# finalexam

November 30, 2023

Q1. Adam uses Shamir's $(t, w)$-threshold scheme with $t = 5$ and $w = 10$ to share a key with 10 friends. The base is $p = 31847$, and the public keys are $(413, 432, 451, 470, 489, 508, 527, 546, 565, 584)$.

Also, the first five $f(x)$ are $(25439, 14847, 24780, 5910, 12734)$.

Identify the key and the polynomial coefficients.

Recall that the polynomial is $(t-1)$ th degree. Rewrite the thing as a matrix and invert it.

[21]:
```
import numpy as np
import galois

p = 31847
zp = galois.GF(p)
x = zp([413, 432, 451, 470, 489])
y = zp([25439, 14847, 24780, 5910, 12734])
pows = np.array([0, 1, 2, 3, 4])

xpowed = np.power(zp([x, x, x, x, x]).transpose(), pows)

a = np.matmul(np.linalg.inv(xpowed), y)
a
```

[21]:
```
GF([31318, 11223,  3945,  7745,  9872], order=31847)
```

A1. The key is 31318 and the polynomial coefficients are $(11223, 3945, 7745, 9872)$.

Q2. Suppose Blom KPS for $k = 2$ is set up for five users ABCDE with $p = 97, r_A = 14, r_B = 38, r_C = 92, r_D = 69, r_E = 70$. The secret polynomials given to each user are as follows:

- $g_A(x) = 15 + 15x + 2x^2$
- $g_B(x) = 95 + 77x + 83x^2$
- $g_C(x) = 88 + 32x + 18x^2$
- $g_D(x) = 62 + 91x + 59x^2$
- $g_E(x) = 10 + 82x + 52x^2$

Compute the key for all distinct pairs of users.

[13]:
```
from itertools import combinations, permutations

user_pairs = list(combinations(['A', 'B', 'C', 'D', 'E'], 2))
```

```
blom_keys = list()

p = 97
k = 2
r = [14, 38, 92, 69, 70]

letter_map = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4}

polynomials = [
    [15, 15, 2],
    [95, 77, 83],
    [88, 32, 18],
    [62, 91, 59],
    [10, 82, 52]
]

def compute_polynomial(coeffs: list[int | float], x) -> int | float:
    result = 0
    for i, coeff in enumerate(coeffs):
        result += coeff * (x**i)
    return result

for u, v in user_pairs:
    blom_keys.append(compute_polynomial(polynomials[letter_map[u]], 
 ↪r[letter_map[v]]) % p)

list(zip(user_pairs, blom_keys))
```

```
[13]: [(('A', 'B'), 78),
 (('A', 'C'), 87),
 (('A', 'D'), 96),
 (('A', 'E'), 1),
 (('B', 'C'), 39),
 (('B', 'D'), 58),
 (('B', 'E'), 32),
 (('C', 'D'), 15),
 (('C', 'E'), 27),
 (('D', 'E'), 70)]
```

A2.

- AB: 78
- AC: 87
- AD: 96
- AE: 1
- BC: 39
- BD: 58
- BE: 32

- CD: 15
- CE: 27
- DE: 70

Q3. Solve for $x$ such that $13x \equiv 4 \mod 99$ and $15x \equiv 56 \mod 101$

[12]:
```
def extended_gcd(a, b):
    if a == 0:
        return b, 0, 1
    else:
        g, x, y = extended_gcd(b % a, a)
        return int(g), int(y - (b // a) * x), int(x)

_, x1, _ = extended_gcd(13, 99)
x1 = x1 % 99
a1 = (x1 * 4) % 99

_, x2, _ = extended_gcd(15, 101)
x2 = x2 % 101
a2 = (x2 * 56) % 101

a = [a1, a2]
n = [99, 101]
M = 99 * 101

x = 0
for ai, ni in zip(a, n):
    mi = M // ni
    si = extended_gcd(ni, mi)[2]
    x += (ai * si * mi)
x = x % M
x, x % 99, x % 101, (a1 * 13) % 99, (a2 * 15) % 101
```

[12]:
```
(7471, 46, 98, 4, 56)
```

A3.

- $x \equiv 4 \times 13^{-1} \mod 99 \implies x \equiv 46 \mod 99$
- $x \equiv 56 \times 15^{-1} \mod 101 \implies x \equiv 98 \mod 101$
- $x = 7471$

Q4. Suppose that we are working with the SPN in slide 5 of Block Cipher Notes, and that the S-box is defined. Calculate

[30]:
```
q4_sbox = {
    0x0: 0xE, 0x1: 0x2, 0x2: 0x1, 0x3: 0x3,
    0x4: 0xD, 0x5: 0x9, 0x6: 0x0, 0x7: 0x6,
    0x8: 0xF, 0x9: 0x4, 0xA: 0x5, 0xB: 0xA,
    0xC: 0x8, 0xD: 0xC, 0xE: 0x7, 0xF: 0xB
}
```

```
slides_sbox = {
    0x0: 0xE, 0x1: 0x4, 0x2: 0xD, 0x3: 0x1,
    0x4: 0x2, 0x5: 0xF, 0x6: 0xB, 0x7: 0x8,
    0x8: 0x3, 0x9: 0xA, 0xA: 0x6, 0xB: 0xC,
    0xC: 0x5, 0xD: 0x9, 0xE: 0x0, 0xF: 0x7
}

"""
def get_lin_approx_table(s_box: dict[int, int]):
    lin_approx = np.zeros((16, 16), dtype=np.uint8)
    for a in range(16):
        for b in range(16):
            for x in range(16):
                # xor a_i*x_i
                if (((a & x) ^ (b & s_box[x]))).bit_count() % 2 == 0:
                    lin_approx[a, b] += 1

    return lin_approx
"""

def get_diff_approx_table(sbox: dict[int, int]) -> np.ndarray:
    diff_approx = np.zeros((16, 16), dtype=np.uint8)
    for xp in range(16):
        for yp in range(16):
            for x in range(16):
                for xstar in range(16):
                    if x ^ xstar == xp and sbox[x] ^ sbox[xstar] == yp:
                        diff_approx[xp, yp] += 1
    return diff_approx

dtable = get_diff_approx_table(q4_sbox)
dtable
#dtable[7,8]
```

[30]:
```
array([[16,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  2,  0,  4,  0,  2,  0,  0,  0,  0,  2,  4,  0,  0,  2],
       [ 0,  2,  0,  0,  0,  0,  0,  2,  0,  0,  2,  0,  0,  2,  2,  6],
       [ 0,  2,  0,  4,  0,  2,  0,  0,  0,  2,  0,  4,  0,  2,  0,  0],
       [ 0,  4,  2,  2,  0,  2,  0,  2,  2,  0,  0,  2,  0,  0,  0,  0],
       [ 0,  0,  0,  4,  0,  0,  0,  4,  0,  0,  0,  0,  2,  2,  2,  2],
       [ 0,  0,  0,  0,  2,  0,  2,  0,  2,  0,  2,  0,  2,  2,  2,  2],
       [ 0,  0,  4,  2,  2,  0,  0,  0,  4,  2,  0,  0,  0,  0,  2,  0],
       [ 0,  2,  0,  0,  2,  4,  2,  2,  0,  2,  0,  0,  0,  2,  0,  0],
       [ 0,  6,  0,  0,  0,  0,  2,  0,  0,  0,  2,  4,  0,  2,  0,  0],
       [ 0,  0,  2,  0,  0,  0,  0,  2,  4,  0,  4,  2,  0,  0,  2,  0],
       [ 0,  0,  0,  0,  2,  2,  2,  2,  0,  0,  0,  0,  4,  0,  4,  0],
```

```
[ 0,  0,  2,  0,  0,  2,  4,  0,  2,  0,  0,  0,  2,  2,  2,  0],
[ 0,  0,  2,  2,  2,  0,  2,  0,  0,  2,  6,  0,  0,  0,  0,  0],
[ 0,  0,  2,  2,  0,  0,  0,  0,  2,  6,  0,  0,  0,  0,  0,  4],
[ 0,  0,  0,  0,  2,  4,  0,  2,  0,  2,  0,  2,  2,  2,  0,  0]],
dtype=uint8)
```

Q4b. Find a differential trail using 4 active S-boxes that has a propagation ratio of 27/2048. Use $S_1^1, S_4^1, S_4^2, S_4^3$.

A4b. The ratio is equal to $3^3 \times 2/8^4$.

The differential trail I got was $S_1^1$: $R_p(1001, 0001) = \frac{3}{8}, S_4^1$: $R_p(1001, 0001) = \frac{3}{8}, S_4^2$: $R_p(1001, 0001) = \frac{3}{8}, S_4^3$: $R_p(0001, 1100) = \frac{2}{8}$

Q4c. Identify a differential attack that will identify 8 subkey bits.

A4c.

Using the differential trail from 4b, the output of $S_4^3$ propagates to $u_1^4$ and $u_2^4$ which propagate to their respective Y blocks. So, we can backtrack from $y_{<1>}$ and $y_{<2>}$, determine which two 4-bit numbers cause $u^4$ to differ in y', which we determined to be 0x1100, and determine the 1st and 2nd blocks of the 5th subkey, which adds up to 8 bits. The Python function below follows the structure of Algorithm 4.3 but uses it for this differential trail.

```
def differential_attack(pairs: list[tuple[int, int]], s_box_inv: dict[int, int]):
    counts = np.zeros((16, 16), dtype=np.float64)

    for x, y, xstar, ystar in pairs:
        # S^3_4's output propagates to y<1> and y<2>...
        if y >> 12 == ystar >> 12 and y >> 8 & 0b1111 == ystar >> 8 & 0b1111:
            for l1 in range(16):
                for l2 in range(16):
                    yb1 = y >> 12
                    yb2 = y >> 8 & 0xF
                    ystarb1 = ystar >> 12
                    ystarb2 = ystar >> 8 & 0xF

                    v4b1 = l1 ^ yb1
                    v4b2 = l2 ^ yb2
                    u4b1 = s_box_inv[v4b1]
                    u4b2 = s_box_inv[v4b2]
                    v4b1star = l1 ^ ystarb1
                    v4b2star = l2 ^ ystarb2
                    u4b1star = s_box_inv[v4b1star]
                    u4b2star = s_box_inv[v4b2star]
                    u4b1prime = u4b1 ^ u4b1star
                    u4b2prime = u4b2 ^ u4b2star

                    if u4b1prime == 0x1 and u4b2prime == 0x1:
                        counts[l1, l2] += 1
```

```
    maxie = -1
    maxkey = None
    for a in range(16):
        for b in range(16):
            counts[a, b] = abs(counts[a, b] - (len(pairs) / 2))

            if counts[a, b] > maxie:
                maxie = counts[a, b]
                maxkey = (a,b)
    return maxkey
```

Q5. Set up an ElGamal scheme with $p = 2579$ and $\alpha = 2$. Suppose $a = 1249$ and $k = 179$. Compute $\beta = \alpha^a \mod p$ and encode $x = 1458$ and $x = 458$.

[33]:
```
from math import gcd

def modpow(x: int, n: int, m: int) -> int:
    if n == 0:
        return 1

    current = x
    result = 1
    while n > 0:
        if (n & 1):
            result = (result * current) % m
        current = (current ** 2) % m
        n = n >> 1

    return result

def elgamal_encrypt(x: int, alpha: int, beta: int, k: int, p: int) ->␣
 ↪tuple[int, int]:
    return (modpow(alpha, k, p), (x*modpow(beta, k, p)) % p)

def elgamal_decrypt(y1: int, y2: int, a: int, p: int) -> int:
    _, y1_inv, _ = extended_gcd(modpow(y1, a, p), p) # calculate the inverse of␣
 ↪y1
    return (y2 * y1_inv) % p

p = 2579

alpha = 2
a = 1249
k = 179
beta = modpow(alpha, a, p)

xs = [1458, 458]
ys = list(map(lambda x: elgamal_encrypt(x, alpha, beta, k, p), xs))
```

```
txs = list(map(lambda y: elgamal_decrypt(*y, a, p), ys))

beta, ys
```

[33]: (1113, [(2029, 682), (2029, 239)])

A5. $\beta = 1113$, $y_{1458} = (2029, 682)$, $y_{458} = (2029, 239)$