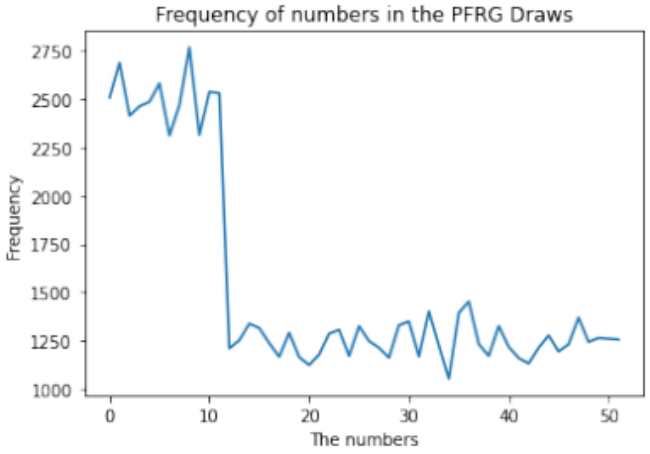
Problem 6 Solution: The Tiger Casino

By Aditya Gollapudi

This was perhaps the most difficult (and open-ended) problem of the competition. This solution will take you through some of the important insights that would have helped you devise a system to exploit the Tiger Casino. Note that a lot of these insights aren't exclusive to this problem — they may be applicable elsewhere when trying to attack a cryptosystem![1]

Frequency Analysis. The first thing to do with a unknown source of randomness is to try and examine the frequency of various elements. If you do this for the randomness files given to you, you should obtain a graph to the one below (that I shamelessly stole from Alan Chung's writeup as I was too lazy to reinstall my `matplotlib` package):



Looking at this plot, we immediately notice something odd: the numbers from 0-11 are *significantly* more likely to occur than 12-51! As we expect the underlying PRG to generate a uniform distribution this strongly suggests that the underlying PRG is generating numbers in the range whose size is a multiple of 52 plus 11. If we look closer, we'll see that the numbers from 0-11 are around *twice* as likely to occur as the rest, suggesting that the PRG is outputting something actually in the range 0-63, and is just double-mapping some of those to the same card. A very reasonable guess (that turns out to be correct) is that HRES is just taking a randomly generated number in $[0, 63]$ and taking it's remainder mod 52.

[1] We won't ask further questions on why you want to do this.

Exploiting Non-Uniformity. It is possible to have a very slight edge against the house as the probabilities are skewed. In particular, you can evaluate the probability of winning based on hitting or not hitting for every hand knowing that 0-11 occur with probability $\frac{1}{32}$ and the remainder occur with probability $\frac{1}{64}$. In our experience, if you do this then you can get a slight 1-2% edge, but not much more. (Shout out to Danny Chen for successfully implementing this strategy!)

Determining the Type of PRG. Being able to produce a larger edge will require you to break the random number generator (and eventually predict it). If you did some research on the three possibilities we provided you, a number of things should have pointed to the fact that the underlying implementation is a *Linear-Feedback Shift Register*. The biggest thing, perhaps, was what you learned from frequency analysis: we had strong reason to believe that the PRG is generating a number from 0-63, which is equivalent to generating six random bits. The only PRG that absolutely *needs* to output a power of 2 is the Linear-Feedback Shift Register (as it outputs a sequence of bits).

There is, moreover, stronger evidence that it cannot be any of the others. If we're using the Middle Square Method (MSM) and we output a number in 0-63 we know that we must be operating in base 2, 4, 8, or 64 as those are the only natural numbers a where there exists an integer n such that $a^n = 64$. For example, if the MSM actually operated in base 10, we would expect to produce a number from 0-99, which is inconsistent with the frequency analysis. However, if we try any of these bases and simply square the numbers in the `.txt` files, we see that it doesn't match the sequences. Similarly, for the Linear Congruential Generator, we note that as we are operating in base-64, there are only $64^2 = 4096$ possible values of the (A, B) pair, where the LCG takes the form $s_{t+1} = (As_t + B) \mod 64$. We can simply brute force these pairs with a script and see that they too yield results inconsistent with the randomness given to you. Thus, the PRG is a LFSR!

Exploiting the Period. If you've followed all the steps up till this point, you'll recall that we're using an LFSR with at most sixteen bits. It turns out that this implies that the period (i.e. the number of elements before the randomness starts cycling) is no more than $2^{16} - 1$.[2][3] With randomly chosen keys, however, the period could be much smaller (in fact, among the training data there were two files with period 511). And if the randomness in a file has a small period, you can just play something like a standard Blackjack strategy and have an edge of about -2%. Then, if you notice the cards forming a cycle, you can exploit this to have big (over 50%) winrates. This would be a nearly optimal strategy if the PRG was never re-seeded and you were allowed to play infinitely many hands, since you would be playing many versions

[2] Why the minus one? Well, if an LFSR ever gets into a state with all 0s, it will stay there forever; thus, the LFSR can't enter that state.
[3] I encourage you see if you can extend this argument to show that all cycles must be one less than a power of 2.

of the same game that you know how to win! (See subsequent sections for how to make money once you can predict every single future card draw.)

This strategy, if implemented optimally, would have likely generated a positive return with our test cases; however, the margin would be much smaller, because in some cases you would have never had the chance to discover the period (because each test case only generated $10,000$, and the period could be as great as $65,535$).

Breaking a General LFSR. To sustainably generate larger margins, you likely would have had to reverse engineer the LFSR to perfectly predict the future. Fortunately, it's quite well-known how to go about this. The LFSR can be written as $c_1X_1 \oplus \cdots \oplus c_{16}X_{16}$ where each c_i is either 0 or 1, and the $\oplus$ sign represents the XOR operation. We also know that the 17th bit of the LFSR is this function of the prior 16 bit. Extending this observation lets us set up a linear system of equations (think of XOR as the plus for bits), which can be solved in a number of ways, e.g. inverting a matrix. This takes at most 16×17 bits of output from the LFSR (why?) before we have a system we can solve for sure, but this number 1) in the average case isn't even this large, and 2) can be greatly reduced in the worst case.

For those of you with a little bit more math, you may wish to consider the Berlekamp-Massey Algorithm, which can find the value of the constants after $2 \times 16 = 32$ bits. You will notice that to get 16 distinct equations you need at least 2×16 bits, so the algorithm is in fact optimal in how many bits it requires from the PRG. Note that the Berlekamp-Massey algorithm doesn't precisely solve for the coefficients as a linear system; it does, however, use similar ideas while attaining more more efficiency by taking advantage of some nice properties of error/discrepancy polynomials and syndromes.

Reverse Engineering the Cards. We'll assume that we now have some blackbox algorithm from above (one of the many) that will reverse engineer an LFSR if given enough bits. If we had outputs from 0-63 we could just interpret them as 6 binary bits and be done. Unfortunately if we see a number n from 0-11 the PRG could have output either n or $n + 52$. To get around this, we don't really need much more than brute-force: just consider for both possibilities (or if this happens k times, all 2^k possibliities) and in the end, see which guess is consistent with reality in the end.

More precisely, if you're using the Berlekamp-Massey algorithm, you only need 32 bits; and since there are 6 bits per card, the overflow of a number happens at most $\lceil 32/6 \rceil = 6$ times. So at worst, you need to keep track of $2^6 = 64$ universes, which is extremely doable. Even if you use the very inefficient algorithm

requiring 17×16 bits in the worst case, note that overflow happens in expectation $24/64 = 3/8$ of the time so we expect to have to consider $2^{(3/8)\cdot(17\cdot16/6)} \sim 2^{16}$ possibilities, which is likewise very brute-forceable. (In this case, if you want to show that most of the time we won't be far from requiring 2^{16} possibilities, consider applying a concentration bound.) In either case, we can then tell which of the universes is actually the case by comparing their predicted output with the actual output of the PRG.

Taking Advantage When You Know the Future. If you assume you have some blackbox that reconstructs the output of the PRG exactly, how can you make money? The easiest way to do so is to simulate all hands, as you know all the cards and the dealer is deterministic. So you can consider, say, the next 5 hands and ask what happens if you hit 0, 1, 2, 3, 4 or 5 times on the first hand, the second hand, etc. for a total of 6^5 possibilities and see which choice maximizes how many out of the 5 hands you win. This, of course, is one of the absolute most brute-forceable techniques you could apply, but it nevertheless yields very good margins (between 40-50% net win rate).

For a more sophisticated technique, note that the state space of a given moment in time can be fully described by the hand we are on, the number of cards dealt so far, and the number of cards dealt in the current hand. The first number ranges from 1-500, the second from 1-10,000, and the last from 4-18 (let's just say 0-20 to keep the math nice). This, in total, is only 100 million possibilities, so if we can just apply a brute-force dynamic programming over all of them to find the globally optimal strategy. The base cases are where we have finished all 500 hands (as they have known value). For the remaining states, the value of that state is the maximum value of two possible states one step in the future (i.e. the state that you transition to when you hit and the state you transition to when you don't hit).

Plaudits

- Congratulations to Qinshi Wang (grad), Kaifeng Lyu (grad), and Shunyu Yao (grad) for submitting the only full solution with the best overall win rate (2195 net wins over 4000 hands)! Enjoy your extra $219.50.
- Congratulations to Dingli Yu (grad), Yuping Luo (grad), and Danny Chen (grad) for both breaking the LFSR and successfully exploiting the probabilities. Unfortunately, due to some poor luck on the final test case the number of net wins was still negative; however, in expectation they achieved a 1-2% edge by just exploiting the probability imbalance described in the previous pages.
- Congratulations to Aleksa Milojevic (MAT '23), Alex Lopez (MAT '22), and Kiril Bangachev (MAT '22) for breaking the LFSR!

Grading Criteria and Notes

25 points	Full solution (should have at least a 30% edge/net win rate)
15 points	Working probabilistic solution (around or above 0% edge) AND LFSR Break (code that finds the keys of the LFSR when given a sequence of card draws)
13 points	Working probabilistic solution OR Working LFSR break
8 points	Full rigorous writeup AND description of a working attack on the modified LFSR
6 points	Identify that the PRG is statistically imbalanced AND that PRG is an LFSR
4 points	Identify that the PRG is an LFSR
2 points	Identify that the PRG is statistically imbalanced (either by identifying skew or finding period)

Checking our Work: If you would like to make sure we tested your code properly (in fact, we encourage you to do so), run the test client in the appropriate language with your player class using command line arguments `500 10000 testingOutput[i].txt`, where `i` ranges from 1-8. In the published set of files, the test randomness can be found in a different folder (`test_randomness`).