622.755 – Crypto Engineering Winter Term, 2020/21

Handout 3: Countermeasures: AES

Elisabeth Oswald

1 General Comments

This handout supports the delivery of the Crypto Engineering unit. For each week, there are slides, some videos and sometimes handouts. I suggest if there is a handout to read the handout first. It will give you some background information and explain important concepts, give examples, and suggest some exercises as well as hints for further reading. Anything under further reading will enhance your understanding, but is not strictly required to be able to get on with the coursework.

The videos are designed to talk you through the notes as well as the exercises and make explicit links to the coursework. I will often use analogies in videos to explain things, whereas the notes, whilst explaining the same ideas, will be more formal.

For this week you need to have a detailed understanding for AES, AES implementations, and differential attacks, such as provided in Crypto A and in the previous weeks of this unit.

There are a number of associated videos available from YouTube via an unlisted playlist:
`https://www.youtube.com/playlist?list=PLQ5hY1xU49CBkXsaK5TgUiQsy4jAZ7-xp`.
The videos are numbered, and you are looking out for number 16.

The text, and the pictures in this handout are largely taken from the book that I co-authored [1].

2 What you should get out of this.

This handout is about a detailed study of countermeasures against power/EM attacks on AES. After engaging with this content you should be able to:

- implement basic hiding countermeasures in software,
- reason about the impact on DPA outcomes,
- implement a basic masking scheme in software, and
- comment on the security of basic masking in the presence of non-HW leaks.

3 Countermeasures

Side channel attacks work because the leakage of cryptographic devices depends on the intermediate values of the executed cryptographic algorithms. The goal of countermeasures such as *masking* or *hiding* is to remove or at least reduce this dependency. Another approach (via appropriate *protocols* or *leakage resilient schemes*) is to mitigation is to reduce the amount of information leaked before keys are updated.

The basic idea of hiding is to remove the data dependency of the leakage. This means that either the execution of the algorithm is randomized or the leakage characteristics of the device are changed in such a way that an attacker cannot easily find a data dependency. The power consumption in particular can be changed in two ways to achieve this goal: the device can be built in such a way that every operation requires approximately the same amount of power, or it can be built in such a way that the power consumption is more or less random. In both cases, the data dependency of the power consumption is reduced significantly. However, in practice the data dependency cannot be removed completely—there always remains a certain amount of data dependency.

The basic idea of masking is to secret share the intermediate values that are processed by the cryptographic device. The motivation behind this approach is that the power that is needed to process shared out (and therefore random looking) intermediate values is independent of the actual intermediate values. A big advantage of masking is that the leakage characteristics of the device do not need to be changed. For instance, the power consumption can still be data dependent. Attacks are prevented because the device processes secret shared (i.e. random looking) intermediate values only.

4 Hiding

The goal of hiding countermeasures is to make the leakage of cryptographic devices independent of the intermediate values that are processed. This can either be done via making the intermediate values leak in a constant or a random way. Leakage such as power occurs over a period of time. Thus a designer can both consider these options along the amplitude dimension and the time dimension. If a designer has the ability to develop a device from scratch, they can for instance opt to produce it via special purpose logic style (rather than relying on CMOS technology), which could be designed to leak less information (this would change the characteristics in the amplitude domain). Such technologies do indeed exist, but they are very expensive and not suitable to be applied to general purpose processors. We will hence focus on different approaches that are useful when a complete (re)design of a device isn't an option.

Let's then focus on hiding approaches that work via the time dimension. I mean by this approaches that change the order of fundamental operations that occur when

an algorithm is executed. This will cause an immediate problem for DPA style attacks: remember that we assumed then that the same operation occurs in the same time points over many runs of an algorithm (we compute the distinguisher over each time point across all traces).

4.1 Random Insertion of Dummy Operations/Sequences

A dummy operation or sequence of operations is an operation that works on random data. The result of a dummy operation is discarded. It is imperative that the inserted operations can be easily distinguished and thereby removed.

The basic idea is to randomly insert dummy operations before, during, and even after the execution of the cryptographic algorithm. Each time the algorithm is executed, randomly generated numbers are used to decide how many dummy operations are inserted at various different positions. It is important that the total number of inserted operations is equal for all executions of the algorithm. In this way, attackers cannot get any information about the number of inserted operations by measuring the execution time of the algorithm.

In an implementation that is protected by this approach, the position of each operation depends on the number of dummy operations that have been inserted before this operation. This number randomly varies from execution to execution. The more this position varies, the more random the power consumption appears. However, it is also clear that the more dummy operations are inserted, the lower is the throughput of the implementation. This is why in practice, a suitable compromise needs be found for every implementation.

In the case of AES there are a number of ways in which one could implement this idea. For instance, if a software implementation is based on manipulating one column at a time, then a good choice for a sequence of dummy instruction would be to the sequence that implements just one column. In other words, the code fragment that implements one column would be called more than 4 times (4 would be needed to process the entire AES state). Any of the additional calls would operate on random data and thus would be the dummies. If, to provide another example, a software implementation would process all byte oriented operations and then MixColumns, then one could inject dummy byte operations (i.e. additional AddRoundKey and SubByte operations on random data) and thereafter additional MixColumns operations.

4.2 Shuffling

The term shuffling refers to the re-ordering of operations of the cryptographic algorithm. The basic idea of this approach is to randomly change the sequence of those operations

of a cryptographic algorithm that can be performed in arbitrary order. For instance, the 16 S-box look-ups are independent of each other. Hence, they can be performed in arbitrary order. Shuffling these operations means that during each execution of AES, randomly generated numbers are used to determine the sequence of the 16 S-box look-ups.

Shuffling randomises the power consumption in a similar way as the random insertion of dummy operations. However, shuffling does not affect the throughput as much as the random insertion of dummy operations. A disadvantage of shuffling is that it can only be applied to a certain extent and some algorithms lend itself more to it than others. In practice, shuffling and the random insertion of dummy operations are often combined.

4.3 Effectiveness and Cost

The cost of hiding mainly depends on the amount of randomness that is required, as well as the extra cost incurred by dummy operations.

The effectiveness of hiding intuitively depends on how much the target intermediate value "moves around" in the leakage traces. The fact that the intermediate value is no longer at the same time ct in each trace means that there is an impact on the correlation of the correct key ck. It is possible to mathematically model this impact. I explained before that the correlation of the correct key directly determines the number of traces that are needed for a successful DPA style attack. Therefore modelling the impact enables us to understand how hiding improves the resilience of an implementation in very concrete terms.

Recall that the correlation between the hypothetical power consumption for the correct key hypothesis H_{ck} and the power consumption at the moment of time ct is given by $\rho_{ck,ct}$ which corresponds to $\rho(H_{ck}, P_{total})$. To account (in our notation) explicitly for the fact that for some traces at time point ct something "random" is happening, we add a "hat": $\hat{P}_{total}$. We use $\hat{p}$ to quantify the probability that the "right" thing happens in time point ct, and therefore $1-\hat{p}$ is the probability that something "random" is happening. In any either, either the right thing happens or something random, therefore we can write the covariance between H_{ck} and $\hat{P}_{total}$ as follows:

$$Cov(H_{ck}, \hat{P}_{total}) = \hat{p} \cdot Cov(H_{ck}, P_{total}) + (1 - \hat{p}) \cdot Cov(H_{ck}, P_{other})$$

We can plug this covariance into the definition for $\rho(H_{ck}, P_{total})$ as shown in (1) and simplify the expression because $Cov(H_{ck}, P_{other}) = 0$, and P_{other} and P_{total} are indepen-

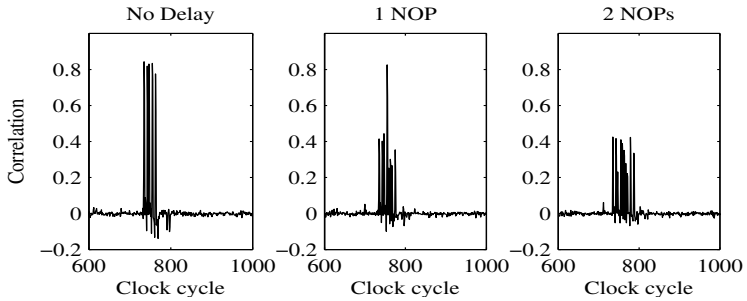

Figure 1: Introduction of a small delay prior to AES. Taken from [1]

dent.

$$\begin{aligned}
\rho(H_{ck}, \hat{P}_{total}) &= \frac{\hat{p} \cdot Cov(H_{ck}, P_{total}) + (1 - \hat{p}) \cdot Cov(H_{ck}, P_{other})}{\sqrt{Var(H_{ck}) \cdot Var(\hat{P}_{total})}} \\
&= \frac{\hat{p} \cdot Cov(H_{ck}, P_{total})}{\sqrt{Var(H_{ck}) \cdot Var(\hat{P}_{total})}} \cdot \sqrt{\frac{Var(P_{total})}{Var(P_{total})}} \\
&= \rho(H_{ck}, P_{total}) \cdot \hat{p} \cdot \sqrt{\frac{Var(P_{total})}{Var(\hat{P}_{total})}} \qquad (1)
\end{aligned}$$

Example. Figure 1 shows DPA results of three attacks. The leftmost picture is the result of a "reference attack' on an unprotected AES implemented. The middle picture is the result of a standard DPA attack on an implementation, where prior to the execution of AES either 0 or 1 dummy operation was inserted. The rightmost picture is the result of a standard DPA attack on an implementation where prior to the execution either 0 or 2 dummy operations were inserted. It is clearly visible that the middle picture illustrates that too little displacement can render the approach useless (the maximum correlation is still that of the attack on an unprotected implementation). The reason for that is that some operations might take longer than the introduced delay, in which case there will be an overlap between the time periods when the displaced operation takes place, rendering the approach useless. The rightmost figure illustrates nicely the application of (1). In this example/analysis we assume that an adversary doesn't change their strategy, i.e. they still perform a standard DPA attack. But is this realistic? Sure an adversary will attempt to compensate for countermeasures.

One way of dealing with traces in which the target intermediate value does not occur at some fixed time across all traces is to use alignment techniques. Trace alignment is about ensuring that the targeted intermediate value occurs at some small interval across all traces. Discussing alignment fully is beyond the scope of this unit, but a basic technique is to "integrate" trace points over some set interval. The exact type of

"integration" can vary, and for simplicity we will assume that an adversary will simply sum over l trace points. In principle the target intermediate value can occur in any of the l points. Therefore we say that it will occur in the first P_1 (we could have chosen any of the l points), and we analyse the correlation in that respect. All other points are considered to be noise, i.e. they are statistically independent of each other and also of H_{ck}. Therefore, it holds for $i = 2, \ldots, l$ that $E(H_{ck} \cdot P_i) - E(H_{ck}) \cdot E(P_i) = 0$. The correlation coefficient for the correct key hypothesis is thus:

$$\begin{aligned}
\rho(H_{ck}, \sum_{i=1}^{l} P_i) &= \frac{E(H_{ck} \cdot \sum_{i=1}^{l} P_i) - E(H_{ck}) \cdot E(\sum_{i=1}^{l} P_i)}{\sqrt{Var(H_{ck}) \cdot (Var(\sum_{i=1}^{l} P_i))}} \\
&= \frac{E(H_{ck} \cdot P_1 + H_{ck} \cdot \sum_{i=2}^{l} P_i) - E(H_{ck}) \cdot (E(P_1) + E(\sum_{i=2}^{l} P_i))}{\sqrt{Var(H_{ck}) \cdot Var(P_1)} \sqrt{\frac{\sum_{i=1}^{l} Var(P_i)}{Var(P_1)}}} \\
&= \frac{E(H_{ck} \cdot P_1) - E(H_{ck}) \cdot E(P_1)}{\sqrt{Var(H_{ck}) \cdot Var(P_1)} \sqrt{\frac{\sum_{i=1}^{l} Var(P_i)}{Var(P_1)}}} \\
&= \frac{\rho(H_{ck}, P_1)}{\sqrt{\frac{\sum_{i=1}^{l} Var(P_i)}{Var(P_1)}}} \qquad (2)
\end{aligned}$$

If the variances in the trace points are roughly equal, then we can simplify this expression further and it becomes

$$\rho(H_{ck}, \sum_{i=1}^{l} P_i) = \frac{\rho(H_{ck}, P_1)}{\sqrt{l}}. \qquad (3)$$

Example. Figure 2 illustrates the effect of integration on an AES implementation were some SubBytes operations are executed in a random order. This means that, e.g. if two SubBytes operations are shuffled, both can occur at "2" positions in the execution (with probability 1/2) and thus we expect to see two DPA peaks. If an adversary performs a standard DPA attack on such traces we thus see the results as in the left panel of Fig. 2. If an adversary first integrates over a suitable interval in the power traces, they can improve the correlation, such as predicted by (2). The effect of this is shown in the right panel of Fig. 2.

5 Secret Sharing — Masking

Secret sharing is a well known technique in cryptography. The basic idea of secret sharing is to split up sensitive variables into several shares. Each share looks random: therefore if an adversary gets access to some shares (but not all) they learn nothing about the sensitive value. This idea also works in the context of leakage: if we secret share intermediate values, and their shares look random, then they may also leak information that

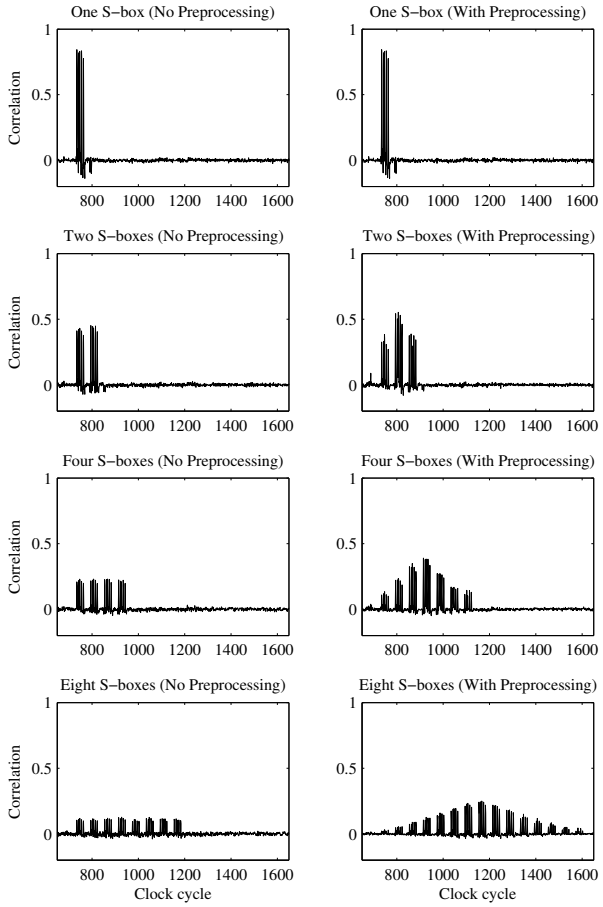

Figure 2: Shuffling of AES SubBytes. Taken from [1]

looks random (i.e. it is unrelated to the sensitive value) and therefore an adversary does not learn any information about the sensitive value. However, the problem in the context with leakage is that a device must process all the shares, and therefore it seems clear from the outset that if the adversary attempts to "bring together" leakage information from different shares they will eventually succeed.

Nevertheless secret sharing is one of the standard countermeasures that gets implemented in practice. Often it is used under the term of masking, in particular if the simplest option of splitting an intermediate into two shares is being used. I will stick with this convention and now discuss the simplest and most widely form of masking.

We split each (intermediate) value v into two shares by generating a random value m (the mask, which is one of the shares) and exclusive oring it with v to generate the masked value v_m (the other share): $v = (v_m, m)$ whereby $v = v_m \oplus m$. The mask is generated internally, it must be drawn at random from a uniform distribution, and it must not be revealed to the adversary. To protect all intermediate values, the default strategy is to mask the AES state as well as the key schedule at the start of encryption (decryption).

Linear operations (linear w.r.t. the Boolean exclusive-or) preserve masks in an simple manner. Let's look at AddRoundKey as an example. AddRoundKey is based on exclusive-oring a state byte with the corresponding key byte. Thus AddRoundKey($p \oplus m$, $k \oplus m_1$) = AddRoundKey(p, k) $\oplus(m \oplus m_1)$. Shiftrows doesn't change mask values at all. MixColumns is in fact also linear w.r.t. to the Boolean exclusive or: MixColumn$(p_1 \oplus m_1, p_2 \oplus m_2, p_3 \oplus m_3, p_4 \oplus m_4) =$ MixColumns$(p_1, p_2, p_3, p_4) \oplus$ MixColumns(m_1, m_2, m_3, m_4).

Non-linear operations (w.r.t. the Boolean exclusive-or) do not preserve masks in a simple manner; this hence concerns the AES SubBytes operation: $S(x \oplus m) \neq S(x) \oplus S(m)$. If SubBytes is implemented via a table look-up, then a relatively simple and pain-free work around is as follows. We produce a masked table S_m with the property $S_m(v \oplus m) = S(v) \oplus m$. Generating such a table is a simple process (we just run through all input values v and store $S(v) \oplus m$ in S_m. However, it is unavoidable to run through all inputs v, look up $T(v)$ and store $T(v) \oplus m$ for all m in the masked table. Obviously, if we wished to mask all state bytes (and their SubBytes step) with different masks, then we would need to produce such a table for each mask m (i.e. we would end up storing 16 tables). Consequently, the computational effort and the amount of memory increases with the number of masks.

5.1 Example

I now provide an example of a masked AES software implementation. The masking scheme uses Boolean masks only, and it is tailored to a typical 8-bit AES software im-

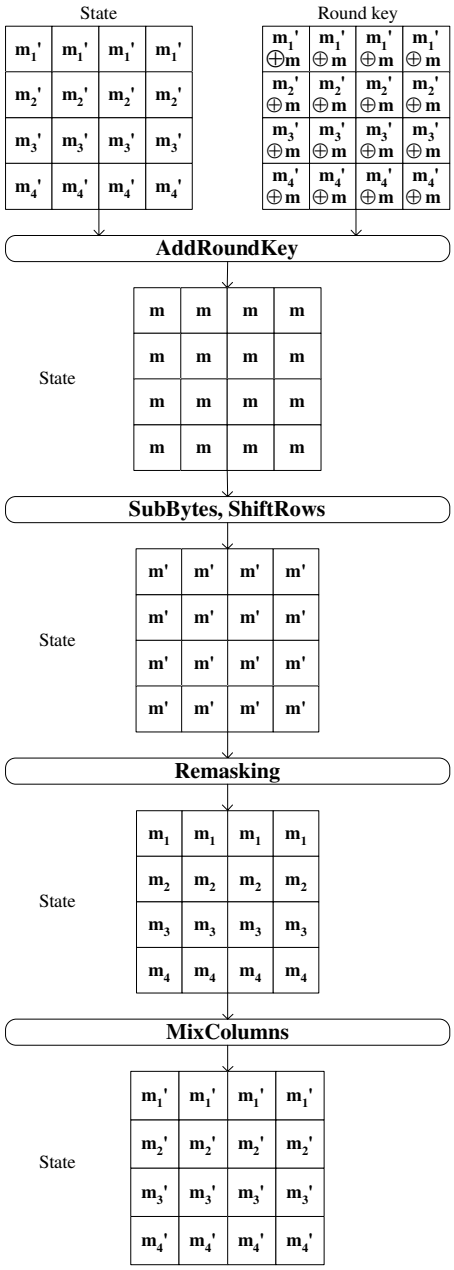

Figure 3: The AES round functions change the masks of the AES state bytes. Taken from [1]

plementation (no T tables).

AddRoundKey: Since the round key bytes k are masked with m in the scheme, performing AddRoundKey automatically masks the bytes d of the state: $d \oplus (k \oplus m) = (d \oplus k) \oplus m$.

SubBytes: We use a masked S-box table for this operation. To deal with potential HD leakage I suggest to produce a masked table that applies a fresh mask to the output.

ShiftRows: This operation does not affect (and is not affected by) masking.

MixColumns: The MixColumns operation requires more attention because MixColumns uses the bytes from different rows of a column. Hence, MixColumns requires at least two masks. If only two masks are used for the bytes of a column, then MixColumns needs to be done very carefully to make sure that all intermediate values stay masked. Instead, I suggest to mask each row with a separate mask.

We use six independent masks in our scheme. The first two masks, m and m', are the input and output masks for the masked SubBytes operation. The remaining four masks m_1, m_2, m_3, and m_4 are the input masks of the MixColumns operation. At the beginning of each AES encryption we compute a masked S-box table S_m such that $S_m(x \oplus m) = S(x) \oplus m'$ and then we compute the output masks for the MixColumns operation by applying this operation to (m_1, m_2, m_3, m_4). We denote the resulting output masks of MixColumns by (m'_1, m'_2, m'_3, m'_4).

Figure 3 shows how the masks flow from one AES step to the next. At the beginning of each round, the plaintext is masked with m'_1, m'_2, m'_3, and m'_4. Then, the AddRoundKey operation is performed and because the round key is masked too the masks change from m'_1, m'_2, m'_3, and m'_4 to m. Next, the table look-up with the S-box table S_m changes the masks to m'. ShiftRows has no influence on the masks. Before MixColumns, we change the mask from m' to m_1 in the first row, to m_2 in the second row, to m_3 in the third row, and to m_4 in the fourth row. MixColumns changes the masks m_i to m'_i for $i = 1, \ldots, 4$, which means we have reinstated the masks from the beginning of the round. Consequently, we can mask an arbitrary number of rounds in this way. At the end of the last encryption round, we remove all masks.

5.2 Effectiveness and Cost

The cost of masking mainly depends on the number of masks and it also increases substantially with the number of masks (i.e. the amount of randomness).

Security proofs for masking assume that the leakage of masked intermediate values is independent of the unmasked intermediate values, if the masked and the unmasked values are independent of each other. However, this assumption does not always hold

in practice. Often, the instantaneous power consumption of a device depends not only on one, but on several values. Such a combined power consumption of two or more intermediate values can make implementations insecure, even if all intermediate values are provably secure.

For example, assume a device that leaks the Hamming distance of two intermediate values. Hence, if two masked intermediate values, which have the same mask, are processed consecutively, the power consumption is related to the Hamming distance of the unmasked values because $HD(v_m, w_m) = HW(v_m \oplus w_m) = HW(v \oplus w)$.

6 Food for thought and Exercises

Task A: Correlation attacks on Hiding

Adjust your implementation so you perform the SubBytes operation and/or AddRoundKey in a random order. Then try and attack your implementation. What does an SPA tell you? Can you succeed with a DPA attacks? If so what can you say about the number of needed traces?

Task B: Correlation attacks on Simulated Masking

An easy way to assess attacks/countermeasures is to perform some simulations. A simulation is about "mimicking" what happens in a real attack. For instance you could write a Python script that take a value, computes SubBytes and leaks the (noisy) Hamming weight of the output. You could now secret share the SubBytes input and leak on the mask and the masked value, produce a masked SubBytes table and therefore also produce a masked SubBytes output and leak on that (so your leakage traces would have three (noisy) values $HW(m), HW(v \oplus m), HW(S(v) \oplus m)$. Consider an attack on these traces that is based on only using a subset of all traces, whereby the subset is determined by a threshold for m. How well does this simple attack work?

Task C: Non Uniform Masking

Consider what would happen if masks were not chosen at random from a uniform distribution. Would this be secure?

7 Quiz

If you have worked through these notes and spent some time on the tasks, you should be able to answer the following questions.

1. True or false: Masking is a technique that is related to homomorphic encryption. (Explain your answer.)
2. True or false: Masking requires a random number generator. (Explain your answer.)
3. True or false: Hiding is about concealing the signal by adding a lot of noise. (Explain your answer.)
4. True or false: Shuffling differs from masking because it requires less randomness. (Explain your answer.)
5. True or false: Implementing dummy instructions requires to think about SPA resistance of the dummies. (Explain your answer.)
6. True or false: If an implementation we first process $x \oplus m$ and then at a separate point in time m, we may leak information on x. (Explain your answer.)

References

[1] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards* . Springer, 2007.