinctions among the classifications. Now, individuals and small groups can muster the resources equivilant to several large organizations. Data that was once considered to be be vulnerable only to an attack by a "large, well-equipped organization" may now be vulnerable to a just few people with friends on the Internet and no budget. This ability may especially affect policies that have assumed a feasable attack on DES would require an investment in specialized hardware.

## 3.3 Inefficiency of Software Key Searches

The ease of development and deployment of a software-based system has its tradeoff in the amount of computational power necessary to accomplish a given task. Hardware implementations are always much faster and more efficient than their software counterparts.

Rocke Verser estimates [9] that a 10,000 cell FPGA should be able to process nearly 100 million keys per second.

Michael Wiener presented a design for a DES key search machine [10]. A \$1,000,000 version of the machine would be capable of finding DES keys in 3.5 hours, on average. A late-1997 version of this machine [11] would be capable of finding DES keys in 35 minutes, on average. A \$10,000 version of this machine would be capable of finding DES keys in 2.5 days, on average.

A dedicated, funded attacker is going to use more efficient methods of key search than idle cycles of general purpose computers. However, a hardware-based attack requires a substantial investment in the hardware. Even though a software-based attack like DESCHALL is inefficient in comparasion, we were able to take advantage of the huge numbers of general-purpose computers already deployed, as essentially no cost.

## 3.4 Scalability

DESCHALL's relatively simple architecture was able to accomplish a tremendous amount of work. On our peak day, we sustained a search rate of just under seven billion ($10^9$) keys per second. On that day alone, more than 600 trillion ($10^{12}$) keys were searched. Figure 4 shows our search rate, in keys per day, from mid-March until we found the key in mid-June.

Much of the amount of work accomplished was due to the number of hosts participating. As shown in Figure 5, fewer than 1,000 hosts per day were involved in early April, and nearly 14,000 per day ran the client at peak.

Figure 6 shows that our progress over the course of the project was steady, and that the architecture we had in place was adequate to meet the demands made of it.

Although we had additional keyserver capacity available should a need suddenly arise, it turned out to be unnecessary. Elaborate systems, including such things as hierarchical keyservers, turned out to end up either unimplemented or problematic in starting and maintaining. We found it best to plan to scale the system as high as it could go without requiring additional components, but have those additional components ready, just in case of an unexpected increase in system load.

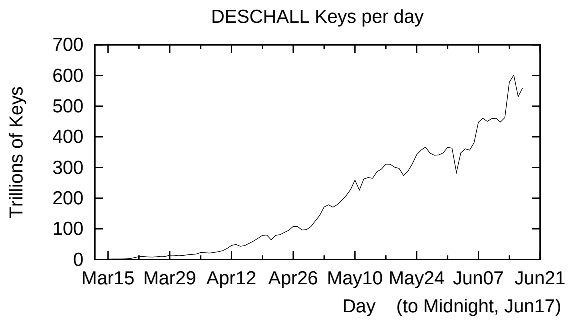

Figure 4: Keys Searched per Day

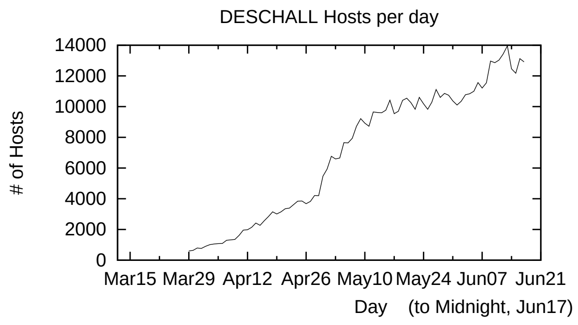

Figure 5: Participating Hosts per Day

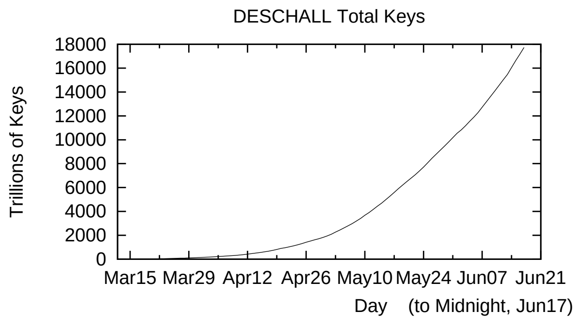

Figure 6: Total Keys Searched

## 4 Conclusions

A number of conclusions can be drawn from DESCHALL:

1. Small key cryptosystems do not provide adequate security against any but the most trivial of attacks.
2. Whereas previous attacks against "live targets"—cryptosystems enjoying "real" use—required the attacker to be relatively well-funded, the kind of power necessary to attack real targets is becoming available to those who are not well-funded, but dedicated enough to make an investment of their time.
3. The potential for performing very large computations without the use of expensive, dedicated hardware, or supercomputers can be seen. Over 7.2 quintillion ($10^{18}$) instructions were executed. Succinctly, massive Internet computing power is here.

## 5 Future Directions

Although DESCHALL was hardly the first distributed Internet computing project, we believe it was the largest such project to date, demonstrating how large amounts of Internet-accessible computing resources can be mobilized relatively easily. It is likely that this form of computing will become more commonplace than it is today, as more people become involved on a regular basis. Some other large-scale distributed computing projects are now underway, including RSA's 64-bit RC5 Challenge [5, 2], RSA's "DES Challenge II" [6, 1], The RSA Factoring Challenge [7], and The Great Internet Mersenne Prime Search [12].

While using this type of effort to do a brute force attack on a 64-bit key would be difficult today, it will certainly be possible in the not so distant future, as more people become involved in these efforts, and as CPU speeds increase according to Moore's Law. At the same time, attacks on smaller key lengths will become easier and easier. Although current distributed projects, like DESCHALL, have not been a direct threat to computer security (i.e., we didn't decipher actual, sensitive data), there is no reason that a DESCHALL-type project could not be assembled by the "underworld" of computer crackers for their own use.

It is our hope that those responsible for deployment and management of cryptosystems take heed to the warnings here, and demand strong cryptography with large keys.

## 6 Acknowledgments

The authors would like to thank RSA Data Security Inc for hosting the Secret Key Challenge, Rocke Verser for starting the DESCHALL project, Karl J. Runge for the excellent statistical data, Darrell Kindred for the tremendous bitslice clients, and everyone involved with DESCHALL and competing efforts for keeping things interesting and fun.

## References

[1] Adam L. Beberg, *et al.* Project Monarch.
`http://www.distributed.net/des/`

[2] Adam L. Beberg, *et al.* Project Bovine.
`http://www.distributed.net/rc5/`

[3] Eli Biham. *A Fast New DES Implementation in Software.* CS 0891, Fast Software Encryption 4, 1997.
`http://www.cs.technion.ac.il/users/biham/cgi-binw/tr-get.cgi/1997/CS/CS0891.ps.gz`

[4] Germano Caronni and Matt Robshaw. How Exhausting is Exhaustive Search? RSA Laboratories' *CryptoBytes*, Vol. 2, No. 3, pages 1–6.

[5] RSA Data Security. *RSA Laboratories Secret-Key Challenge.*
`http://www.rsa.com/rsalabs/97challenge/`

[6] RSA Data Security. *RSA Laboratories DES Challenge II.* `http://www.rsa.com/rsalabs/des2/`

[7] RSA Data Security. *RSA Factoring Challenge.* `http://www.rsa.com/rsalabs/html/factoring.html`

[8] Rocke Verser. DESCHALL Project Home Page. `http://www.frii.com/~rcv/deschall.htm`

[9] Rocke Verser. Email to Matt Curtin. January 17, 1998.

[10] Michael J. Wiener. "Efficient DES Key Search", presented at the Rump session of Crypto '93. Reprinted in Practical Cryptography for Data Internetworks, W. Stallings, editor, IEEE Computer Society Press, pp. 31–79 (1996).

[11] Michael J. Wiener. Efficient DES Key Search—An Update. RSA Laboratories' *CryptoBytes*, Vol. 3, No. 2, pages 6–8.

[12] George Woltman. *Great Internet Mersenne Prime Search.* `http://www.mersenne.org/`