# Homework 2

## Introduction

You are interning at the super secretive TLA (Three Letter Agency) again. This time, the TLA is analyzing a hash function used by rival intelligence agencies, called BAH (Bad Algorithm for Hashing). Your goal is to cryptanalyze the BAH hash function. As before, you will work in teams; we recommend keeping the same teams as for Project 1.

Here is the pseudocode for the BAH hash function:

```
function BAH(M)
    M ← PadToMultiple(M, 256)
    arr ← ToArray(M, 2)
    state ← Array(512)
    for i ← 0 ; i < length(arr)/128 ; i ← i + 1 do
        m ← Join(arr[128 × i, 128 × (i + 1)], Array(384))
        state ← (state + m) mod 4
        state ← Transform(state)
    end for
    return ToBits(state[0, 128], 2)
end function

function Transform(state)
    state1 ← state
    for i ← 0 ; i < 20 ; i ← i + 1 do
        state1 ← Round(state1)
    end for
    return state1
end function

function Round(state)
    state1 = Array(512)
    for i ← 0 ; i < 512 ; i ← i + 1 do
        j ← (((i + 1) ∗ 289) mod 513) − 1
        j ← (j + 151) mod 512
        state1[i] = state[j]
    end for
```

```
    state2 = Array(512)
    for i ← 0 ; i < 256 ; i ← i + 1 do
        state2[2 × i]     ← S0[ state1[2i] ][ state1[2i + 1] ]
        state2[2 × i + 1] ← S1[ state1[2i] ][ state1[2i + 1] ]
    end for
    return state2
end function
```

$S0 = ((3,1,1,1),(2,1,0,3),(2,2,2,3),(3,0,0,0))$
$S1 = ((2,2,3,0),(1,1,2,1),(2,0,3,0),(3,1,3,0))$

Here,

- `ToArray`$(M, \ell)$ takes a bitstring $M$ that is assumed to be a multiple of $\ell$ bits. It then partitions $M$ into $\ell$-bit pieces, which it then interprets as integers from 0 to $2^\ell - 1$. It then outputs the array formed by these $\ell$-bit integers.
- `ToBits`$(A, \ell)$ is the opposite of `ToArray`. It takes an integer array $A$. For each integer, it converts the integer into a $\ell$-bit string (reducing mod $2^\ell$ if necessary) and then concatenates all the bits together.
- `PadToMultiple`$(M, \ell)$ adds a concatenates to $M$ a padding of the form $10^n$ so that the result has a length that is a multiple of $\ell$.
- `Array`$(n)$ creates a new array of length $n$ where each entry is initialized to 0
- `arr` mod `b` performs the modular reduction component-wise over the elements of the array. $arr1 + arr2$ performs addition entry-wise over the arrays $arr1$ and $arr2$.
- Arrays are 0-indexed. $arr[i]$ means get the $i$th entry, while $arr[i, j]$ means get entries $i$ through $j - 1$. $arr[i][j]$ means get the $i$th entry $arr[i]$ of $arr$, which itself is an array; then get the $j$th entry of $arr[i]$.

# 1 Part 1: Basic observations

Take a look at the pseudocode for $BAH$, and describe the overall structure of the hash function. Examples: is it Merkle-Damgard, Sponge, or something else? What is the output length? If it is Merkle-Damgard, what is the compression function (Davies-Meyer or something else)? If it a sponge, what is the state size? For any underlying substitution-permutation network, how many rounds are used? What is the domain/range of the S-box. Anything else of note about the function? (Hint: this function may not operate over binary)

## 2 Part 2: Theory

Your goal in this section is to develop an attack (that is, find a collision) on the BAH hash function using differential cryptanalysis.

(a) One of the building blocks for the hash function is a substitution-permutation network. Suppose you knew a differential for the network. Not any differential will necessarily suffice for finding a collision. Determine some properties of a high-probability differential that would help you find collisions. Explain how to find collisions with such "good" differentials. You should answer this question without looking at the specific implementation of the underlying SPN — instead, just use the overall structure from Part 1.

(b) Next, explain what kinds of differentials for the underlying S-box would be useful for constructing good differentials overall. Again, just look at the overall structure from Part 1.

(c) Devise a plan for finding two inputs with a "good" differential. Any one differential might have low probability, but you will be satisfied finding any two messages with *any* "good" differentials.

(d) Estimate the computational cost of your attack. That is, roughly how many messages will you need to try before finding a collision? This should be based on the parameters of the hash function, not by applying empirical analysis.

## 3 Part 3: Attack

Now you will actually turn your plan into action.

(a) Find a collision consisting of two 128-bit messages

(b) Find a collision consisting of two 384-bit messages

(c) Find a collision consisting of two 512-bit messages

(d) Find a collision consisting of two $2^{20}$-bit messages

(e) Find a collision consisting of two messages which are strings consisting of only English words and spaces. To hash a string, convert each character into a byte using the ASCII encoding. The messages do not need to be grammatically correct English, and there are no length restrictions for the messages. However, in each message, no word can appear twice, and any two words are separated by exactly one space, and no two spaces are allowed next to each other.

(f) Find a collision between two messages of the form:

Transfer $ddddddddddd.dd from account aaaaaaaaaa to account bbbbbbbbbb on mm/dd/yyyy

Here, ddddddddddd.dd is an amount of money (given as a 10 digit dollar amount and 2 digit cent amount. E.g. 0094000000.01), aaaaaaaaaa and bbbbbbbbbb are account numbers (given as 10 decimal digits), and mm/dd/yyyy is a 8-digit date, two digits for month, two digits for date, and four digits for year. Your date should be a valid date.

For this part, you will need an implementation the BAH hash function. Here, the efficiency of the implementation will affect how long it will take you to find a collision, so a fast implementation will be desirable. To help debug your implementation, we have provided a few input/output pairs of the hash function in the file `examples.txt`. At the end of the file, there is also an example input/output pair of the round function.

You will also probably need to make some optimizations to your collision-finding procedure in order to more-quickly find collisions. For example, is it possible to re-use already existing computations or aborting computations early in some situations. Please write up any optimizations in your writeup.

Keep in mind that even with optimizations, it still may take many minutes of computation time before you successfully find a collision, depending on your implementation. Therefore, it recommended to start with finding collisions in "round-reduced" versions of the function. That is, consider a weakened version BAH-$s$, which is identical in every way, except the number of rounds is set to $s$ instead of the actual round number $r$. Try finding a collision with a small number of rounds first. This will give you a sense of how long it will take to find collisions for the overall function, and will be a quick sanity check that your attack strategy will work before spending a while finding a full collision.

For your writeup, please describe how you approached each problem. Were there any difficulties you encountered in translating theory to practice? What optimizations did you perform?

# Bonus

Notice that in parts 3(a) through 3(c), we skipped finding a collision between 256-bit messages. For up to 10 bonus points, explain why 256-bit messages are presumably more difficult, and nonetheless find a collision for two 256-bit messages. Submit your collision in a clearly marked file. You will receive partial bonus points if you are able to identify why a collision for 256 bits is more difficult, but unable to find such a collision.

# 4 Part 4: Fixing

The weakness you exploited above came from the underlying S-box being poorly designed. In this part, you will try to fix the hash function by giving it a new S-box. You are only allowed to change the S-box — everything else about the function stays the same. In particular, the domain and range of the S-box cannot be changed, only the particular mapping from inputs to outputs.

Your S-box must still be a permutation (otherwise, can you see how to attack the scheme?). Design an S-box that resists your differential cryptanalysis from above. Explain your design choices.

# 5 Deliverables

Your submission will contain the following 8 files:

- `writeup.pdf`, which will contain a write-up for each of the 4 parts above. For Parts 1 and 2, answer the questions given. For part 3, you will you will likely use some optimizations; include a description of each of your optimizations. For part 4, explain how you came up with your new S-box. Give an estimate for the computational cost of finding a "good" differential.
- `a.txt`, `b.txt`, ..., `f.txt`. These will contain the collisions you found in part 3. The files should have the form `message1,message2`. For parts (a) through (d), each message should be represented in binary. For parts (e) and (f), each message should be a character string.
- `sbox.txt`, which will contain your new S-box. This file should have the form

  S0=((3,1,1,1),(2,1,0,3),(2,2,2,3),(3,0,0,0))
  S1=((2,2,3,0),(1,1,2,1),(2,0,3,0),(3,1,3,0))

  (these are the S-boxes from the BAH function, which you should replace with your own).

Additionally, please submit any code you used to find collisions.

# 6 Grading

Your grade will be determined as follows:

- Writeup: 100 points. This will be based on the thoroughness of your answers to all of the questions. The breakdown between the different parts is the following:
  - Part 1: 20 points
  - Part 2: 40 points
  - Part 3: 20 points
  - Part 4: 20 points
- Cryptanalysis: 30 points. Here, you will get 5 points for each collision found.
- New S-box: 20 points. Here, *the teaching staff* will try to find a collision for BAH with your new S-box. We will attempt to find collisions in round-reduced version of the BAH hash function. If we find a collision in $s$ rounds, you will receive a score of $20 \times (1 - s/r)$. Note that we will be able to find collisions for some number of rounds, so it will be impossible to achieve a perfect score for this portion.