Crypto Engineering (GBX9SY03) TP – Square attack on rounds of the AES block cipher

Hatim Ridaoui

October 2025

1 Exercise 1: Warming up

Q1.

What xtime does and why we need it. In AES, the mixcolumns step multiplies each 4byte column by an mds matrix over the finite field $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(m(X))$ with $m(X) = X^8 + X^4 + X^3 + X + 1$. All the required byte products (by 2 and by 3) can be buil from a single primitive: multiplication by X (i.e., by 2 in the polynomial basis). xtime computes $X \cdot a \bmod m(X)$ for a byte a. products by 3 are then $xtime(a) \oplus a$, and mixcoluns can be implemented with a few calls to xtime and XORs.

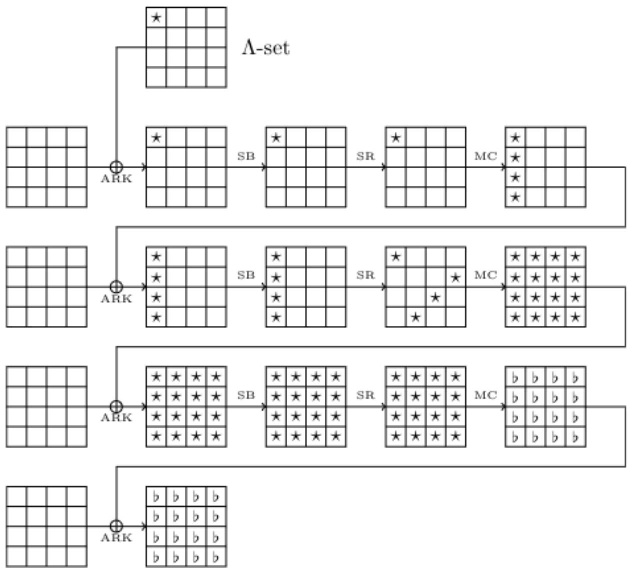


Figure 1: The 3-round Square distinguisher

Reference implementation The file defines

```
uint8_t xtime(uint8_t p) {
    uint8_t m = p >> 7;
    m ^= 1;
    m -= 1;
    m &= 0x1B;
    return (p << 1) ^ m;
}
```

The constant `0x1B` is the binary encoding of $X^4 + X^3 + X + 1$, i.e., the lower-degree part of $m(X)$ once the X^8 term has been removed for reduction.

Correctness proof Let $p \in \{0, \dots, 255\}$. In polynomial form,

$$\texttt{xtime}(p) = \begin{cases} (p \ll 1) & \text{if the MSB of } p \text{ is } 0, \\ (p \ll 1) \oplus 0x1B & \text{if the MSB of } p \text{ is } 1, \end{cases}$$

which is exactly $X \cdot p \bmod m(X)$.

The code realizes this *branchlessly* by synthesizing the mask:

1. `m = p >> 7` sets $m \in \{0, 1\}$ to the MSB of p.
2. `m ^= 1; m -= 1;` maps m to $0x00$ if MSB$=0$, and to $0xFF$ if MSB$=1$ (since $0 \oplus 1 = 1$, $1 - 1 = 0$; and $1 \oplus 1 = 0$, $0 - 1 \equiv 255$).
3. `m &= 0x1B;` yields either $0x00$ or $0x1B$ accordingly.
4. Finally `(p<<1) ^ m` implements the two cases above.

Thus the function returns $p \cdot X \bmod (X^8 + X^4 + X^3 + X + 1)$ for all p, in constant time with respect to the data.

Variant for another irreducible polynomial Consider the irreducible aswell module

$$m'(X) = X^8 + X^6 + X^5 + X^4 + X^3 + X + 1.$$

When reducing $X \cdot p$ modulo $m'(X)$, the high bit case requires XOR with the binary encoding of $X^6 + X^5 + X^4 + X^3 + X + 1$, namely

$(1 \ll 6) + (1 \ll 5) + (1 \ll 4) + (1 \ll 3) + (1 \ll 1) + (1 \ll 0) = 64 + 32 + 16 + 8 + 2 + 1 = 123 =$ `0x7B`.

A drop-in, constant-time variant is therefore:

```
static inline uint8_t xtime_mprime(uint8_t p) {
    uint8_t m = p >> 7;  // MSB
    m ^= 1;
    m -= 1;              // 0x00 if MSB=0, 0xFF if MSB=1
    m &= 0x7B;           // lower part of m'(X) without X^8
    return (p << 1) ^ m; // multiply by X modulo m'(X)
}
```

If we keep the same matrix coefficients (2 and 3) but switch the modulus from m to m', mix still evaluates over the field structure—only the `xtime` reduction constant changes (from `0x1B` to `0x7B`).

Q2. See code in main-q2.c

Let $K^{(r)} \in \{0,1\}^{128}$ be the r-th round key of AES-128, written as bytes $K^{(r)}[0], \ldots, K^{(r)}[15]$. The forward expansion from $K^{(r)}$ to $K^{(r+1)}$ is

$$\begin{aligned}
K^{(r+1)}[0] &= K^{(r)}[0] \oplus S\Big(K^{(r)}[13]\Big) \oplus \mathrm{RC}[r],\\
K^{(r+1)}[1] &= K^{(r)}[1] \oplus S\Big(K^{(r)}[14]\Big),\\
K^{(r+1)}[2] &= K^{(r)}[2] \oplus S\Big(K^{(r)}[15]\Big),\\
K^{(r+1)}[3] &= K^{(r)}[3] \oplus S\Big(K^{(r)}[12]\Big),\\
K^{(r+1)}[i] &= K^{(r)}[i] \oplus K^{(r+1)}[i-4], \qquad i = 4, \ldots, 15,
\end{aligned}$$

which is exactly the byte-wise form of $K^{(r+1)}[0..3] = K^{(r)}[0..3] \oplus \mathrm{SubWord}(\mathrm{RotWord}(K^{(r)}[12..15])) \oplus (\mathrm{RC}[r], 0, 0, 0)$ and $K^{(r+1)}[i] = K^{(r)}[i] \oplus K^{(r+1)}[i-4]$ for $i \geq 4$.

What we implemented) Given $K^{(r+1)}$, recover $K^{(r)}$ as follows

$$\boxed{K^{(r)}[i] = K^{(r+1)}[i] \oplus K^{(r+1)}[i-4] \ \text{ for } i = 4, \ldots, 15}$$

This uses only $K^{(r+1)}$ and is an immediate rearrangement of the forward rule Once $K^{(r)}[12..15]$ are known, invert the first four bytes:

$$\begin{aligned}
K^{(r)}[3] &= K^{(r+1)}[3] \oplus S\Big(K^{(r)}[12]\Big),\\
K^{(r)}[2] &= K^{(r+1)}[2] \oplus S\Big(K^{(r)}[15]\Big),\\
K^{(r)}[1] &= K^{(r+1)}[1] \oplus S\Big(K^{(r)}[14]\Big),\\
K^{(r)}[0] &= K^{(r+1)}[0] \oplus S\Big(K^{(r)}[13]\Big) \oplus \mathrm{RC}[r].
\end{aligned}$$

These equations come from the first four forward relations by solving for $K^{(r)}[0..3]$.

Correctness (i) For $i \geq 4$, the forward rule is $K^{(r+1)}[i] = K^{(r)}[i] \oplus K^{(r+1)}[i-4]$, hence $K^{(r)}[i] = K^{(r+1)}[i] \oplus K^{(r+1)}[i-4]$. (ii) Substituting the recovered $K^{(r)}[12..15]$ into the four equations above yields a unique $K^{(r)}[0..3]$; re-applying the forward schedule reproduces $K^{(r+1)}$ byte-for-byte. Therefore the two transformations are inverses.

Implementation Our function mirrors the algebraic derivation:

```
for (i = 15; i >= 4; --i) prev[i] = next[i] ^ next[i-4];
prev[3] = next[3] ^ S[ prev[12] ];
prev[2] = next[2] ^ S[ prev[15] ];
prev[1] = next[1] ^ S[ prev[14] ];
prev[0] = next[0] ^ S[ prev[13] ] ^ RC[round];
```

Verification We performed two checks for all rounds $r = 0..9$ on 10,000 random trials:

$$\texttt{prev(next(}K^{(r)}\texttt{,r),r)} = K^{(r)} \quad \text{and} \quad \texttt{next(prev(}K^{(r+1)}\texttt{,r),r)} = K^{(r+1)}.$$

Both passed (*Key schedule inversion tests: OK*).

Q3. See code in main-q3.c

Construction. We init E as AES-128 reduced to three *full* rounds including mixcolumns. Define

$$F(k_1 \| k_2, x) \;=\; E_{k_1}(x) \;\oplus\; E_{k_2}(x).$$

Standard beyond birthday construction: when E is a PRP, the PRF security of F reduces to the PRP security of E.

If $k_1 = k_2 = k$, then $F(k\|k, x) = E_k(x) \oplus E_k(x) = 0^{128}$ for every x, i.e. a constant zero function that is trivially distinguishable and useless for encryption and or MAC. So we must enforce $k_1 \neq k_2$.

Let Λ be a Λ-set of 256 plaintexts (one byte varies over all values, the others fixed). For 3-round AES and any fixed key k,

$$\bigoplus_{x \in \Lambda} E_k(x) = 0^{128}.$$

Therefore,

$$\bigoplus_{x \in \Lambda} F(k_1 \| k_2, x) = \left(\bigoplus_{x \in \Lambda} E_{k_1}(x) \right) \oplus \left(\bigoplus_{x \in \Lambda} E_{k_2}(x) \right) = 0^{128} \oplus 0^{128} = 0^{128}.$$

Thus the same 3-round square distinguisher that breaks E also breaks F.

Validation We implemented F with three AES rounds and generated multiple Λ-sets (varying which byte ranges over all 256 values). For random distinct keys $k_1 \neq k_2$, the XOR of the 256 outputs of F is always 0^{128}, exactly as predicted and as a sanity check, with $k_1 = k_2$ the function collapses to a constant 0^{128} for *every* input.

2 Exercise 2 : Key-recovery attack for 3 1/2-round AES

Q1. See code in main-attack.c

Hi level idea A Λ-set is a set of 256 plaintexts that are identical except for one byte which takes all values $0, \dots, 255$. When such a set is encrypted through three full rounds of AES the xorof the 256 corresponding ciphertext bytes at certain positions is zero, one can "peel off" the last (half) round by guessing a byte of the last round key, apply the inverse subbytes on the resulting byte and test the integral property. Wrog guesses do not produce a zero XOR: the correct guess does.

Algorithm

1. Repeat for several independent Λ-sets (to filter false positives):
   (a) Build a Λ-set: choose a random base plaintext P and let plaintexts P_v be equal to P except at byte position t where $P_v[t] = v$ for $v \in \{0, \dots, 255\}$.
   (b) Query the $3\frac{1}{2}$-round oracle on all P_v to obtain ciphertexts C_v.
   (c) For each target ciphertext byte index j (0..15) and for each guess $g \in \{0, \dots, 255\}$ for the corresponding last-round key byte:
      - Compute $X_g = \bigoplus_{v=0}^{255} \mathrm{Sinv}(C_v[j] \oplus g)$.
      - If $X_g = 0$ then keep g as a candidate for byte j, otherwise discard it.
2. Intersect candidate sets across multiple independent Λ-sets, a small number of independent sets (e.g. 4–8) usually leaves a unique candidate per byte.
3. Assemble the recovered last-round key $K^{(4)}$ from the unique byte candidates.
4. Invert the key schedule: apply the inverse of the AES key expansion step-by-step (round 4 $\to$ round 3 $\to$ round 2 $\to$ round 1 $\to$ master key) using the relations

$$K^{(r)}[i] = K^{(r+1)}[i] \oplus K^{(r+1)}[i-4] \quad (i = 4..15)$$

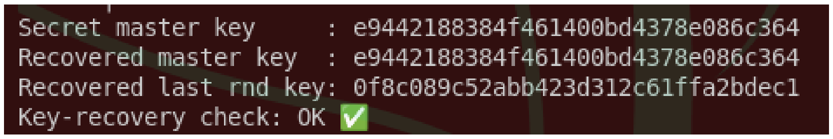
and

$$\begin{aligned} K^{(r)}[3] &= K^{(r+1)}[3] \oplus S\big(K^{(r)}[12]\big), \\ K^{(r)}[2] &= K^{(r+1)}[2] \oplus S\big(K^{(r)}[15]\big), \\ K^{(r)}[1] &= K^{(r+1)}[1] \oplus S\big(K^{(r)}[14]\big), \\ K^{(r)}[0] &= K^{(r+1)}[0] \oplus S\big(K^{(r)}[13]\big) \oplus \mathrm{RC}[r]. \end{aligned}$$

5. Verify the recovered master key by encrypting a few random plaintexts with it and checking equality with the oracle outputs.

As noted in the subject, some toy s-boxes or poorly chosen matrices can produce many false positives (the distinguisher may behave badly). When experimenting with non-standard components, increase the number of Λ-sets (or vary the varying byte position) and re-run the filtering until unique candidates remain.

Result Running the reference implementation with the provided AES files on a random secret produced:



```
Secret master key     : e9442188384f461400bd4378e086c364
Recovered master key  : e9442188384f461400bd4378e086c364
Recovered last rnd key: 0f8c089c52abb423d312c61ffa2bdec1
Key-recovery check: OK ✅
```

Figure 2: Result printed on stdin

This confirms the attack recovers the exact master key and last-round key, and that the recovered master key reproduces the oracle ciphertexts.

Q2.

We modified the `xtime()` function inside `aes-128_enc.c` to use a different reduction polynomial (for example $X^8 + X^6 + X^5 + X^4 + X^3 + X + 1$ instead of the standard $X^8 + X^4 + X^3 + X + 1$). This effectively changes the internal multiplication rules in $\mathbb{F}_{2^8}$, thus defining a different field representation.

After recompiling and running the attack program, we observed that the implementation produced a different last round key and ciphertexts, confirming that it is a distinct cipher. The key recovery attack still succeeded (`Key-recovery check: OK`), showing that the square distinguisher remains valid under a change of field representation.

We did *not* modify the subbytes transformation nor the MixColumns matrix.

In conclusion, changing the field representation of $\mathbb{F}_{2^8}$ changes the algebraic structure of AES but not its overall substitution permutation network properties. So the square attack continues to apply as expected.

3 Compilation and execution

How to compile and run

```
# from project root
# 1) build everything


# or build the q2 unit-test program only
```

```
make q2
./q2                # should print: "Key schedule inversion tests: OK"

# build & run the attack (Q3)
make attack
./attack             # runs the square-attack on the 3.5-round oracle

# clean objects
make clean
```

Expected output examples

```
# q2 test
Key schedule inversion tests: OK

# attack run (example)
Secret master key     : e9...364
Recovered master key  : e9...364
Recovered last rnd key: 0f...c1
Key-recovery check: OK
```