Crypto Blunders (v. 1.1)

Steve Burnett, RSA Security Inc.

Abstract: Cryptography has emerged as an enormously important component of the networked world. People are hesitant to trust the World Wide Web and eCommerce without the protections crypto provides. As companies build Virtual Private Networks, demand secure communications and require stronger authentication techniques, more and more applications are built with crypto as core components. Many cryptographic algorithms are almost unbreakable . . . if used properly. If applied incorrectly, it doesn't matter how strong an algorithm or key is, the tools of crypto will provide no protection. This paper will describe some of the blunders people have made over the years in their use of cryptography. Some of these mistakes made headlines, some did not. Some might even be a little humorous -- to those not involved. If nothing else, readers will learn what not to do in their products.

Blunder #1

In 1917, the respected journal Scientific American described the Vigenére Cipher as "impossible of translation." The problem with that statement was that the Union Army in the US Civil War had broken the Vigenére Cipher in the 1860's. [K]

In 1977, Scientific American was the first to publish the RSA algorithm in Martin Gardner's column, "Mathematical Recreations." This article also gave the first RSA Challenge. Ron Rivest, Adi Shamir and Len Adleman (the "R", "S" and "A") encrypted a message using their new algorithm and offered $100 to anyone who could decode it. Gardner claimed the code was a cipher which human ingenuity could not resolve. Not quite as confident as Gardner, Rivest remarked it would take "40 quadrillion years" to crack. They paid up 17 years later[1]. [see Levy]

This leads us to

Crypto Blunder #1: Declare your algorithm unbreakable.

[1] It is only fair to point out that the Ron Rivest 40 quadrillion year statement was actually prefaced with, "Given current technology . . ." Technology, in the form of faster computers and new factoring techniques made the crack possible. Furthermore, at the time, research into developing good security estimates for public key algorithms was practically nonexistent. In fact, after studying the problem further, Rivest did revise his estimate downward. To this day, RSA is still secure, as long as the key is long enough. The 1977 challenge was on a 428-bit key, most uses of RSA today rely on 1024-bit keys.

No crypto algorithm is unbreakable. The best inventors are honest and simply assert that it will, on average, take the attacker too long to crack any individual message. As soon as you declare a cipher unbreakable, cryptanalysts come out of nowhere to shoot down your claim. Surely modern crypto designers have learned that lesson. And yet . . .

A recent survey of crypto products finds several interesting claims. The web site http://www.atlantic-coast.com/ube/ is offering for sale the software package "UnBreakable Encryption." That's the name of the product, "UnBreakable Encryption." Or go to http://www.meganet.com to find out how you can purchase "VME", which promises "Our technology, VME™ (Virtual Matrix Encryption), is quite simply the only unbreakable encryption commercially available." Another company, Top Secret Crypto (http://topsecretcrypto.com) announces, "Until now, unbreakable encryption methods have been possessed by only a few government agencies, such as the National Security Agency (NSA) and the Soviet KGB. With Top Secret Crypto you now have that ability."

Recently, Harvard professor Michael Rabin along with Ph.D. student Yan Zong Ding have announced a new algorithm. "This is the first provably unbreakable code that is really efficient," Dr. Rabin said. "We have proved that the adversary is helpless."[N] "It provides everlasting security." [H] When Dr. Rabin says "provably unbreakable," he's talking about a mathematical proof which the researchers have offered and, they claim, has shown their scheme to be unbreakable.

Unfortunately, "mathematical proof" or "security proof" is not synonymous with "unbreakable". In 1997, IBM announced a new algorithm (dubbed Atjai Dwork, after the inventors, Miklos Atjai and Cynthia Dwork). IBM offered mathematical proofs to bolster its claim of invincibility. Unfortunately, the algorithm was broken the next year. How was anyone able to break an algorithm with the force of mathematical proof behind it? Not by attacking the "math", but the underlying assumptions of the proof. [Co1]

Will The Ding/Rabin algorithm suffer the same fate? Will someone attack their assumptions to dispute the results? It turns out that the new algorithm is simply a one-time pad with a "new" key sharing technique (more on that later). Part of the mathematical proof of unbreakability depends on an assumption of storage capacity of the adversary, but part of it also depends on the invincibility of the one-time pad, the topic of the next blunder.

Blunder #2

There is a belief that "The one-time pad is the only unbreakable encryption algorithm." This is a rather casual description. To be at least a little more rigorous, it would be better to say, "The one-time pad encryption scheme has provable security properties if the pad is random, used only once and the adversary does not have access to the pad."

Here is a simple version of the one-time pad algorithm. Generate a series of random numbers and print two copies of these numbers on two pads. Each correspondent receives one copy of the pad.

To encrypt a message, take the first letter of plaintext and add to it the first random number of the pad. This is the first letter of ciphertext. Then add the second pad number to the second letter of plaintext. And so on. Each letter of plaintext is encrypted with a number from the pad.

	P	**L**	**A**	**I**	**N**	**T**	**E**	**X**	**T**	**. . .**
Pad:	**5**	**10**	**3**	**21**	**0**	**7**	**14**	**14**	**8**	**. . .**
	U	**V**	**D**	**D**	**N**	**A**	**S**	**L**	**B**	**. . .**

Figure 1: An example of a one-time pad. The word "PLAINTEXT" is encrypted to "UVDDNASLB". P+5=U, L+10=V and so on.

To decrypt, subtract the pad numbers from the ciphertext letters. The pad is secure because anyone intercepting a message could try any number of possible pads that "decrypt" to something reasonable. For instance, an attacker could try the pad "7,17,8,25,22,14,4,20,23" with the ciphertext in Figure 1. That would produce plaintext of "NEVERMORE". That's not the correct answer, but it is reasonable. How can anyone know when they've stumbled onto the correct pad?

	U	**V**	**D**	**D**	**N**	**A**	**S**	**L**	**B**	**. . .**
Pad:	**7**	**17**	**8**	**25**	**22**	**14**	**4**	**20**	**23**	**. . .**
	N	**E**	**V**	**E**	**R**	**M**	**O**	**R**	**E**	**. . .**

Figure 2: An example of an incorrect pad producing a reasonable answer. The ciphertext "UVDDNASLB" can be decrypted to "NEVERMORE". U-7=N, V-17=E and so on.

But if the correspondents use the same pad twice, an attacker will be able to try a plausible pad on both messages. Although many pads will produce reasonable answers for each message, only one pad (the correct pad) will produce reasonable results for both messages. This is why it is imperative to use a pad only once.

The next condition is randomness. If the pad is not random, that means some patterns in the pad will exist. An attacker will try only those pads that possess those patterns. Or maybe if portions of the pad can be deduced because some of the plaintext is known or guessed, an attacker may be able to figure out what the next pad numbers will be because the next numbers are not random.

In the 1930's and 1940's, The Soviet Union was using one-time pads to encrypt messages sent to diplomatic missions throughout the world. In 1942, the Soviet crypto center accidentally printed duplicate copies of one-time pads. US cryptanalysts discovered this flaw in 1943 and were able to extract information from many messages sent between 1942 and 1948. [CIA]

This leads us to

Crypto Blunder #2: Worship at the altar of the one-time pad.

Surely modern users of cryptography have learned the lesson that one cannot be casual about one-time pads. And yet . . .

RC4® is a cipher that is similar to a one-time pad. It encrypts by performing an XOR operation on each byte of input with a byte of "key stream." The algorithm generates the key stream "on-the-fly". As you need more stream bytes, RC4 gives them to you. It generates its key stream from the encrypting key. That is, from a 128-bit key, you can build a practically unlimited amount (well, 10^{100} bytes) of key stream.

This key stream is similar to a one-time pad in that it is pseudo-random. Pseudo-random means the output passes tests of randomness, but because it is possible to recreate key streams, the values are not truly random.

If you use the same key twice, you will generate the same key stream. Using the same RC4 key twice is essentially the same as using a one-time pad twice.

In 1998, Microsoft® released an implementation of PPTP, the "Point-to-Point Tunneling Protocol". This was software that allowed users to make "Point-to-Point Protocol (PPP) connections to be tunneled through an IP network, creating a Virtual Private Network (VPN)." [Co2]

In PPTP there is a client and a server. Messages between the two entities can be encrypted using RC4. Microsoft decided to make the encryption of client to server messages independent of the encryption of server to client messages. The client used one RC4 instantiation to encrypt messages to the server, and the server used another RC4 instantiation to encrypt messages to the client.

The problem was that the Microsoft implementation used the same key for both directions. They were using the same RC4 key twice for two messages. [Co2]

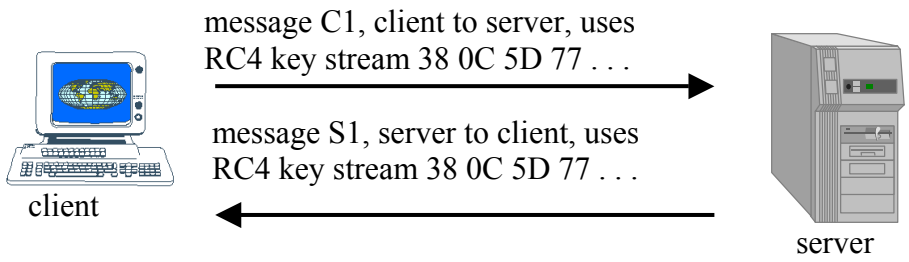


Figure 3: Microsoft's PPTP would encrypt a message from the client to the server using an RC4 key stream. Then it would encrypt the message from the server to the client using the same RC4 key stream, similar to using a one-time pad twice.

Think of it this way, each side had two pads, an encrypting pad for when they sent messages, and a decrypting pad for when they received messages. But the pads were the same. For example, the first message the client would send would use, say, the first 128 bytes of the pad. The server's first message, a response to the client, would use the same first 128 bytes of that pad.

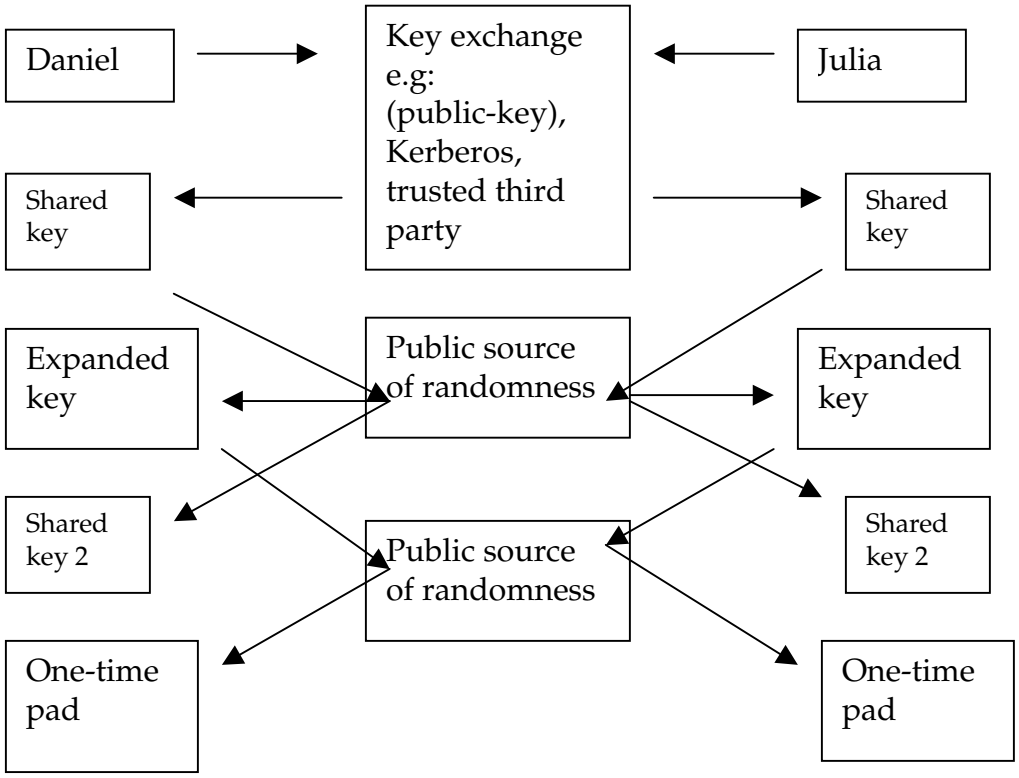
When the US was trying to decrypt Soviet messages, the cryptanalysts did not know which messages were encrypted using duplicate pads, it was hard work to find such message pairs. And not all messages used a duplicate pad. Microsoft was kind enough to let eavesdroppers know which messages were encrypted using the same RC4 key, they were messages in the same session. And all sessions contained such message pairs.

Over the past few years, various people have proposed one-time pads for personal use. Eventually, the problem of pad-sharing is addressed. One common proposal is to use existing music CD's or movie DVD's. You want to share a pad with a friend? Just decide on a movie, and get your pad from the DVD. That is a huge source of pad bytes, right? The problem is that the data is not random. The attacker tries a pad that produces a reasonable answer. Now run that pad through a DVD player. Does it produce a movie? No, it's not the correct pad[2].

In the Ding/Rabin scheme mentioned earlier, the claim of unbreakability is based partly on the one-time pad. Converting plaintext to ciphertext is indeed done by using a one-time pad, but the problem is that the pad is available to the adversary as well. The

[2] Actually, that's a simplified explanation of an attack on DVD pads, but not that far off, either.

proposal is for the two participants, call them Daniel and Julia[3], to share a secret key, use this shared key along with a public source of randomness (such as a satellite beaming a continuous stream of random bytes) in an expansion algorithm to create two things. First, they create a common expanded key which will be megabytes in length, and second, another shared secret key to produce a new expanded key later on. Then when the two want to communicate with each other, they use the expanded key in the expansion algorithm to create a one time-pad. When they run out of expanded key, they use the second shared secret key to generate a new expanded key.



The expansion algorithm must be secure, otherwise, the eavesdropper (call her Satomi[4]) might later on figure out what the original shared secret was, and use the same expansion algorithm to determine the expanded key. The security of the expansion algorithm lies in volume. The expansion algorithm instructs the participants to read bytes of data from the public source, extract only one bit every so often based on the input key (shared secret key is input to produce expanded key, expanded key is input to produce pad), and throw away the rest. Daniel and Julia will know which bits to extract, because they both possess

[3] This author is tired of Alice and Bob.
[4] This author is tired of Eve as well.

the shared secret key or the expanded key. Any other bits that come from the public source they throw away. Satomi doesn't know which bits to extract, so she would have to save all the random bytes (terabytes worth of data). That is, she knows the correct bits are in there somewhere, but where? Maybe a known plaintext attack will help her find them. But she has to have something to work with (a broken key or message) before she can extract bits. Until then, she has to save all the bits.

The security proof says there is a point beyond which Satomi simply cannot store the data necessary to extract the appropriate bits. If you make sure your expansion algorithm is breakable only beyond that point, then Satomi cannot extract bits.

Will this algorithm be broken? Maybe, maybe not. Julien Marcil is a cryptographer with RSA Security in Europe. His graduate research is in the same field as the Ding/Rabin algorithm. He states, "a proof of security depends totally on the model in which you work. The proof that you can or cannot break a cryptosystem is a mathematical statement. Whether the model fits reality is another question."

Blunder #3

The 1700's saw in Europe a rise in the use of what were known as "Black Chambers". These were places where mail was intercepted. Since a government was the operator of the post office, it was easy to divert mail sent to and from foreign embassies and officials. Experts (hiding in these black chambers) would carefully open the letters, copy the contents, replace the items into the envelopes and, if necessary, apply a forged seal.

All governments knew which countries had black chambers and generally established procedures to encrypt correspondence. The algorithms chosen were often cracked, so important secrets were sometimes uncovered. [K]

Frequently, the encryptors knew when the algorithms they had chosen had been broken. Oddly enough, many continued to use those broken methods.

Odder yet was that the Vigenére cipher had been invented over 100 years prior to this era and had not been broken yet. And still, governments chose not to employ it.

This leads us to

Crypto Blunder #3: Don't use the best possible algorithms available.

Today there are many encryption algorithms available, some cost money, others are free. Whether a company is willing to spend money or not, there are plenty of options. So

surely developers today would not choose an unsecure or unknown algorithm. And yet . . .

When Microsoft was designing the Operating System "NT", they came up with a way to protect passwords. It involved hashing the password. At the time, the two most well-known and used hash algorithms were MD2 and MD5. These two had been very widely studied and tested.

Microsoft decided to use its own algorithm, called "LANMAN hash." It turned out to be so weak, researchers were able to crack passwords in seconds. [La]

When Microsoft decided to use a different hash algorithm, they chose MD4. Researchers had very early expressed doubts about the security of MD4 and recommended no one use it. In fact, even the developers of MD4 suggested it not be used.

The question is, with MD5 available (and no intellectual property rights attached), why did Microsoft choose to create their own algorithm? And when they finally decided to upgrade, why did they choose MD4? At this point, there are more algorithms to choose from, namely SHA-1 and RIPEMD. Why not use those?

Another story on algorithm choice is the DVD hack. This story is a little longer.

DVD stands for "Digital Video Disc". Just as music CD's replaced records and tapes, DVD's are supposed to replace video tape, since they provide a better quality viewing experience. Actually, DVD's are touted to replace CD's (both music and computer) as well.

Film companies, however, are worried about piracy since computers can easily copy the contents of the discs. For protection from thieves, then, movies are encrypted, and each licensed DVD player, whether a hardware device or a software package, has its own unique unlock key. Each movie is encrypted using a different key (the movie key), which is then encrypted using the unlock keys of each licensed DVD player. That means each disc comes preloaded with hundreds of copies of the movie key, each copy encrypted with a different unlock key.

When the legitimate consumer uses a valid DVD player, the player will find a special location on the disc containing all the copies of the movie key. It finds the copy that had been encrypted with its (that particular player's) unlock key. Using its unlock key, the licensed player can decrypt the movie key and using the movie key it can decrypt the movie and then, of course, play it.

This collection of encrypted movie keys is on a section of the disc that is supposedly copy protected. That is, while the movie itself can be copied from the disc to another medium (a computer hard drive or another disc), the movie key cannot.

If a licensed DVD player is fed a movie which does not have the list of movie keys, it will not play. So if a movie has been decrypted and copied, licensed players will not play that movie.

Suppose a thief wanted to sell copies of DVD movies to people with licensed players. This thief would have to make sure the portion of the disc containing all the copies of the encrypted movie key exists on the pirate disc. If the legal disc is truly read protected and the thief cannot extract them from the disc, then the thief must figure out what all the unlock keys are.

If the algorithm is secure, that won't be possible in a reasonable amount of time.

One group of programmers in Norway were able to quickly find many unlock keys. They discovered that one of the DVD players had neglected to protect its unlock key. That was one key. Using that key as a sarting point, they were able to quickly find the unlock keys for well over one hundred other players.

They attributed the ease of finding unlock keys to two design flaws. First, the encryption algorithm was extremely weak. It was a new, proprietary algorithm called CSS. As Jon Johansen, one of the Norwegian programmers, put it, "I wonder how much they paid for someone to actually develop that weak algorithm. It's a very weak encryption algorithm." [P]

Why did the designers choose an unproven algorithm? They probably did not know in advance that it would be so weak, but they had no way of knowing whether it would be strong enough either. With so many good, well-tested algorithms on the market (Triple-DES, RC2®, RC5™, Blowfish, CAST, and on and on), why use a new, proprietary algorithm? Was money an issue? After all, some of the known good algorithms are not free, they have to be licensed. But the DVD people paid someone to build a new algorithm, it might have been cheaper to license technology than invent a new, untested algorithm. And besides, some of the good algorithms are indeed free.

The second problem was that the DVD designers used a 40-bit key. It had been shown as early as 1995 that 40-bit keys were woefully inadequate to protect secrets. Why then use 40-bit keys? For export reasons? The US government used to place severe limitations on the export of crypto. Using 40-bit keys made export easier. However, At the time DVD was released, it was fairly simple to get permission to release 48-bit encryption. With a little more work, it was possible to get 56-bit or even 64-bit export licenses. Furthermore,

when an application only decrypted, never encrypted anything, it was often possible to get 128-bit export permission.

Why did the DVD management make the decisions they made? Why did they choose not to use the best algorithms available?

Blunder #4

During the 1920's and 30's, the Japanese used what was known as the "Red" cipher to encrypt messages sent to and from diplomatic missions. US codebreakers had figured out ways to quickly decrypt those messages. In response, the Japanese developed a new cipher. The US cryptanalysts called this new system "Purple". It was introduced right before World War II began.

Purple was far superior to Red. The Japanese were confident the US could not break their messages.

But the US was able to crack the system. How? Because when the machine was first introduced, users made mistakes in implementation. One particularly damaging mistake came about in 1941, when "the Manila legation repeated a telegram 'because of a mistake on the plugboard.'" [K]

This leads us to

Crypto Blunder #4: Implement the algorithm incorrectly.

No matter how good the algorithm is, it won't provide any security if it is implemented improperly. By now, surely engineers have learned that you have to be very careful in implementation. And yet . . .

Briefly, here is how the RSA algorithm works,

Find two primes, p and q, multiply them together to produce a modulus n.
Decide on a public exponent, e, and find the private exponent,
d = inverse of e mod $(p-1)(q-1)$.
To encrypt message m and produce ciphertext c, perform exponentiation:
$c = m^e \bmod n$.
To decrypt:
$d = c^d \bmod n$.

A customer of RSA Security once called Tech Support claiming that the software they purchased was not encrypting the data. The ciphertext was the same as the plaintext. He

sent a sample program displaying the error. The problem was that he had chosen 1 as his public exponent. He was encrypting by finding

$c = m^1 \bmod n$.

Blunder #5

During World War II, the German military used the encryption machine Enigma. The British codebreaking unit at Bletchley Park had broken the Enigma cipher using captured machines. However, Admiral Dönitz, commander of the German Navy, suspected Enigma messages were not secure, and in February of 1942, instituted a more rigorous use of the device. The British were able to break Army and Air Force messages, but not those of the Navy.

An Enigma machine must be set with a key before it can work. A key for that cipher was simply a series of letters. The Army and Air Force instructed the radio operators to come up with "random" keys. They were using 3 or maybe 6 letter keys, easily breakable using the brute-force machine designed by Alan Turing. Actually, often the codebreakers did not even need to use the brute-force attack, they could guess many of the keys. Some operators used their own initials or the initials of their wives or girlfriends, others used letters that spelled out words, such as BERLIN[5].

What Dönitz had introduced was a better keying system. The keys were longer, they were generated in a relatively random manner, and they were changed often. The extra precautions were working, The codebreakers were not able to break the Navy's message. They needed the keys.

The keys were printed on water-soluble paper. If captured, officers and radio operators were instructed to destroy the machines and codebooks. Simply throwing them overboard would be sufficient.

But in October of 1942, the British captured the submarine U-559. The radio operator made sure he saved the letters he had written to his girlfriend, but did not bother to destroy the codebook. The British retrieved the book and broke the Naval code.

Just as important as the encryption algorithm itself is the key, which leads us to

Crypto Blunder #5: Don't protect the key.

[5] This is a pair of blunders in itself, expecting humans to be random and using small passwords.

Have we learned that lesson? Peter Gutmann of the University of New Zealand showed in 1995 that the Password-Based Encryption (PBE) technique used in Windows for Workgroups software was not secure. The technique was being used to protect passwords in password list (PWL) files. Backing up his claim that the technique was flawed, Gutmann produced code that could crack passwords, sometimes in a matter of seconds. Microsoft responded by saying, "The password list file is encrypted with an algorithm that meets the U.S. government Data Encryption Standard (DES). This encryption technology is the highest security allowed in software exported from the United States." No one contended that the algorithm was bad, just that the PBE scheme was weak.

In 1996, Gutmann announced the attack on the PWL files could be extended to recover server private keys in Netscape products. Netscape was using similar techniques to protect keys that Microsoft was using to protect passwords. Netscape did replace their key protection code at about the same time (unrelated to the announcement, according to Gutmann).

Finally, in 1997, Gutmann demonstrated that private keys in Micorsoft's Internet Explorer were being protected by the same technique as the old PWL files and were susceptible to attack. Microsoft offered a new key-protection method, but Gutmann showed that it was not much better.[G]

When designing a system, it is vitally important to spend as much time or even more on the problem of key protection as is spent on the actual encryption algorithm itself.

Conclusion

There have been many other crypto blunders throughout history, this paper simply provided a few choice stories. With any luck, though, readers will be able to learn from others' past mistakes and avoid their own crypto blunders.

References

[Co1] Counterpane Systems, “Crypto-Gram”, Sept. 15, 1998, see http://www.counterpane.com/crypto-gram-9809.html

[Co2] Counterpane Systems, “Cryptanalysis of Microsoft’s PPTP Authentication Extensions”, Oct. 19, 1999, see http://www.counterpane.com/pptpv2-paper.html

[CIA] US Central Intelligence Agency: Center for the Study of Intelligence, Venona: Soviet Espionage and the American Response, 1939-1957, 1996, see http://www.cia.gov/csi/books/venona/venona.htm

[G] Gutmann, Peter, “How to recover private keys for Microsoft Internet Explorer . . .", found at http://www.cs.auckland.ac.nz/~pgut001/pubs/breakms.txt

[H] Harvard Gazette, April 26, 2001

[K] Kahn, David, The Codebreakers, 1996

[Mo] Momsen, Bill, Codebreaking and Secret Weapons in World War II, © 1993 - 1999 Nautical Brass, see http://members.aol.com/nbrass/3enigma.htm

[La] Laamanen, Petteri, NT Server Security, Helsinki University of Technology, see http://www.tcm.hut.fi/Opinnot/Tik-110.501/1997/nt_server.html

[N] New York Times, Feb. 20, 2001

[Levy] Levy, Steven, “Wisecrackers,” Wired, March 1996, found in the “Wired Archive 4.03”

[P] Petrizio, Andy, “Why the DVD Hack Was a Cinch,” Wired, Nov. 2, 1999, found on WiredNews, http://www.wired.com/news/technology/0,1282,32263,00.html

RC2 and RC4 are registered trademarks and RC5 and RSA are trademarks of RSA Security Inc. All other trademarks mentioned herein are the property of their respective owners.

© copyright 2001, RSA Security Inc. All rights reserved.