Chapter 14 What Can Go Wrong

The mathematics behind cryptosystems is only part of the picture. Implementation is also very important. The best systems, when used incorrectly, can lead to serious problems. And certain design considerations that might look good at the time can later turn out to be bad.

We start with a whimsical example that (we hope) has never been used in practice. Alice wants to send a message to Bob over public channels. They decide to use the three-pass protocol (see Section 3.6).

Here is a physical description. Alice puts her message in a box, puts on her lock, and send the box to Bob. Bob puts on his lock and sends the box back to Alice, who takes her lock off the box and sends the box to Bob. He takes off his lock and opens the box to retrieve the message. Notice that the box always is locked when it is in transit between Alice and Bob.

In Section 3.6, we gave an implementation using modular exponentiation. But Alice and Bob know that the one-time pad is the most secure cryptosystem, and they want to use only the best. Therefore, Alice and Bob each choose their own one-time pads, call them K_A and K_B. Alice's message is M. She encrypts it as $C_1 = M \oplus K_A$ and sends C_1 to Bob, who computes $C_2 = C_1 \oplus K_B$ and sends C_2 back to Alice. Then Alice removes her "lock" K_A by computing $C_3 = C_2 \oplus K_A$ and sends C_3 to Bob. He removes his lock by computing $M = C_3 \oplus K_B$.

Meanwhile, Eve intercepts C_1, C_2, C_3 and computes

$$C_1 \oplus C_2 \oplus C_3 = (M \oplus K_A) \oplus (M \oplus K_A \oplus K_B) \oplus (M \oplus K_B) = M.$$

The moral of the story: even the best cryptosystem can be insecure if not used appropriately.

In this chapter, we describe some situations where mistakes were made, leading to security flaws.

14.1 An Enigma “Feature”

For its time, the Enigma machine (see Section 2.7) was an excellent system, and the German troops’ belief in this led to security breaches. After all, if your post is being overrun or your ship is sinking, are you going to try to escape, or are you going to risk your life to destroy a crypto machine that you’ve been told is so good that no one can break the system?

But there were also other lapses in its use. One of the most famous took advantage of an inherent “feature” of Enigma. The design, in particular the reflector (see Section 2.7), meant that a letter could not be encrypted as itself. So an A would never encrypt as an A, and a B would never encrypt as a B, etc. This consequence of the internal wiring probably looked great to people unfamiliar with cryptography. After all, there would always be “complete encryption." The plaintext would always be completely changed. But in practice it meant that certain guesses for the plaintext could immediately be discarded as impossible.

One day in 1941, British cryptographers intercepted an Enigma message sent by the Italian navy, and Mavis Batey, who worked at Bletchley Park, noticed that the ciphertext did not contain the letter L. Knowing the feature of Enigma, she guessed that the original message was simply many repetitions of L (maybe a bored radio operator was tapping his finger on the keyboard while waiting for the next message). Using this guess, it was possible to determine the day’s Enigma key. One message was “Today’s the day minus three.” This alerted the British to a surprise attack on their Mediterranean fleet by the Italian navy. The resulting Battle of Cape

Matapan established the dominance of the British fleet in the eastern Mediterranean.

14.2 Choosing Primes for RSA

The security of the RSA cryptosystem (see Chapter 9) relies heavily on the inability of an attacker to factor the modulus $n = pq$. Therefore, it is very important that p and q be chosen in an unpredictable way. This usually requires a good pseudorandom number generator to give a starting point in a search for each prime, and a pseudorandom number generator needs a random seed, or some random data, as an input to start its computation.

In 1995, Ian Goldberg and David Wagner, two computer science graduate students at the University of California at Berkeley, were reading the documentation for the Netscape web browser, one of the early Internet browsers. When a secure transaction was needed, the browser generated an RSA key. They discovered that a time stamp was used to form the seed for the pseudorandom number generator that chose the RSA primes. Using this information, they were able to deduce the primes used for a transaction in just a few seconds on a desktop computer.

Needless to say, Netscape quickly put out a new version that repaired this flaw. As Jeff Bidzos, president of RSA Data Security pointed out, Netscape "declined to have us help them on the implementation of our software in their first version. But this time around, they've asked for our help" (*NY Times*, Sept. 19, 1995).

Another potential implementation flaw was averted several years ago. A mathematician at a large electronics company (which we'll leave nameless) was talking with a colleague, who mentioned that they were about to put out an implementation of RSA where they saved time by

choosing only one random starting point to look for primes, and then having p and q be the next two primes larger than the start. In horror, the mathematician pointed out that finding the next prime after $\sqrt{n}$, a very straightforward process, immediately yields the larger prime, thus breaking the system.

Perhaps the most worrisome problem was discovered in 2012. Two independent teams collected RSA moduli from the web, for example from X-509 certificates, and computed the gcd of each pair (see [Lenstra2012 et al.] and [Heninger et al.]). They found many cases where the gcd gave a nontrivial factor. For example, the team led by Arjen Lenstra collected 11.4×10^6 RSA moduli. They computed the gcd of each pair of moduli and found 26965 moduli where the gcd gave a nontrivial factor. Fortunately, most of these moduli were at the time no longer in use, but this is still a serious security problem. Unless the team told you, there is no way to know whether your modulus is one of the bad ones unless you want to compute the gcd of your modulus with the 6.4×10^6 other moduli (this is, of course, much faster than computing the gcd of all pairs, not just those that include your modulus). And if you find that your modulus is bad, you have factored someone else's modulus and thus have broken their system.

How could this have happened? Probably some bad pseudorandom number generators were used. Let's suppose that your pseudorandom number generator can produce only 1 million different primes (this could easily be the case if you don't have a good source of random seeds; [Heninger et al.] gives a detailed analysis of this situation). You use it to generate 2000 primes, which you pair into 1000 RSA moduli. So what could go wrong?

Recall the Birthday Paradox (see Section 12.1). The number of "birthdays" is $N = 10^6$ and the number of "people" is $2000 = 2\sqrt{N}$. It is likely that two "people"

have the same “birthday.” That is, two of the primes are equal (it is very unlikely that they are used for the same modulus, especially if the generating program is competently written). Therefore, two moduli will share a common prime, and this can be discovered by computing their gcd.

Of course, there are enough large primes that a good pseudorandom number generator will potentially produce so many primes that is is unlikely that a prime will be repeated. As often is the case in cryptography, we see that security ultimately relies on the quality of the pseudorandom number generator. Generating randomness is hard.

14.3 WEP

As wireless technology started to replace direct connections in the 1990s, the WEP (Wired Equivalent Privacy) Algorithm was introduced and was the standard method used from 1997 to 2004 for users to access the Internet via wireless devices via routers. The intent was to make wireless communication as secure as that of a wired connection. However, as we'll see, there were several design flaws, and starting in 2004 it was replaced by the more secure WPA (Wi-Fi Protected Access), WPA2, and WPA3.

The basic arrangement has a central access point, usually a router, and several laptop computers that want to access the Internet through the router. The idea is to make the communications between the laptops and the router secure. Each laptop has the same WEP key as the other users for this router.

A laptop initiates communication with the router. Here is the protocol.

1. The laptop sends an initial greeting to the router.

2. Upon receiving the greeting, the router generates a random 24-bit IV (= initial value) and sends it to the laptop.

3. The laptop produces a key for RC4 (a stream cipher described in Section 5.3) by concatenating the WEP key for this router with the IV received from the router:

$$\text{RC4Key} \;=\; \text{WEPKey} \,||\, \text{IV}.$$

   and uses it to produce the bitstream *RC4*.

4. The laptop also computes the checksum $ch = CRC32\,(Message)$. (See the description of CRC-32 at the end of this section.)

5. The laptop forms the ciphertext

$$C = \text{ RC4 } \oplus \text{ (Message, ch)},$$

which it sends to the router along with the IV (so the router does not need to remember what IV it sent).

6. The router uses the WEPKey and the IV to form

$$RC4Key = WEPKey \parallel IV,$$

and then uses this to produce the bitstream $RC4$.

7. The router computes $C \oplus \text{ RC4 } = (\text{message, ch})$.

8. If $ch = CRC32\,(Message)$, the message is regarded as authentic and is sent to the Internet. If not, it is rejected.

Notice that, except for the authentication step, this is essentially a one-time pad, with RC4 supplying the pseudorandom bitstream.

Usually, the WEP key length was 40 bits. Why so short? This meant there were $2^{40} \approx 10^{12}$ possible keys, so a brute force attack could find the key. But back when the algorithm was developed, U.S. export regulations prohibited the export of more secure cryptography, so the purpose of 40 bits was to allow international use of WEP. Later, many applications changed to 104 bits. However, it should be emphasized that WEP is considered insecure for any key size because of the full collection of design flaws in the protocol, which is why the Wi-Fi Alliance subsequently developed new security protocols to protect wireless communications.

The designers of WEP knew that reusing a one-time pad is insecure, which is why they included the 24-bit IV. Since the IV is concatenated with the 40-bit WEP key to form the 64-bit RC4Key (or $24 + 104 = 128$ bits in more recent versions), there should be around $2^{24} \approx 1.7 \times 10^7$ keys used to generate RC4 bitstreams. This might seem secure, but a highly used router might be expected to use almost every possible key in a short time span.

But the situation is even worse! The Birthday Paradox (see Section 12.1) enters again. There are $N = 2^{24}$ possible IV values, and $\sqrt{N} = 2^{12} = 4096$. Therefore, after, for example, 10000 communications, it is very likely that some IV will have been used twice. Since the IV is sent unencrypted from the router to the laptop, this is easy for Eve to notice. There are now two messages encrypted with the same pseudorandom one-time pad. This allows Eve to recover these two messages (see Section 4.3). But Eve also obtains the RC4 bitstream used for encryption. The IV and this bitstream are all that Eve needs in Step (e) of the protocol to do the encryption. The WEP key is not required once the bitstream corresponding to the IV is known. Therefore, Eve has gained access to the router and can send messages as desired.

There is even more damage that Eve can do. She intercepts a ciphertext $C = \text{RC4} \oplus (\text{Message, ch})$, and we are not assuming this RC4 was obtained from a repeated IV value. So the C is rather securely protecting the message. But suppose Eve guesses correctly that the message says

Send to IP address 69.72.169.241. *Here is my credit card number, etc.*

Eve's IP address is 172.16.254.1. Let M_1 be the message consisting of all 0s except for the XOR of the two IP addresses in the appropriate locations. Then

$M_2 = M \oplus M_1$
$=$ *Send to IP address* 172.16.254.1. *Here is my credit card number, etc.*

Moreover, because of the nature of CRC-32,

$$ch(M_2) = ch(M) \oplus ch(M_1).$$

Eve doesn't know M, and therefore, doesn't know M_2. But she does know $ch(M_1)$. Therefore, she takes the original $C = RC4 \oplus (M, ch(M))$ and forms

$$C \oplus (M_1, ch(M_1)) = RC4 \oplus (M, ch(M)) \oplus (M_1, ch(M_1)).$$

Since $M \oplus M_1 = M_2$ and

$$ch(M) \oplus ch(M_1) = ch(M \oplus M_1) = ch(M_2),$$

she has formed

$$RC4 \oplus (M_2, ch(M_2)).$$

This will be accepted by the router as an authentic message. The router forwards it to the Internet and it is delivered to Eve's IP address. Eve reads the message and obtains the credit card number.

This last flaw easily could have been avoided if a cryptographic hash function had been used in place of CRC-32. Such functions are highly non-linear (so it is unlikely that $ch(M \oplus M_1) = ch(M) \oplus ch(M_1)$), and it would be very hard to figure out how to modify the checksum so that the message is deemed authentic.

The original version of WEP used what is known as Shared Key Authentication to control access. In this version, after the laptop initiates the communication with a greeting, the router sends a random challenge bitstring to the laptop. The laptop encrypts this using the WEP key as the key for RC4 and then XORing the challenge with the RC4 bitstring:

$$\text{RC4(WEPKey)} \oplus Challenge.$$

This is sent to the router, which can do the same computation and compare results. If they agree, access is granted. If not, the laptop is rejected.

But Eve sees the Challenge and also RC4(WEPKey) $\oplus$ Challenge. A quick XOR of these two strings yields RC4(WEPKey). Now Eve can respond to any challenge, even if she does not know the WEP key, and thereby gain access to the router.

This access method was soon dropped and replaced with the open access system described above, where anyone

can try to send a message to the system, but only the ones that decrypt and yield a valid checksum are accepted.

14.3.1 CRC-32

CRC-32 (= 32-bit Cyclic Redundancy Check) was developed as a way to detect bit errors in data. The message M is written in binary and these bits are regarded as the coefficients of a polynomial mod 2. For example, the message 10010101 becomes the polynomial $x^7 + x^4 + x^2 + 1$. Divide this polynomial by the polynomial

$$p(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1.$$

Let $r(x)$ be the remainder, which is a polynomial of degree at most 31. Reduce the coefficients of $r(x)$ mod 2 to obtain a binary string of length 32 (if the degree is less than 31, the binary string will start with some 0s corresponding to the 0-coefficients for the high degree terms). For an example of such polynomial division, see Section 3.11. This binary string is the checksum $ch(M)$.

Adding two polynomials mod 2 corresponds to computing the XOR of the corresponding bitstrings formed from their coefficients. For example, the polynomial $x^7 + x^4 + x^2 + 1$ corresponds to 10010101 and $x^4 + x + 1$ corresponds to 00010011, with a few 0s added to match the first string. The sum of the two polynomials mod 2 is $x^7 + x^2 + x$, which corresponds to 10000110 = 10010101⊕00010011. Since dividing the sum of two polynomials by $p(x)$ yields a remainder that is the sum of the remainders that would be obtained by dividing each polynomial individually, it can be deduced that

$$ch(M_1 \oplus M_2) = ch(M_1) \oplus ch(M_2)$$

for any messages M_1, M_2.

14.4 Exercises

1. The Prime Supply Company produces RSA keys. It has a database of 10^8 primes with 200 decimal digits and a database of 10^{10} primes with 250 decimal digits. Whenever it is asked for an RSA modulus, it randomly chooses a 200-digit prime p and a 250-digit prime q from its databases, and then computes $n = pq$. It advertises that it has 10^{18} possible moduli to distribute and brags about its great level of security. After the Prime Supply Company has used this method to supply RSA moduli to 20000 customers, Eve tries computing gcd's of pairs of moduli of these clients (that is, for each pair of clients, with RSA moduli n' and n'', she computes $\gcd(n', n'')$). What is the likely outcome of Eve's computations? Explain.

2. The Modulus Supply Company sells RSA moduli. To save money, it has one 300-digit prime p and, for each customer, it randomly chooses another 300-digit prime q (different from p and different from q supplied to other customers). Then it sells $n = pq$, along with encryption and decryption exponents, to unsuspecting customers.

    1. Suppose Eve suspects that the company is using this method of providing moduli to customers. How can she read their messages? (As usual, the modulus n and the encryption exponent e for each customer are public information.)

    2. Now suppose that the customers complain that Eve is reading their messages. The company computes a set S of 10^6 primes, each with 300 digits. For each customer, it chooses a random prime p from S, then randomly chooses a 300-digit prime q, as in part (a). The 100 customers who receive moduli $n = pq$ from this update are happy and Eve publicly complains that she no longer can break their systems. As a result, 2000 more customers buy moduli from the company. Explain why the 100 customers probably have distinct primes p, but among the 2000 customers there are probably two with the same p.

3. Suppose $p(x) = x^4 + x + 1$ is used in place of the $p(x)$ used in CRC-32. If S is a binary string, then express it as a polynomial $f(x)$, divide by $p(x)$, and let $r(x)$ be the remainder mod 2. Change this back to a binary string of length 4 and call this string $c(S)$. This produces a check-sum in a manner similar to CRC-32.

    1. Compute $c(1001001)$ and $c(1101100)$.

2. Compute $c(1001001 \oplus 1101100)$ and show that it is the XOR of the two individual XORs obtained in part (a).

4. You intercept the ciphertext *TUCDZARQKERUIZCU*, which was encrypted using an Enigma machine. You know the plaintext was either `ATTACKONTHURSDAY` or `ATTACKONSATURDAY`. Which is it?

5. 1. You play the CI Game from Chapter 4 with Bob. You give him the plaintexts `CAT` and `DOG`. He chooses one of these at random and encrypts it on his Enigma machine, producing the ciphertext `DDH`. Can you decide which plaintext he encrypted?

   2. Suppose you play the CI Game in part (a) using plaintexts of 100 letters (let's assume that these plaintexts agree in only 10 letters and differ in the other 90 letters). Explain why you should win almost every time. What does this say about ciphertext indistinguishability for Enigma?