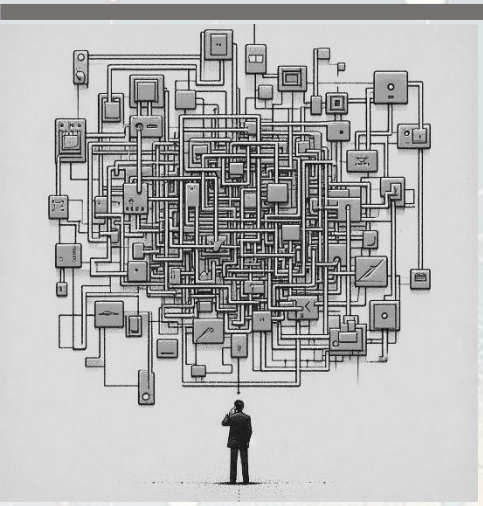
Speed Optimization of AES Function



AES is a widely used symmetric encryption algorithm standardized by the National Institute of Standards and Technology (NIST). Algorithmic optimization involves refining the mathematical operations and structures of the AES algorithm to enhance its speed and resource efficiency.

Focusing more on algorithmic aspects (cf. hardware-specific optimizations), we can ensure consistent performance across diverse computing environments, ranging from embedded systems to high-performance computing clusters.

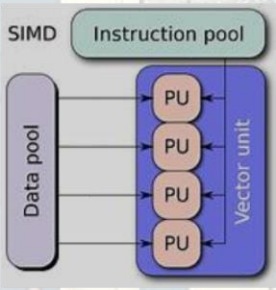
The goals of AES optimization can vary, including speed (making it faster), memory (reducing the footprint), or area (minimizing physical space). Optimization can occur at different levels. At the algorithmic level, it involves improving the steps in an algorithm to enhance its execution. At the implementation level, it involves optimizing code by utilizing native instruction sets and compute architecture. Top-5 optimization tricks are as follows: -

Bil-Slicing: Instead of having a single variable storing an n-bit number, we have n variables (slices). Each variable stores one bit of the number. This allows for parallel execution of operations across multiple blocks of data. Further, a bit-sliced implementation, which is based solely on logical operations and does not involve tables, is immune to timing attacks.

Number Theory Tricks: For example, suppose we have n numbers a1, a2, ..., an and we want to compute their inverses 1/a1, 1/a2, ..., 1/an modulo a prime p. Instead of computing each inverse separately,, Montgomery's trick allows us to compute all inverses with just one inversion and 3n multiplications <https://doi.org/10.1007/978-3-642-21969-6_10>

Look Up Tables (LUT): LUTs can significantly improve the speed of AES execution by replacing complex computations with simple array indexing. Operations like SubBytes (byte substitution) and MixColumns (mixing of column data) can be precomputed and stored in lookup tables. When the algorithm runs, instead of performing these computations, it simply retrieves the precomputed results from the lookup tables.

Vectored Computation: In AES, SIMD can be used to process multiple blocks of data at once, significantly increasing throughput. E.g., MixColumns step can be optimized for four blocks as described in the ARMv8 ASIMD implementation*. This involves processing four 128-bit AES states simultaneously.
*https://doi.org/10.1007/978-3-030-40921-0_5



Special Instructions: In modern x86 architectures, AES New Instructions (AES-NI) is a set of instructions that enable fast and secure data encryption and decryption. AES-NI consists of seven instructions and supports all usage and modes of operations of AES. ***AESENC*** performs one round of an AES encryption flow. ***AESENCLAST*** performs the last round of an AES encryption flow. Other instructions in AES NI supports decryption and key expansion functions. Similarly, in ARM processors, particularly those based on the ARMv8-A architecture, support AES computation with dedicated instructions

AES optimization is the art of finding harmony between speed, inter-operability, memory footprint, and security.

Sara Malik <smk@PakCrypt.Org> is an InfoSec professional in PakCrypt outreach program.