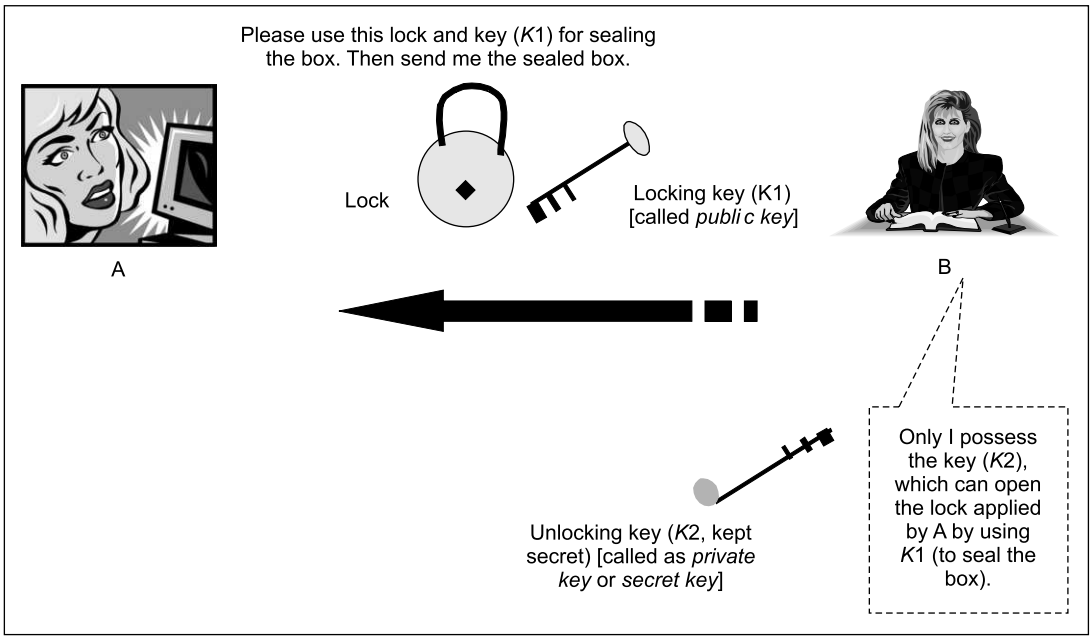

Fig. 2.62 Use of key pair

sends his/her lock and public key to B. B can use that to seal the message and send the sealed message to A. A can then use her private key to open the lock. Similarly, when B wants to receive a message securely from A, B sends his/her lock and public key to A, using which B can secure the message. Since only B has his/her own private key, he/she can open the lock and access the message.

Extending this basic idea, if 1,000 people want to be able to securely communicate with each other, only 1,000 locks, 1,000 public keys and the corresponding 1,000 private keys are required. This is in stark contrast to the symmetric key operation wherein for 1,000 participants, we needed 499,500 lock-and-key pairs (please refer to our earlier discussion).

Therefore, in general, when using asymmetric key operation, the recipient has to send the lock and his/her public key to the sender. The sender uses these to apply the lock and sends the sealed contents to the recipient. The recipient uses his/her private key to open the lock. Since only the recipient possesses the private key, all concerned are assured that only the intended recipient can open the lock.

2.7 STEGANOGRAPHY

Steganography is a technique that facilitates hiding of a message that is to be kept secret inside other messages. This results in the concealment of the secret message itself! Historically, the sender used methods such as invisible ink, tiny pin punctures on specific characters, minute variations between handwritten characters, pencil marks on handwritten characters, etc.

Of late, people hide secret messages within graphic images. For instance, suppose that we have a secret message to send. We can take another image file and we can replace the last two rightmost bits of each byte of that image with (the next) two bits of our secret message. The resulting image would not look

too different, and yet carry a secret message inside! The receiver would perform the opposite trick: it would read the last two bits of each byte of the image file, and reconstruct the secret message. This concept is illustrated in Fig. 2.63.

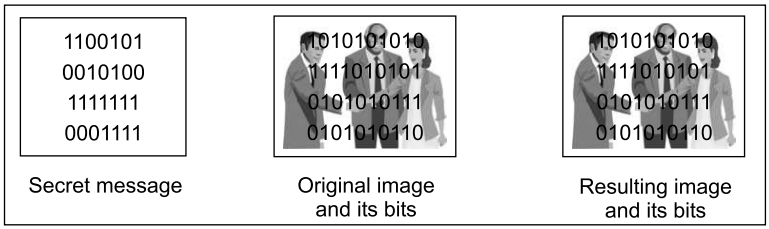

Fig. 2.63 Steganography example

2.8 KEY RANGE AND KEY SIZE

We have already seen one way to classify attacks on information itself, or information sources (such as computers or networks). However, the encrypted messages can be attacked, too! Here, the cryptanalyst is armed with the following information:

- The encryption/decryption algorithm
- The encrypted message
- Knowledge about the key size (e.g. the value of the key is a number between 0 and 100 billion)

As we have seen previously, the encryption/decryption algorithm is usually not a secret—everybody knows about it. Also, one can access an encrypted message by various means (such as by listening to the flow of information over a network). Thus, only the actual value of the key remains a challenge for the attacker. If the key is found, the attacker can resolve the mystery by working backwards to the original plain-text message, as shown in Fig. 2.64. We shall consider the brute-force attack here, which works on the principle of trying every possible key in the **key range**, until you get the right key.

It usually takes a very small amount of time to try a key. The attacker can write a computer program that tries many such keys in one second. In the best case, the attacker finds the right key in the first attempt itself, and in the worst case, it is the 100 billionth attempt. However, the usual observation is that the key is found somewhere in between the possible range. Mathematics tells us that on an average, the key can be found after about half of the possible values in the key range are checked. Of course, this is just a guideline, and may or may not work in real practice for a given situation.

As Fig. 2.64 shows, the attacker has access to the cipher-text block and the encryption/decryption algorithm. He/she also knows the key range (a number between 0 and 100 billion). The attacker now starts trying every possible key, starting from 0. After every decryption, he/she looks at the generated *plain text* (actually, it is not truly plain text, but decrypted text, but we shall ignore this technical detail). If the attacker notices that the decryption has yielded unintelligent plain text, she continues the process with the next key in the sequence. Finally, she is able to find the right key with a value 90,171,451,191, which yields the plain text *To: Payroll*.

How does the attacker determine if the plain text, and therefore the key, are the right ones? This can be determined depending on the value of the plain text. If the plain text seems reasonable (i.e. very close

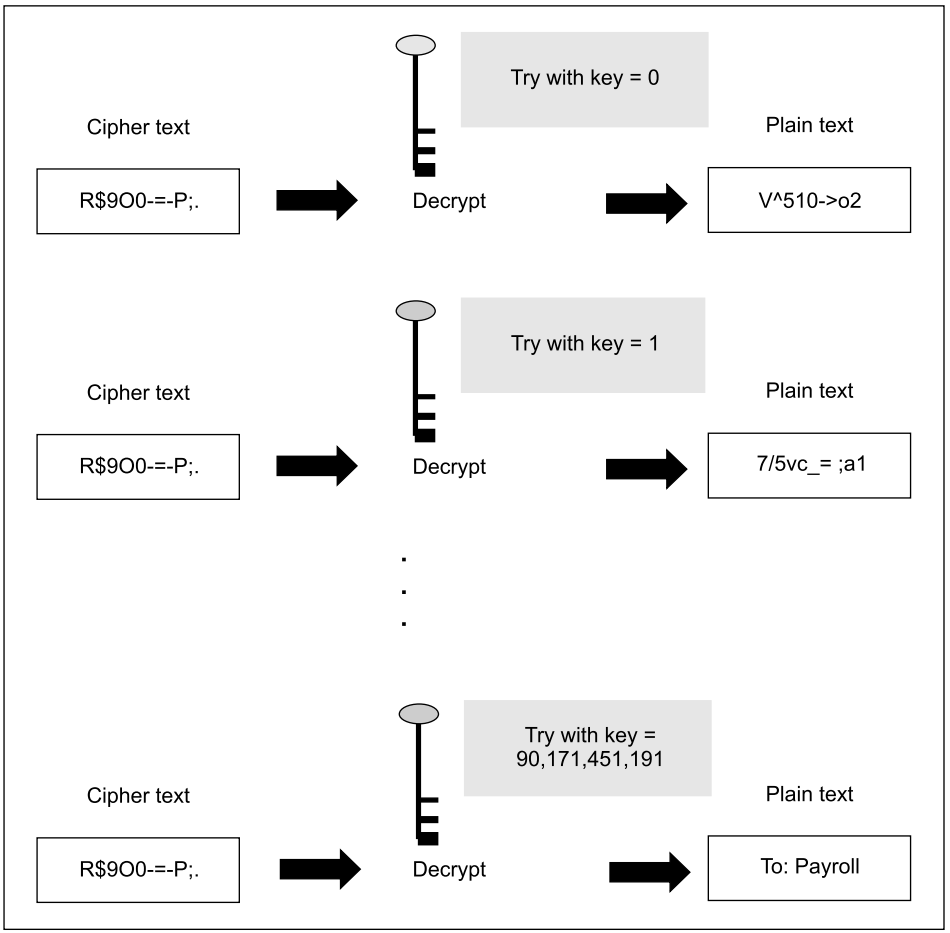

Fig. 2.64 Brute-force attack

to actual English words/sentences/numbers that make sense), it is highly probable that the plain text is indeed what corresponds to the cipher text.

This means that our key, and therefore, the plain-text message, are now cracked! How can we prevent an attacker from succeeding in such attempts? As we know, currently, our key range is 0 to 100 billion. Also, let us assume that the attacker took only 5 minutes to successfully crack our key. However, we want our message to remain secret for at least 5 years. This means that the attacker must spend at least 5 years in trying out every possible key, in order to obtain our original plain-text message. Therefore, the solution to our problem lies in expanding or increasing the *key range* to a size, which requires the attacker to work for more than 5 years in order to crack the key. Perhaps our key range should be from 0 to 100 billion billion billion billion.

In computer terms, the concept of *key range* leads us to the principle of **key size**. Just as we measure the value of stocks with a given index, gold in troy ounces, money in dollars, pounds or rupees; we measure the strength of a cryptographic key with *key size*. We measure key size in bits, and represent it using the binary number system. Thus, our key might be of 40 bits, 56 bits, 128 bits, and so on. In order to

protect ourselves against a brute-force attack, the key size should be such that the attacker cannot crack it within a specified amount of time. How long should it be? Let us study this.

At the simplest level, the key size can be just 1 bit. This means that the key can be either 0 or 1. If the key size is 2, the possible key values are 00, 01, 10, 11. Obviously, these examples are merely to understand the theory, and have no practical significance.

From a practical viewpoint, a 40-bit key takes about 3 hours to crack. However, a 41-bit key would take 6 hours, a 42-bit key takes 12 hours, and so on. This means that every additional bit doubles the amount of time required to crack the key. Why is this so? This works on the simple theory of binary numbers wherein every additional bit doubles the number of possible states of the number. This is shown in Fig. 2.65.

A 2-bit binary number has four possible states:

00
01
10
11

If we have one more bit to make it a 3-bit binary number, the number of possible states also doubles to eight, as follows:

000
001
010
011
100
101
110
111

In general, if an *n-bit* binary number has *k* possible states, an *n* + 1 *bit* binary number will have 2*k* possible states.

Fig. 2.65 Understanding key range

Thus, with every incremental bit, the attacker has to perform double the number of operations as compared to the previous key size. It is found that for a 56-bit key, it takes 1 second to search 1 percent of the key range. Taking this argument further, it takes about 1 minute to search about half of the key range (which is what is required, on an average, to crack a key). Using this as the basis, let us have a look at the similar values (time required for a search of 1 percent and 50 percent of the key space) for various key sizes. This is shown in Fig. 2.66.

Key size on bits	*Time required to search 1 percent of the key space*	*Time required to search 50 percent of the key space*
56	1 second	1 minute
57	2 seconds	2 minutes
58	4 seconds	4 minutes
64	4.2 minutes	4.2 hours
72	17.9 hours	44.8 days
80	190.9 days	31.4 years
90	535 years	321 centuries
128	146 billion millennia	8 trillion millennia

Fig. 2.65 Efforts required to break a key

We can represent the possible values in the key range using hexadecimal notation, and see visually how an increase in the key size increases the key range, and therefore, the complexity for an attacker. This is shown in Fig. 2.67.

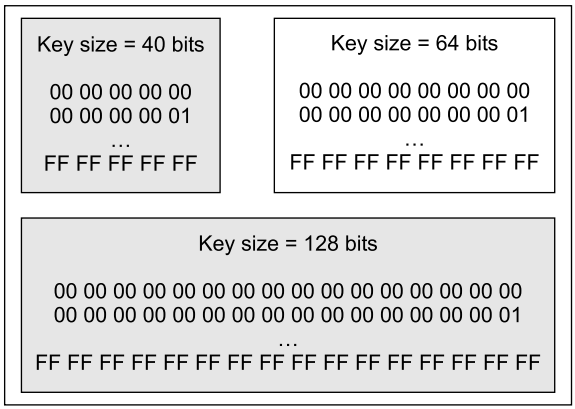

Fig. 2.67 Key sizes and ranges

Clearly, we can assume with reasonable confidence that a 128-bit key is quite safe (because 2^{128} means about 340,000,000,000,000,000,000,000,000,000,000,000,000 possible keys!), considering the capabilities of computers today. Obviously, as computing power and techniques improve, these numbers change. May be, in a few years time, a 128-bit key will be cracked. Then, we would need to rely on keys of size at least 256 bits, or 512 bits.

One may think that with the technological progress chasing key sizes so fast, how long can this go on and on? Today, 56-bit keys are not safe, tomorrow, 128-bit keys may not be sufficient, another day, 256-bit keys can be cracked, and so on! How far can we go? Well, it is argued that we may not have to look beyond 512-bit keys at any point of time in the future. That is, 512-bit keys will always be safe. What substantiates this argument?

Suppose that every atom in the universe is actually a computer. Then, we would have 2^{300} such computers in the world. Now, if each of these computers could check 2^{300} keys in one second, it would take 2^{162} millennia to search 1 percent of a 512-bit key! The Big Bang theory suggests that the amount of time that has passed since the universe came into existence is less than 2^{24} millennia, which is significantly less than the key search time. Thus, 512-bit keys will always be safe.

■ 2.9 POSSIBLE TYPES OF ATTACKS ■

Based on the discussion so far, when the sender of a message encrypts a plain-text message into its corresponding cipher text, there are five possibilities for an attack on this message, as shown in Fig. 2.68.

Let us discuss these attacks now.

1. Cipher-Text Only Attack

In this type of attack, the attacker does not have any clue about the plain text. She has some or all of the cipher text. (Interestingly, we should point out that if the attacker does not have an access even to the cipher text, there would be no need to encrypt the plain text to obtain cipher text in the first place!).