Crypto Engineering '21

✧

Block cipher design ~ AES

Pierre Karpman
pierre.karpman@univ-grenoble-alpes.fr
https://membres-ljk.imag.fr/Pierre.Karpman/tea.html

2021–99–99

What's up?

From the previous lectures, we know (somehow) how to provide:

- Confidentiality/Semantic security of a message
- Authenticity of communications
- Integrity

if given access to the right primitives. But:

How do you design primitives?

Today's focus: block ciphers → AES

Advanced Encryption Standard: AES

The AES is:

- A family of three block ciphers of block size 128 bits; key size 128, 192 or 256 bits
- Designed in '98 by Daemen and Rijmen
- Winner of an academic competition run by the (American) NIST
- Standardized in 2001 by the NIST

First things first

Building a BC, general objectives:

- Be secure
- Be efficient
- Be easy to implement
- Be versatile

General strategy:

- Use small/simple building blocks
- Use an *iterative* structure

Justifying the strategy

- It is hard/impossible(?) to define a BC in a single operation
- Complex operations are expensive
- The ability to do fine-tuning is useful

$\Rightarrow$

Most BCs are based on iterations of a small set of simple operations. Typically:

- Modular addition + bitwise XOR + rotations (ARX)
- Lookup tables
- Simple (non-)linear functions
- Bit permutations

Iterative structure: details

BCs usually use:

- A *round function* $\rho : \{0,1\}^{\kappa'} \times \{0,1\}^n \to \{0,1\}^n$
  - Takes as input a *round key* and an "intermediate state" that gets updated
- A *key schedule* $\sigma : \{0,1\}^{\kappa} \times \mathbb{N} \to \{0,1\}^{\kappa'}$
  - Takes as input a *master key* and a round number and returns a round key

Resulting structure $\rightsquigarrow$ blackboard

Rationale:

- It is "easy" to define a small round function, a key schedule
- More rounds $\Rightarrow$ better security (*mostly* true)

A particular round structure: SPNs

SPN: Substitution-Permutation Networks. Build a round function from:

- Non-linear (over $\mathbb{F}_2$) *Substitution boxes* (S-boxes): locally break any exploitable structure
- Linear (ditto) *permutations* or more generally, matrices: ensure that local changes spread globally

Many tradeoffs possible for the size/quality of components

Sometimes traced back (?) to Shannon's idea of composing "confusion" and "diffusion"

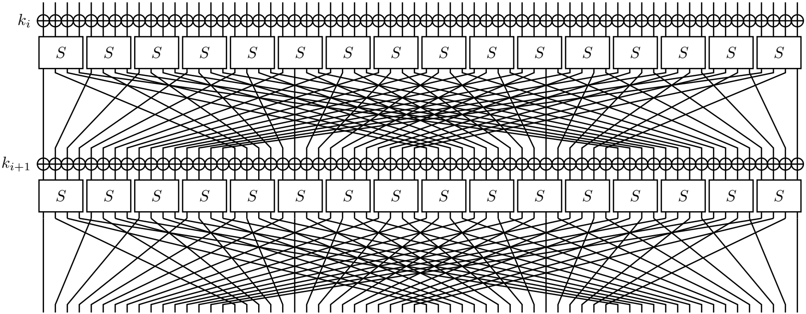
SPN as in the AES

- Use a square state of $16 = 4 \times 4$ bytes
- S-boxes are over 8 bits ("SubBytes")
- Permutation is the composition of
  - Inter-column light diffusion ("ShiftRow")
  - Column-wise heavy diffusion ("MixColumn")
- The round key is just XORed to the entire state ("AddRoundKey") (no details about the rest today)
- Full structure $\rightsquigarrow$ blackboard

Remark: This is a rather heavy round function (only ten rounds for AES-128)

Intermission: PRESENT round function

Some (other) SPNs have a *very* simple round function. Ex. PRESENT:



Back to AES: more details

SubBytes:

- The S-box $\mathcal{S}$ is well-chosen to provide very strong protection against *differential* and *linear* cryptanalysis
- It has a strong algebraic structure over $\mathbb{F}_{2^8}$, masked by an affine mapping over $\mathbb{F}_2$

MixColumn:

- Defined as a matrix-vector multiplication over $\mathbb{F}_{2^8}^4$
- The matrix is the *redundancy part* of an $[8, 4, 5]_{\mathbb{F}_{2^8}}$ linear code, that is *maximum distance separable* (MDS)

$\mathbb{F}_{2^8}$ arithmetic

- MixColumn requires operations over $\mathbb{F}_{2^8}$ (the finite field with 256 elements)
- The representation of $\mathbb{F}_{2^8}$ used in AES is as $\mathbb{F}_2[X]/\langle X^8 + X^4 + X^3 + X + 1\rangle$
- Using "integer notation", the MixColumn matrix **M** is then:

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

Brief differential focus example

Why this choice for the matrix (/S-box)?
Important differential properties:

- For all $\Delta_{\text{in}}, \Delta_{\text{out}} \in \mathbb{F}_2^8$,
  $\#\{x \in \mathbb{F}_2^8 \text{ s.t. } \mathcal{S}(x) \oplus \mathcal{S}(x \oplus \Delta_{\text{in}}) = \Delta_{\text{out}}\} \in \{0, 2, 4\}$
- So $\max_{(\Delta_{\text{in}}, \Delta_{\text{out}})} \Pr[\mathcal{S}(x) \oplus \mathcal{S}(x \oplus \Delta_{\text{in}}) = \Delta_{\text{out}} : x \xleftarrow{\$} \mathbb{F}_2^8] = 2^{-6}$
- $\rightsquigarrow$ "*It is hard to control the behaviour of input differences to an S-box*"
- $\min_{\vec{x} \neq \vec{0}} wt(\vec{x}) + wt(\mathbf{M} \cdot \vec{x}) = 5$
- $\rightsquigarrow$ "*It is hard to restrict differences to a few S-boxes*"

AES(-128): how secure?

- Many attacks exist against the AES
  - Square, Impossible differential, MiTM, Yoyo, etc.
- Some are very efficient but only work on a few rounds (cf. TP)
- No key-recovery attack on 7/10 rounds takes time $< 2^{100}$
- Some better attacks exist in very strong models (usually not a problem)
- Still today, 10 rounds offer a good security/efficiency tradeoff for most use-cases

What about implementation now?

Naive needs:

- ShiftRow: cabling/moves
- MixColumn: multiplication by constants in $\mathbb{F}_{2^8}$
- SubBytes: table lookups

Implementation (cont.)

Naive MixColumn (`xtime`) issues:

- Not efficient
- Leaks information about inputs
- $\rightsquigarrow$ Can do better

Common AES implementation techniques:

- All by table lookups
- Block-wise vectorization w/ shuffles; very nice! (Hamburg, 2009)
- Parallel vectorization/ "bitslicing"
- Use hardware instructions ('cause it's already implemented...)

Implementation: always looking up

Table lookups details:

- Not the best approach, but pretty easy
- Idea: $\vec{\alpha} \cdot \mathbf{A} = \sum_i \vec{\alpha}_i \cdot \mathbf{A}_i$
- Use this to compute $\mathbf{M} \cdot \vec{x} = \vec{x}^t \cdot \mathbf{M}^t$
  - For every row $\mathbf{M}_i^t$, for every $\alpha \in \mathbb{F}_{2^8}$, precompute `T[i][`$\alpha$`]` $= \alpha \mathbf{M}_i^t$
  - Requires $256 \cdot 4 \cdot 4 = 4$kB of static data
  - Then compute `MixColumn(x)` as `T[0][x[0]] ^ T[1][x[1]] ^ T[2][x[2]] ^ T[3][x[3]]`
- Optimizations:
  - Fold in the S-box calls into `T`
  - Possible tradeoff: use the (circulant) structure of the matrix to store only one row

Table drawbacks

Table implementations are "classical", but they

- Need memory (not the best for constraint devices)
- May leak information (via e.g. cache attacks)

Cache attacks main observations:

- Table accesses depend on secret data
- Access times may depend on micro-architectural effects (e.g. presence/absence of data in cache)
- $\rightsquigarrow$ Can learn key material by measuring running time

In some context, additional protection against other side-channel attacks may also be needed! (cf. φ security)

AES extensions

The AES inspired many later designs, e.g.:

- LED (Guo et al., 2011; lightweight variant)
- Kiasu (Jean et al., 2014; tweakable variant)
- AESQ (Biryukov & Khovratovich, 2014; wide permutation variant)
- Etc.

But the original cipher is still up to date → the sensible default choice for a block cipher

Light summary

Symmetric encryption relies on:

- Primitives ((Tweakable) block ciphers, MACs, hash functions, permutations, ...)
- Operating modes
- Everything has to be implemented at some point (!)

$\Rightarrow$ Many things to study; many things that can go wrong