Exercise Sheet

Low Entropy Attacks

No mandatory homework.

Exercise 1: RNG Entropy
A physical "random" bit generator is measured to output a 0 with probability 0.45 and a 1 with probability 0.55. A key is generated by concatenating 40 output bits from this generator. What is the entropy of the resulting 40-bit key?

Exercise 2: Plaintext Entropy
We have learned that the entropy of a key can be interpreted as the average number of bits required to store it if perfect compression is used. Of course, the same holds for the entropy of plaintexts.
Using the plaintext characteristics (frequency of letters for the English language), derive the average number of bits required to store one letter of British English if perfect compression is used?

Exercise 3: Low Entropy Attack
Under file sharing on CampusNet, you can find the file `ciphertext_week6.bin`. This is a file which has been encrypted using AES-128, the U.S. encryption standard. It is your task to decrypt this file, assuming that you know the following:

- The 128-bit key was generated in the period June 22-28, 2014, using the GCC generator described in the lecture:
  1. The inner state s_0 of the generator is initialised to the current system time (seconds since 00:00:00 UTC on January 1, 1970).
  2. The update function has the form
  $$s_i = (s_{i-1} \cdot 1103515245) + 12345) \bmod 2^{31}$$
  3. Each key byte $k[0], \ldots, k[15]$ is generated by first running the update function and then taking the 8 least significant bits of the current inner state.
- The text was encrypted using AES-128 in ECB mode (meaning that the plaintext was split into blocks of 16 bytes and processed blockwise during encryption). The plaintext length is a multiple of the block size and no padding was applied.
- The text is rather recent, from a newspaper and in (British) English, using normal text format (small/capital letters, whitespaces, punctuation etc.). It has something to do with the NSA, so you might want to use certain names.

Use your knowledge from the lecture to break this cipher using a clever key search given the low entropy of the key material.
Note: The ciphertext file is a collection of pretty random-looking bytes. In particular, it contains the byte that is interpreted as 'EOF' (end of file) by many file handlers. So check whether your program stops parsing the file too early; if it does, it has probably encountered the wrong 'EOF' symbol. In this case, you'll have to find a workaround (e.g. you can read in the file in binary format in C) - all tricks are allowed as long as they work.