Crypto Blunders

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 e-commerce 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 virtually 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 1860s. [K]

During a visit to the United States in 1930, German officials demonstrated their code machine, Enigma, to Maj. P.W. Evans of the US Army Signal Corps. They boasted to Maj. Evans that this encryption device was "unbreakable." [Mo] During World War II, the Luftwaffe High Command sent a message to a field officer assuring him Enigma was "unbreakable." This message was encrypted using Enigma. How do we know such a message was sent? Because British code breakers at Bletchley park were able to decrypt it shortly after it was intercepted. [Lipp]

In 1977, Scientific American is 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. [see Levy]

This leads us to

Crypto Blunder #1:
Declare your algorithm unbreakable.

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/index.htm to find out how you can purchase "VME", which promises "100% security for $100 only."

Are these products destined for the same fate as other unbreakable algorithms? Counterpane Systems, Bruce Schneier's company, reported on TriStrata, a company in Redwood City, California, that offered unbreakable encryption, only to quickly amend their claims after further scrutiny. This is an example of someone learning their lesson the hard way. [Co1]

Incidentally, it is only fair to add this post-script to the Ron Rivest 40 quadrillion year statement. The claim 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.

Blunder #2

Speaking of unbreakable cryptography, 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 and used only once."

Here is a simple version of the one-time pad algorithm. Generate a series of random numbers, printing two copies on two pads. Each correspondent receives one of the copies of the pad, this list of random numbers.

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.

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?

	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.

	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 1930s and 1940s, 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: Use a one-time pad more than once.

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].

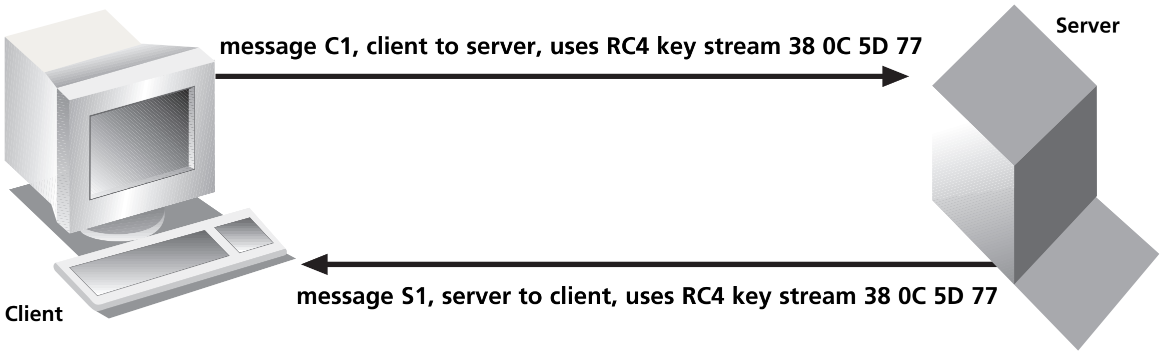
message C1, client to server, uses RC4 key stream 38 0C 5D 77
Server
message S1, server to client, uses RC4 key stream 38 0C 5D 77
Client

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.

Blunder #3

The 1700s 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 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 CDs replaced records and tapes, DVDs are supposed to replace video tape, since they provide a better quality viewing experience. Actually, DVDs are touted to replace CDs (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 starting 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 encryption exported.

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 1920s and 30s, the Japanese used what was known as the "Red" cipher to encrypt messages sent to and from diplomatic missions. US code breakers 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 crypto engineers (the people who build the systems that cryptographers design) have learned that you have to be very careful in implementation. And yet . . .

When Sun Microsystems released JDK 1.1 (the "second" generation of the Java language), they included an implementation of DSA. The letters stand for "Digital Signature Algorithm." Sun chose to make it a part of the JDK because they wanted a digital signature algorithm as part of the base release and DSA was really the only algorithm they could use. Other algorithms (such as RSA, El Gamal and elliptic curves) are subject to US export controls or have intellectual property restrictions or both. DSA cannot encrypt or perform key exchange, only create and verify signatures, so is exportable. Furthermore, it is widely believed that DSA has no intellectual property attachments, such as patent license requirements.*

There is a part of DSA called "random *k*." This random value is used to create the digital signature. Each signature must use a different random *k*. If an implementation uses the same random *k* for two signatures, it is very easy to determine the private key. Normally, to break DSA, one must solve the discrete log problem. But if the same random *k* is used in two signatures, to break DSA, one must simply apply some high school algebra.

The developers at Sun who wrote the Java DSA code originally used a hard-coded random *k*. They figured they would later on solve the problem of generating a new *k* for each signature. They forgot. The code was released with that hard-coded random *k*. All signatures used the same *k*.

It's not that the programmers were ignorant of the way DSA worked, they just made an implementation mistake. Sun did fix it in JDK version 1.1.2. Unfortunately, the fix meant that it took four or five seconds to compute a single signature (most applications need a signature computation to take just a few milliseconds).

For some applications, an implementation mistake can simply mean something doesn't look right, for instance a window does not resize properly or a font is wrong. But with cryptography, an implementation mistake makes the application useless. Actually, it makes the application dangerous.

Incidentally, here are some of the hard-coded random *k*'s as computed by the author. Sun used a different random *k* for the different key sizes.

DSA with 512-bit keys used *k* =
66 d1 f1 17 51 44 7f 6f 2e f7 95 16 50 c7 38 e1
85 0b 38 59

DSA with 1024-bit keys used *k* =
65 a0 7e 54 72 be 2e 31 37 8a ea 7a 64 7c db ae
c9 21 54 29

*Author's note: I am not a lawyer. Consult an expert on intellectural property if you want to know whether DSA is free to use or whether licensing is required. There is dispute as to whether a license to a patent by Claus Schnorr is required before using DSA in a commercial application.

There are more random *k*'s for more key sizes. Computation of those is left as an exercise to the reader.

Blunder #5

The Swiss company Crypto AG sells cryptographic products. Many governments bought their hardware products for use in encrypting messages to and from diplomatic missions throughout the world.

In 1992, a Crypto AG sales representative, Hans Buehler, was arrested in Iran. The Iranian officials accused him of spying. They contended the Crypto AG machinery had a "back door" built in. Buehler reported that he "…was questioned for five hours a day for nine months. I was never beaten, but I was strapped to wooden benches and told I would be beaten."

It turns out there was indeed a back door in the Crypto AG equipment. Each time someone used the product, the encryption key could be "clandestinely transmitted with the enciphered message." [Ma]

This leads us to

Crypto Blunder #5: Put a back door into your product.

By now, anyone trying to sell crypto products knows not to put back doors into the devices. People don't like to buy crypto products with back doors. Some people are so disgusted with back doors they arrest and "intensely question" people who do put back doors into their products. And yet . . .

In 1993, the US government offered the Clipper chip. This was a cryptographic device to be used on phones, in computers, on networks, anywhere information would be encrypted. Coming from the US government, it seemed probable there would be a back door. Probable? The US government advertised that feature. It announced up front that anything encrypted could be decrypted by the US government.

The Clipper is no longer in production.

Incidentally, the Crypto AG sales representative, Hans Buehler, did not know there was a back door in the product he was selling.

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, *TriStrata*, Crypto-Gram Newsletter, Oct. 15, 1998, see http://www.counterpane.com/crypto-gram-9810.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

[K] Kahn, David, The Code breakers, 1996

[Ma] Madsen, Wayne, *Crypto AG: The NSA's Trojan Whore?*, Covert Action Quarterly, no. 63, Winter 1997-1998, see http://mediafilter.org/caq/cryptogate/

[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

[Lipp] Lippman, David, World War II Plus 55, December 27, 1941 - January 3, 1942, see http://home.flash.net/~hfwright/dl27de41.htm

[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

RSA SECURITY™

Corporate Headquarters:
20 Crosby Drive, Bedford, MA 01730 USA
Tel 877 RSA 4900 or 781 301 5000, Fax 781 301 5170

OEM and Developer Solutions Group:
California, USA, Tel 800 PUBLIKEY or 650 295 7600, Fax 650 295 7700

Europe, Middle East and Africa Headquarters:
United Kingdom, Tel +44 118 936 2600, Fax +44 118 936 2790

Asia/Pacific Headquarters:
Singapore, Tel +65 733 5400, Fax +65 733 2400
Japan, Tel +81 3 3539 7511, Fax +81 3 3539 7514

www.rsasecurity.com   info@rsasecurity.com