NSUCRYPTO 2020: Bases

Ioan Dragomir[1], Gabriel Tulba-Lecu[2], and Mircea-Costin Preoteasa[3]

[1] ioandr@gomir.pw – Technical University of Cluj-Napoca
[2] gabi_tulba_lecu@yahoo.com – Polytechnic Univeristy of Bucharest
[3] mircea_costin84@yahoo.com – Polytechnic Univeristy of Bucharest

Table of Contents

1 Problem summary

Given $s \geq d > 1$, $r = \sum_{i=0}^{d} \binom{s}{i}$, and a family $\mathcal{F}$ of size s containing r-bit binary vectors, we create a set $\mathcal{B}$ by bitwise AND-ing all subsets of length at most d from $\mathcal{F}$ (there's r of them). By convention, AND-ing the empty set results in an all-one vector. Study when $\mathcal{B}$ is a basis over $\mathbb{F}_2^r$, and propose practical applications for such bases.

For example, for $s = 2$, $d = 2 \Rightarrow r = 4$, an example of a conforming $\mathcal{F}$ is $\{(1100), (0101)\}$, which generates $\mathcal{B} = \{(1111), (1100), (0101), (0100)\}$, whose elements are all linearly independent with respect to xor, and so is a basis. A bad $\mathcal{F}$ would be $\{(1001), (1110)\}$, which does not generate a basis in $\mathbb{F}_2^4$.

2 Equivalence between solutions

One of the first things we notice is that if we have a solution and we permute all the vectors in $\mathcal{F}$ and $\mathcal{B}$ in the same way, the new sets will also be a valid solution, because all operations are bitwise. Using this property, we can reduce our search space by a factor of r factorial.

3 Exploring solutions for low s, d

We will write a program which enumerates all distinct (under permutations, as detailed before) $\mathcal{F}$ sets for a given s, d. We expect, at least for the cases when $s = d \Rightarrow r = 2^s$, for solutions like the following to be valid: $\mathcal{F} = \{(11110000), (11001100), (10101010)\}$. (Notice each vector is an indicator for a given bit plane $2^k,\ 1 \leq k < r$.)

```
s = 2, d = 2 => r = 4
F = {(0011), (0101)}  =>
B = {(0001), (0011), (0101), (1111)}

s = 3, d = 2 => r = 7
F = {(0000111), (0011001), (0101010)}  =>
B = {(0000001), (0000010), (0000111), (0001000),
     (0011001), (0101010), (1111111)}

F = {(0000111), (0011001), (0101011)}  =>
B = {(0000001), (0000011), (0000111), (0001001),
     (0011001), (0101011), (1111111)}

F = {(0000111), (0011011), (0101001)}  =>
B = {(0000001), (0000011), (0000111), (0001001),
     (0011011), (0101001), (1111111)}

F = {(0000111), (0011011), (0101101)}  =>
B = {(0000011), (0000101), (0000111), (0001001),
     (0011011), (0101101), (1111111)}

F = {(0001111), (0010011), (0100101)}  =>
B = {(0000001), (0000011), (0000101), (0001111),
     (0010011), (0100101), (1111111)}

F = {(0001111), (0010011), (0110101)}  =>
B = {(0000011), (0000101), (0001111), (0010001),
     (0010011), (0110101), (1111111)}

F = {(0001111), (0110011), (1010101)}  =>
B = {(0000011), (0000101), (0001111), (0010001),
     (0110011), (1010101), (1111111)}

F = {(0010111), (0101011), (0110001)}  =>
B = {(0000011), (0010001), (0010111), (0100001),
     (0101011), (0110001), (1111111)}

s = 3, d = 3 => r = 8
F = {(00001111), (00110011), (01010101)}  =>
B = {(00000001), (00000011), (00000101), (00001111),
     (00010001), (00110011), (01010101), (11111111)}
```

We notice the only solution for $s = d = 2$ and $s = d = 3$ is, indeed, the one we predicted, and that for $s \neq d$ we have solutions related to that idea, which can be generated by the following algorithm.

For the first vector of $\mathcal{F}$, we split the r bits in half, making the first half all zeros and the second half all ones. For the second vector, we split each of the halves from before in two, filling even quarters with zeros and odd quarters with ones. We repeat this s times, or until we cannot divide any interval no more.

Notice that at each halving, if the interval length is odd, we have two equivalent options, whether to put the middle element in the first or second half. Thus, when r is not a power of two, we will have multiple solutions, corresponding to each way we can split the undecided middle elements.

By intuition, we suppose this construction generates all the valid bases.

A Python basis search

```
import sys, math, itertools

def comb(n, k):
        return math.factorial(n) // (math.factorial(k) * math.factorial(n-k))

assert len(sys.argv) == 3, f"Usage: {sys.argv[0]} s d"

s = int(sys.argv[1])
d = int(sys.argv[2])

assert s >= d > 1

r = sum( comb(s, i) for i in range(d+1) )

print(f"s = {s} generating vectors")
print(f"d = {d} AND inputs")
print(f"r = {r} bits")

Fr = list(range(1, 2**r-1))
lfr = len(Fr)

def fmt(x):
        elems = [f"({bin(2**r + q)[-r:]})" for q in sorted(x)]
        return "{" + ", ".join(elems) + "}"

def is_basis(F, r):
        visited = {0}
        for x in F:
                newvis = set()
                for b in visited:
                        bb = b ^ x
                        if bb in visited or bb in newvis: return False
                        newvis.add(bb)
                visited.update(newvis)
        return True
```

```
def orderless(F):
        return tuple(sorted(tuple((f>>i)&1 for f in F) for i in range(r)))

equivalence_classes = set()

for F in itertools.combinations(Fr, s):
        fo = orderless(F)
        if fo in equivalence_classes: continue
        equivalence_classes.add(fo)

        B = []
        for i in range(0, d+1):
                for bois in itertools.combinations(F, i):
                        b = 2**r-1
                        for boi in bois:
                                b &= boi
                        B.append(b)
                        if b == 0:
                                break
                if B[-1] == 0:
                        break
        else:
                if is_basis(B, r):
                        print(f'F = {fmt(F)}  =>  B = {fmt(B)}')
```