Cryptology - Homework 3

Christoffer Luplau - s204498

I did not get to finish all of the assignments. I was unable to get in touch with my group member, and therefore had to do his part myself last night, and unfortunately ran out of time.

Exercise 1

To verify that a mapping is homomorphic, we will check if the homomorphic property holds for any valid input, $x, y \in Z_N$

We can verify this property by selecting a few different pairs of integers $x, y \in Z_N$, and checking whether or not the two properties hold.

Let N = 33.

Pairs of x and y:
x = 10, y = 22
x = 6, y = 15
x = 28, y = 9

First we look at

$$\alpha(x) \cdot \alpha(y) \bmod N^2 = \alpha(x + y \bmod N)$$

And expanded:

$$((1 + xN) \bmod N^2) \cdot ((1 + yN) \bmod N^2) \bmod N^2 = (1 + (x + y \bmod N)N) \bmod N^2$$

x	y	$\alpha(x) \cdot \alpha(y) \bmod N^2$	$\alpha(x + y \bmod N)$
10	22	1057	1057
6	15	694	694
28	9	133	133

As we can see the mapping is homomorphic in the first case.

We now look at the second case:

$$\alpha(x)^y \bmod N^2 = \alpha(x \cdot y \bmod N)$$

And expanded:

$$((1 + xN) \bmod N^2)^y \bmod N^2 = (1 + (x \cdot y \bmod N)N) \bmod N^2$$

x	y	$\alpha(x)^y \bmod N^2$	$\alpha(x \cdot y \bmod N)$
10	22	727	727
6	15	793	793
28	9	694	694

In the cases we have tested, we have verified that the homomorphic property holds

Exercise 2

We find the inverse mapping $\alpha^{-1}(\cdot)$. We know the mapping function $\alpha(x) = (1 + xN) \bmod N^2$, which is a function that maps from Z_N into $Z^*_{N^2}$. By definition we know that $(1 + xN) < N^2$ in our use of $\alpha(x)$ as $x \in Z_N$. The "$\ldots \bmod N^2$" is therefore irrelevant. We need know only consider $(1 + xN)$.

We want to find an $\alpha^{-1}(\cdot)$ such that:

$$x = \alpha^{-1}(\alpha(x))$$

We can build it up step by step

$$\begin{aligned} x &= \alpha^{-1}(\alpha(x)) && \text{Write initial form} \\ &= \alpha^{-1}(1 + xN) && \text{Expand the mapping} \\ &= 1 - (1 + xN) && \text{Subtract the 1 away} \\ &= \frac{(1 - (1 + xN))}{N} && \text{Divide the N away} \end{aligned}$$

We arrive at the inverse mapping:

$$\alpha^{-1}(x) = \frac{(1 - x)}{N}$$

Exercise 3

We verify the multiplicatively homomorphic property much like in **Exercise 1**.

We choose an N, x and y

Let N = 17, x = 7, y = 9

$$\begin{aligned}
\beta(x) &= x^N \bmod N^2 = 7^{17} \bmod 17^2 &&= 75 \\
\beta(y) &= y^N \bmod N^2 = 9^{17} \bmod 17^2 &&= 179 \\
\beta(x) \cdot \beta(y) &= 75 \cdot 179 &&= 13425 \\
\beta(x \cdot y) &= (x \cdot y)^N \bmod N^2 = (7 \cdot 9)^{17} \bmod 17^2 &&= 131
\end{aligned}$$

And we notice that $13425 \bmod 17^2 = 131$. We therefore verified that the mapping is multiplicatively homomorphic $\bmod N^2$

Exercise 4

Exercise 5

Exercise 7

Exercise 8

We know that H is collision-resistant and we assume that there exists a collision in G'. Specifically there exists two inputs, $x = (x_1|x_2|x_3|x_4)$ and $x' = (x'_1|x'_2|x'_3|x'_4)$ where $G'(x) = G'(x')$ and $x \neq x'$

We would then have that: $H(H(x_1|x_2)|H(x_3|x_4)) = H(H(x'_1|x'_2)|H(x'_3|x'_4))$

Since we know that H is collision-resistant, it must be that $x_1|x_2 = x'_1|x'_2$ and $x_3|x_4 = x'_3|x'_4$, but this is not possible since x and x' are distinct. Our assumption that there exists a collision in G' must be false then.

Exercise 9

To construct a collision-resistant hash function G for inputs of length $n \cdot 2^k$ for arbitrary $k \in \mathbb{N}, k > 1$, we can use H from **Exercise 8.**

As the length of the input is of length n, doubled k times, we can recursively iterate on the input, applying the collision-resistant hash function H as in **Exercise 8**. Since $H : \{0,1\}^{2n} \to \{0,1\}^n$ outputs with length half that of the input length, we can recursively apply this function until the output has length n.

Example (values are picked at random):

```
n: 4
input: 123456789ABCDEFG

1234       5678 9ABC       DEFG
|             | |             |
---H(x1|x2)---   ---H(x3|x4)---
       |                |
      abcd             qrst
       |                |
       -----H(x1|x2)-----
               |
             1Aaq
```

Exercise 10

The Merkle-Damgård construction works by splitting the input message into blocks of a fixed size, and processing each block in turn using a compression function. The compression function takes the current state of the hash function and the current message block and produces the next state. As each iteration is reliant on the computed value of the one just before it, this process can not be parallelised. The runtime of the a program for Merkle-Damgård hashing will be linear with the input size, $O(n)$

The Merkle tree hashing algorithm can parallelise each “layer” of recursion (assuming that we can compute as many compressions in parallel as we need), since each computation is only reliant on the computations of the recursive call that it has called. In the example given in **Exercise 4**, we would initially call H with the input string, which would recursively call two instances of H. These two could be computed in parallel, returned to the initial call and then that could be calculated. For a input of length 4 times that of the desired output, we would only create two “recursive layers” and thereby have to call the H function in two batches. An example with the input of length 8 times that of the desired output would only result in three “recursive layers”. This results in a runtime of the Merkle tree hashing which would be logarithmic with regards to the input size, $O(\ln n)$