COMP 3109 Assignment 1

Braeden Hall (101143403)

October 14, 2021

Question 1.

There are 2^{18} machines, each with $32 = 2^5$ cores. Thus, we have $2^{18} \cdot 2^5 = 2^{23}$ cores at our disposal and as such we can test 2^{23} keys per cycle. Since each core runs at 8.5899GHz, we know that each core has $2^{\log_2(8.5899 \cdot 10^9)} \approx 2^{33}$ cycles per second. Together this means that we can test $2^{23} \cdot 2^{33} = 2^{56}$ keys per second.

a) Keylength is 56 bits, so there are 2^{56} possible keys. In the worst case, it will take

$$\frac{2^{56}}{2^{56}} = 1$$

second to find the secret key.

b) Keylength is 80 bits, so there are 2^{80} possible keys. In the worst case, it will take

$$\frac{2^{80}}{2^{56}} = 2^{24}$$

seconds to find the secret key.

$$\frac{2^{24}}{(60)(60)(24)} \approx 194.2$$

So 194.2 days to find the key in the worst case.

c) Keylength is 128 bits, so there are 2^{128} possible keys. In the worst case, it will take

$$\frac{2^{128}}{2^{56}} = 2^{72}$$

seconds to find the secret key.

$$\frac{2^{72}}{(60)(60)(24)(365)(4.031 \cdot 10^9)} \approx 37148.4$$

So 37148.4 orys to find the key in the worst case.

d) Keylength is 256 bits, so there are 2^{256} possible keys. In the worst case, it will take

$$\frac{2^{256}}{2^{56}} = 2^{200}$$

seconds to find the secret key.

$$\frac{2^{200}}{(60)(60)(24)(365)(4.031 \cdot 10^9)} \approx 1.26 \cdot 10^{43}$$

So $1.26 \cdot 10^{43}$ orys to find the key in the worst case.

As per my calculations, it is completely infeasible to expect to find a 128 or 256 bit secret key through exhaustive search with the current level of computing power available. However, if you have access to thousands of high power, fast computers, cracking 56 bit keys is trivial. Lastly, if you have a good amount of free time, you could attempt to take on keys of about 80 bits.

Question 2.

a) Refer to appendix 1 figure 1 for the tree.

b) Refer to appendix 1 figure 2 for the tree.

c) Refer to appendix 1 figure 3 for the tree.

For all parts (a-c), the leaves represent the resulting hand after the algorithm has terminated assuming it followed the leaf's specific path. Since each call to random is an independent event we can calculate probability as follows:
For each leaf n define the set P_n as the set of edges in the path from the root to n. Then

$$Pr(n) = \prod_{e \in P_n} Pr(e)$$

Using the previous equation we can calculate the following probabilities for each of the hands If the same hand appears in multiple leaves n_i, n_j, n_k... etc. we simply add $P_{n_i} + P_{n_j} + P_{n_k} + ...$ to get the probability of that hand):

hand	Method (a)	Method (b)	Method (c)
$[X, Y, Z]$	$\frac{4}{27}$	$\frac{1}{6}$	0
$[X, Z, Y]$	$\frac{5}{27}$	$\frac{1}{6}$	$\frac{3}{8}$
$[Y, X, Z]$	$\frac{5}{27}$	$\frac{1}{6}$	$\frac{3}{8}$
$[Y, Z, X]$	$\frac{5}{27}$	$\frac{1}{6}$	0
$[Z, X, Y]$	$\frac{4}{27}$	$\frac{1}{6}$	0
$[Z, Y, X]$	$\frac{4}{27}$	$\frac{1}{6}$	$\frac{2}{8}$

Question 3.

a) Let $m' = m_1|m_2$, where m_1 is the first 3 bytes of m and m_2 is the second 3 bytes (including the padding).

The original message m is missing 2 bytes (since we need 6 for 2 full blocks), so, in accordance with PKCS #7, we must add the padding 00000010|00000010 to the end of the message.

$m_1 = 11001001|00110100|11001000$
$m_2 = 00110010|00000010|00000010$

Applying CBC mode:

Calculating c_1: $c_1 = Enc_k(m_1 \oplus IV)$, since $IV = 0$ we have
$c_1 = Enc_k(m_1) = Enc_k(11001001|00110100|11001000)$

$q_0 = 00110111 \oplus 00110110 = 00000001$, $q_1 = 11001001 \oplus 11001011 = 00000010$,
$q_2 = 00110100 \oplus 00110111 = 00000011$

Thus, $c_1 = 00000001|00000010|00000011$.

Calculating c_2: $c_2 = Enc_k(m_2 \oplus c_1)$
$m_2 \oplus c_1 = 00110010|00000010|00000010 \oplus 00000001|00000010|00000011 = 00110011|00000000|00000001$

$c_2 = Enc_k(00110011|00000000|00000001)$

$q_0 = 00110111 \oplus 11001100 = 11111011$, $q_1 = 00110011 \oplus 11111111 = 11001100$,
$q_2 = 00000000 \oplus 11111110 = 11111110$

Thus, $c_2 = 11111011|11001100|11111110$.

Therefore, the full ciphertext $c = 00000001|00000010|00000011|11111011|11001100|11111110$.

b) Let $m' = m_1|m_2$
$m_1 = 11001001|00110100|11001000$
$m_2 = 00110010|00000000|00000000$

Applying CBC mode:

Calculating c_1: $c_1 = Enc_k(m_1 \oplus IV)$, since $IV = 0$ we have
$c_1 = Enc_k(m_1) = Enc_k(11001001|00110100|11001000)$

$q_0 = 00110111 \oplus 00110110 = 00000001$, $q_1 = 11001001 \oplus 11001011 = 00000010$,
$q_2 = 00110100 \oplus 00110111 = 00000011$

Thus, $c_1 = 00000001|00000010|00000011$ (same as in part a).

Calculating c_2: $c_2 = Enc_k(m_2 \oplus c_1)$
$m_2 \oplus c_1 = 00110010|00000000|00000000 \oplus 00000001|00000010|00000011 = 00110011|00000010|00000011$

$c_2 = Enc_k(00110011|00000010|00000011)$

$q_0 = 00110111 \oplus 11001100 = 11111011$, $q_1 = 00110011 \oplus 11111101 = 11001110$,
$q_2 = 00000010 \oplus 11111100 = 11111110$

Thus, $c_2 = 11111011|11001110|11111110$

In ciphertext stealing, the ciphertext $c = c_2|\widehat{c_1}$, where $\widehat{c_1}$ is the high bits of c_1 that correspond to the bits of m_2. Therefore $c = 11111011|11001110|11111110|00000001$.

Question 4.

a) Keylength = 56 bits, Blocksize = 64 bits

For $\forall k \in \{0, 1, 2, \ldots, 2^{56}\}$ (all possible possible keys k), we must compute $DEC_k(c)$.

Let us assume that our hash function maps all values $DEC_k(c)$ to a unique index of our hash table. Thus, our hash table will need to have 2^{56} indices.

In each index we must store $(k, DEC_k(c))$, i.e. a 56-bit key and a 64-bit block. This will require $56 + 64 = 120 < 128 = 2^7$ bits in total per index.

So, the total storage for the entire hash table is

$$2^{56} \cdot 2^7 = 2^{63} \text{ bits}$$

or

$$\frac{2^{63}}{(2^3)(2^{10})(2^{10})(2^{10})(2^{10})(2^{10})(2^{10})} = 1 \text{ EB}$$

Therefore, the hash table for double-DES requires approximately (though less than) 1 EB of storage.

b) Keylength = 128 bits, Blocksize = 128 bits

For $\forall k \in \{0, 1, 2, \ldots, 2^{128}\}$ (all possible possible keys k), we must compute $DEC_k(c)$.

Let us assume that our hash function maps all values $DEC_k(c)$ to a unique index of our hash table. Thus, our hash table will need to have 2^{128} indices.

In each index we must store $(k, DEC_k(c))$, i.e. a 128-bit key and a 128-bit block. This will require $128 + 128 = 256 = 2^8$ bits in total per index.

So, the total storage for the entire hash table is

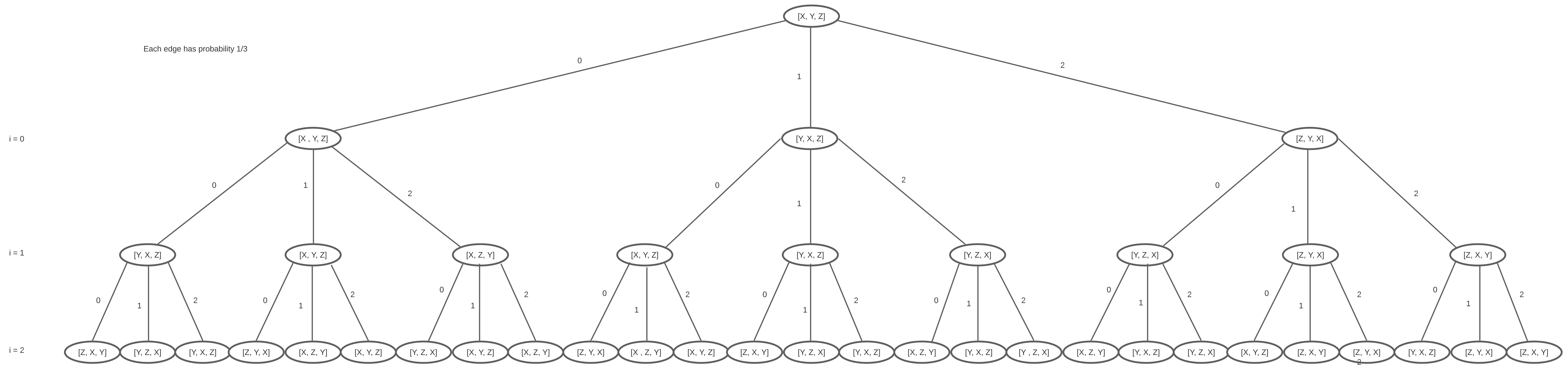
$$2^{128} \cdot 2^8 = 2^{136} \text{ bits}$$

or

$$\frac{2^{136}}{(2^3)(2^{10})(2^{10})(2^{10})(2^{10})(2^{10})(2^{10})} = 2^{73} \approx 9.44 \cdot 10^{21} \text{ EB}$$

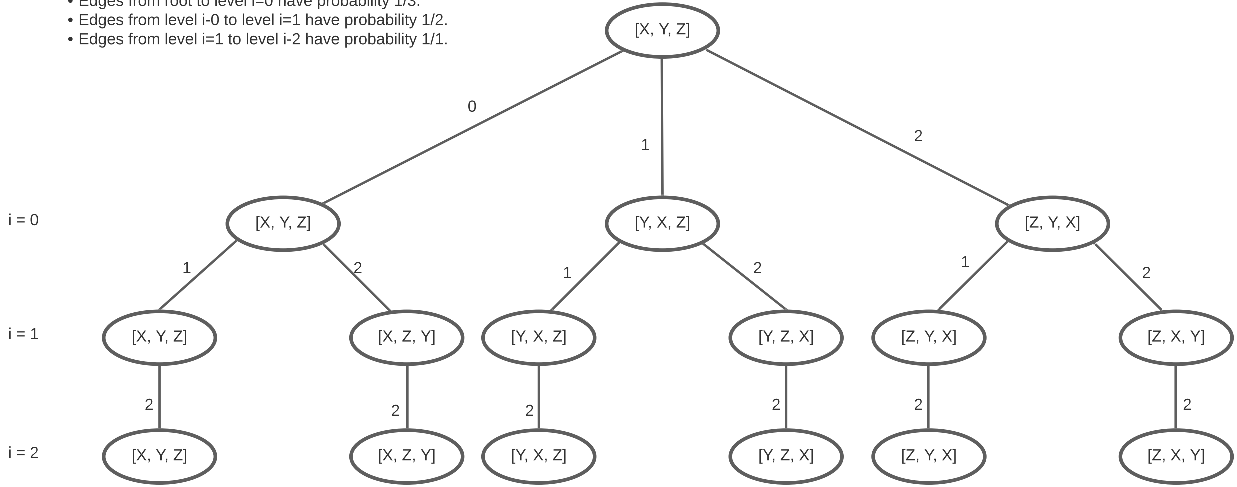
Therefore, the hash table for double-AES requires $9.44 \cdot 10^{21}$ EB of storage.

Appendix 1 Figure 1

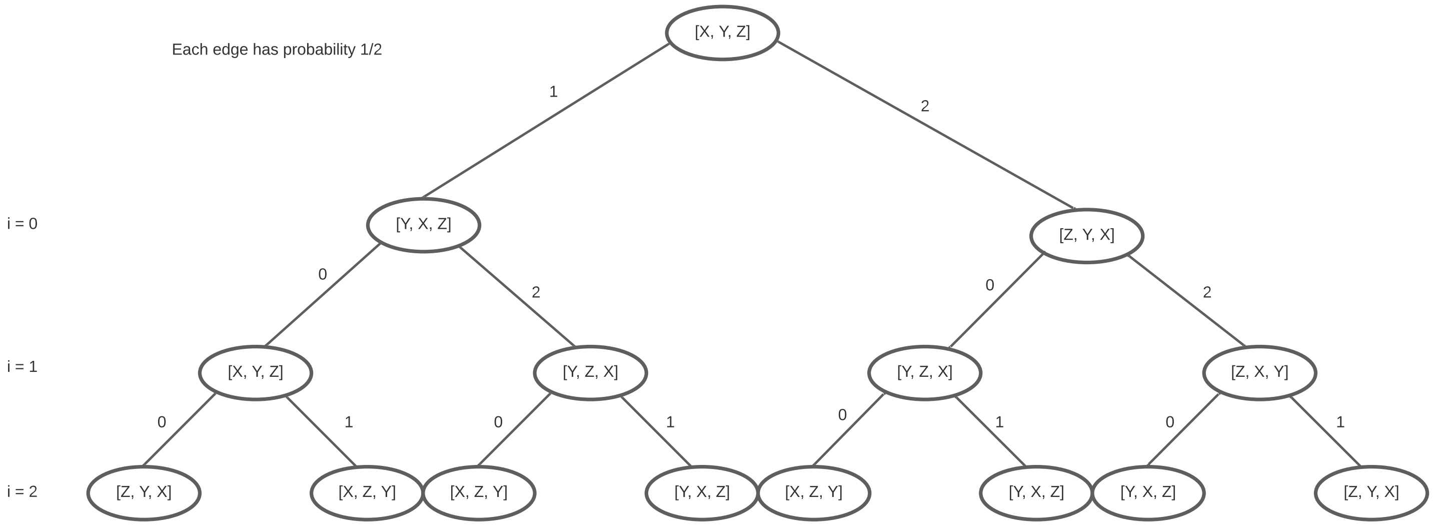


Appendix 1 Figure 2

- Edges from root to level i=0 have probability 1/3.
- Edges from level i-0 to level i=1 have probability 1/2.
- Edges from level i=1 to level i-2 have probability 1/1.



Appendix 1 Figure 3


Each edge has probability 1/2
[X, Y, Z]
1
2
i = 0
[Y, X, Z]
[Z, Y, X]
0
2
0
2
i = 1
[X, Y, Z]
[Y, Z, X]
[Y, Z, X]
[Z, X, Y]
0
1
0
1
0
1
0
1
i = 2
[Z, Y, X]
[X, Z, Y]
[X, Z, Y]
[Y, X, Z]
[X, Z, Y]
[Y, X, Z]
[Y, X, Z]
[Z, Y, X]