
Scan This Book! **Cracking DES**

Secrets of

How federal

Encryption Research,

agencies

Wiretap Politics

subvert

& Chip Design

privacy

EFF
ELECTRONIC FRONTIER FOUNDATION

Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design

by the Electronic Frontier Foundation

Distributed by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472

Printing History:

May 1998: First Edition.

This book is printed on acid-free paper with 85% recycled content, 15% post-consumer waste O'Reilly & Associates is committed to using paper with the highest recycled content available consistent with high quality.

ISBN: 1-56592-520-3

Table of Contents

Preface

In privacy and computer security, real information is too hard to find. Most people don 't know what's really going on, and many people who do know aren't telling.

This book was written to reveal a hidden truth. The standard way that the US Government recommends that we make information secure and private, the "Data Encryption Standard" or DES, does not actually make that information secure or private. The government knows fairly simple ways to reveal the hidden information (called "cracking" or "breaking" DES).

Many scientists and engineers have known or suspected this for years. The ones who know exactly what the government is doing have been unable to tell the public, fearing prosecution for revealing "classified" information. Those who are only guessing have been reluctant to publish their guesses, for fear that they have guessed wrong.

This book describes a machine which we actually built to crack DES. The machine exists, and its existence can easily be verified. You can buy one yourself, in the United States; or can build one yourself if you desire. The machine was designed and built in the private sector, so it is not classified. We have donated our design to the public domain, so it is not proprietary. There is no longer any question that it can be built or has been built. We have published its details so that other scientists and engineers can review, reproduce, and build on our work. There can be no more doubt. **DES is not secure.**

Chapters

The first section of the book describes the Electronic Frontier Foundation's research project to build a machine to crack DES. The next section provides full technical details on the machine that we designed: for review, critique, exploration, and further evolution by the cryptographic research community. The final section includes several hard-to-find technical reports on brute force methods of cracking DES.

Technical description

Chapter 1, *Overview*, introduces our project and gives the basic architecture of the Electronic Frontier Foundation's DES-cracking machine.

Chapter 2, *Design Specification*, by Paul Kocher of Cryptography Research, provides specifications for the machine from a software author's point of view.

Chapter 3, *Hardware Specification*, by Advanced Wireless Technologies, provides specifications for the custom gate array chips, and the boards that carry them, from a hardware designer's point of view.

Technical design details

Chapter 4, *Scanning the Source Code*, explains how you can feed this book through an optical scanner and regenerate the exact source code needed to build the software and the specialized gate array chip that we designed.

Chapter 5, *Software Source Code*, contains a complete listing of the C-language software that runs on a PC and controls the DES-Cracker.

Chapter 6, *Chip Source Code*, contains a complete listing of the chip design language (VHDL) code that specifies how we designed the custom gate array chip.

Chapter 7, *Chip Simulator Source Code*, contains a complete listing of the C-language software that simulates the operation of the chip, for understanding how the chip works, and for generating test-vectors to make sure that the chips are properly fabricated.

Chapter 8, *Hardware Board Schematics*, provides schematic diagrams of the boards which provide power and a computer interface to the custom chips, as well as information on the layout of the boards and the backplanes that connect them.

Related Research Papers

Chapter 9, *Breaking One Million DES Keys*, by Yvo Desmedt, is a 1987 paper proposing an interesting design for a machine that could search for many DES keys simultaneously.

Chapter 10, *Architectural considerations for cryptanalytic hardware*, by Ian Goldberg and David Wagner, is a 1996 study that explores cracking DES and related ciphers by using field-programmable gate array chips.

Chapter 11, *Efficient DES Key Search - An Update*, by Michael J. Wiener, revises for 1998 the technology estimates from his seminal 1993 paper, which was the first to include full schematic diagrams of a custom chip designed to crack DES.

Chapter 12, *About the Authors*, describes the foundation and the companies which collaborated to build this project.

1
Overview

In This chapter:

Politics of Decryption

We began the Electronic Frontier Foundation's DES Cracker project because of our interest in the politics of decryption.* The vulnerability of widely used encryption standards like DES is important for the public to understand.

A "DES Cracker" is a machine that can read information encrypted with the Data Encryption Standard (DES), by finding the key that was used to encrypt it. "Cracking DES" is a name for this search process. It is most simply done by trying every possible key until the right one is found, a tedious process called "brute-force search".

If DES-encrypted information can easily be decrypted by those who are not intended to see it, the privacy and security of our infrastructures that use DES are at risk. Many political, social, and technological decisions depend on just how hard it is to crack DES.

We noticed an increasing number of situations in which highly talented and respected people from the U.S. Government were making statements about how long it takes to crack DES. In all cases, these statements were at odds with our own estimates and those of the cryptographic research community. A less polite way to say it is that these government officials were lying, incompetent, or both. They were stating that cracking DES is much more expensive and time-consuming than we believed it to be. A very credible research paper had predicted that a

* DES, the Data Encryption Standard, encrypts a confidential message into scrambled output under the control of a secret key. The input message is also known as "plaintext", and the resulting output as "ciphertext". The idea is that only recipients who know the secret key can decrypt the ciphertext to obtain the original message. DES uses a 56-bit key, so there are 2^{56} possible keys.

machine could be built for $1.5 million, including development costs, that would crack DES in 3-1/2 hours. Yet we were hearing estimates of thousands of computers and weeks to years to crack a single message.

On Thursday, June 26, 1997 the U.S. House of Representatives' Committee on International Relations heard closed, classified testimony on encryption policy issues. The Committee was considering a bill to eliminate export controls on cryptography. After hearing this testimony, the Committee gutted the bill and inserted a substitute intended to have the opposite effect. A month later, a censored transcript of the hearing was provided; see http://jya.com/hir-hear.htm. Here are excerpts:

Statement of Louis J. Freeh, Director, Federal Bureau of Investigation

. . . And we do not have the computers, we do not have the technology to get either real-time access to that information or any kind of timely access.

If we hooked together thousands of computers and worked together over 4 months we might, as was recently demonstrated decrypt one message bit. That is not going to make a difference in a kidnapping case, it is not going to make a difference in a national security case. We don't have the technology or the brute force capability to get to this information.

Statement of William P. Crowell, Deputy Director, National Security Agency

. . . I would go further and say there have been people who have said that Louis Freeh's organization should just get smarter technically, and if they were just smarter technically, they would be able to break all of this stuff. I would like to leave you with just one set of statistics, and then I think I am going to close with just a few comments on the bill itself.

There is no brute force solution for law enforcement. [blacked out ---] A group of students -- not students -- the Internet gang last week broke a single message using 56-bit DES. It took 78,000 computers 96 days to break one message, and the headline was, DES has weak encryption.

He doesn't consider that very weak. If that had been 64-bit encryption, which is available for export today, and is available freely for domestic use, that same effort would have taken 7,000 years. And if it had been 128-bit cryptography, which is what PGP is, pretty good privacy, it would have taken 8.6 trillion times the age of the universe.

Comments made later in the hearing

Chairman Gilman. Would you need added manpower resource and equipment if there is a need to decrypt? And would that add to your already difficult case of language translation in many of your wiretaps?

Director Freeh. We would certainly need those resources, but I think more importantly is the point that was made here. Contrary to the National Research Council recommendation that the FBI buy more computers and Bill Gates' suggestion to me that we upgrade our research and development [blacked out-----------------------------] American industry cannot do it, and that is decrypt real time encryption over a very minimal level of robustness. [blacked out---------] If you gave me $3 million to buy a Cray computer, it would take me how many years to do one message bit?

Mr. Crowell. 64 bits, 7,000 years.

Director Freeh. I don't have that time in a kidnapping case. It would kill us.

On March 17, 1998, Robert S. Litt, Principal Associate Deputy Attorney General, testified to the U.S. Senate Judiciary Committee, Subcommittee on the Constitution, Federalism, and Property. The subject of the hearing was "Privacy in a Digital Age: Encryption and Mandatory Access". Mr. Litt's whole statement is available at http://www.computerprivacy.org/archive/03171998-4.shtml. The part relevant to DES cracking is:

Some people have suggested that this is a mere resource problem for law enforcement. They believe that law enforcement agencies should simply focus their resources on cracking strong encryption codes, using high-speed computers to try every possible key when we need lawful access to the plaintext of data or communications that is evidence of a crime. But that idea is simply unworkable, because this kind of brute force decryption takes too long to be useful to protect the public safety. For example, decrypting one single message that had been encrypted with a 56-bit key took 14,000 Pentium-level computers over four months; obviously, these kinds of resources are not available to the FBI, let alone the Jefferson City Police Department.

What's Wrong With Their Statements?

Some of the testimony quoted may have been literally true; nevertheless, it is deceptive. All of the time estimates presented by Administration officials were based on use of general-purpose computers to do the job. But that's fundamentally the wrong way to do it, and they know it.

A ordinary computer is ill-suited for use as a DES Cracker. In the first place, the design of DES is such that it is inherently very slow in software, but fast in hardware. Second, current computers do very little in parallel; the designers don't know exactly what instructions will be executed, and must allow for all combinations.

The right way to crack DES is with special-purpose hardware. A custom-designed chip, even with a slow clock, can easily outperform even the fastest general-purpose computer. Besides, you can get many such chips on a single board, rather than the one or two on a typical computer's motherboard.

There are practical limits to the key sizes which can be cracked by brute-force searching, but since NSA deliberately limited the key size of DES to 56 bits, back in the 1970's when it was designed, DES is crackable by brute force. Today's technology might not be able to crack other ciphers with 64-bit or 128-bit keys--or it might. Nobody will know until they have tried, and published the details for scientific scrutiny. Most such ciphers have very different internal structure than DES, and it may be possible to eliminate large numbers of possible keys by taking advantage of the structure of the cipher. Some senior cryptographers estimated what key sizes were needed for safety in a 1996 paper;* they suggest that to protect against brute force cracking, today's keys should have a minimum of 75 bits, and to protect information for twenty years, a minimum of 90 bits.

The cost of brute-force searching also overstates the cost of recovering encrypted text in the real world. A key report on the real impact of encryption on law enforcement[+] reveals that there are no cases in which a lack of police access to encrypted files resulted in a suspected criminal going free. In most cases the plaintext was recovered by other means, such as asking the suspect for the key, or finding another copy of the information on the disk. Even when brute force is the method of choice, keys are seldom truly random, and can be searched in the most likely order.

Export Controls and DES

The U.S. Government currently restricts the ability of companies, individuals, and researchers to export hardware or software that includes the use of DES for confidentiality. These "export controls" have been a severe impediment to the development of security and privacy for networked computers, cellular phones, and other popular communications devices. The use of encryption algorithms stronger than DES is also restricted.

In December 1996, the government formally offered exporters the ability to incorporate DES, but nothing stronger, into their products. The catch is that these companies would have to sign an agreement with the government, obligating them to

---

* Minimal Key Lengths For Symmetric Ciphers To Provide Adequate Commercial Security: A Report By An Ad Hoc Group Of Cryptographers And Computer Scientists. Matt Blaze, Whitfield Diffie, Ronald L. Rivest, Bruce Schneier, Tsutomu Shimomura, Eric Thompson, Michael Wiener, January 1996. Available at http://www.bsa.org/policy/encryption/index.html.

+ Encryption and Evolving Technologies: Tools of Organized Crime and Terrorism, by Dorothy E. Denning and William E. Baugh, Jr. National Strategy Information Center, 1997. ISSN 1093-7269.

---

install "key recovery" into their products within two years. Key recovery technology provides a way for the government to decrypt messages at will, by offering the government a copy of the key used in each message, in a way that the product's user cannot circumvent or control. In short, the government's offer was: collude with us to violate your customers' privacy, or we won't let you export any kind of secure products.

At the same time, the FBI was let into the group that reviews each individual company's application to export a cryptographic product. All reports indicate that the FBI is making good on the threat, by objecting to the export of all kinds of products that pose no threat at all to the national security (having been exportable in previous years before the FBI gained a voice). The FBI appears to think that by making itself hated and feared, it will encourage companies to follow orders. Instead it is encouraging companies to overturn the regulatory scheme that lets the FBI abuse the power to control exports. Industry started a major lobbying group called Americans for Computer Privacy (http://www.computerprivacy.org/), which is attempting to change the laws to completely decontrol nonmilitary encryption exports.

Some dozens of companies to signed up for key recovery, though it is unclear how many actually plan to follow through on their promise to deploy the technology. You will not find many of these companies trumpeting key recovery in their product advertisements. Users are wary of it since they know it means compromised security. If customers won't buy such products, companies know it makes no sense to develop them.

The best course for companies is probably to develop products that provide actual security, in some jurisdiction in the world which does not restrict their export. Some companies are doing so. The government's "compromise" offer discourages hesitant companies from taking this step, by providing a more moderate and conciliatory step that they can take instead. Companies that go to the effort to build overseas cryptographic expertise all use stronger technology than DES, as a selling point and to guard against early obsolescence. If those companies can be convinced to stay in the US, play the government's key-recovery game, and stick with DES, the government continues to win, and the privacy of the public continues to lose.

The success or failure of the government's carrot-and-stick approach depends on keeping industry and the public misled about DES's security. If DES-based products were perceived as insecure, there would be little reason for companies to sign away their customers' privacy birthrights in return for a mess of DES pottage. If DES-based products are perceived as secure, but the government actually knows that the products are insecure, then the government gets concessions from companies, without impacting its ability to intercept communications. Keeping the public ignorant gives the government the best of both worlds.

Political Motivations and EFF's Response

We speculate that government officials are deliberately misleading the public about the strength of DES encryption:

- To encourage the public to continue using DES, so their agencies can eavesdrop on the public.

- To prevent the widespread adoption of stronger standards than DES, which the government would have more trouble decrypting.

- To offer DES exportability as a bargaining-chip, which actually costs the government little, but is perceived to be valuable.

- To encourage policy-makers such as Congressmen or the President to impose drastic measures such as key recovery, in the belief that law enforcement has a major encrypted-data problem and no practical way to crack codes.

As advocates on cryptography policy, we found ourselves in a hard situation. It appeared that highly credible people were either deliberately lying to Congress and to the public in order to advance their own harmful agendas, or were advocating serious infringement of civil liberties based on their own ignorance of the underlying issues. Most troubling is the possibility that they were lying. Perhaps these government executives merely saw themselves as shielding valuable classified efforts from disclosure. As advocates of good government, we do not see that classifying a program is any justification for an official to perjure themselves when testifying about it. (Declining to state an opinion is one thing; making untruthful statements as if they were facts is quite another.)

The National Research Council studied encryption issues and published a very complete 1996 report.* The most interesting conclusion of their report was that "the debate over national cryptography policy can be carried out in a reasonable manner on an unclassified basis". This presumes good faith on the part of the agencies who hide behind classified curtains, though. If it turns out that their public statements are manipulative falsehoods, an honest and reasonable public debate must necessarily exclude them, as dishonest and unreasonable participants.

In the alternative, if poor policy decisions are being made based on the ignorance or incompetence of senior government officials, the role of honest advocates should be to inform the debate.

---

* Cryptography's Role In Securing the Information Society, Kenneth W. Dam and Herbert S. Lin, editors. National Academy Press, Washington, DC, 1996.

In response to these concerns, EFF began a research program. Our research results prove that DES can be cracked quickly on a low budget. This proves that these officials were either lying or incompetent. The book you are holding documents the research, and allows it to be validated by other scientists.

Goals

The goal of EFF's DES Cracker research project is to determine just how cheap or expensive it is to build a machine that cracks DES usefully.

Technically, we were also interested in exploring good designs for plaintext recognizers. These are circuits that can notice when the result of decryption is likely enough to be correct that specialized software--or a human--should look at it. Little research has been published on them,* yet they are a vital part of any efficient system for cryptanalysis.

Merely doing the research would let EFF learn the truth about the expense of cracking DES. But only publishing the research and demonstrating the machine would educate the public on the truth about the strength of DES. Press releases and even technical papers would not suffice; the appearance of schematics for a million-dollar DES Cracker in Michael Wiener's excellent 1993 paper should have been enough. But people still deploy DES, and Congressmen blindly accept the assurances of high officials about its strength.

There are many people who will not believe a truth until they can see it with their own eyes. Showing them a physical machine that can crack DES in a few days is the only way to convince some people that they really cannot trust their security to DES.

Another set of people might not believe our claims unless several other teams have reproduced them. (This is a basic part of the scientific method.) And many people will naturally be interested in how such a box works, and how it was built for only about $200,000. This book was written for such people. It contains the complete specifications and design documents for the DES Cracker, as well as circuit diagrams for its boards, and complete listings of its software and its gate array design. The full publication of our design should enable other teams to rapidly reproduce, validate, and improve on our design.

---

* But see: David A. Wagner and Steven M. Bellovin, "A Programmable Plaintext Recognizer," 1994. Available at http://www.research.att.com/~smb/papers/recog.ps or recog.pdf.

History of DES Cracking

DES Crackers have been mentioned in the scientific and popular literature since the 1970's. Whitfield Diffie's Foreword describes several of them. The most recent detailed description was in a paper by Michael Wiener of Bell Northern Research in 1993. Wiener's paper included a detailed hardware design of a DES Cracker built with custom chips. The chips were to be built into boards, and the boards into mechanical "frames" like those of telephone central office switches. A completed design would have cost about a million dollars and would determine a DES key from known plaintext and known ciphertext in an average of 3-1/2 hours (7 hours in the worst case).

Mr. Wiener updated his conclusions in 1998, adjusting for five years of technological change. His update paper is included in this book, thanks to the courtesy of RSA Data Security, which originally published his update.

Ian Goldberg and David Wagner of the University of California at Berkeley took a different approach. Their design used a "field programmable gate array" (FPGA), which is a chip that can be reprogrammed after manufacturing into a variety of different circuits.

FPGA chips are slower than the custom chips used in the Wiener design, but can be bought quickly in small quantities, without a large initial investment in design. Rather than spend a big chunk of a million dollars to design a big machine, these researchers bought one or two general purpose chips and programmed them to be a slow DES Cracker. This let them quickly measure how many slow chips they would need to pile up to make a practical DES Cracker. Their paper is also included in this book.

EFF's DES Cracker Project

The Electronic Frontier Foundation began its investigation into DES Cracking in 1997. The original plan was to see if a DES Cracker could be built out of a machine containing a large number of FPGA's.

Large machines built out of FPGAs exist in the commercial market for use in simulating large new chip designs before the chip is built. A collection of thousands of relatively incapable FPGA chips can be put together to simulate one very capable custom chip, although at 1/10th or 1/100th of the speed that the eventual custom chip would run at. This capability is used by chip designers to work the "bugs" out of their chip before committing to the expensive and time-consuming step of fabricating physical chips from their design.

EFF never got access to such a chip simulator. Instead, our investigations led us to Paul Kocher of Cryptography Research. Paul had previously worked with a team of hardware designers who knew how to build custom gate array chips cheaply, in batches of a few thousand chips at a time.

Paul and EFF met with the chip designers at Advanced Wireless Technologies, and determined that a workable DES Cracker could be built on a budget of about $200,000. The resulting machine would take less than a week, on average, to determine the key from a single 8-byte sample of known plaintext and ciphertext. Moreover, it would determine the key from a 16-byte sample of ciphertext in almost the same amount of time, if the statistical characteristics of the plaintext were known or guessable. For example, if the plaintext was known to be an electronic mail message, it could find all keys that produce plaintext containing nothing but letters, numbers, and punctuation. This makes the machine much more usable for solving real-world decryption problems.

There is nothing revolutionary in our DES Cracker. It uses ordinary ideas about how to crack DES that have been floating around in the cryptographic research community for many years. The only difference is that we actually built it, instead of just writing papers about it. Very similar machines could have been built last year, or the year before, or five or ten years ago; they would have just been slower or more expensive.

Architecture

The design of the EFF DES Cracker is simple in concept. It consists of an ordinary personal computer connected with a large array of custom chips. Software in the personal computer instructs the custom chips to begin searching, and interacts with the user. The chips run without further help from the software until they find a potentially interesting key, or need to be directed to search a new part of the key space. The software periodically polls the chips to find any potentially interesting keys that they have turned up.

The hardware's job isn't to find the answer. but rather to eliminate most of the answers that are incorrect. Software is then fast enough to search the remaining potentially-correct keys, winnowing the false positives" from the real answer. The strength of the machine is that it replicates a simple but useful search circuit thousands of times, allowing the software to find the answer by searching only a tiny fraction of the key space.

As long as there is a small bit of software to coordinate the effort, the problem of searching for a DES key is "highly parallelizable". This means the problem can be usefully solved by many machines working in parallel, simultaneously. For example, a single DES-Cracker chip could find a key by searching for many years. A thousand DES-Cracker chips can solve the same problem in one thousandth of the time. A million DES-Cracker chips could theoretically solve the same problem in about a millionth of the time, though the overhead of starting each chip would become visible in the time required. The actual machine we built contains 1536 chips.

When conducting a brute-force search, the obvious thing to do is to try every possible key, but there are some subtleties. You can try the keys in any order. If you think the key isn't randomly selected, start with likely ones. When you finally find the right key, you can stop; you don't have to try all the rest of the keys. You might find it in the first million tries; you might find it in the last million tries. On average, you find it halfway through (after trying half the keys). As a result, the timings for brute-force searches are generally given as the average time to find a key. The maximum time is double the average time.

Search units

The search unit is the heart of the EFF DES Cracker; it contains thousands of them.

A search unit is a small piece of hardware that takes a key and two 64-bit blocks of ciphertext. It decrypts a block of ciphertext with the key, and checks to see if the resulting block of plaintext is "interesting". If not, it adds 1 to the key and repeats, searching its way through the key space.

If the first decryption produces an "interesting" result, the same key is used to decrypt the second block of ciphertext. If both are interesting, the search unit stops and tells the software that it has found an interesting key. If the second block's decryption is uninteresting, the search unit adds one to the key and goes on searching the key space.

When a search unit stops after finding an interesting result, software on the host computer must examine the result, and determine whether it's the real answer, or just a "false positive". A false positive is a plaintext that looked interesting to the hardware, but which actually isn't a solution to the problem. The hardware is designed to produce some proportion of false positives along with the real solution. (The job of the hardware isn't to find the answer, but to eliminate the vast majority of the non-answers.) As long as the false positives don't occur so rapidly that they overwhelm the software s ability to check and reject them, they don't hurt, and they simplify the hardware and allow it to be more general-purpose. For the kinds of problems that we're trying to solve, the hardware is designed to waste less than 1% of the search time on false positives.

Recognizing interesting plaintext

What defines an interesting result? If we already know the plaintext, and are just looking for the key, an interesting result would be if the plaintext from this key matches our known block of plaintext. If we don't know the plaintext, perhaps the guess that it's all composed of letters, digits, and punctuation defines "interesting". The test has to be simple yet flexible. We ended up with one that's simple for the hardware, but a bit more complicated for the software.

Each result contains eight 8-bit bytes. First, the search unit looks at each byte of the result. Such a byte can have any one of 256 values. The search unit is set up with a table that defines which of these 256 byte values are "interesting" and which are uninteresting. For example, if the plaintext is known to be all numeric, the software sets up the table so that the ten digits (O to 9) are interesting, and all other potential values are uninteresting.

The result of decrypting with the wrong key will look pretty close to random. So the chance of having a single byte look "interesting" will be based on what fraction of the 256 values are defined to be "interesting". If, say, 69 characters are interesting (A-Z, a-z, 0-9, space, and a few punctuation characters), then the chance of a random byte appearing to be interesting is 69/256 or about 1/4. These don't look like very good odds; the chip would be stopping on one out of every four keys, to tell the software about "interesting" but wrong keys.

But the "interest" test is repeated on each byte in the result. If the chance of having a wrong key's byte appear interesting is 1/4, then the chance of two bytes appearing interesting is 1/4 of 1/4, or 1/16th. For three bytes, 1/4th of 1/4th of 1/4th, or 1/64th. By the time the chip examines all 8 bytes of a result, it only makes a mistake on 1/65536th of the keys ($l/4^8$ keys).

That seems like a pretty small number, but when you're searching through 72,057,594,037,927,936 keys (2^{56} keys, or 72 quadrillion keys), you need all the help you can get. Even having the software examine 1/65536th of the possible keys would require looking at 1,099,511,627,776 keys (2^{40} or about a trillion keys). So the chip provides a bit more help.

This help comes from that second block of ciphertext. If every byte of a result looks interesting when the first block of ciphertext is decrypted, the chip goes back around and decrypts the second block of ciphertext with the same key. This divides the "error rate" by another factor of 65536, leaving the software with only 16,777,216 (2^{24} or about sixteen million) keys to look at. Software on modern computers is capable of handling this in a reasonable amount of time.

(If we only know one block of ciphertext, we just give the chip two copies of the same ciphertext. It will test both copies, and eventually tell us that the block is

interesting. The amount of time it spends checking this "second block" is always a tiny fraction of the total search time.)

In the plaintext recognizer there are also 8 bits that lets us specify which bytes of a plaintext are interesting to examine. For example, if we know or suspect the contents of the first six bytes of a plaintext value, but don't know anything about the last two bytes, we can search for keys which match in just those six bytes.

Known plaintext

The chips will have many fewer "false positives" if the plaintext of the message is known, instead of just knowing its general characteristics. In that case, only a small number of byte values will be "interesting". If the plaintext has no repeated byte values, only eight byte values will be interesting, instead of 69 as above.

For example, if the plaintext block is "hello th", then only the six byte values "h", "e", "l", "o", space, and "t" are interesting. If a plaintext contains only these bytes, it is interesting. We'll get some "false positives" since many plaintexts like "tholo tt" would appear "interesting" even though they don't match exactly.

Using this definition of "interesting", a byte resulting from a wrong key will look interesting only about 8/256ths of the time, or 1/32nd of the time. All eight bytes resulting from a wrong key will look interesting only 1/32nd to the eighth power (1/32nd of 1/32nd of 1/32nd of 1/32nd of 1/32nd of 1/32nd of 1/32nd of 1/32nd) of the time, or 1/1,099,511,627,776th of the time ($1/2^{40}$ of the time). In other words, a search unit can try an average of a trillion keys before reporting that a wrong key looks interesting. This lets it search for a long time without slowing down or bothering the software.

Speed

Once you get it going, a search unit can do one decryption in 16 clock cycles. The chips we have built can run with a clock of 40 Mhz (40 million cycles per second). Dividing 16 into 40 million shows that each search unit can try about 2.5 million keys per second.

In building the search units, we discovered that we could make them run faster if we used simpler circuitry for adding 1 to a key. Rather than being able to count from a key of O all the way up to a key of all ones, we limited the adder so that it can only count the bottom 32 bits of the key. The top 24 bits always remain the same. At a rate of 2.5 million keys per second, it takes a search unit 1717 seconds (about half an hour) to search all the possible keys that have the same top 24 bits. At the end of half an hour, the software has to stop the chip, reload it with a new value in the top 24 bits, and start it going again.

Feedback Modes

The chip can also decrypt ciphertext that was encrypted in "Cipher Block Chaining" mode. In this mode, the ciphertext of each block is exclusive-OR'd into the plaintext of the next block before it is encrypted. (An "initialization vector" is exclusive-OR'd into the first block of plaintext.) The search unit knows how to exclusive-OR out an Initialization Vector (IV) after decrypting the first cyphertext, and to exclusive-OR out the first cyphertext after decrypting the second one. The software specifies the IV at the same time it provides the cyphertext values.

Blaze Challenge

In June, 1997 Matt Blaze, a cryptography researcher at AT&T, proposed a different sort of cryptographic challenge. He wanted a challenge that not even the proponent knew how to solve, without either doing a massive search of the key-space, or somehow cryptanalyzing the structure of DES.

His challenge is merely to find a key such that a ciphertext block of the form XXXXXXXX decrypts to a plaintext block of the form YYYYYYYY, where X and Y are any fixed 8-bit value that is repeated across each of the eight bytes of the block.

We added a small amount of hardware to the search units to help with solving this challenge. There is an option to exclusive-OR the right half of the plaintext into the left half, before looking to see if the plaintext is "interesting". For plaintexts of the form YYYYYYYY, this will result in a left half of all zeros. We can then set up the plaintext recognizer so it only looks at the left half, and only thinks zeroes are interesting. This will produce a large number of false positives (any plaintext where the left and right halves are equal, like ABCDABCD), but software can screen them out with only about a 1% performance loss.

Structure Of The Machine

Now that you know how a single search unit works, let's put them together into the whole machine.

Each search unit fits inside a custom chip. In fact, 24 search units fit inside a single chip. All the search units inside a chip share the same ciphertext blocks. initialization vector, and the same plaintext-recognizer table of "interesting" result values. Each search unit has its own key, and each can be stopped and started independently.

The chip provides a simple interface on its wires. There are a few signals that say whether any of the search units are stopped, some address and data wires so that the software can read and write to the search units, and wires for electrical power and grounding.

Since each search unit tries 2.5 million keys per second, a chip with 24 search units will try 60 million keys per second. But there are a lot of keys to look at. For a single chip, it would take 6,950 days (about 19 years) to find the average key, or 38 years to search the entire key space. Since we don't want to wait that long, we use more than one chip.

Each chip is mounted onto a large circuit board that contains 64 chips, along with a small bit of interface circuitry. The board blinks a light whenever the software is talking to that board. 64 other lights show when some search unit in each chip has stopped. In normal operation the software will talk to the board every few seconds, to check up on the chips. The chips should only stop every once in a while, and should be quickly restarted by the software.

The boards are designed to the mechanical specifications of "9U" VMEbus boards (about 15" by 15"). VMEbus is an industrial standard for computer boards, which was popular in the 1980s. We used the VMEbus form factor because it was easy to buy equipment that such boards plug into; we don't actually use the VMEbus electrical specifications.

9U VMEbus boards are much larger than the average interface card that plugs into a generic PC, so a lot more chips can be put onto them. Also, 9U VMEbus boards are designed to supply a lot of power, and our DES Cracker chips need it.

Since each chip searches 60 million keys per second, a board containing 64 chips will search 3.8 billion keys per second. Searching half the key space would take the board about 109 days. Since we don't want to wait that long either, we use more than one board.

The boards are mounted into chassis, also called "card cages". In the current design, these chassis are recycled Sun workstation packages from about 1990. Sun Microsystems built a large number of system.s that used the large 9U VMEbus boards, and provide excellent power and cooling for the boards. The Sun-4/470 chassis provides twelve slots for VMEbus boards, and can easily be modified to handle our requirements. Subsequent models may use other physical packaging.

Each chassis has a connector for a pair of "ribbon cables to connect it to the next chassis and to the generic PC that runs the software. The last chassis will contain a "terminator", rather than a connection to the next chassis, to keep the signals on the ribbon cable from getting distorted when they reach the end of the line.

Since each board searches 3.8 billion keys per second, a chassis containing 12 boards will search 46 billion keys per second. At that rate, searching half the key space takes about 9 days. One chassis full of boards is about 25% faster than the entire worldwide network of machines that solved the RSA "DES-II" challenge in February 1998, which was testing about 34 billion keys per second at its peak.

Since an informal design goal for our initial DES Cracker was to crack an average DES key in less than a week, we need more than 12 boards. To give ourselves a comfortable margin, we are using 24 boards, which we can fit into two chassis. They will search 92 billion keys per second, covering half the key space in about 4.5 days. If the chips consume too much power or produce too much heat for two chassis to handle,* we can spread the 24 boards across three chassis.

Table 1-1: Summary of DES Cracker performance

Device	How Many In Next Device	Keys/Sec	Days/avg search
Search Unit	24	2,500,000	166,800
Chip	64	60,000,000	6,950
Board	12	3,840,000,000	109
Chassis	2	46,080,000,000	9.05
EFF DES Cracker		92,160,000,000	4.524

We designed the search unit once. Then we got a speedup factor of more than 36,000 to 1 just by replicating it 24 times in each chip and making 1500 chips. This is what we meant by "highly parallelizable"

Budget

The whole project was budgeted at about US$210,000. Of this, $80,000 is for the labor of designing, integrating, and testing the DES Cracker. The other $130,000 is for materials, including chips, boards, all other components on the boards, card cages, power supplies, cooling, and a PC.

The software for controlling the DES Cracker was written separately, as a volunteer project. It took two or three weeks of work.

The entire project was completed within about eighteen months. Much of that time was used for preliminary research, before deciding to use a custom chip rather than FPGA's. The contract to build custom chips was signed in September, 1997, about eight months into the project. The team contained less than ten people, none of whom worked full-time on the project. They include a project manager, software designer, programmer, chip designer, board designer, hardware technicians, and hardware managers.

* At publication time, we have tested individual chips but have yet not built the full machine If the chips' power consumption or heat production is excessive in a machine containing 1500 chips, we also have the option to reduce the chips' clock rate from 40 MHz down to, say, 30 MHz. This would significantly reduce the power and heat problems, at a cost of 33% more time per search (6 days on average).

We could have reduced the per-chip cost, or increased the chip density or search speed, had we been willing to spend more money on design. A more complex design could also have been flexible enough to crack other encryption algorithms. The real point is that for a budget that any government, most companies, and tens of thousands of individuals could afford, we built a usable DES Cracking machine. The publication of our design will probably in itself reduce the design cost of future machines, and the advance of semiconductor technology also makes this cost likely to drop. In five years some teenager may well build her own DES Cracker as a high school science fair project.

Who Else Is Cracking DES?

If a civil liberties group can build a DES Cracker for $200,000, it's pretty likely that governments can do the same thing for under a million dollars. (That's a joke.) Given the budget and mission of the US National Security Agency, they must have started building DES Crackers many years ago. We would guess that they are now on their fourth or fifth generation of such devices. They are probably using chips that are much faster than the ones we used; modern processor chips can run at more than 300 Mhz, eight times as fast as our 40 Mhz chips. They probably have small "field" units that fit into a suitcase and crack DES in well under a day; as well as massive central units buried under Ft. Meade, that find the average DES key in seconds, or find thousands of DES keys in parallel, examining thousands of independent intercepted messages.

Our design would scale up to finding a DES key in about half an hour, if you used 333,000 chips on more than 5,200 boards. The boards would probably require about 200 parallel port cards to communicate with them; an IBM-compatible PC could probably drive four such cards, thus requiring about 50 PC's too. The software required would be pretty simple: the hard part would be the logistics of physical arrangement and repair. This is about 200 times as much hardware as the project we built. A ridiculously high upper bound on the price of such a system would be 200 times the current project price, or S40 million

Of course, if we were going to build a system to crack DE.S in half an hour or less, using a third of a million chips, it would be better to go back to the drawing board and design from scratch. We'd use more modern chip fabrication processes; a higher-volume customer can demand this. We d spend more on the initial design and the software, to produce a much cheaper and simpler total system, perhaps allowing boards full of denser, faster, lower-voltage chips to use a small onboard processor and plug directly into an Ethernet. We'd work hard to reduce the cost of each chip, since there would be so many of them. We'd think about how to crack multiple DES keys simultaneously.

It would be safe to assume that any large country has DES Cracking machines. After the publication of this book wakes them up, probably more small countries and some criminal organizations will make or buy a few DES Crackers. That was not the intent of the book; the intent was to inform and warn the targets of this surveillance, the builders of equipment, and the policy makers who grapple with encryption issues.

What To Do If You Depend On DES

Don't design anything else that depends on single DES.

Take systems out of service that use permanently fixed single-DES keys, or superencrypt the traffic at a higher level. Superencryption requires special care, though, to avoid providing any predictable headers that can be used to crack the outer DES encryption.

Start changing your software and/or hardware to use a stronger algorithm than DES.

Three-key Triple-DES is an obvious choice, since it uses the same block size and can possibly use the same hardware; it just uses three keys and runs DES three times (encrypting each block with the first key, decrypting it with the second, then encrypting it with the third). The strength of Triple-DES is not known with any certainty, but it is certainly no weaker than single DES, and is probably substantially stronger. Beware of "mixed up" variants or modes of Triple-DES; research by Eli Biham* and David Wagner+ shows that they are significantly weaker than the straightforward Triple-DES, and may be even weaker than single-DES. Use three copies of DES in Electronic Code Book (ECB) mode as a basic primitive. You can then build a mode such as Cipher Feedback mode using the primitive ECB 3DES.

The US Government is tardily going through a formal process to replace the DES. This effort, called the Advanced Encryption Standard, will take several years to decide on a final algorithm, and more years for it to be proven out in actual use, and carefully scrutinized by public cryptanalysts for hidden weaknesses. If you are designing products to appear five to ten years from now, the AES might be a good source of an encryption algorithm for you.

The reason that the AES is tardy is because the NSA is believed to have blocked previous attempts to begin the process over the last decade. In recent years NSA

* "Cryptanalysis of Triple-Modes of Operation", Eli Biham, Technion Computer Science Department Technical Report CS0885, 1996.

+"Cryptanalysis of some Recently Proposed Multiple Modes of Operation", David Wagner, University of California at Berkeley, http://www.cs.berkeley.edu/~daw/multmode-fse98.ps. Presented at the 1998 Fast Software Encryption workshop.

has tried, without success, to get the technical community to use classified, NSA-designed encryption algorithms such as Skipjack, without letting the users subject these algorithms to public scrutiny. Only after this effort failed did they permit the National Institute of Standards and Technology to begin the AES standardization process.

Conclusion

The Data Encryption Standard has served the public pretty well since 1975. But it was designed in an era when computation cost real money, when massive computers hunkered on special raised flooring in air-conditioned inner sanctums. In an era when you can carry a supercomputer in your backpack, and access millions of machines across the Internet, the Data Encryption Standard is obsolete.

The Electronic Frontier Foundation hopes that this book inspires a new level of truth to enter the policy debates on encryption. In order to make wise choices for our society, we must make well-informed choices. Great deference has been paid to the perspective and experience of the

National Security Agency and Federal Bureau of Investigation in these debates. This is particularly remarkable given the lack of any way for policy-makers or the public to check the accuracy of many of their statements.* (The public cannot even hear many of their statements, because they are classified as state secrets.) We hope that the crypto policy debate can move forward to a more successful and generally supported policy. Perhaps if these agencies will consider becoming more truthful, or policy-makers will stop believing unverified statements from them, the process can move more rapidly to such a conclusion.

---

* DES cracking is not the only issue on which agency credibility is questionable. For example, the true extent of the law enforcement problem posed by cryptography is another issue on which official dire predictions have been made, while more careful and unbiased studies have shown little or no impact. The validity of the agencies' opinion of the constitutionality of their own regulations is also in doubt, having been rejected two decades ago by the Justice Department, and declared unconstitutional in 1997 by a Federal District Court. The prevalence of illegal wiretapping and communications interception by government employees is also in question; see for example the Los Angeles Times story of April 26, 1998, "Can the L.A. Criminal-Justice System Work Without Trust?"

2
Design for DES Key Search Array

In This chapter:

Cryptography Research
and
Advanced Wireless Technologies, Inc.

On-Chip Registers

Each chip contains the following registers. They are addressed as specified in Figure 2-1.

Ciphertext0 (64 bits = 8 bytes)

The value of the first ciphertext being searched. Ciphertext0 is identical in all search units and is set only once (when the search system is first initialized).

Ciphertext1 (64 bits = 8 bytes)

The value of the second ciphertext being searched. Ciphertext1 is identical in all search units and is set only once (when the search system is first initialized).

PlaintextByteMask (8 bits)

The plaintext byte selector. One-bits in this register indicate plaintext bytes that should be ignored when deciding whether or not the plaintext produced by a particular key is possibly correct. This mask is helpful when only a portion of the plaintext's value is known. For example, if the first S bytes equal a known header but the remaining three are unknown, a PlaintextByteMask of 0x07 would be used.

PlaintextXorMask (64 bits = 8 bytes)

This register is XORed with decryption of ciphertext0. This is normally filled with

Figure 2-1: Register Addressing

Register(s)	Description & Comments
0x00-0x1F	PlaintextVector
0x20-0x27	PlaintextXorMask
0x28-0x2F	Ciphertext0
0x30-0x37	Ciphertext1
0x38	PlaintextByteMask
0x39-0x3E	Unused (reserved)
0x3F	SearchInfo
0x40-0x47	Search unit 0 key counter (0x40-0x46) and SearchStatus (0x47)
0x48-0x4F	Search unit 1 key counter (0x48-0x4E) and SearchStatus (0x4F)
0x50-0x57	Search unit 2 key counter (0x50-0x56) and SearchStatus (0x57)
0x58-0x5F	Search unit 3 key counter (0x58-0x5E) and SearchStatus (0x5F)
0x60-0x67	Search unit 4 key counter (0x60 0x66) and SearchStatus (0x67)
0x68-0x6F	Search unit 5 key counter (0x68 0x6E) and SearchStatus (0x6F)
0x70-0x77	Search unit 6 key counter (0x70-0x76) and SearchStatus (0x77)
0x78-0x7F	Search unit 7 key counter (0x78-0x7E) and SearchStatus (0x7F)
0x80-0x87	Search unit 8 key counter (0x80-0x86) and SearchStatus (0x87)
0x88-0x8F	Search unit 9 key counter (0x88 0x8E) and SearchStatus (0x8F)
0x90-0x97	Search unit 10 key counter (0x90-0x96) and SearchStatus (0x97)
0x98-0x9F	Search unit 11 key counter (0x98-0x9E) and SearchStatus (0x9F)
0xA0-0xA7	Search unit 12 key counter (0xA0-0xA6) and SearchStatus (0xA7)
0xA8-0xAF	Search unit 13 key counter (0xA8-0xAE) and SearchStatus (0xAF)
0xB0-0xB7	Search unit 14 key counter (0xB0 0xB6) and SearchStatus (0xB7)
0xB8-0xBF	Search unit 15 key counter (0xB8-0xBE) and SearchStatus (0xBF)
0xC0-0xC7	Search unit 16 key counter (0xC0-0xC6) and SearchStatus (0xC7)
0xC8-0xCF	Search unit 17 key counter (0xC8-0xCE) and SearchStatus (0xCF)
0xD0-0xD7	Search unit 18 key counter (0xD0-0xD6) and SearchStatus (0xD7)
0xD8-0xDF	Search unit 19 key counter (0xD8-0xDE) and SearchStatus (0xDF)
0xE0-0xE7	Search unit 20 key counter (0xE0-0xE6) and SearchStatus (0xE7)
0xE8 0xEF	Search unit 21 key counter (0xE8-0xEE) and SearchStatus (0xEF)
0xF0-0xF7	Search unit 22 key counter (0xF0-0xF6) and SearchStatus (0xF7)
0xF8-0xFF	Search unit 23 key counter (0xF8-0xFE) and SearchStatus (0xFF)

the CBC mode IV.

PlaintextVector (256 bits = 8 bytes)

Identifies allowable plaintext byte values (ignoring those masked by the PlaintextByteMask). If, for any plaintext byte P[i=0..7], bit P[i] is not set, the decryption key will be rejected. PlaintextVector is identical in all search units and is set only once (when the search system is first initialized).

SearchInfo (8 bits)

The bits in SearchInfo describe how the correct plaintext identification function works. Bits of SearchInfo are defined as follows:

bit 0 = UseCBC

If this bit is set, Ciphertext0 is XORed onto the plaintext produced by decrypting Ciphertext1 before the plaintext is checked. This bit is used when checking CBC-mode ciphertexts.

bit 1 = ExtraXOR

If set, the right half of the resulting plaintext is XORed onto the left before any plaintext checking is done. ExtraXOR and UseCBC cannot be used together.

bit 2 = ChipAllActive

If cleared, one or more search units in this chip have halted (e.g., SearchActive is zero). This value is computed by ANDing the SearchActive bits of all search units' SearchStatus bytes. The inverse of this value is sent out on a dedicated pin, for use in driving a status LED which lights up whenever the chip halts.

bit 3 = BoardAllActive

This pin is the AND of the ChipAllActive lines of this chip and all later chips on the board. This is implemented by having each chip n take in chip n+1's BoardAllActive line, AND it with its own ChipAllActive line, and output the result to chip n-1 for its BoardAllActive computation. This makes it possible to find which chip on a board has halted by querying log2N chips, where N is the number of chips on the board. If BoardAllActiveEnable is not set to 1, BoardAllActive simply equals the BoardAllActiveInput pin, regardless of the chip's internal state.

bit 4 = BoardAllActiveEnable

If this value is set to 0 then BoardAllActive always equals the BoardAllActiveInput pin, regardless of whether all search units on the board are active. If this bit is set to 1, then the BoardAllActive register (and output) are set to reflect the internal state of the chip ANDed with the input pin.

bits 5-7 = Unused

KeyCounter (56 bits)

The value of the key currently being checked The KeyCounter is updated very frequently (i.e., once per key tested). A unique KeyCounter value is assigned to every search unit. When the search unit halts after a match, KeyCounter has already been incremented to the next key; the match was on the previous key.

SearchCommandAndStatus (8 bits)

The bits in SearchStatus describe the current search state of a specific search unit. A unique SearchStatus register is allocated for each search unit. Bits of SearchStatus are allocated as follows:

bit 0 = SearchActive

Indicates whether the search is currently halted (0=halted, 1=active). The computer sets this bit to begin a search, and it is cleared by the search unit if a matching candidate key is found. The host computer checks the status of this bit periodically and, if it is zero, reads out the key then restarts the search. (See also ChipAllActive and BoardAllActive in the SearchInfo register.)

bit 1 = CiphertextSelector

Indicates whether the search engine is currently checking Ciphertext0 or Ciphertext1. (0=Ciphertext0, 1=Ciphertext1). If this bit is clear, the search engine decrypts Ciphertext0 and either sets CiphertextSelector to 1 (if the plaintext passes the checks) or increments KeyCounter (if the plaintext does not pass). If this bit is set, the search engine decrypts Ciphertext1 and either sets SearchActive to 0 (if the plaintext passes the checks) or sets CiphertextSelector to 0 and increments KeyCounter (if the plaintext does not pass).

bits 2-7 = Unused

Commands

In order to be able to address each search unit separately, each can be addressed uniquely by the combination of its location on the chip, the location of the chip on the board, and board's identifier. The BoardID is interpreted off-chip; each chip has a board select pin, which notifies the chip when the board has been selected. Chip ID matching is done inside each ASIC; the ID pins of the ASIC are wired to the chip's ID.

All commands are originated by the computer go via a bus which carries 8 bits for BoardID/ChipID/Register address, 8 bits for data, and a few additional bits for controls .

To do a search, the host computer will program the search units as shown in Figure 2-2. (N is the total number of search units, numbered from 0 to N-1, each with a unique BoardID/ChipID/Register address.)

Search Unit Operation

Each search unit contains a DES engine, which performs DES on two 32-bit registers L/R using the key value in KeyCounter. Each search unit goes through the process detailed in Figure 2-3, and never needs to halt. If registers are updated during the middle of this process, the output is meaningless (which is fine, since an incorrect output is statistically almost certain to not be a match).

Figure 2-2: Example algorithm for programming the search array using host computer

This is a very simple algorithm intended only as an example. The actual software will use more intelligent search techniques, using the BoardAllActive and ChipAllActive lines.

Load Ciphertext0, Ciphertext1, PlaintextXorMask, PlaintextByteMask, PlaintextVector, and SearchInfo into each chip.

For i = 0 upto N-1

Set SearchStatus in search unit i to 0 while loading the key.
Set KeyCounter of search unit i to ((256)(i) / N).

```
    Set SearchStatus in search unit i to 1 to enable SearchActive.

EndFor

While correct key has not been found:

    For i = 0 upto N-1:
        Read SearchStatus from search unit i.
        Check SearchActive bit.
        If SearchActive is set to 0:
            Read KeyCounter from search unit i.
            Subtract 1 from the low 32 bits of the key.
            Perform a DES operation at the local computer to check the key. If the key is correct,
            the search is done.
            Set the SearchActive bit of SearchStatus to restart the search.

        EndIf

    EndFor

EndWhile
```

Sample Programming Descriptions

This section describes how the system will be programmed for some typical operations.

Known ciphertext/plaintext (ECB, CBC, etc.)

If a complete ciphertext/plaintext block is known, this mode is used. This works for most DES modes (ECB, CBC, counter, etc.), but does require a full plaintext/ ciphertext pair.

PlaintextVector

For this search, there are 8 (or fewer) unique plaintext bytes in the known plaintext. The bits corresponding to these bytes are set in PlaintextVector, but all other bits are set to 0.

Figure 2-3: Search unit operation

1. If CiphertextSelector is 0, then Let L/R = Ciphertext0.
If CiphertextSelector is 1, then Let L/R = Ciphertext1.

2. Decrypt L/R using the key in KeyCounter, producing a candidate plaintext in L/R.

3. If ExtraXOR is 1, then Let L = L XOR R.
If CiphertextSelector is 0, then

Let L/R = L/R XOR PlaintextXorMask.

```
If CiphertextSelector is 1 and UseCBC is 1, then:

Let L/R = L/R XOR Ciphertext0.

4. If SearchActive = 1 AND (
(PlaintextByteMask[0x80] = 0 AND PlaintextVector[byte 0 of L] is 0) OR
(PlaintextByteMask[0x40] = 0 AND PlaintextVector[byte 1 of L] is 0) OR
(PlaintextByteMask[0x20] = 0 AND PlaintextVector[byte 2 of L] is 0) OR
(PlaintextByteMask[0x10] = 0 AND PlaintextVector[byte 3 of L] is 0) OR
(PlaintextByteMask[0x08] = 0 AND PlaintextVector[byte 0 of R] is 0) OR
(PlaintextByteMask[0x04] = 0 AND PlaintextVector[byte 1 of R] is 0) OR
(PlaintextByteMask[0x02] = 0 AND PlaintextVector[byte 2 of R] is 0) OR
(PlaintextByteMask[0x01] = 0 AND PlaintextVector[byte 3 of R] is 0)) then:

Let CiphertextSelector = 0.
Increment KeyCounter.

else

If CiphertextSelector is 1 then Let SearchActive = 0.
Let CiphertextSelector = 1.

5. Go to step 1.
```

Ciphertext0

Equals the ciphertext block.

Ciphertext1

Equals the ciphertext block.

SearchInfo

UseCBC and ExtraXOR are both set to 0.

PlaintextByteMask

Set to 0x00 (all bytes used).

PlaintextXorMask

Set to 0x0000000000000000.

Because the plaintext byte order does not matter, there are 8 acceptable values for each ciphertext byte, or $8^8 = 2^{24}$ = 16.7 million possible ciphertexts which will satisfy the search criteria. The probability that an incorrect ciphertext will pass is $2^{24} / 2^{64}$, so over a search of 2^{55} keys there will be an average of $(2^{55})(2^{24} / 2^{64})$, or 32768 false positives which will need to be rejected by the controlling computer. Because the Ciphertext0 and Ciphertext1 selections are identical, any false positives that pass the first test will also pass the second test. (The performance penalty is negligible; the search system will do two DES operations on each of the 32768 false positive keys, but only one DES operation on all other incorrect keys.)

ASCII text (ECB or CBC)

A minimum of two adjacent ciphertexts (16 bytes total) are required for ASCII-only attacks.

PlaintextVector

Set only the bits containing acceptable ASCII characters. For normal text, this would normally include 55 of the 256 possible characters occur (10=line feed, 13=carriage return, 32=space, 65-90=capital letters, and 97-122=lowercase letters).

Ciphertext0

Equals the first ciphertext.

Ciphertext1

Equals the second ciphertext.

SearchInfo

UseCBC is set to 0 if ECB, or set to 1 if the ciphertext was produced using CBC. ExtraXOR is set to 0.

PlaintextByteMask

Set to 0x00 (all bytes used).

PlaintextXorMask

Set to 0x0000000000000000 for ECB, to IV for CBC.

The probability that the two (random) candidate plaintexts produced by an incorrect key will contain only the ASCII text characters listed above is $(55/256)^{16}$. In a search, there will thus be an average of 2^{55} $(55/256)^{16}$ = 742358 false positives which need to be rejected by the computer. For one key in about 220,000, the first check will pass and an extra DES will be required. (The time for these extra DES operations is insignificant.) Idle time lost while waiting for false positives to be cleared is also insignificant. If the computer checks each search unit's SearchActive flag once per second, a total of 0.5 search unit seconds will be wasted for every false positive, or a total of 103 search-unit hours, out of about 4 million search-unit hours for the whole search.

When programming CBC mode, note that the PlaintextXorMask must be set to the IV (or the previous ciphertext, if the ciphertext being attacked is not in the first block).

Matt Blaze's Challenge

The goal is to find a case where all plaintext bytes are equal and all ciphertext bytes are equal.

PlaintextVector

Set only bit 0.

Ciphertext0

Set to a fixed value with all bytes equal

Ciphertext1

Same as Ciphertext0.

SearchInfo

UseCBC is set to 0. ExtraXOR is set to 1.

PlaintextByteMask

Set to 0x0F (only left half examined).

PlaintextXorMask

Set to 0x0000000000000000.

If the right and left half are equal, as must be the case if all plaintext bytes are the same, then when the ExtraXOR bit's status causes the L=L XOR R step, L will become equal to 0. The plaintext byte mask selects only the left half and the PlaintextVector makes sure the 4 bytes are 0.

False positives occur whenever L=R, or with one key in 2^{32}. Because this search is not guaranteed to terminate after 2^{56} operations, the average time is 2^{56} (not 2^{55}). The number of false positives is expected to be $2^{56} / 2^{32} = 2^{24} = 16.8$ million. Each search unit will thus find a false positive every 2^{32} keys on average, or about once every half hour. At 1 second polling of search units, (0.5)(16.8 million)/3600 = 2333 search unit hours will be idle (still under 1% of the total). The host computer will need to do the 16.8 million DES operations (on average), but even a fairly poor DES implementation can do this in just a few minutes.

Scalability and Performance

The architecture was intended to find DES keys in less than 10 days on average. The performance of the initial implementation is specified in Figure 2-4. Faster results can be easily obtained with increased hardware; doubling the amount of hardware will halve the time per result. Within the design, boards of keysearch ASICs can be added and removed easily, making it simple to make smaller or larger systems, where larger systems cost more but find results more quickly. Larger systems will have additional power and cooling requirements.

Figure 2-4: Performance Estimate

Total ASICs	1536
Search units per ASIC	24
Total search units	36864
Clock speed (Hz)	4.00E+07
Clocks per key (typical)	16
DES keys per search unit per second	2.50E+06
Total DES keys per second	9.22E+10
Search size (worst case)	7.21E+16
Seconds per result (worst case)	7.82E+05
Days per result (worst case)	9.05
Search size (average case)	3.60E+16
Seconds per result (average case)	3.91E+05
Days per result (average case)	**4.52**

Host Computer Software

Cryptography Research will write the following software:

Simulation

Cryptography Research will develop software to generate test vectors for the chip for testing before the design is sent to the fab. This software will test all features on the chip and all modes of operation. This program will have a simple command line interface.

Host computer

The host computer software program will implement the standard search tasks of breaking a known plaintexts, breaking encrypted ASCII text (ECB and CBC modes), and solving the Matt Blaze challenge. These programs will be written in standard ANSI C, except for platform-specific I/O code. The host program will also have a test mode, which loads search units with tasks that are known to halt reasonably quickly (e.g., after searching a few million keys) and verifies the results to detect of any failed parts. (The software will include the capability of bypassing bad search units during search operations.) Users who wish to perform unusual searches will need to add a custom function to determining whether candidate keys are actually correct and recompile the code.

The initial version of this program will have a simple command line interface and will be written for DOS. A Linux port will also be written, but may not be ready by the initial target completion date. (Because the only platform-specific code will be the I/O functions, it should be very easy to port to any platform with an appropriate compiler.) Software programs will identify the participants in the project (AWT, EFF, and Cryptography Research).

Cryptography Research will also produce a version with a prettier user interface to make the demonstration more elegant (platform-to-be-determined).

All software and source code will be placed in the public domain.

Glossary

BoardID

An 8-bit identifier unique for each board. This will be set with a DIP switch on the board. The host computer addresses chips by their ChipID and BoardID.

CBC mode

A DES mode in which the first plaintext block is XORed with an initialization vector (IV) prior to encryption, and each subsequent plaintext is XOR with the previous ciphertext.

ChipID

A value used by the host computer to specify which chip on a board is being addressed.

Ciphertext

Encrypted data.

Ciphertext0

The first of the two ciphertexts to be attacked.

Ciphertext1

The second of the two ciphertexts to be attacked.

Pre-ANSI C can be supported if required. Any GUI code will probably be written in C++ .

CiphertextSelector

A register used to select the current ciphertext being attacked. The selector is needed because a single DES engine needs to be able to test two ciphertexts to determine whether both are acceptable matches before deciding that a key is good match.

DES

The Data Encryption Standard.

ExtraXOR

A register to make the search units perform an extra operation which XORs the right and left halves of the result together. This is used to add support for Matt Blaze's DES challenge.

Host computer

The computer that controls the DES search array.

KeyCounter

Each search unit has a KeyCounter register which contains the current key being searched. These registers are each 7 bytes long, to hold a 56-bit key.

Plaintext

Unencrypted data corresponding to a ciphertext.

PlaintextByteMask

An 8-bit register used to mask off plaintext bytes. This is used to mask off bytes in the plaintext whose values aren't known or are too variable to list in the PlaintextVector.

PlaintextVector

A 256-bit register used to specify which byte values can be present in valid plaintexts. It is the host computer's responsibility to ensure that only a reasonable number of bits are set in the PlaintextVector; setting too many will cause the DES search units to halt too frequently.

PlaintextXorMask

A 64-bit register XORed onto the value derived by decrypting ciphertext 0. Normally this mask is either zero or set to the CBC mode initialization vector (IV).

SearchActive

A bit for each search unit which indicates whether it is currently searching, or whether it has stopped at a candidate key. Stopped search units can be restarted by loading a key which does not halt and resetting this bit.

SearchInfo

A register containing miscellaneous information about how DES results should be post-processed and also indicating whether any search units on the chip or on the board have halted.

UseCBC

A bit in SearchInfo which directs the search engine to do CBC-mode post-processing after decryption (e.g., XOR the decryption of ciphertext1 with ciphertext0 to produce plaintext1).

3
Design for DES Key Search Array Chip-Level Specification

In This chapter:

- *ASIC Description*
- *Board description*
- *Read and Write Timing*
- *Addressing Registers*
- *All-active Signal*
- *ASIC Register Allocation*

Advanced Wireless Technologies, Inc.
and
Cryptography Research

ASIC Description

Select1

Selects Cipher text 1

C0

Cipher text 0

C1

Cipher text 1

Search

Search is active

K

Key

Mask

Plain text bit mask and DES output

Match=0

a Zero is found in any bit position of plain text vector as specified in step 4 of Search Unit Operation (see Chapter 2)

CBC & Extra XOR

Perform step 3 of Search Unit Operation (see Chapter 2)

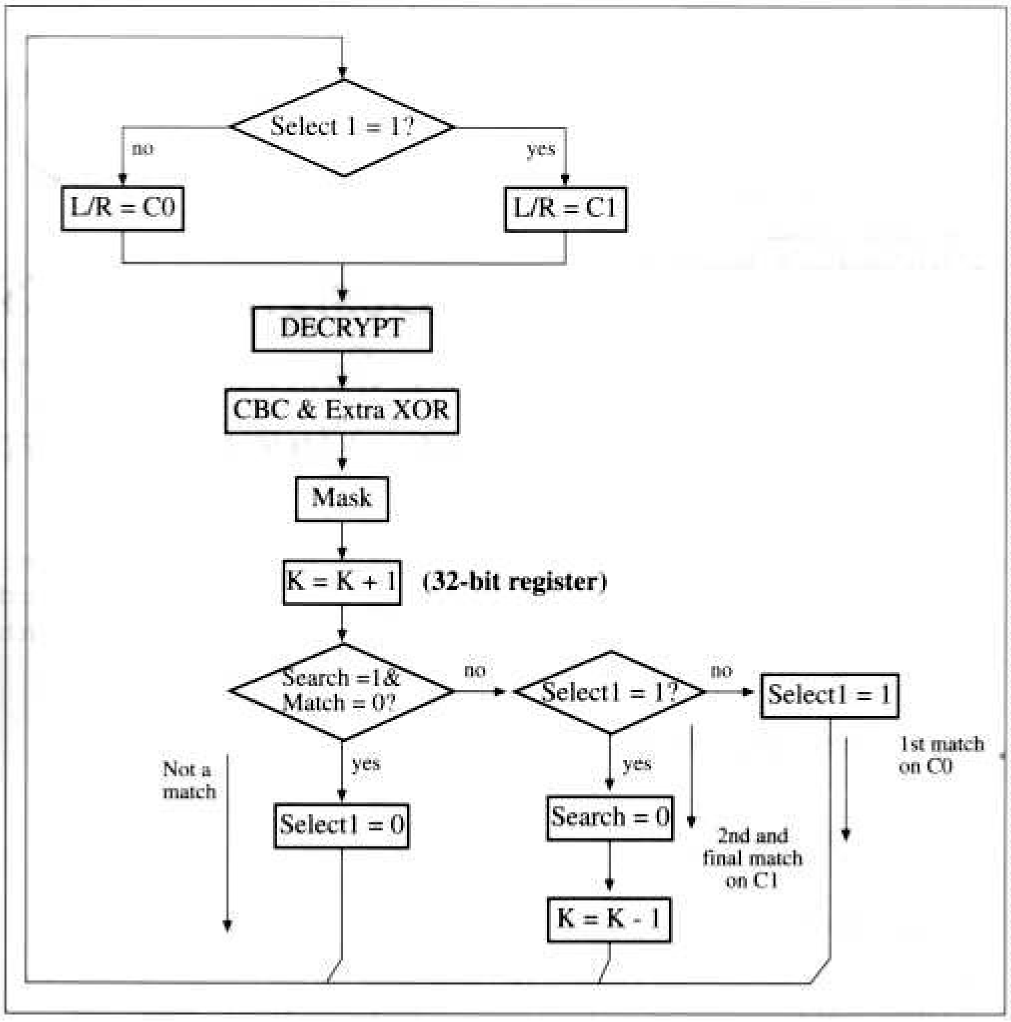


Figure 3-1. Search Unit Operation Flow Chart

To determine the maximum number of bit required for the Key:

K= $\log_2$(Maximum combinations/number of chips)

= $\log_2(2^{56}/(24$ cpc * 64 cpb * 24 boards) = $\log_2(1.\ 95E12)$ = 42 bits

If we are going to use 32-bit counters, then it will overflow every:

2^{32} * 16 cycles * 25ns = 1. 72 * 10^{12}ns = 1720 sec = 28. 7 minutes

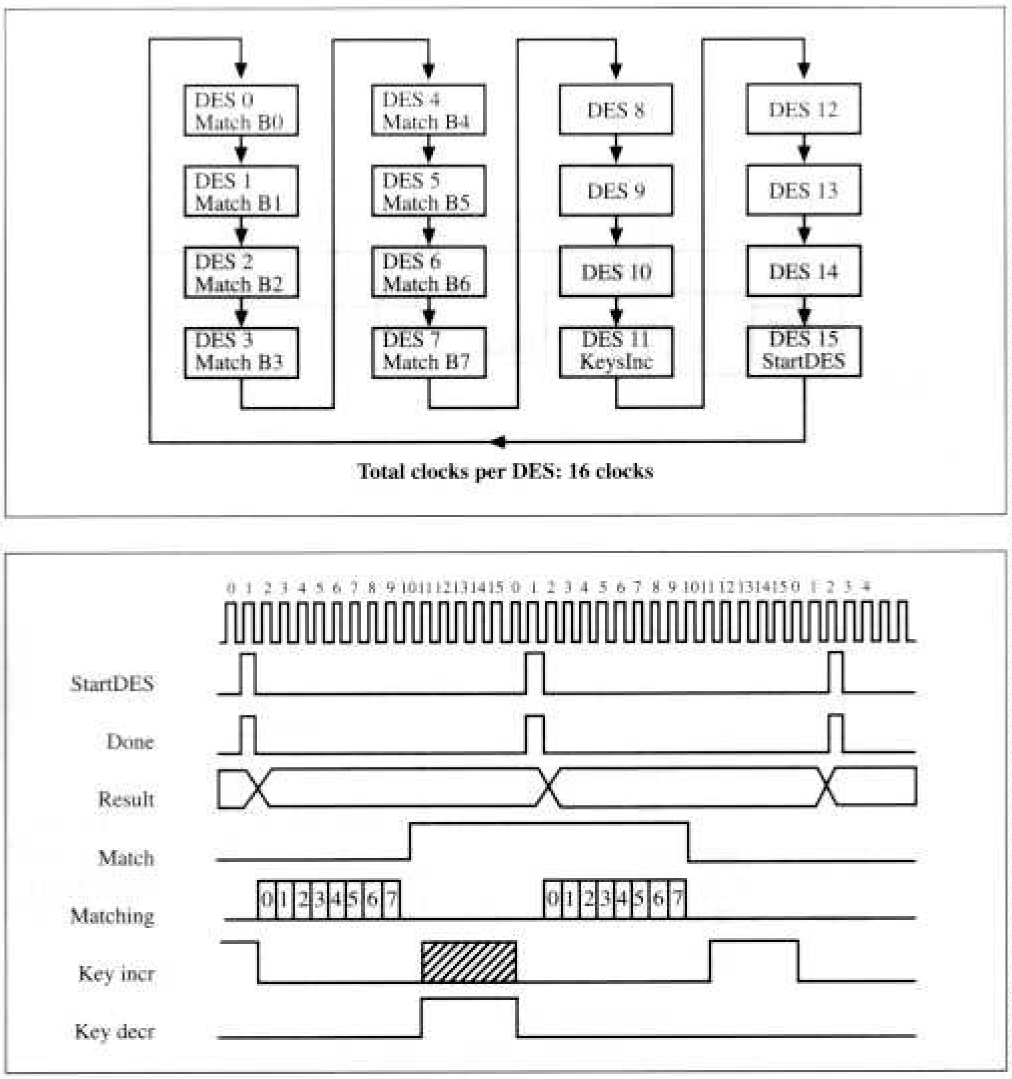


Figure 3-2 State Diagram tor the Search Unit

Board description

The PC will interface with the ASICs through a parallel card. The parallel card has three ports, assigned:

Port A: Address(7:0)
Port B: Data(7:0)
Port C: Control, 8 signals

To reduce the routing resources on the boards and ASICs we multiplex the address lines. To access register on the ASIC, it is required that the software latch the

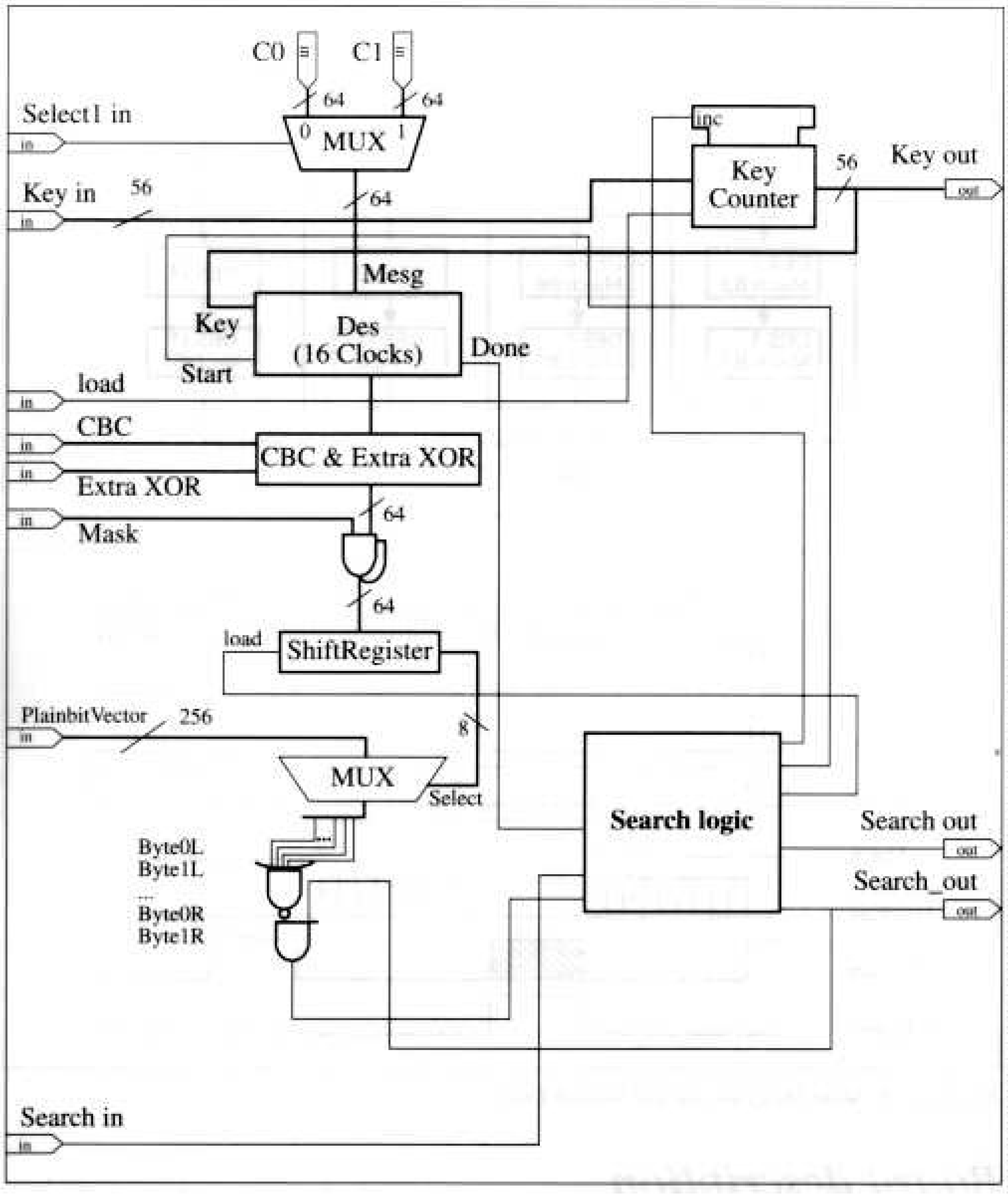
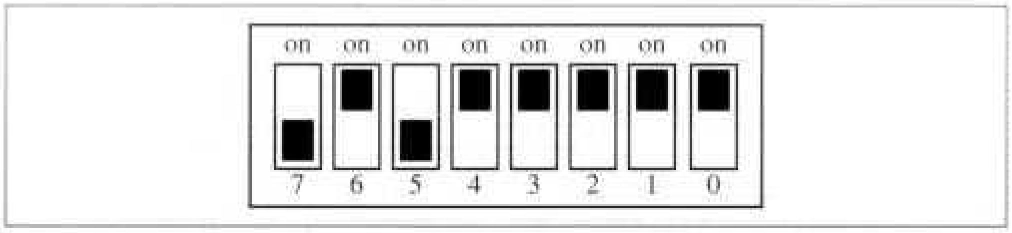


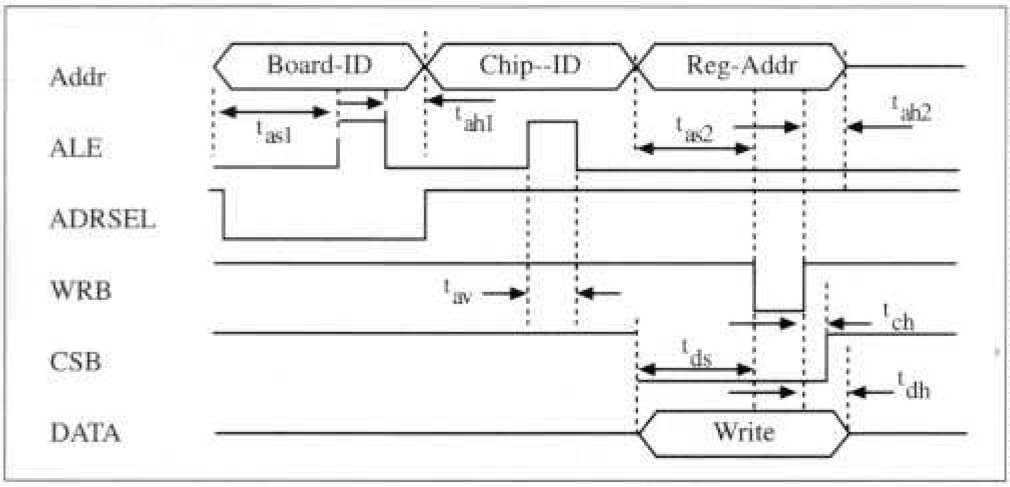
Figure 3-3: Search Unit's Block Diagram

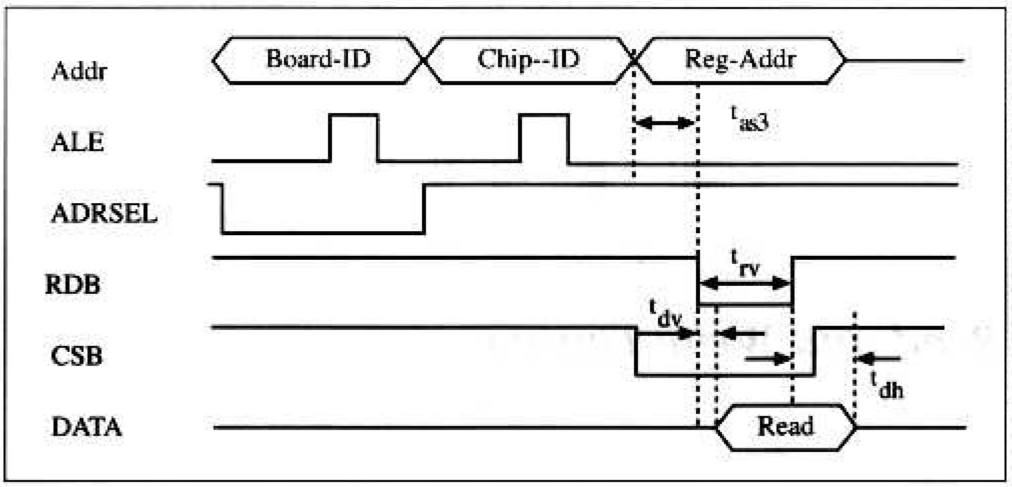
address three times: Board-ID(7:0), Chip-ID(6:0) and then Register address.

Having switches on the board makes the design flexible and expandable. Each board has its own unique Board-ID configured on switches: for example a board with an ID of hexadecimal 5F has its board ID switches configured as follows:



Read and Write Timing





t_{as1} 10 ns Min Board-ID and Chip-ID Address setup

t_{as2} 10 ns Min Write Register-Address setup

t_{as3} 10 ns Min Read Register-Address setup

t_{ah1} 10 ns Min Board-ID and Chip-ID Address invalid (hold)

t_{ah2} 10 ns Min Write strobe trailing edge to Address invalid (hold)

t_{av} 10 ns Min ALE valid

t_{ds} 10 ns Min Data valid to Write strobe goes low (setup)

t_{ch} 10 ns Min Chip select hold

t_{dh} 10 ns Min Write strobe goes high to data invalid (Data hold)

t_{rv} 10 ns Min Read strobe duration

t_{dv} 100 ns Max Read strobe goes low to data valid

t_{dh} 100 ns Max Read strobe goes high to data invalid (Data hold)

Addressing Registers

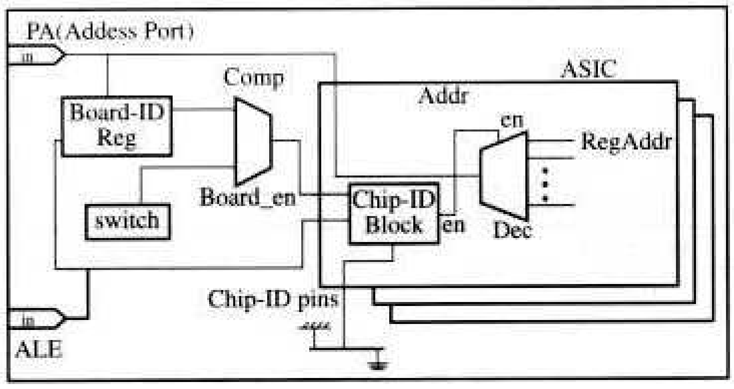
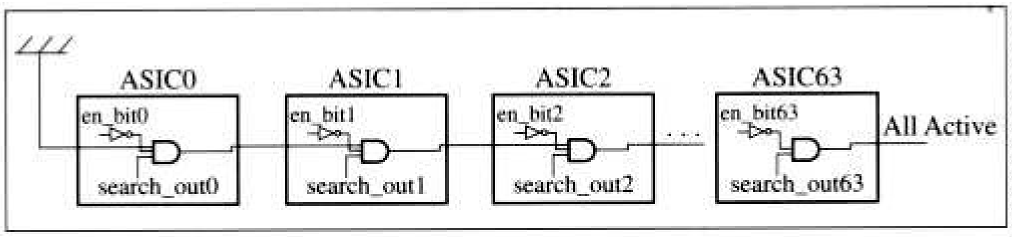


Figure 3-4 Address Bus Scheme

All-active Signal

If low the SearchActive bit together. We will place one AND gate per ASIC and cascade them.



ASIC Register Allocation

Registers Common to All Search Units

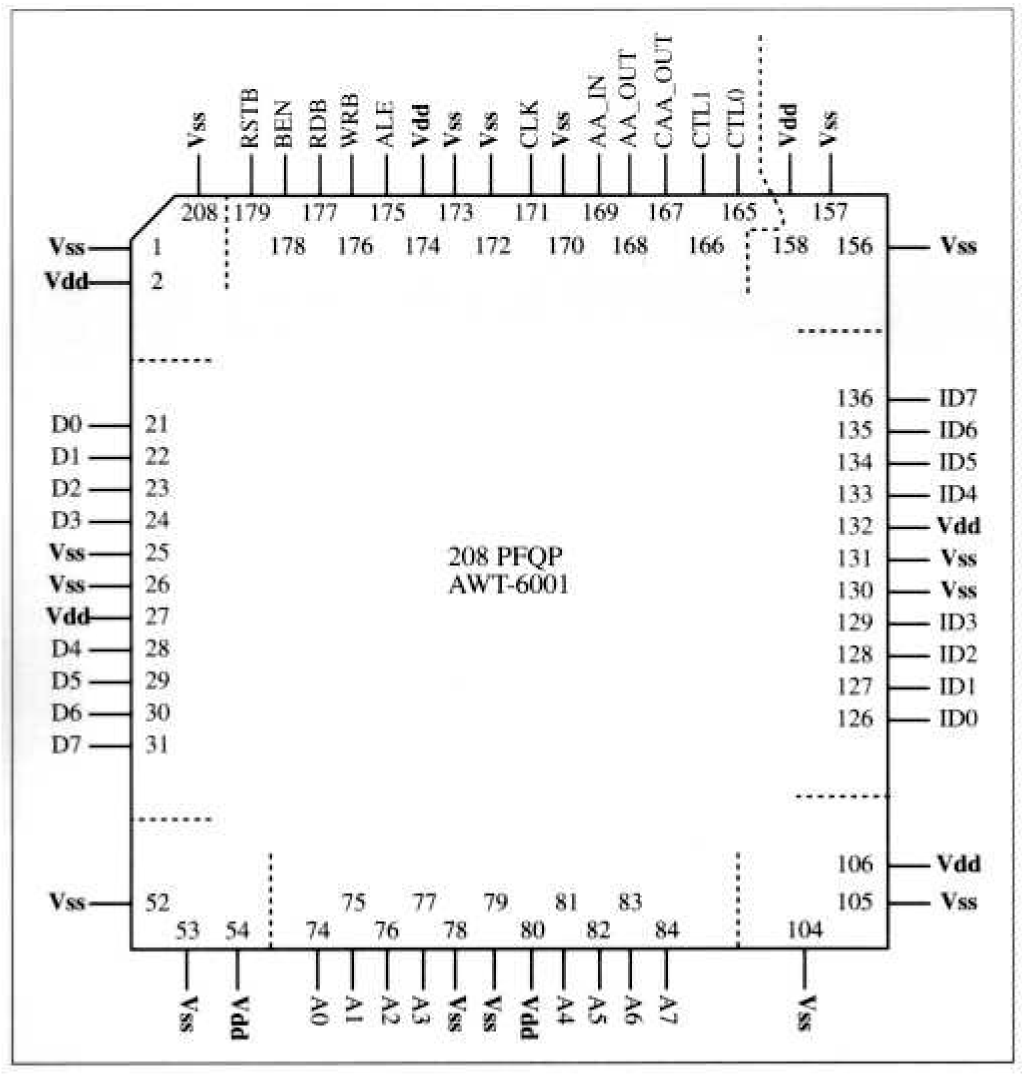
0x00-0x1f	PlaintextVector
0x20-0x27	PlaintextXorMask
0x28-0x2f	CipherText0
0x30-0x37	CipherText1
0x38	PlaintextByteMask
0x39-0x3e	Reserved
0x3f	SearchInfo

Additional Registers for Search Units

0x40-0x47	Search Unit 0: Key counter (first 7 bytes) and Search Status
0x48-0x4f	Search Unit 1: Key counter (first 7 bytes) and Search Status
. . .	
0xf8-0xff	Search Unit 23: Key counter (first 7 bytes) and Search Status

Number of register required:

58 common registers + 8 * *n* registers; *n* = the total number of search units in an ASIC In this case *n* = 24, therefore 58 + 192 = 250 registers



Note: The unspecified pins are Non-Connects

CNTRL0 = ALE = ADDSEL1

CNTRL1 = CSB = ADDSEL2

4
Scanning the Source Code

In This chapter:

- *The Politics of Cryptographic Source Code*
- *The Paper Publishing Exception*
- *Scanning*
- *Bootstrapping*

The next few chapters of this book contain specially formatted versions of the documents that we wrote to design the DES Cracker. These documents are the primary sources of our research in brute-force cryptanalysis, which other researchers would need in order to duplicate or validate our research results.

The Politics of Cryptographic Source Code

Since we are interested in the rapid progress of the science of cryptography, as well as in educating the public about the benefits and dangers of cryptographic technology, we would have preferred to put all the information in this book on the World Wide Web. There it would be instantly accessible to anyone worldwide who has an interest in learning about cryptography.

Unfortunately the authors live and work in a country whose policies on cryptography have been shaped by decades of a secrecy mentality and covert control. Powerful agencies which depend on wiretapping to do their jobs--as well as to do things that aren't part of their jobs, but which keep them in power--have compromised both the Congress and several Executive Branch agencies. They convinced Congress to pass unconstitutional laws which limit the freedom of researchers--such as ourselves--to publish their work. (All too often, convincing Congress to violate the Constitution is like convincing a cat to follow a squeaking can opener, but that doesn't excuse the agencies for doing it.) They pressured agencies such as the Commerce Department, State Department, and Department of Justice to not only subvert their oaths of office by supporting these unconstitutional laws, but to act as front-men in their repressive censorship scheme, creating unconstitutional regulations and enforcing them against ordinary researchers and authors of software.

The National Security Agency is the main agency involved, though they seem to have recruited the Federal Bureau of Investigation in the last several years. From the outside we can only speculate what pressures they brought to bear on these other parts of the government. The FBI has a long history of illicit wiretapping, followed by use of the information gained for blackmail, including blackmail of Congressmen and Presidents. FBI spokesmen say that was "the old bad FBI" and that all that stuff has been cleaned up after J. Edgar Hoover died and President Nixon was thrown out of office. But these agencies still do everything in their power to prevent ordinary citizens from being able to examine their activities, e.g. stonewalling those of us who try to use the Freedom of Information Act to find out exactly what they are doing.

Anyway, these agencies influenced laws and regulations which now make it illegal for U.S. crypto researchers to publish their results on the World Wide Web (or elsewhere in electronic form).

The Paper Publishing Exception

Several cryptographers have brought lawsuits against the US Government because their work has been censored by the laws restricting the export of cryptography. (The Electronic Frontier Foundation is sponsoring one of these suits, Bernstein v. Department of Justice, et al).* One result of bringing these practices under judicial scrutiny is that some of the most egregious past practices have been eliminated.

For example, between the 1970's and early 1990's, NSA actually did threaten people with prosecution if they published certain scientific papers, or put them into libraries. They also had a "voluntary" censorship scheme for people who were willing to sign up for it. Once they were sued, the Government realized that their chances of losing a court battle over the export controls would be much greater if they continued censoring books, technical papers, and such.

Judges understand books. They understand that when the government denies people the ability to write, distribute, or sell books, there is something very fishy going on. The government might be able to pull the wool over a few judges' eyes about jazzy modern technologies like the Internet, floppy disks, fax machines, telephones, and such. But they are unlikely to fool the judges about whether it's constitutional to jail or punish someone for putting ink onto paper in this free country.

---

* See http://www.eff.org/pub/Privacy/ITAR_export/Bernstein_case/.

Therefore, the last serious update of the cryptography export controls (in 1996) made it explicit that these regulations do not attempt to regulate the publication of information in books (or on paper in any format). They waffled by claiming that they "might" later decide to regulate books-- presumably if they won all their court cases -- but in the meantime, the First Amendment of the United States Constitution is still in effect for books, and we are free to publish any kind of cryptographic information in a book. Such as the one in your hand.

Therefore, cryptographic research, which has traditionally been published on paper, shows a trend to continue publishing on paper, while other forms of scientific research are rapidly moving online.

The Electronic Frontier Foundation has always published most of its information electronically. We produce a regular electronic newsletter, communicate with our members and the public largely by electronic mail and telephone, and have built a massive archive of electronically stored information about civil rights and responsibilities, which is published for instant Web or FTP access from anywhere in the world.

We would like to publish this book in the same form, but we can't yet, until our court case succeeds in having this research censorship law overturned. Publishing a paper book's exact same information electronically is seriously illegal in the United States, if it contains cryptographic software. Even communicating it privately to a friend or colleague, who happens to not live in the United States, is considered by the government to be illegal in electronic form.

The US Department of Commerce has officially stated that publishing a World Wide Web page containing links to foreign locations which contain cryptographic software "is not an export that is subject to the Export Administration Regulations (EAR)."* This makes sense to us--a quick *reductio ad absurdum* shows that to make a ban on links effective, they would also have to ban the mere mention of foreign Universal Resource Locators. URLs are simple strings of characters, like http://www.eff.org/; it's unlikely that any American court would uphold a ban on the mere naming of a location where some piece of information can be found.

Therefore, the Electronic Frontier Foundation is free to publish links to where electronic copies of this book might exist in free countries. If we ever find out about such an overseas electronic version, we will publish such a link to it from the page at http://www.eff.org/pub/Privacy/Crypto_misc/DESCracker/.

---

* In the letter at http://samsara.law.cwru.edu/comp_law/jvd/pdj-bxa-gjs070397.htm, which is part of Professor Peter Junger's First Amendment lawsuit over the crypto export control regulations.

Scanning

When printing this book, we used tools from Pretty Good Privacy, Inc (which has since been merged into Network Associates, Inc.). They built a pretty good set of tools for scanning source code, and for printing source code for scanning. The easiest way to handle the documents we are publishing in this book is to use their tools and scanning instructions.

PGP published the tools in a book, naturally, called "Tools for Publishing Source Code via OCR", by Colin Plumb, Mark H. Weaver, and Philip R. Zimmermann, ISBN # 1-891064-02-9. The book was printed in 1997, and is sold by Printers Inc. Bookstore, 301 Castro St, Mountain View, California 94041 USA; phone +1650 961 8500; http://www.pibooks.com/.

The tools and instructions from the OCR Tools book are now available on the Internet as well as in PGP's book. See http://www.pgpi.com/project/, and follow the link to "proof-reading utilities". If that doesn't work because the pages have been moved or rearranged, try working your way down from the International PGP page, http://www.pgpi.com/.

PGP's tools produce per-line and per-page checksums, and make normally invisible characters like tabs and multiple spaces explicit. Once you obtain these tools, we strongly suggest reading the textual material in the book, or the equivalent README file in the online tool distribution. It contains very detailed instructions for scanning and proofreading listings like those in this book. The instructions that follow in this chapter are a very abbreviated version.

The first two parts of converting these listings to electronic form is to scan in images of the pages, then convert the images into an approximation of the text on the pages. The first part is done by a mechanical scanner; the second is done by an Optical Character Recognition (OCR) program. You can sometimes rent time at a local "copy shop" on a computer that has both a scanner and an OCR program.

When scanning the sources, we suggest "training" your OCR program by scanning the test-file pages that follow, and some of the listings, and correcting the OCR program's idea of what the text actually said. The details of how to do this will depend on your particular OCR program. But if you straighten it out first about the shapes of the particular characters and symbols that we're using, the process of correcting the errors in the rest of the pages will be much easier.

Some unique characters are used in the listings; train the OCR program to convert them as follows:

Right pointing triangle (used for tabs) - currency symbol (byte value octal 244)

Tiny centered triangle "dot" (used for multiple spaces) - center dot or bullet (byte value octal 267)

Form feed - yen (byte value octal 245)

Big black square (used for line continuation) - pilcrow or paragraph symbol (byte value octal 266).

Once you've scanned and OCR'd the pages, you can run them through PGP's tools to detect and correct errors, and to produce clean online copies.

Bootstrapping

By the courtesy of Philip R. Zimmermann and Network Associates, to help people who don't have the PGP OCR tools, we have included PGP's bootstrap and bootstrap2 pages. (The word *bootstrap* refers to the concept of "pulling yourself up by your bootstraps", i.e. getting something started without any outside help.) If you can scan and OCR the pages in some sort of reasonable way, you can then extract the corrected files using just this book and a Perl interpreter. It takes more manual work than if you used the full set of PGP tools.

The first bootstrap program is one page of fairly easy to read Perl code. Scan in this page, as carefully as you can: you'll have to correct it by hand. Make a copy of the file that results from the OCR, and manually delete the checksums, so that it will run as a Perl script. Then run this Perl script with the OCR result (with checksums) as the argument. If you've corrected it properly, it will run and produce a clean copy of itself, in a file called bootstrap. (Make sure none of your files have that name.) If you haven't corrected it properly, the perl script will die somehow and you'll have to compare it to the printed text to see what you missed.

When the bootstrap script runs, it checks the checksum on each line of its input file. For any line that is incorrect, the script drops you into a text editor (set by the EDITOR environment variable) so you can fix that line. When you exit the editor, it starts over again.

Once the bootstrap script has produced a clean version of itself, you can run it against the scanned and OCR'd copy of the bootstrap2 page. Correct it the same way, line by line until bootstrap doesn't complain. This should leave you with a clean copy of bootstrap2.

The bootstrap2 script is what you'll use to scan in the rest of the book. It works like the bootstrap script, but it can detect more errors by using the page checksum. Again, it won't correct most errors itself, but will drop you into an editor to correct them manually. (If you want automatic error correction, you have to get the PGP book.)

All the scannable listings in this book are in the public domain, except the test-file, bootstrap, and bootstrap2 pages, which are copyrighted, but which Network Associates permits you to freely copy. So none of the authors have put restrictions on your right to copy their listings for friends, reprint them, scan them in, publish them, use them in products, etc. However, if you live in an unfree country, there may be restrictions on what you can do with the listings or information once you have them. Check with your local thought police.

pp. 4-7 to 4-14 in preparation: Six pages of test files, 1 page of bootstrap, 1 page of bootstrap2. [Note: see bootstrap and bootstrap 2 as part of OCR tools, http://www.pgpi.com/project/]

Chapters 5, 6 and 7

(source code in zip archive)
(source code in .tar.gz archive)

5

Software Source Code

6

Chip Source Code

7

Chip Simulator Source Code

8
Hardware Board Schematics

Note: Yvo Desmedt granted permission on August 1, 1998 to publish this chapter. Mr. Desmedt stated that he is responsible only for this chapter and not the book. His current addresses are the University of Wisconsin - Milwaukee, and the University of London. <desmedt@cs.uwm.edu>. And, "Note that it is said in the book that this paper was presented at Eurocrypt '87, which is incorrect. A more general paper, with a different title was presented at Eurocrypt '86. The chapter may have been presented at a rump session, but I do not remember it."

9
Breaking One Million DES Keys by Yvo Desmedt

In This chapter.

This paper was presented at Eurocrypt 1987 by Yvo Desmedt and Jean-Jacques Quisquater, under the title "An Exhaustive Key Search Machine Breaking One Million DES Keys". We publish it here for the first time, since no proceedings were made. It points out some research directions in parallel brute force codebreaking that are still useful today.

Abstract

The DES is in the commercial and industrial world the most used cryptoalgorithm. A realistic exhaustive key search machine will be proposed which breaks thousands of keys each hour, when DES is used in its standard 8 byte modes to protect privacy. Also authenticity protection with DES is sometimes insecure.

Introduction

The DES is the NBS* and ANSI+ standard for encryption. It has been proposed to become an ISO# standard, under the name DEA1. From the beginning Diffie and Hellman mentioned that one DES key could be broken under a known plaintext attack using an exhaustive keysearch machine.§ However the design was criticized because practical problems as size and power dissipation were not taken into

* "Data Encryption Standard", FIPS (National Bureau of Standards Federal Information Processing Standards Publ.), no. 46, Washington D.C., January 1977.

+ "Data Encryption Algorithm", ANSI X3.92-1981, (American National Standards Institute), New York, December 31, 1980.

# "Data Encipherment, Specification of Algorithm DEA1", ISO/DP 8227 (Draft Proposal), 1983.

§ Diffie, W., and Hellman, M.E.: "Exhaustive cryptanalysis of the NBS Data Encryption Standard", Computer, vol. 10, no. 6, pp. 74 -84, June 1977.

consideration. Hoornaert* proposed last year a realistic exhaustive keysearch machine, which solved all practical problems. Instead of breaking DES in half a day (as in the Diffie-Hellman machine), the cheap version ($1 million) needs maximum 4 weeks to find the key. In practice however companies or secret agencies want to break several keys at once. Indeed for doing industrial espionage, companies want to break as many communications as possible of their main competitors. Secret agencies want to be able to eavesdrop all communications and to follow up industrial developments in other countries which may be used for military purposes. The above machine is unpractical or expensive for this purpose. Instead of using thousands of machines for breaking thousands of keys, one modified machine is enough.

The basic idea

At first sight if one wants to break one million keys with an exhaustive machine one needs one million pairs (plaintext,ciphertext)=(Mi,Ci) and do the job for each different pair. If all these pairs have the same plaintext M, the exhaustive machine can do the same job by breaking all these one million ciphertexts, as in the case it had only to break one. This assumption is very realistic, indeed in letters some pattern as e.g."Yours Sincerely" are common. For all standard+ 8 bytes modes a partially known plaintext attack is sufficient. In the case of ECB a ciphertext only attack is sufficient. Indeed the most frequent combination of 8 bytes can easily be detected and used. Evidently more machines can handle more different plaintext patterns. So, a few machines can break millions of keys. The number of different patterns can be reduced by using a chosen plaintext attack!

Details of such a machine

Although we did not built it, in this section sufficient details are given to show that such a machine is feasible. The machine will be based on a small extension of the DES chips used in Hoornaert's machine. We will call the ciphertexts for which one wants to break the key: "desired" ciphertexts. In one machine, each of the (e.g.) 25 thousand DES chips will calculate ciphertexts for variable keys starting from the same 8 byte "plaintext" pattern. The machine has to verify if such a ciphertext is the same as some "desired" ciphertext. If so, it has to communicate the corresponding key to the Key Handling Machine (KHM) and the "number" of the "desired" ciphertext. However each used DES chip generates each second about

---

* Hoornaert, F., Goubert, J., and Desmedt, Y.: "Efficient hardware implementations of the DES", Advances in Cryptology, Proceedings of Crypto 84, Santa Barbara, August 1984 (Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1985), pp. 147-173.

+ DES modes of operation", FIPS (NBS Federal Information Processing Standards Publ.), no. 81, Washington D.C., December 2, 1980.

---

one million pairs (ciphertext, key). This gives a major communication problem. Indeed all this information (about 110Mbit/sec.= (56 key bits + 64 ciphertext bits) x 1M DES/sec.) cannot be communicated constantly outside the chip. To avoid this communication problem, the chip will internally exclude ciphertexts which certainly are not equal to a "desired" ciphertext. So only a fraction has to be communicated to the outside world. Hereto the "desired" ciphertexts were previously ordered based on their first 20 bits, which are used as address of the desired ciphertexts. If more than one of these "desired" ciphertexts have the same 20 first bits then one of them will later be transferred to the exhaustive machine. The others will be put on a waiting list. In the exhaustive machine bits of the desired ciphertexts are spread in RAMs, as explained later, using the 20 first bits as address. Each extended DES chip is put on a hybrid circuit together with 4 RAMs of 1Mbit and a refresh controller (see also fig. 1*[not provided]*). For each enumerated key the DES chip communicates the 20 first bits of the corresponding generated ciphertext to the RAMs as address. The 4 bits information stored in the RAMs correspond to the next 4 bits of the desired ciphertexts. The RAMs communicate to the modified DES chip these 4 bits. Only if these 4 bits are equal to the corresponding ones in the generated ciphertext, the generated pair (ciphertext, key) is communicated outside the DES chip to a local bus (see fig. 1). So in average the communication rate is reduced, by excluding the ciphertexts which are certainly not desired. About 10 of these hybrids are put on a small PCB. A custom designed chip checks the next 10 bits (the bits 25 till 34) of the ciphertexts using the same idea as for the 4 bits (the bits 21 till 24). Hereto 10 RAMs each of 1Mbit are used, the address is again the first 20 bits of the generated ciphertext. Only if the check succeeds the pair (ciphertext, key) is communicated to the outside world via a global bus. This reduces the communication between the local bus and the global bus with a factor 1000. About 2500 similar PCBs are put in the machine. The last 30 bits of the ciphertext are checked further on. Hereto similar hardware controls several PCBs. Finally a small machine can do the final check. The machine KHM checks the correctness of the key on other (plaintext, ciphertext) pairs or on the redundancy in the language. Once each (e.g.) hour the machine KHM will update the broken keys and put the ones which are on the waiting list into the exhaustive machine (if possible). Suppose that one hybrid cost $80, then the price of $3 million (25,000 x hybrid + custom chips + PCBs + etc) for this machine is realistic.

Obtained results and remarks

The described machine breaks about one million keys in 4 weeks, or in average about 3000 keys each hour. By updating the broken keys better results can be obtained.* Practical problems as buffering, synchronization, MTBF, power dissipation, size, reloading of the RAMs and so on are solved by the author. Optimizations under several circumstances and variants of the machine are possible. In view of the existing rumors that a trapdoor was built in DES by NSA, the feasibility of this machine shows that a trapdoor was not needed in order to break it. Old RAM technology allowed to design similar (or larger) machines which break less keys (e.g. thirty-two thousand keys). This attack can be avoided if the users of DES use the CFB one byte mode appropriately, or use new modes+ or triple encryption with two different keys. DES-like algorithms can be designed which are more secure against the described attack and which use a key of only 48 bit, and which have the same encryption/decryption speed as DES (if used with fixed key).[#] The protection of the authenticity of (e.g. short) messages with DES is sometimes insecure.[§] These results combined with the above one, shows that the authentication of standardized messages with DES may be worthless. Remark finally that the DES chip used in this machine does not use the state of the art of VLSI. Indeed about only 10,000 transistors are used in it. Megabits RAMs are easily available.

Conclusion

Every important company or secret agency over the world can easily build such a machine. Because it is not excluded that such machines are already in use by these organizations, the author advises the users to be careful using DES. Because the most used modes are breakable, the users have to modify their hard- or software in a mode which avoids this attack. Meanwhile only low-sensitive information can be transmitted with DES. If the authenticity of the messages is protected with DES under its standardized use, short messages have to be enlarged.

---

* Desmedt, Y., "Optimizations and variants of exhaustive key search machines breaking millions of DES keys and their consequences on the security of privacy and authenticity with DES", Internal Report, ESAT Laboratory, Katholieke Universiteit Leuven, in preparation.

+ Quisquater, J.-J., Philips Research Laboratory, Brussels, paper in preparation.

# Quisquater, J.-J., Desmedt, Y., and Davio, M.: "A secure DES* scheme with < 48 bit keys", presented at the rump session at Crypto '85, Santa Barbara, August, 1985

§ Desmedt, Y.: "Unconditionally secure authentication schemes and practical and theoretical consequences", presented at Crypto '85, Santa Barbara, August, 1985, to appear in the proceedings: Advances in Cryptology (Springer-Verlag, Berlin, 1986).

Acknowledgement

The author is sponsored by the Belgian NFWO. The author is very grateful to F. Hoornaert, IMEC-ESAT, Leuven, and J.-J. Quisquater, Philips Research Laboratory, Brussels, for many suggestions and improvements.

Y.Desmedt
ESAT Laboratory
Katholieke Universiteit Leuven
Kard. Mercierlaan 94
B-3030 Heverlee, Belgium

10
Architectural Considerations for Cryptanalytic Hardware

Ian Goldberg and David Wagner

[iang,daw]@cs.berkeley.edu

http://www.shmoo.com/crypto/Cracking_DES/CH10/main.html (HTML)

http://www.cs.berkeley.edu/~iang/isaac/hardware/paper.ps (Postscript)

11
Efficient DES Key Search An Update by Michael J. Wiener

In This chapter:

An exciting moment in the history of DES was reached in June 1997 when a group coordinated by Rocke Verser solved RSA Data Security's DES challenge by exhaustive key search on a large number of computers. This result was useful because it served to underscore in a public way how vulnerable DES has become. However, it may also have left the false impression that one cannot do much better than attacking DES in software with a large distributed effort. The design of DES is such that it is fairly slow in software, but is compact and fast when implemented in hardware. As a result, using software to attack DES gives poor performance compared to what can be achieved in hardware. This applies not only to DES, but also to most other block ciphers, attacks on hash functions, and attacks on elliptic curve cryptosystems. Avoiding efficient hardware-based attacks requires the use of algorithms with sufficiently long keys, such as triple-DES, 128-bit RC5,* and CAST-128.[+]

In this article we assess the cost of DES key search using hardware methods and examine the effectiveness of some proposed methods for thwarting attacks on

---

Michael J. Wiener, Entrust Technologies, 750 Heron Road, Suite E08, Ottawa, Ontario, Canada K1V 1A7

This article first appeared in RSA Laboratories' Autumn 1997 Cryptobytes newsletter; it is reprinted with permission from the author and RSA Data Security, Inc.

* R. Rivest, "The RC5 Encryption Algorithm", Fast Software Encryption--Lecture Notes in Computer Science (1008), pp. 86-96. Springer, 1995.

+ C. Adams, "Constructing Symmetric Ciphers Using the CAST Design Procedure", Designs, Codes and Cryptography, vol. 12, no. 3, pp. 283-316, Nov. 1997. Also available as "The CAST-128 Encryption Algorithm", RFC 2144, May 1997.

Advancing Technology

The best known way to attack DES is to simply try all of the possible 56-bit keys until the correct key is found. On average, one expects to go through about half of the key space. In 1993, a design for an exhaustive DES key search machine including a detailed chip design was published.* A $1 million version of this machine used 57600 key search chips, each capable of testing 50 million keys per second. Overall, the machine could find a DES key in, on average, three and a half hours.

About four and a half years have passed since this design was completed, and according to Moore's Law, processing speeds should have doubled three times in that period. Of course, estimating in this fashion is a poor substitute for the careful analysis and design effort that went into the earlier design. The original chip design was done in a 0.8 micron CMOS process, and with the geometries available today, it is possible to fit four instances of the original design into the same silicon area. In keeping with the conservative approach to estimates in the 1993 paper, we assume here that the updated key search chip's clock speed would increase to only 75 MHz from the original 50 MHz, making the modern version of the chip six times faster for the same cost. It is interesting to note that just 21 of these chips would give the same key searching power as the entire set of computers used by the team who solved the DES challenge.

Today's version of the $1 million machine could find a DES key in, on average, about 35 minutes (one-sixth of 3.5 hours). This time scales linearly with the amount of money spent as shown in the following table.

Key Search Machine Cost	**Expected Search Time**
$10,000	2.5 days
$100,000	6 hours
$1,000,000	35 minutes
$10,000,000	3.5 minutes

Note that the costs listed in the table do not include the cost to design the chip and boards for the machine. Because the one-time costs could be as high as half a million dollars, it does not make much sense to build the cheaper versions of the machine, unless several are built for different customers.

This key search engine is designed to recover a DES key given a plaintext-ciphertext pair for the standard electronic-codebook (ECB) mode of DES. However, the machine can also handle the following modes without modification: cipher-block

---

* 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). Currently available at ftp://ripem.msu.edu/pub/crypt/docs/des-keysearch.ps.

---

chaining (CBC), 64-bit cipher feedback (CFB), and 64- bit output feedback (OFB). In the case of OFB, two consecutive plaintexts are needed. The chip design can be modified to handle two other popular modes of DES, 1-bit and 8-bit CFB, at the cost of a slightly more expensive chip. Fewer chips could be purchased for a $1 million machine causing the expected key search time to go up to 40 minutes for all modes, except 1-bit CFB, which would take 80 minutes, on average.

Programmable Hardware

The costs associated with chip design can present a significant barrier to smalltime attackers and hobbyists. An alternative which has much lower start-up costs is the use of programmable hardware. One such type of technology is the Field Programmable Gate Array (FPGA). One can design a circuit on a PC and download it to a board holding FPGAs for execution. In a report in early 1996,* it was estimated that $50000 worth of FPGAs could recover a DES key in, on average, four months. This is considerably slower than what can be achieved with a chip design, but is much more accessible to those who are not well funded.

Another promising form of programmable hardware is the Complex Programmable Logic Device (CPLD). CPLDs offer less design freedom and tend to be cheaper than FPGAs, but the nature of key search designs seems to make them suitable for CPLDs. Further research is needed to assess whether CPLDs are useful for DES key search.

Avoiding Known Plaintext

The designs described to this point have relied on the attacker having some known plaintext. Usually, a single 8-byte block is sufficient. One method of preventing attacks that has been suggested is to avoid having any known plaintext. This can be quite difficult to achieve. Frequently, data begins with fixed headers. For example, each version of Microsoft Word seems to have a fixed string of bytes that each file begins with.

For those cases where a full block of known plaintext is not available, it is possible to adapt the key search design. Suppose that information about plaintext is available (e.g., ASCII character coding is used), but no full block is known. Then instead of repeatedly encrypting a known plaintext and comparing the result to a ciphertext, we repeatedly decrypt the ciphertext and test the candidate plaintexts against our expectations. In the example where we expect 7-bit ASCII plaintext, only about 1 in 256 keys will give a plaintext which has the correct form. These

* M. Blaze, W. Diffie, R. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener, "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security", currently available at http://www.bsa.org/policy/encryption/cryptographers.html.

keys would have to be tried on another ciphertext block. The added logic to handle this would add just 10 to 20% to the cost of a key search chip.

Even if we only know a single bit of redundancy in each block of plaintext, this is enough to cut the number of possible keys in half. About 56 such blocks are needed to uniquely identify the correct key. This does not mean that the run-time is 56 times greater than the known-plaintext case. On average, each key is eliminated with just two decryptions. Taking into account the cost of the added logic required makes the expected run-time for a $1 million machine about 2 hours in this case.

Frequent Key Changes

A commonly suggested way to avoid key search attacks is to change the DES key frequently. The assumption here is that the encrypted information is no longer useful after the key is changed, which is often an inappropriate assumption. If it takes 35 minutes to find a DES key, why not change keys every 5 minutes? The problem with this reasoning is that it does not take exactly 35 minutes to find a key. The actual time is uniformly distributed between 0 and 70 minutes. We could get lucky and find the key almost right away, or we could be unlucky and take nearly 70 minutes. The attacker's probability of success in the 5-minute window is 5/70 = 1/14. If after each key change the attacker gives up and starts on the next key, we expect success after 14 key changes or 70 minutes. In general, frequent key changes cost the attacker just a factor of two in expected run-time, and are a poor substitute for simply using a strong encryption algorithm with longer keys.

Conclusion

Using current technology, a DES key can be recovered with a custom-designed $1 million machine in just 35 minutes. For attackers who lack the resources to design a chip and build such a machine, there are programmable forms of hardware such as FPGAs and CPLDs which can search the DES key space much faster than is possible using software on PCs and workstations. Attempts to thwart key search attacks by avoiding known plaintext and changing keys frequently are largely ineffective. The best course of action is to use a strong encryption algorithm with longer keys, such as triple-DES, 128-bit RC5, or CAST-128.

12
Authors

In This chapter:

- *The Electronic Frontier Foundation*
- *John Gilmore*
- *Cryptography Research*
- *Paul Kocher*
- *Advanced Wireless Technologies*

The Electronic Frontier Foundation

Electronic Frontier Foundation
1550 Bryant Street, Suite 725
San Francisco CA 94103 USA
+1 415 436 9333 (voice)
+1 415 436 9993 (fax)
http://www.eff.org/
info@eff.org

The Electronic Frontier Foundation (EFF) is a nonprofit public-interest organization protecting rights and promoting liberty online. It was founded in 1990 by Mitchell Kapor, John Perry Barlow, and John Gilmore.

The Foundation seeks to educate individuals, organizations, companies, and governments about the issues that arise when computer and communications technologies change the world out from under the existing legal and social matrix.

The Foundation has been working on cryptography policy for many years. It was a significant force in preventing the adoption of the "Clipper chip" and its follow-on "key escrow" proposals, and continues to advocate for wide public availability and use of uncompromised and unbreakable encryption technology. EFF is backing the lawsuit in which Professor Daniel Bernstein seeks to overturn the United States export laws and regulations on cryptography, arguing that the First Amendment to the US Constitution protects his right to publish his cryptography research results online without first seeking government permission. EFF's research effort in creating this first publicly announced DES Cracker, and the publication of its full technical details, are part of EFF's ongoing campaign to understand, and educate the public about, the social and technical implications of cryptographic technology.

EFF encourages you to join us in exploring how our society can best respond to today's rapid technological change. Please become an EFF member; see http://www.eff.org/join/

John Gilmore

John Gilmore is an entrepreneur and civil libertarian. He was an early employee of Sun Microsystems, and co-founded Cygnus Solutions, the Electronic Frontier Foundation, the Cypherpunks, and the Internet's "alt" newsgroups. He has twenty-five years of experience in the computer industry, including programming, hardware and software design, and management. He is a significant contributor to the worldwide open sourceware (free software) development effort. His advocacy efforts on encryption policy aim to improve public understanding of this fundamental technology for privacy and accountability in open societies. He is currently a board member of Moniker pty ltd, the Internet Society, and the Electronic Frontier Foundation.

John leads the EFF's efforts on cryptography policy, managed the creation of the DES cracker and wrote much of the text in this book.

John can be reached at the email address gnu@des.toad.com; his home page is http://www.cygnus.com/~gnu/

Cryptography Research

Cryptography Research
870 Market Street, Suite 1088
San Francisco, CA 94102 USA
+1 415 397 0123 (voice)
+1 415 397 0127 (fax)
http://www.cryptography.com/

Cryptography Research is Paul Kocher's San Francisco-based consulting company. Cryptography Research provides consulting, design, education, and analysis services to many leading firms and start-ups. Kocher and the company are widely known for their technical work and research, including the development of leading cryptographic protocols (such as SSL 3.0), cryptanalytic work (including the discovery of timing attacks against RSA and other cryptosystems), and numerous presentations at major conferences. To reach Cryptography Research please write to mailto:info@cryptograpy.com.

Cryptography Research managed the hardware and software design for the DES cracker, and wrote the chip simulator and the driver software.

Paul Kocher, Josh Jaffe, and everyone else at Cryptography Research would like to thank John Gilmore and the EFF for funding this unique project, and AWT for their expert hardware work!

Paul Kocher

Paul Kocher is a cryptographer specializing in the practical art of building secure systems using cryptography. He currently serves jointly as President of Cryptography Research (http://www.cryptography.com/) and Chief Scientist of ValiCert (http://www.valicert.com/). Paul has worked on numerous software and hardware projects and has designed, implemented, and broken many cryptosystems. Paul can be reached via e-mail at paul@cryptography.com.

Advanced Wireless Technologies

Advanced Wireless Technologies, Inc.
3375 Scott Blvd, Suite 410
Santa Clara, CA 95054 USA
+1 408 727 5780 (voice)

+1 408 727 8842 (fax)
http://www.awti.com/

Advanced Wireless Technologies, Inc. (AWT) is dedicated to providing Application-Specific Integrated Circuit (ASIC) and board level design solutions for high tech industries at highest quality and lowest cost. AWT's design philosophy is to reduce product development cost/risk and recurring cost. AWT employs a thorough design flow from system architecture to system integration and test.

AWT was founded in 1993. Its engineering team is composed of a highly qualified, tenured employee base, including technical management staff. The employees are knowledgeable, motivated, highly competent, and have from 3 to 25 years of experience in system engineering, chip design, and complete subsystem design.

AWT offers digital ASIC/Gate Array and Board design services to support customers' specific requirements. The company can participate in any development phase from specifications definition to design implementation and prototype testing.

In addition to providing engineering services AWT has developed leading products for use in the communications industry. AWT's standard products include IP Cores, ASICs, and board level products in the fields of demodulation, forward error correction, and encryption decryption.

AWT designed and built the hardware for the DES Cracker, including the custom ASIC, logic boards, and interface adapters. If you're interested in purchasing a DES Cracker unit, contact AWT.

AWT invites you to visit at http://www.awti.com/ or call +1 408 727 5780 for your specific engineering needs.

Note: URLs for other parts welcomed
