4. Let $a = 39$: $z = 39^{45} \equiv 78 \bmod 91$, hence, $\tilde{p}$ is composite.

Since the numbers 12, 17 and 38 give incorrect statements for the prime candidate $\tilde{p} = 91$, they are called "liars for 91".

◇

7.7 RSA in Practice: Padding

What we described so far is the so-called "schoolbook RSA" system which has several weaknesses. In practice RSA has to be used with a padding scheme. Padding schemes are extremely important, and if not implemented properly, an RSA implementation may be insecure. The following properties of schoolbook RSA encryption are problematic:

- RSA encryption is deterministic, i.e., for a specific key, a particular plaintext is always mapped to a particular ciphertext. An attacker can derive statistical properties of the plaintext from the ciphertext. Furthermore, given some pairs of plaintext–ciphertext, partial information can be derived from new ciphertexts which are encrypted with the same key.
- Plaintext values $x = 0$, $x = 1$, or $x = -1$ produce ciphertexts equal to 0, 1, or -1.
- Small public exponents e and small plaintexts x might be subject to attacks if no padding or weak padding is used. However, there is no known attack against small public exponents such as $e = 3$.

RSA has another undesirable property, namely that it is *malleable*. A crypto scheme is said to be malleable if the attacker Oscar is capable of transforming the ciphertext into another ciphertext which leads to a known transformation of the plaintext. Note that the attacker does not decrypt the ciphertext but is merely capable of manipulating the plaintext in a predictable manner. This is easily achieved in the case of RSA if the attacker replaces the ciphertext y by $s^e y$, where s is some integer. If the receiver decrypts the manipulated ciphertext, he computes:

$$(s^e y)^d \equiv s^{ed} x^{ed} \equiv sx \bmod n.$$

Even though Oscar is not able to decrypt the ciphertext, such targeted manipulations can still do harm. For instance, if x were an amount of money which is to be transferred or the value of a contract, by choosing $s = 2$ Oscar could exactly double the amount in a way that goes undetected by the receiver.

A possible solution to all these problems is the use of padding, which embeds a random structure into the plaintext before encryption and avoids the above mentioned problems. Modern techniques such as *Optimal Asymmetric Encryption Padding (OAEP)* for padding RSA messages are specified and standardized in Public Key Cryptography Standard #1 (PKCS #1).

Let M be the message to be padded, let k be the length of the modulus n in bytes, let $|H|$ be the length of the hash function output in bytes and let $|M|$ be the

length of the message in bytes. A hash function computes a message digest of fixed length (e.g., 160 or 256 bit) for every input. More about hash functions is found in Chap. 11. Furthermore, let L be an optional label associated with the message (otherwise, L is an empty string as default). According to the most recent version PKCS#1 (v2.1), padding a message within the RSA encryption scheme is done in the following way:

1. Generate a string PS of length $k - |M| - 2|H| - 2$ of zeroed bytes. The length of PS may be zero.
2. Concatenate $Hash(L)$, PS, a single byte with hexadecimal value `0x01`, and the message M to form a data block DB of length $k - |H| - 1$ bytes as

$$DB = Hash(L)||PS||\texttt{0x01}||M.$$

3. Generate a random byte string $seed$ of length $|H|$.
4. Let $dbMask = MGF(seed, k - |H| - 1)$, where MGF is the mask generation function. In practice, a hash function such as SHA-1 is often used as MFG.
5. Let $maskedDB = DB \oplus dbMask$.
6. Let $seedMask = MGF(maskedDB, |H|)$.
7. Let $maskedSeed = seed \oplus seedMask$.
8. Concatenate a single byte with hexadecimal value `0x00`, $maskedSeed$ and $maskedDB$ to form an encoded message EM of length k bytes as

$$EM = \texttt{0x00}||maskedSeed||maskedDB.$$

Figure 7.3 shows the structure of a padded message M.

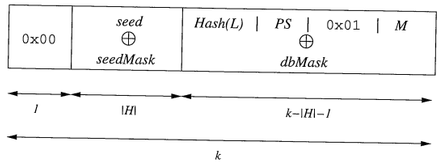

Fig. 7.3 RSA encryption of a message M with Optimal Asymmetric Encryption Padding (OAEP)

On the decryption side, the structure of the decrypted message has to be verified. For instance, if there is no byte with hexadecimal value `0x01` to separate PS from M, a decryption error occurred. In any case, returning a decryption error to the user (or a potential attacker!) should not reveal any information about the plaintext.