**Kevin Dang Nguyen**

**Rena Jing**

**ECS 132 - Project 2**

**Basic Step:** First we will consider that the IV is a 8 bit vector.

1. Randomly generate vectors and find the average number of trials that are required to achieve an A collision.

* About 20. We add onto our array of random bytes until we get a matching collision.

1. Repeat the previous step for S-collision.

* About 260. We used a specific byte vector and we keep adding randomly generated vectors until there’s a matching collision.

1. For a given number of randomly generated 8-bit IVs, derive the probability of collision for both A collision and S-collision. This will be based on reading and understanding the birthday paradox.

* Look at attached graph. Similar to the birthday paradox for a given number of random bytes (n), we can simulate the combinatorial solution. The paradox gives us the probability for a given number n that they have the same birthday/collision. As n increases so does the probability of a matching birthday/collision.

**Scalability:** In this step we will study the scalability of the results as a function of the size of the ICV vector.

Determine the average number of trials required for for an S-collision using IVs of size 10, 12, 14 and 16.

S\_collision(10) # about 1226.03

S\_collision(12) # about 4178.66

S\_collision(14) # about 15750.13

S\_collision(16) # about 63542.38

Instead of 8 bit, we used a function (S\_collision) to adjust the length of the IV size.

**Estimation:** Try to estimate the number of trials required when the IV is increased to 24 bits.

S\_collision(24) # runs for a very long time !!

**Practical Scenario:** Consider a 52 Mbps wireless LAN. Consider an AP and a mobile station. Suppose the packet size is 1500 bytes. Determine the time required to observe a collision when a simple counter based approach is used to change the IV for IV sizes of 8 bits and 16 bits. Compare it with the probabilistic approach.

time\_8 = 416000000/(2^8) # 1625000

time\_8

time\_16 = 416000000/(2^16) # 6347.656

time\_16

**Discussion and Extra Points**

1. A possible strategy is alternating between ascending and descending counters,

which is more efficient than ascending, A-collision, and S-collision. It is

more efficient than S-collisions and ascending collisions because both require

traversal of the elements from the beginning of the array, whilst ascending

and descending counters can simply estimate where the IV value is centered

around and gauge based on the estimate the byte it is closer to, thus decreasing

the range that we have to search in.

2. A-collision is the most efficient strategy because it has the shortest run

times out of ascending order and S-collision according to the simulations. Also,

since you are increasing the number of reference points, there is a higher

probability of a collision as opposed to having just one reference point in

an S-collision. The A-collision strategy is more efficient because at the worst

case, say if the byte was the larger than all the values in the array for the ascending order strategy, the ascending-order strategy requires a traversal through the entire

array, which would be inefficient as opposed to having randomly generated

bytes.