Applied Cryptography - AttackHW Stage 3

Patrick Lee

1 Introduction

This document specifies the implementation of AES-128 Encryption such that it is secure against key recovery by way of the correlation power analysis attack (described by Brier et al. **[1]**) implemented in **Stage 2**. The chosen counter-measure makes use of bothh temporal re-ordering in certain operations randomised boolean masking to obfuscate the state matrix during encrpytion, increasing the difficulty in successfully modelling the relationship between chosen plaintexts, the encryption key and the observable power consumption during encryption. The countermeasures require a source of randomness from device $\mathcal{D}$, which is assumed to be entropic.

The implemented counter-measure was shown to be successfull in preventing key recovery by a CPA attack making use of 1000 power traces, each with approximately 80,000 samples and attacking the first round of encryption.

2 Design and Implementation

The design used, in principle, reflects much of that described by Herbst et al. **[2]**, but excluding both the dummy rounds introduced at the beginning and end of encryption. The paper also suggested that temporal re-ordering could be applied to all four of the AddRoundKey, SubBytes, ShiftRows and MixColumns round functions. In this specification, only the first two are randomised.

2.1 Temporal re-ordering

During the AddRoundKey and SubBytes functions, the sequence of the processing of each byte of the state matrix can be randomised as each byte is processed independently. As the attack attempts to model the hamming weight of the result of the SubBytes function, applying this temporal re-ordering to the SubBytes function should reduce the correlation found by the attacker, as position of the actual power consumption samples in each trace will not be where the attacker expects relative to the modelled hamming weights. Clavier et al. showed in **[3]** that the number of power measurements required for a successful attack increased by a factor of k^2 where $\frac{1}{k}$ is the probability of a hypothetical measurement being temporally aligned with its corresponding power measurement, in this case $k = 16$. Put another way, it would be expected that an attack requiring 1000 power traces would instead require $> 256,000$ traces to achieve the same success.

Herbst et al. suggested in **[2]** the addition of a random number of 'dummy' AES operations, acting on a 'dummy' state, alongside the temporal re-ordering. This would make it impossible for the attacker to know which AES round is the 'real' first round, with information leakage further minimized by assigning the 'dummy' state a base address with the same hamming weight as that of the real state. That being said, the 'real' first round of encryption would still be apparent somewhere in the full power consumption trace. If n dummy rounds were added, the required attack time would increase by a factor of n. As this isn't a drastic increase in difficulty for the attacker, and adding several dummy rounds would constitute a fair amount of overhead, as well as the increased memory footprint from storing the 'dummy' state, this countermeasure was not included in the implementation.

An implementation of the fisher-yates shuffle **[4]** is used in dis-aligning the execution of the AddRoundKey and SubBytes functions during the first encryption round. An update of the PRNG described in **2.1** is made prior to each of these, so the shuffle can use a fresh set of random bytes to derive a shuffled set of indices. This update call, and the requirement for two further iterations over the state matrix, as well as the modulus operation involved, contribute to a small overhead.

2.2 Masking

In a masked implementation all intermediate values in the state matrix values a are concealed by a random mask, causing a power consumption which is not predictable by the adversary in the CPA attack.

Prior to each encryption, 6 random mask bytes are derived from the randomness supplied by the user.

The MixColumnMasks are created by passing the ColumnMasks through the MixColumns function, (M1', M2', M3', M4') = MixColumns(M1, M2, M3, M4). These are used to account for the changing in masking produced by the MixColumns function when run on the state matrix. The masked SBox, S' is precomputed such that $S'(x \oplus M) = S(x) \oplus M'$. Before the start of the encryption, the state matrix containing the plaintext is masked columnwise with the MixColumnsMasks, such that $S_{i,j} = S_{i,j} \oplus M_j$. The round-key is also masked such that $RK_{i,j} = RK_{i,j} \oplus M_j \oplus M_4$. This ensures that the state matrix will be masked and unmasked appropriately by the round-key in the AddRoundKeyFunction. It can be assumed that the round-key will remain masked in this manner (apart from before the final AddRoundKey where it will be of the form $RK_{i,j} = RK_{i,j} \oplus M_j$).

Consequently, the expansion step needs to remove and apply masks appropriately. As peforming XOR between two key bytes, each with the same masks applied, will essentially produce an unmasked result, it suffices just to remask each key byte after expansion for the latter 3 rows of the round-key matrix. For the first row, the masked SBox is required to account for the analagous use of the SBox in the regular KeyExpansion algorithm. These two cases are handled as $i = 0 : RK'_{i,j} = S'(RK_{i,(j'+12)} \oplus M_{j'}) \oplus M'_4 \oplus RK_{i,j}$ where $j' = j + 1 \pmod 4$, and $i > 0 : RK'_{i,j} = RK_{i-1,j} \oplus RK_{i,j} \oplus M_4 \oplus M_j$ respectively. In the final round, the mask M_4 needs to be removed to ensure that the AddRoundKey function will unmask the state matrix completely. This is can simply be performed as $RK'_{i,j} = RK'_{i,j} \oplus M_4$.

2.3 Random Number Generation

As both **2.1** and **2.2** require a source of randomness, a psuedo-random number generato was developed, based off the X9.31 in **[5]**, which would take the randomness given by the user and expand it into cryptographically secure psuedo-random numbers. After beinng given a seed, the implementation performs 3 AES encryptions, each of which is a standard encryption with no implemented countermeasures, on each update. This implementation requires 2 different random number sequences - one for each of the fisher-yates permutations. The random values for the masks are derived from the first of these sequences. It was ultimately decided to only use 2 rounds for each of the AES encryptions involved. Doing produced a considerable reduction in latency (see **Figure 1**) and using a minimum of two rounds is shown in **[6]** to result in full diffusion - changing a single bit of the AES input would result in a change in every bit of the output. Whilst, admittedly sacrificng some robustness in the implementation of this update function, the performance increase makes this trade-off favourable.

The specified version of this algorithm makes use of the system time, effectively as a nonce in its encryption process. As this is unavailable on device $\mathcal{D}$, it must be provided by the host $\mathcal{H}$, for every use of the update function. As 2 update cycles are performed, 48 bytes of randomness are required from $\mathcal{H}$ - 16 for the initial state or 'seed' and 2 'nonce' values. As the generator is deterministic, the security of this implementation relies on the $\mathcal{H}$ sending a different series of randomness bytes each time.

3 Performance and Security Analysis

The results in **Figure 1** show that the nature of the PRNG (**2.3**) has a large impact on the efficiency of this implementation, regardless of whether one or both of **2.1** and **2.2** are used. The correlations in the CPA attack shown in **Figure 2** (and the corresponding key recovery successes) show that masking by itself is not as effective in protecting the key as desired, although still has a considerable effect on the correlation, and therefore efficacy of the CPA attack. A likely source of weakness in the masking implementation is in the masked key expansion. There could be some information leakage from bytes of the round key being unmasked and re-masked with the same masks that are used on the state matrix. If able, I would have explored this further to perhaps redesign the key expansion in an attempt to mitigate this. Given that the key is fully protected under the combination of both temporal dis-alignment and masking, a revised masking implementation should provide very strong protection against CPA.

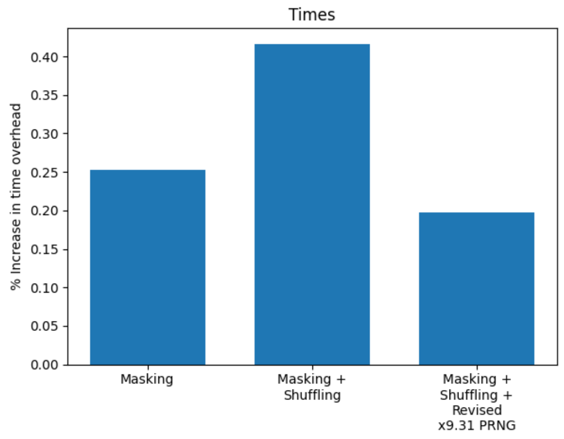

Figure 1

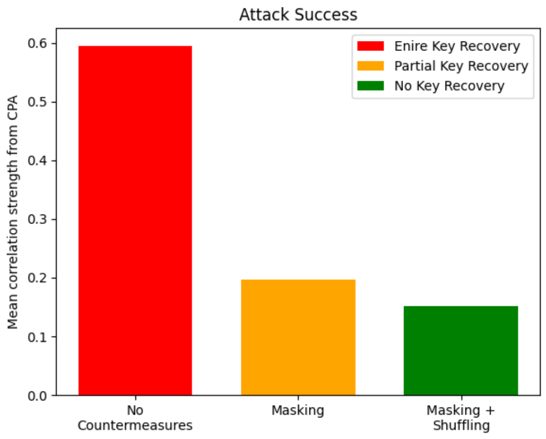

Figure 2

References

[1] E. Brier, C. Clavier, and F. Olivier *Correlation Power Analysis with a Leakage Model* In: *Cryptographic Hardware and Embedded Systems (CHES).* LNCS 3156. Springer-Verlag, 2004.

[2] C. Herbst, E. Oswald, and S. Mangard *An AES Smart Card Implementation Resistant to Power Analysis Attacks* In: *Applied Cryptography and Network Security (ACNS).* . LNCS 3989. Springer-Verlag, 2006.

[3] Clavier, C., Coron, JS., Dabbous, N. (2000) Differential Power Analysis in the Presence of Hardware Countermeasures. In: Koç, Ç.K., Paar, C. (eds) Cryptographic Hardware and Embedded Systems — CHES 2000. CHES 2000. Lecture Notes in Computer Science, vol 1965. Springer, Berlin, Heidelberg. `https://doi.org/10.1007/3-540-44499-8_20`

[4] Fisher-Yates shuffle `https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle`

[4] *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry*. American National Standards Institute (ANSI) Standard X9.31. 1993.

[5] The Design of Rijndael. ISC. Springer, Heidelberg (2020). `https://doi.org/10.1007/978-3-662-60769-59`