Problem Set 3 Solutions

Problem 6. *Alice wants to deal a one-byte secret to shareholders 1, 2, and 3 such that any two of them can reconstruct the secret, but no single player knows anything about it. She decides to use Shamir secret sharing over the smallest prime field that will work for this situation. She represents bytes and points in this field in the natural way. Players 1 and 2 end up with shares of* $209 = 0xD1$ *and* $34 = 0x22$*, respectively. What secret was shared? What was player 3's share?*

The smallest prime larger than 256 is 257, so we will work in the field $\mathbb{Z}_{257}$. Since the problem asks for a 2-out-of-3 secret sharing, the sharing algorithm would be $f(x) = a_0 + a_1 x \mod 257$ where $a_0 \in \mathbb{Z}_{256}$ is the shared secret and $a_1 \in \mathbb{Z}_{257}$ is a random parameter. Then $a_0 + a_1 = 209 \mod 257$, while $a_0 + 2a_1 = 34 \mod 257$. Subtracting the first equation from the second (doing everything mod 257) gives $a_1 = 82$. Plugging this back into the first equation gives $a_0 = 127 = 0x7F$. Now knowing the polynomial chosen to have been $f(x) = 127 + 82x$ gives $f(3)$, the share for player 3, of $116 = 0x74$.

Problem 7. *Suppose you'd like to k-out-of-n secret share a 5-gigabyte DVD M among* $n \geq 3$ *shareholders, obtaining shares* $S_1, \ldots, S_n$*. Obviously it would be highly inefficient to regard M as a point from a (truly gigantic) finite field. Describe two simpler/faster approaches, and argue informally that they should work. The first should involve use of the field* $\mathbb{F}_{2^8}$ *and no complexity assumptions. The second should involve using Shamir's secret-sharing scheme on no more than 16 bytes.*

Part 1. Fix the access structure (the numbers k and n). Let $(\text{Share}, \text{Recover})$ denote Shamir's secret sharing scheme for this access structure and over $\mathbb{F}_{2^8}$. Build from this a secret-sharing scheme $(\text{Share1}, \text{Recover1})$ with message space BYTE^+ as follows: to compute $\text{Share}(M_1 \cdots M_m)$ with each M_i a byte, let $(S_i^1, \cdots, Y_i^n) \leftarrow \text{Share}(M_i)$ for each $i \in [1..m]$; then output $(S_1^1 \cdots S_m^1, \cdots, S_1^n \cdots S_m^n)$. That is, independently share each of the m bytes, and then concatenate corresponding bytes to make shares. Similarly, let $\text{Recover1}(S_1^1 \cdots S_m^1, \ldots, S_1^n \cdots S_m^n)$ be $\text{Recover}(S_1^1 \cdots S_m^1) \cdots \text{Recover}(S_1^n \cdots S_m^n)$.

To verify that this works, we need to check correctness and privacy. Correctness means that authorized sets of players holding good shares will recover what was previously shared out. This is so because they correctly recover each of the m bytes. Privacy means that unauthorized subsets of players get shares uncorrelated to what was shared out, apart from its length. This follows because each byte of what's recovered is uncorrelated to the corresponding byte that was shared out by Share (by the privacy of $(\text{Share}, \text{Recover})$), and uncorrelated to every other byte (by Share1's definition).

Part 2. Same setup and nomenclature as above, fixing k, n. Now assume we have a good PRG $G : \text{BYTE}^{16} \to \text{BYTE}^{\infty}$ (that is, an object like RC4, but with better security). We construct a secret-sharing scheme $(\text{Share2}, \text{Recover2})$ as follows. Algorithm $\text{Share2}(M)$ generates a random 16-byte string K and computes both $C \leftarrow M \oplus G(K)$ and $(S_1, \ldots, S_n) \leftarrow \text{Share1}(K)$. It returns the shares $(S_1 C, \ldots, S_n C)$. Algorithm $\text{Recover2}(S_1', \ldots, S_n')$ parses each provided S_i' into $S_i C_i$ (remember some shares may not be provided); outputs *fail* if the C_i values are not all the same value C; runs $\text{Recover}(S_1, \ldots, S_n)$ to get a value K; and outputs $C \oplus G(K)$ as the recovered message.

We must again argue correctness and privacy. The first is again trivial. For the second, note that we do not achieve "perfect" privacy: shares *are* information-theoretically correlated to the original secret. So we would need a new, *computational* notion of privacy for a secret sharing scheme. It would say, roughly, that reasonable adversaries can't distinguish unauthorized collections of shares corresponding to two different, equal-length messages. The PRG property of G should make this highly plausible: it tells us that C itself is going to be pseudorandom (as the xor of a pseudorandom string and some uncorrelated string), and, to the adversary, you are supplementing this by something that has *no* correlation (information-theoretically) to the message or PRG's key.

Problem 8.[1] The RC4 algorithm maps a key $K \in \text{BYTE}^k$ to an infinite string RC4(K), where $k \in [1..256]$. Investigate empirically the probability p_i that the second byte of RC4 output is $i \in \{0, \ldots, 9\}$ (written as a byte). For concreteness, assume a key length of $k = 16$ bytes. Now describe a simple adversary to distinguish RC4 output from truly random bits. Estimate your adversary's advantage, defined as the probability that it outputs 1 when given truly random bits minus the probability that it outputs 1 when given pseudorandom (RC4-generated) bits.

We implemented the RC4 algorithm and ran it with independently sampled 16-byte keys. Specifically, our program makes in total 2^{25} trials and counts the occurrences of the events that the second output byte equals to any of $\{0, \ldots, 9\}$. Finally, the program outputs the computed frequency for each of the 10 outcomes. The following is from a sample output:

```
frequency for byte 0: 0.007838
frequency for byte 1: 0.003847
frequency for byte 2: 0.003855
frequency for byte 3: 0.003883
frequency for byte 4: 0.003864
frequency for byte 5: 0.003871
frequency for byte 6: 0.003864
frequency for byte 7: 0.003900
frequency for byte 8: 0.003899
frequency for byte 9: 0.003873
```

From the above result, it would appear that the second output byte of RC4 is strongly biased towards 0: p_0 is approximately $1/128 = 0.0078125$, which is twice the anticipated probability. This suggests a simple adversary for distinguishing RC4 output from truly random bits: adversary $\mathcal{A}$, given a challenge string $x = x_1 x_2 \cdots$ having two or more bytes, would just read its second byte x_2 and then return 1 if and only if it's 0:

$\underline{\mathcal{A}(x_1 x_2 \cdots)}$ *Each x_i is a byte*

If $x_2 = \texttt{0x00}$ then return 1
else return 0

From our empirical study, we anticipate that if $\mathcal{A}$ is given real RC4 output, it will output 1 with probability about $1/128$; while if it is given a truly random string, it will output 1 with probability exactly $1/256$. Thus we expect that $\mathbf{Adv}(\mathcal{A}) \approx 1/256 \approx= 0.0039$.

[1] This problem requires a little programming, and it requires you to lookup a definition of RC4.