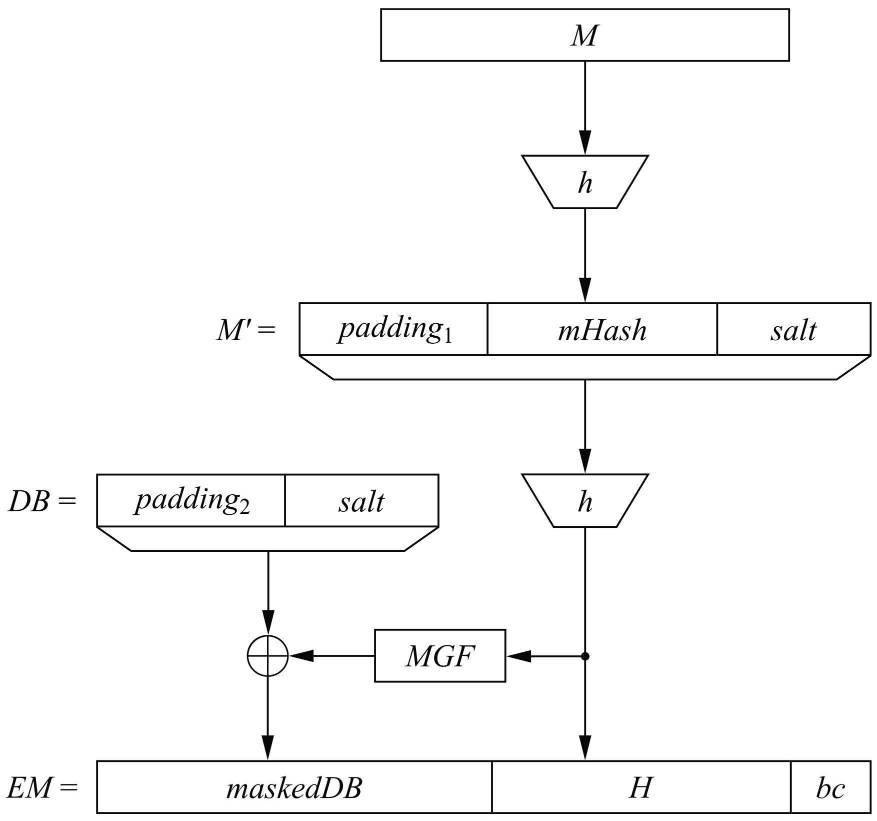

Fig. 10.2 Principle of EMSA-PSS encoding

10.3 The Elgamal Digital Signature Scheme

The Elgamal signature scheme, which was published in 1985, is based on the difficulty of computing discrete logarithms (cf. Chap. 8). Unlike RSA, where encryption and digital signature are almost identical operations, the Elgamal digital signature is quite different from the encryption scheme with the same name.

10.3.1 Schoolbook Elgamal Digital Signature

Key Generation

As with every public-key scheme, there is a set-up phase during which the keys are computed. We start by finding a large prime p and constructing a discrete logarithm problem as follows:

Key Generation for Elgamal Digital Signature

1. Choose a large prime p.
2. Choose a primitive element α of $\mathbb{Z}_p^*$ or a subgroup of $\mathbb{Z}_p^*$.
3. Choose a random integer $d \in \{2, 3, \ldots, p-2\}$.
4. Compute $\beta = \alpha^d \bmod p$.

The public key is now formed by $k_{pub} = (p, \alpha, \beta)$, and the private key by $k_{pr} = d$.

Signature and Verification

Using the private key and the parameters of the public key, the signature

$$\text{sig}_{k_{pr}}(x, k_E) = (r, s)$$

for a message x is computed during the signing process. Note that the signature consists of two integers r and s. The signing consists of two main steps: choosing a random value k_E, which forms an ephemeral private key, and computing the actual signature of x.

Elgamal Signature Generation

1. Choose a random ephemeral key $k_E \in \{0, 1, 2, \ldots, p-2\}$ such that $\gcd(k_E, p-1) = 1$.
2. Compute the signature parameters:

$$r \equiv \alpha^{k_E} \bmod p,$$
$$s \equiv (x - d \cdot r) k_E^{-1} \bmod p-1.$$

On the receiving side, the signature is verified as $\text{ver}_{k_{pub}}(x, (r, s))$ using the public key, the signature and the message.

Elgamal Signature Verification

1. Compute the value
$$t \equiv \beta^r \cdot r^s \bmod p$$
2. The verification follows from:

$$t \begin{cases} \equiv \alpha^x \bmod p & \Longrightarrow \text{valid signature} \\ \not\equiv \alpha^x \bmod p & \Longrightarrow \text{invalid signature} \end{cases}$$

In short, the verifier accepts a signature (r,s) only if the relation $\beta^r \cdot r^s \equiv \alpha^x \bmod p$ is satisfied. Otherwise, the verification fails. In order to make sense of the rather arbitrary looking rules for computing the signature parameters r and s as well as the verification, it is helpful to study the following proof.

Proof. We'll prove the correctness of the Elgamal signature scheme. More specifically, we show that the verification process yields a "true" statement if the verifier uses the correct public key and the correct message, and if the signature parameters (r,s) were chosen as specified. We start with the verification equation:

$$\begin{aligned}\beta^r \cdot r^s &\equiv (\alpha^d)^r (\alpha^{k_E})^s \bmod p \\ &\equiv \alpha^{d\,r + k_E\,s} \bmod p.\end{aligned}$$

We require that the signature is considered valid if this expression is identical to α^x:

$$\alpha^x \equiv \alpha^{d\,r + k_E\,s} \bmod p. \tag{10.1}$$

According to Fermat's Little Theorem, the relationship (10.1) holds if the exponents on both sides of the expression are identical modulo $p-1$:

$$x \equiv d\,r + k_E\,s \bmod p-1$$

from which the construction rule of the signature parameters s follows:

$$s \equiv (x - d \cdot r) k_E^{-1} \bmod p-1.$$

□

The condition that $\gcd(k_E, p-1) = 1$ is required since we have to invert the ephemeral key modulo $p-1$ when computing s.

Let's look at an example with small numbers.

Example 10.2. Again, Bob wants to send a message to Alice. This time, it should be signed with the Elgamal digital signature scheme. The signature and verification process is as follows:

Alice		**Bob**
		1. choose $p = 29$ 2. choose $\alpha = 2$ 3. choose $d = 12$ 4. $\beta = \alpha^d \equiv 7 \bmod 29$
	$\xleftarrow{(p,\alpha,\beta)=(29,2,7)}$	
		compute signature for message $x = 26$: choose $k_E = 5$, note that $\gcd(5,28) = 1$ $r = \alpha^{k_E} \equiv 2^5 \equiv 3 \bmod 29$ $s = (x - d\,r)\,k_E^{-1} \equiv (-10) \cdot 17 \equiv 26 \bmod 28$
	$\xleftarrow{(x,(r,s))=(26,(3,26))}$	
verify: $t = \beta^r \cdot r^s \equiv 7^3 \cdot 3^{26} \equiv 22 \bmod 29$ $\alpha^x \equiv 2^{26} \equiv 22 \bmod 29$ $t \equiv \alpha^x \bmod 29 \Longrightarrow$ valid signature		

◇

10.3.2 Computational Aspects

The key generation phase is identical to the set-up phase of Elgamal encryption, which we introduced in Sect. 8.5.2. Because the security of the signature scheme relies on the discrete logarithm problem, p needs to have the properties discussed in Sect. 8.3.3. In particular, it should have a length of at least 1024 bits. The prime can be generated using the prime-finding algorithms introduced in Sect 7.6. The private key should be generated by a true random number generator. The public key requires one exponentiation using the square-and-multiply algorithm.

The signature consists of the pair (r,s). Both have roughly the same bit length as p, so that the total length of the package $(x,(r,s))$ is about three times as long as only the message x. Computing r requires an exponentiation modulo p, which can be achieved with the square-and-multiply algorithm. The main operation when computing s is the inversion of k_E. This can be done using the extended Euclidean algorithm. A speed-up is possible through precomputing. The signer can generate the ephemeral key k_E and r in advance and store both values. When a message is to be signed, they can be retrieved and used to compute s. The verifier performs two exponentiations that are again computed with the square-and-multiply algorithm, and one multiplication.