A custom software implementation of the AES-128 decryption [Nat01] for a smart-card emula-
tor (Fig. 1) has been developed and attacked success-
cessfully using a new timing attack. AES is widely used in
today’s embedded applications and is consid-
ered to be secure against all cryptanalytic attacks
known to date. Despite its mathematical security, its implementa-
tions may be vulnerable to Side-channel Analysis (SCA) attacks [Koc96],
which exploit the non-intended leakage of information during the physical op-
eration of the cipher. The smart-card emulator decrypts arbitrary ciphertext using a secret key. The
non-linear byte substitution InvSubBytes has been optimized to save 20% code size by direct calculation of the substitution values (i.e. no look-up table). But the execution time per decryption typically increases from 20 ms to 200 ms.

The Modified Dynamic S-box Implementation

InvSubBytes consists of two steps, an affine transformation and the compu-
tation of the multiplicative inverse in $\mathbb{F}_8$ [2]. The multiplicative inverse has
been computed using a simple brute-force search algorithm. It unintentionally
introduces a data-dependent execution time when it puts out 0, as the search
is not executed in this case, thus it needs significantly less time for that.
The search itself shows nearly constant execution time and is based on the
computation of the logarithm of the input by successively computing the
power of a primitive element [LN94].

Timing Attack

The execution time depends on the number of zeros occurring during the de-
cryption at the output of InvSubBytes.

Idea: measure the execution time and store the plaintext of N random decryp-
tions, estimate the number of zeros, gather the secret key

We know that in the last round of the AES decryption

\[ k_i = p_i \otimes \text{InvSubBytes}(s_i), \quad k_i : \text{Byte i of the secret key,} \quad (1) \]

\[ p_i : \text{Byte i of the plaintext,} \quad s_i : \text{Byte i of the internal state.} \quad (2) \]

If InvSubBytes($s_i$) = 0;

\[ k_i = p_i, \quad (3) \]

i.e. in this case we know the correct key byte.

After the number of zeros is estimated, we don’t know which of the 160 inver-
sions resulted in zero, but we can compute the probability that it occurred in the
last round. If that is the case, it provides so-called correct information, as (3) is true for some i. Otherwise it provides so-called misleading information, c.f. Fig. 2.

In order to combine the information acquired by the N individual measure-
ments, a maximum likelihood (ML) estimator has been used. A matrix with
the likelihoods corresponding to $k_i = 0, \ldots, 255$ has been created. After all
measurements have been processed, the most likely entries in the matrix
form the key.

References


Acknowledgement

Thanks to Prof. Georg Sigl and Oscar Guillen Hernandez for their valuable review and feedback.