Chapter 9

MASKING

The goal of every countermeasure is to make the power consumption of a cryptographic device independent of the intermediate values of the cryptographic algorithm. Masking achieves this by randomizing the intermediate values that are processed by the cryptographic device. An advantage of this approach is that it can be implemented at the algorithm level without changing the power consumption characteristics of the cryptographic device. In other words, masking allows making the power consumption independent of the intermediate values, even if the device has a data-dependent power consumption. Masking is one of the countermeasures that has been extensively discussed in the scientific community. Numerous articles have been published that explain different types of masking schemes. Even security proofs have been delivered for some of the schemes. Recently, masking has also been applied to the cell level.

In this chapter, we discuss how masking works and under which assumptions it leads to secure implementations. We also discuss how to implement masking at the architecture and the cell level.

9.1 General Description

In a masked implementation, each intermediate value v is concealed by a random value m that is called mask: $v_m = v * m$. The mask m is generated internally, *i.e.* inside the cryptographic device, and varies from execution to execution. Hence, it is not known by the attacker.

> A masked intermediate value v_m is an intermediate value v that is concealed by a random value m: $v_m = v * m$. The attacker does not know the random value.

The operation $*$ is typically defined according to the operations that are used in the cryptographic algorithm. Hence, the operation $*$ is most often the Boolean exclusive-or function $\oplus$, the modular addition $+$, or the modular multiplication $\times$. In case of modular addition and modular multiplication, the modulus is chosen according to the cryptographic algorithm.

Typically, the masks are directly applied to the plaintext or the key. The implementation of the algorithm needs to be slightly changed in order to process the masked intermediate values and in order to keep track of the masks. The result of the encryption is also masked. Hence, the masks need to be removed at the end of the computation in order to obtain the ciphertext. A typical *masking scheme* specifies how all intermediate values are masked and how to apply, remove, and change the masks throughout the algorithm.

It is important that every intermediate value is masked all the time. This must be guaranteed also for intermediate values that are calculated based on previous intermediate values. For instance, if two masked intermediate values are exclusive-ored, we need to ensure that the result is masked as well. For this reason, we typically use several masks. Hence, different intermediate values are concealed by different masks. It turns out that it is not advisable to use a new mask for each intermediate value because the number of masks decreases the performance. Consequently, the number of masks needs to be chosen carefully in order to achieve a reasonable performance.

In the remainder of this section we review several important concepts that occur in the context of masking. In particular, we discuss different types of masking (Boolean vs. arithmetic), and we explain how secret sharing relates to masking. Furthermore, we explain the meaning of blinding and discuss the security of masking.

9.1.1 Boolean vs. Arithmetic Masking

We distinguish between Boolean and arithmetic masking. In Boolean masking, the intermediate value is concealed by exclusive-oring it with the mask: $v_m = v \oplus m$. In arithmetic masking the intermediate value is concealed by an arithmetic operation (addition or multiplication). Often, one uses the modular addition: $v_m = v + m \pmod{n}$. The modulus n is defined according to the cryptographic algorithm. The other arithmetic operation that is frequently used is the modular multiplication: $v_m = v \times m \pmod{n}$.

Some algorithms are based on Boolean and arithmetic operations. Therefore, they require both types of masking. This is problematic because switching from one type of masking to another type often requires a significant amount of additional operations, see [CG00], and [Gou01].

In addition, cryptographic algorithms use linear and non-linear functions. A linear function f has the property that $f(x * y) = f(x) * f(y)$. For example, if the operation $*$ is the exclusive-or operation $\oplus$, then a linear function has