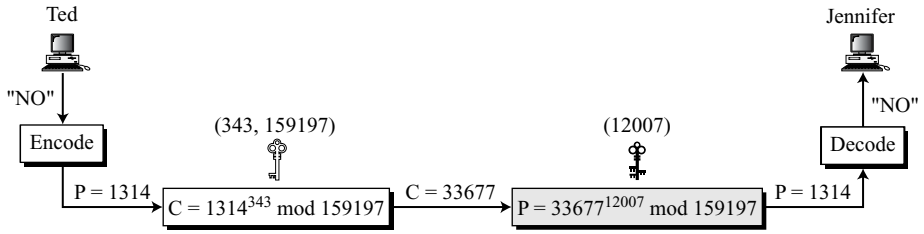
Example 10.7

Jennifer creates a pair of keys for herself. She chooses $p = 397$ and $q = 401$. She calculates $n = 397 \times 401 = 159197$. She then calculates $\phi(n) = 396 \times 400 = 158400$. She then chooses $e = 343$ and $d = 12007$. Show how Ted can send a message to Jennifer if he knows e and n.

Solution

Suppose Ted wants to send the message "NO" to Jennifer. He changes each character to a number (from 00 to 25), with each character coded as two digits. He then concatenates the two coded characters and gets a four-digit number. The plaintext is 1314. Ted then uses e and n to encrypt the message. The ciphertext is $1314^{343} = 33677 \bmod 159197$. Jennifer receives the message 33677 and uses the decryption key d to decipher it as $33677^{12007} = 1314 \bmod 159197$. Jennifer then decodes 1314 as the message "NO". Figure 10.7 shows the process.

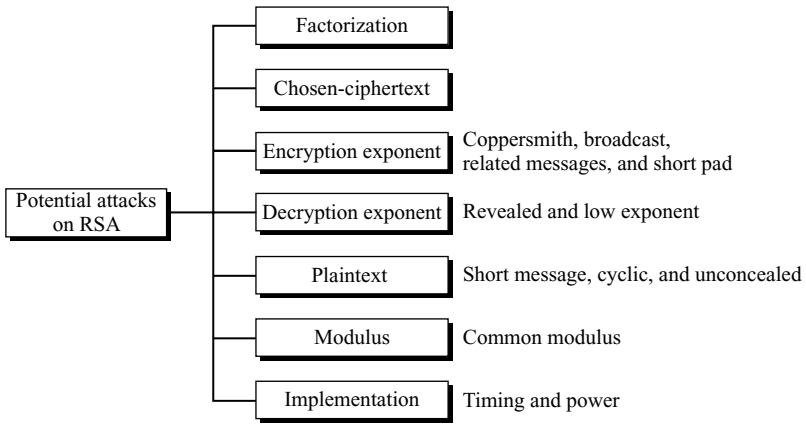
Figure 10.7 *Encryption and decryption in Example 10.7*



Attacks on RSA

No devastating attacks on RSA have been yet discovered. Several attacks have been predicted based on the weak plaintext, weak parameter selection, or inappropriate implementation. Figure 10.8 shows the categories of potential attacks.

Figure 10.8 *Taxonomy of potential attacks on RSA*



Factorization Attack

The security of RSA is based on the idea that the modulus is so large that it is infeasible to factor it in a reasonable time. Bob selects p and q and calculates $n = p \times q$. Although n is public, p and q are secret. If Eve can factor n and obtain p and q, she can calculate $\phi(n) = (p - 1)\ (q - 1)$. Eve then can calculate $d = e^{-1} \bmod \phi(n)$ because e is public. The private exponent d is the trapdoor that Eve can use to decrypt any encrypted message.

As we learned in Chapter 9, there are many factorization algorithms, but none of them can factor a large integer with polynomial time complexity. To be secure, RSA presently requires that n should be more than 300 decimal digits, which means that the modulus must be at least 1024 bits. Even using the largest and fastest computer available today, factoring an integer of this size would take an infeasibly long period of time. This means that RSA is secure as long as an efficient algorithm for factorization has not been found.

Chosen-Ciphertext Attack

A potential attack on RSA is based on the multiplicative property of RSA. Assume that Alice creates the ciphertext $C = P^e \bmod n$ and sends C to Bob. Also assume that Bob will decrypt an arbitrary ciphertext for Eve, other than C. Eve intercepts C and uses the following steps to find P:

a. Eve chooses a random integer X in Z_n^*.
b. Eve calculates $Y = C \times X^e \bmod n$.
c. Eve sends Y to Bob for decryption and get $Z = Y^d \bmod n$; This step is an instance of a chosen-ciphertext attack.
d. Eve can easily find P because

$$Z = Y^d \bmod n = (C \times X^e)^d \bmod n = (C^d \times X^{ed}) \bmod n = (C^d \times X) \bmod n = (P \times X) \bmod n$$
$$Z = (P \times X) \bmod n \quad \rightarrow \quad P = Z \times X^{-1} \bmod n$$

Eve uses the extended Euclidean algorithm to find the multiplicative inverse of X and eventually the value of P.

Attacks on the Encryption Exponent

To reduce the encryption time, it is tempting to use a small encryption exponent e. The common value for e is $e = 3$ (the second prime). However, there are some potential attacks on low encryption exponent that we briefly discuss here. These attacks do not generally result in a breakdown of the system, but they still need to be prevented. To thwart these kinds of attacks, the recommendation is to use $e = 2^{16} + 1 = 65537$ (or a prime close to this value).

Coppersmith Theorem Attack The major low encryption exponent attack is referred to as the **Coppersmith theorem attack.** This theorem states that in a modulo-n polynomial $\boldsymbol{f}(x)$ of degree e, one can use an algorithm of the complexity log n to find the roots if one of the roots is smaller than $n^{1/e}$. This theorem can be applied to the RSA

cryptosystem with $C = \boldsymbol{f}(P) = P^e \bmod n$. If $e = 3$ and only two thirds of the bits in the plaintext P are known, the algorithm can find all bits in the plaintext.

Broadcast Attack The **broadcast attack** can be launched if one entity sends the same message to a group of recipients with the same low encryption exponent. For example, assume the following scenario: Alice wants to send the same message to three recipients with the same public exponent $e = 3$ and the moduli n_1, n_2, and n_3.

$$C_1 = P^3 \bmod n_1 \qquad C_2 = P^3 \bmod n_2 \qquad C_3 = P^3 \bmod n_3$$

Applying the Chinese remainder theorem to these three equations, Eve can find an equation of the form $C' = P^3 \bmod n_1 n_2 n_3$. This means that $P^3 < n_1 n_2 n_3$. This means $C' = P^3$ is in regular arithmetic (not modular arithmetic). Eve can find the value of $C' = P^{1/3}$.

Related Message Attack The **related message attack,** discovered by Franklin Reiter, can be briefly described as follows. Alice encrypts two plaintexts, P_1 and P_2, and encrypts them with $e = 3$ and sends C_1 and C_2 to Bob. If P_1 is related to P_2 by a linear function, then Eve can recover P_1 and P_2 in a feasible computation time.

Short Pad Attack The **short pad attack,** discovered by Coppersmith, can be briefly described as follows. Alice has a message M to send to Bob. She pads the message with r_1, encrypts the result to get C_1, and sends C_1 to Bob. Eve intercepts C_1 and drops it. Bob informs Alice that he has not received the message, so Alice pads the message again with r_2, encrypts it, and sends it to Bob. Eve also intercepts this message. Eve now has C_1 and C_2, and she knows that they both are ciphertexts belonging to the same plaintext. Coppersmith proved that if r_1 and r_2 are short, Eve may be able to recover the original message M.

Attacks on the Decryption Exponent

Two forms of attacks can be launched on the decryption exponent: **revealed decryption exponent attack** and **low decryption exponent attack.** They are discussed briefly.

Revealed Decryption Exponent Attack It is obvious that if Eve can find the decryption exponent, d, she can decrypt the current encrypted message. However, the attack does not stop here. If Eve knows the value of d, she can use a probabilistic algorithm (not discussed here) to factor n and find the value of p and q. Consequently, if Bob changes only the compromised decryption exponent but keeps the same modulus, n, Eve will be able to decrypt future messages because she has the factorization of n. This means that if Bob finds out that the decryption exponent is compromised, he needs to choose new value for p and q, calculate n, and create totally new private and public keys.

In RSA, if d is comprised, then p, q, n, e, and d must be regenerated.

Low Decryption Exponent Attack Bob may think that using a small private-key d, would make the decryption process faster for him. Wiener showed that if $d < 1/3\ n^{1/4}$, a special type of attack based on *continuous fraction,* a topic discussed in number theory, can jeopardize the security of RSA. For this to happen, it must be the case that $q < p < 2q$. If these two conditions exist, Eve can factor n in polynomial time.

In RSA, the recommendation is to have $d \geq 1/3\ n^{1/4}$ to prevent low decryption exponent attack.

Plaintext Attacks

Plaintext and ciphertext in RSA are permutations of each other because they are integers in the same interval (0 to $n - 1$). In other words, Eve already knows something about the plaintext. This characteristic may allow some attacks on the plaintext. Three attacks have been mentioned in the literature: short message attack, cycling attack, and unconcealed attack.

Short Message Attack In the **short message attack,** if Eve knows the set of possible plaintexts, she then knows one more piece of information in addition to the fact that the ciphertext is the permutation of plaintext. Eve can encrypt all of the possible messages until the result is the same as the ciphertext intercepted. For example, if it is known that Alice is sending a four-digit number to Bob, Eve can easily try plaintext numbers from 0000 to 9999 to find the plaintext. For this reason, short messages must be padded with random bits at the front and the end to thwart this type of attack. It is strongly recommended that messages be padded with random bits before encryption using a method called OAEP, which is discussed later in this chapter.

Cycling Attack The **cycling attack** is based on the fact that if the ciphertext is a permutation of the plaintext, the continuous encryption of the ciphertext will eventually result in the plaintext. In other words, if Eve continuously encrypts the intercepted ciphertext C, she will eventually get the plaintext. However, Eve does not know what the plaintext is, so she does not know when to stop. She needs to go one step further. When she gets the ciphertext C again, she goes back one step to find the plaintext.

Intercepted ciphertext: C
$C_1 = C^e \mod n$
$C_2 = C_1^e \mod n$
…
$C_k = C_{k-1}^e \mod n \rightarrow$ If $C_k = C$, stop: the plaintext is $P = C_{k-1}$

Is this a serious attack on RSA? It has been shown that the complexity of the algorithm is equivalent to the complexity of factoring n. In other words, there is no efficient algorithm that can launch this attack in polynomial time if n is large.

Unconcealed Message Attack Another attack that is based on the permutation relationship between plaintext and ciphertext is the **unconcealed message attack.** An

unconcealed message is a message that encrypts to itself (cannot be concealed). It has been proven that there are always some messages that are encrypted to themselves. Because the encryption exponent normally is odd, there are some plaintexts that are encrypted to themselves such as P = 0 and P = 1. Although there are more, if the encrypting exponent is selected carefully, the number of these message is negligible. The encrypting program can always check if the calculated ciphertext is the same as the plaintext and reject the plaintext before submitting the ciphertext.

Attacks on the Modulus

The main attack on RSA, as discussed previously, is the factorization attack**.** The factorization attack can be considered an attack on the low modulus. However, because we have already discussed this attack, we will concentrate on another attack on the modulus: the common modulus attack.

Common Modulus Attack The **common modulus attack** can be launched if a community uses a common modulus, n. For example, people in a community might let a trusted party select p and q, calculate n and $\phi(n)$, and create a pair of exponents (e_i, d_i) for each entity. Now assume Alice needs to send a message to Bob. The ciphertext to Bob is $C = P^{e_B} \bmod n$. Bob uses his private exponent, d_B, to decrypt his message, $P = C^{d_B} \bmod n$. The problem is that Eve can also decrypt the message if she is a member of the community and has been assigned a pair of exponents (e_E and d_E), as we learned in the section "Low Decryption Exponent Attack". Using her own exponents (e_E and d_E), Eve can launch a probabilistic attack to factor n and find Bob's d_B. To thwart this type of attack, the modulus must not be shared. Each entity needs to calculate her or his own modulus.

Attacks on Implementation

Previous attacks were based on the underlying structure of RSA. As Dan Boneh has shown, there are several attacks on the implementation of RSA. We mention two of these attacks: the timing attack and the power attack.

Timing Attack Paul Kocher elegantly demonstrated a ciphertext-only attack, called the **timing attack.** The attack is based on the fast-exponential algorithm discussed in Chapter 9. The algorithm uses only squaring if the corresponding bit in the private exponent d is 0; it uses both squaring and multiplication if the corresponding bit is 1. In other words, the timing required to do each iteration is longer if the corresponding bit is 1. This timing difference allows Eve to find the value of bits in d, one by one.

Assume that Eve has intercepted a large number of ciphertexts, C_1 to C_m. Also assume that Eve has observed how long it takes for Bob to decrypt each ciphertext, T_1 to T_m. Eve, who knows how long it takes for the underlying hardware to calculate a multiplication operation, calculated t_1 to t_m, where t_i is the time required to calculate the multiplication operation Result = Result $\times C_i \bmod n$.

Eve can use Algorithm 10.5, which is a simplified version of the algorithm used in practice, to calculate all bits in d (d_0 to d_{k-1}).

The algorithm sets $d_0 = 1$ (because d should be odd) and calculates new values for T_i's (decryption time related to d_1 to d_{k-1}). The algorithm then assumes the next bit is 1