# Project 2

Alex Miller

November 13, 2021

Let $O_i(q)$ denote a response to a query to the task $i$ oracle for a query $q$.

## 1

**Flag**: 'wilted warfare'
**Method**:
Finding the value of the flag can be done by:

1. Finding the index of the biased byte, as well as its bias by:
   (a) Submitting large queries of null bytes to the task one oracle for many trials; this is equivalent to have many examples of output from $PRG_{k,r}$. Once we have this data we can
   (b) perform a per-byte frequency analysis on the results and determine the position of the biased index $i$.

   The number of trials alters the observability of the bias; I ran a single 250 trial bias test and noticed that the 31st byte of output was $0^8$ around 10 percent of the time. From this I deduced that the byte at index $i = 30$ was the bias, and that the bias was towards $0^8$
2. Finding the flag length by querying an empty byte array to the task one oracle: this works because the task one oracle returns a string exactly as long as $query||FLAG$.
3. For each byte $j$ in the flag:
   (a) Submitting a sufficient number of queries to the task one s.t. $j$ is byte aligned with the biased byte
   (b) such that we can be confident the most commonly observed encrypted byte at index $i$ was encrypted by the bias of $PRG$, which in this case would just be the real byte in the flag. I chose to submit 150 queries per flag byte; additionally, I only return the result of a recovered byte if it is printable, other wise I repeat the process.

   Note that this attack won't work if the length of the flag is not $\leq i + 1$; if this is not true you cannot byte align every byte of the flag with the biased index.

This attack makes 250 queries to find the bias, and then a further 150 queries per byte of the flag (unless an error occurs; technically, this code can run forever if we keep recovering invalid, non printable bytes).

## 2

**Flag:** 'Adults are always asking children what they want to be when they grow up because t'
**Method:**
Finding the value of the flag can be done by:

1. Finding the length of the flag by:
   (a) Feeding an empty query $q_0$ to the task two oracle, prior to queries of increasing length, and observing when the oracle responds with additional block of output.
   (b) The length of the flag is $|O_2(q_0)| - |q'|$, where $q'$ is the smallest query s.t. $|O_2(q_0)| \neq |O_2(q')|$
2. For each byte $j$ in the flag, in revers order:
   (a) Aligning byte $j$ with the left edge of a strategically picked block in the plaintext. This can be done with padding.
   (b) Assume we know the other 15 bytes in the block with $j$, call these bytes $B$; then we can submit $2^8 = 256$ guesses for $j$, each some $j'$, of the form $j'||B$ to $O_2$. We know we have found the correct $j' = j$ when the encrypted result of block we guess is equivalent to the encrypted block containing the aligned plaintext $j||B$.

(c) Once we know $j'$ we update our $B$ s.t. $B = j'||B[0:14]$, and move on to the next byte in the flag.

This attack work because, once we know then length of the flag, and can isolate the last byte in the flag in the first byte of the plaintext, we know an initial $B$, which is equivalent to the cmsc248 padding for a single byte string. From there we recover the whole flag.

This attack makes 16 or less queries in order to find the length of the flag, and an additional, in the worst case, 256 queries per flag byte to recover the entirety of the flag. We note here this could be reduced by restricting the search space of potential flag bytes to ascii printable characters, but not doing so lets us claim our solution is correct for any flag, not only ones which are ascii printable.

# 3

**Flag:** 'wHo'D SaVe a MIllIon of Us?'
**Method:**
Finding the value of the flag can be done by:

1. Setting some value $q_{base}$ such that $q_{base}$ is equivalent to a large string preceding the flag in $M$. In this case $q_{base}$ = b'password=' initially.
2. For each byte of the flag:
   (a) Constructing a set of guesses $G = \{q_{base}||q' : q' \in \{0,1\}^8\}$
   (b) For each $g \in G$, submitting $O_3(g)$ and observing $|O_3(g)|$ (the length of the response)
   (c) Determining the $g = q_{base}||q' \in G$ s.t. $|O_3(g)|$ is shortest, and updating $q_{base} = q_{base}||q'$

   This attack works because each time we guess a correct $q'$, the observed length of $|O_3(q_{base}||q')|$ should be shorter than any other guess, since $q_{base}$ precedes the byte $q'$ in the flag, and therefore causes $q'$ to be compressed by ZLIB.

This attack makes 256 queries per byte of the flag, in the worst case. We note here this could be reduced by restricting the search space of potential flag bytes to ascii printable characters, but not doing so lets us claim our solution is correct for any flag, not only ones which are ascii printable.

# 4

**Flag:** 'greMlInS 2: tHE NEw baTch'
**Method:**
This attack is much like the last, except that it has to deal with the effects of $O_4$ padding $M$ prior to compression.
Finding the value of the flag can be done by:

1. Finding appropriate padding $p$ for our base query $q_{base}$, which is initialized to the same value b'password=', s.t. $|O_4(q_{base})| = |O_4(p||q_{base})| < |O_4(p'||q_{base})|$, where $p'$ is the next largest pad tested by our $find_pad_len$ method. $p$ acts as a 'good pad' for which to test bytes.
2. For each byte of the flag:
   (a) Constructing a set of guesses $G = \{p||q_{base}||q' : q' \in \{0,1\}^8\}$
   (b) For each $g \in G$, submitting $O_3(g)$ and observing $|O_3(g)|$ (the length of the response)
   (c) Determining the $g = p||q_{base}||q' \in G$ s.t. $|O_3(g)|$ is shortest, and updating $q_{base} = q_{base}||q'$ accordingly

   This attack works because each time we guess a correct $g = p||q_{base}||q'$, the observed length of $|O_3(g)|$ should be shorter than any other guess, since $q_{base}$ precedes the byte $q'$ in the flag, and therefore causes $q'$ to be compressed by ZLIB. And since $q_{base}$ is byte aligned by $p$, so is $q_{base}||q'$ after compression. A wrong guess would not be compressed, and would add an additional byte to the compressed $M$; padding would ensure that we observe the overflow into the next block.

This attack takes 16 queries to find a good pad $p$. This attack makes 256 queries per byte of the flag, in the worst case. We note here this could be reduced by restricting the search space of potential flag bytes to ascii printable characters, but not doing so lets us claim our solution is correct for any flag, not only ones which are ascii printable.

## 5

**Flag:** 'Admin access granted.'
**Method:**
To gain admin access:

1. First make a empty query to $O_{5a}$ in order to get Prof. Cash's cipher text containing the encrypted form of his credentials; those are aligned with the edge of the last block, lets call the value stored at that last block *creds*
2. Then construct a query $q$ such that $|q|$ = the length of 'davidcash&uid=133', since such a query would align the plaintext '...&role=' with edge of the block it is encrypted to, because that's true of the sequence 'davidcash&uid=133'. My CNetID is the same length as 'davidcash', so I set $q$ to 'amiller68&uid=133'.
3. Query $O_{5b}(q)$ and take the first two blocks which will contain the encrypted plaintext 'username=amiller68&uid=133&role='. call this value $u$
4. Because ECB encryption is not randomized, we can substitute portions of cipher text from different oracle queries to construct forged plaintext to the decryption oracle. Since we know $u$ decrypts to 'username=amiller68&uid=133&role=', and *creds* decrypts to 'professor' (with the correct padding), we can get the $O_{6c}$ to give us the success message by querying $u||creds$, which decrypts to 'username=amiller68&uid=133&role=professor', which contains a username that is not Prof. Cash's, and a credential for a professor'.

## 6

**Flag:** 'Success!'
**Method:**
To gain admin access:

1. First query $O_{6b}(0^{256})$, two blocks of null bytes. Under the encryption scheme $\Pi = \{Enc, Dec\}$, this should decrypt to $m = m[1]||m[2] = AES_k(0^{128}) \oplus k||AES_k(0^{128}) \oplus 0^{128} = AES_k(0^{128}) \oplus k||AES_k(0^{128})$, so we can recover the key value $k$ because $m[1] \oplus m[2] = AES_k(0^{128}) \oplus k \oplus AES_k(0^{128} = k$
2. Since $\Pi$ is just CBC encryption in which the key is set to the IV, by recovering the key we can forge valid cipher texts. We encrypt $AES_{CBC}(k, pad('\text{let me in please}'), k)$ and send that to $O_{6c}$, which fulfills the requirements for generating the success message.

## 7

**Flag:** 'What my girlfriend thought, first 4 dates: 1. Nice shirt. 2. Wow. A second nice sh'
**Method:**
To find the value of the flag:

1. Query the $O_{7a}$ oracle to get the encrypted cipher text $C$
2. Then, because we have access to a padding oracle, we can implement a padding oracle attack to recover intermediary the cipher text $C_i$, which will help us recover the flag's plaintext.
   For every byte $c$ at index $i_c$ of the encrypted flag $C$ (every byte not in the IV), working in reverse:
   (a) We try to determine the value of the intermediary cipher text byte $c_i$ at index $i_c$ in $C_i$ by manipulating $C$ in the creation of $C'$, s.t. $C'$ helps us by letting us know whether the guesses we make about the value $c_i$ are correct; in this case, we are trying to make the resulting decrypted message text from $C'$, $M'$ have good padding at the corresponding byte $m_i$, as well as any other bytes that need to be altered in order to complete the padding.
   (b) This is done by constructing a set of guesses $G = \{format(C, C_i[i_c] = x, i_c) : x \in \{0,1\}^8\}$; $G$ is a set of forged cipher texts that modify $C$ in order to make the padding oracle return *true*, if after $C_i[i_c] = x$
   (c) As more of $C_i$ is uncovered, we can recover more and more of a given block; when all the intermediary cipher text bytes of a block are recovered, we can drop the block from our queries to the oracle, and repeat the process in the next block.
3. Once we uncover all the intermediate cipher text we can, we XOR it against the appropriate section of the original cipher text $C$ in order to recover the flag.

This attack takes one query to $O_{7a}$ in order to get the starter cipher text $C$ and then 256 queries per byte of the flag, in the worst case.