

# Crypto -HW 3

Hagai Ben Yehuda, ID num: 305237000

Jonathan Bauch, ID num: 204761233

## 1

### 1.a

Let  $m_1, m_2 \in \{0, 1\}^n$  be two distinct messages. Assume that we have obtained  $MAC_k(m_2) = F_k(m_2)$  and  $MAC_k(m_1||m_2) = F_k(m_1) \oplus F_k(m_2)$  ( $||$  means concatenation) then we can calculate

$$MAC_k(m_1||m_2) \oplus MAC_k(m_2) = F_k(m_1) \oplus F_k(m_2) \oplus F_k(m_2) = F_k(m_1) = MAC_k(m_1)$$

Thus we can obtain the  $MAC$  for a new message showing that this scheme is indeed insecure for variable length messages.

### 1.b

Let  $m_1, m_2 \in \{0, 1\}^n$  be two distinct messages. Assume that we have obtained

$$MAC_k(m_1||m_1) = (F_k(m_1), F_k(F_k(m_1)))$$

and

$$MAC_k(m_2||m_2) = (F_k(m_2), F_k(F_k(m_2)))$$

Then we have also obtained

$$MAC_k(m_1||m_2) = (F_k(m_1), F_k(F_k(m_2)))$$

(we have the first part from the result for  $MAC_k(m_1||m_1)$  and the second part from  $MAC_k(m_2||m_2)$ ) thus we have obtained the  $MAC$  for a new message  $(m_1||m_2)$  showing that this scheme is insecure.

### 1.c

Let  $m_1, m_2 \in \{0, 1\}^{n-1}$  be two distinct messages. Assume that we have obtained

$$MAC_k(m_1||m_1) = (F_k(0||m_1), F_k(1||m_1))$$

and

$$MAC_k(m_2||m_2) = (F_k(0||m_2), F_k(1||m_2))$$

Then we can construct

$$MAC_k(m_1||m_2) = (F_k(0||m_1), F_k(1||m_2))$$

(where the first part comes from  $MAC_k(m_1||m_1)$  and the second one from  $MAC_k(m_2||m_2)$ ). Thus we have forged a  $MAC$  for a previously unseen message proving that the scheme is insecure.

## 2

Let  $m_1, m'_1, m_2 \in \{0, 1\}^{n-1}$  be distinct messages. First we obtain

$$MAC(m_1) = E_k(E_k(m_1) \oplus (0\dots 01))$$

then we obtain

$$MAC(m'_1) = E_k(E_k(m'_1) \oplus (0\dots 01))$$

Now set

$$m'_2 := m_2 \oplus MAC(m_1) \oplus MAC(m'_1)$$

And obtain

$$\begin{aligned} MAC(m'_1 || 0\dots 01 || m'_2) &= E_k(E_k(E_k(E_k(m'_1) \oplus 0\dots 01) \oplus m'_2) \oplus 0\dots 011)) \\ &= E_k(E_k(MAC(m'_1) \oplus m'_2) \oplus 0\dots 011)) \\ &= E_k(E_k(MAC(m_1) \oplus m_2) \oplus 0\dots 011)) \\ &= E_k(E_k(E_k(E_k(m_1) \oplus 0\dots 01) \oplus m_2) \oplus 0\dots 011)) \\ &= MAC(m_1 || 0\dots 01 || m_2) \end{aligned}$$

Thus we can send  $(m_1 || 0\dots 01 || m_2, MAC(m'_1 || 0\dots 01 || m'_2))$ , as we have shown this is indeed the  $MAC$  for this message and we haven't asked for it directly ( $m'_1$  was chosen to be different from  $m_1$ ), showing that this scheme is also not secure.

## 3

### 3.a

No, for instance if  $h$  is the identity on  $\{0, 1\}^n$  then it is obviously collision resistant, but is not a  $OWF$  as for every  $x \in Im(h)$ ,  $h^{-1}(x) = x$  and thus for every element in  $h$ 's image, a source can be found in constant time with probability 1.

### 3.b

Indeed, define  $A'$  as follows:

- Choose  $x \in \{0, 1\}^n$  uniformly from  $\{0, 1\}^n$ .
- Return  $(A(h(x)), x)$ .

Correctness: the probability that  $A'$  indeed returns (at the first position)  $x'$  such that  $h(x') = h(x)$  is  $\geq \epsilon$  from the assumption on  $A$ . thus the only condition in which  $x$  and  $x'$  are not an example of a collision is if  $x = x'$ , we now note that:

$$\begin{aligned} \Pr_{x \leftarrow U_n} (A' \text{ returns the same element in both indices}) &= \\ \Pr_{x \leftarrow U_n} (A \text{ finds an inverse for } h(x)) * \Pr_{x \leftarrow U_n} (x = x' | h(x) = h(x')) &= \\ \Pr_{x \leftarrow U_n} (A \text{ finds an inverse for } h(x)) * \frac{1}{2^n} \end{aligned}$$

where the first equality is due to the fact that  $x$  was chosen in random and the second equality is derived from the regularity of  $h$ .

Thus we obtain:

$$\begin{aligned}
& \Pr_{x \leftarrow U_n} (A' \text{ returns a valid collision}) = \\
& \Pr_{x \leftarrow U_n} (A \text{ finds an inverse for } h(x) \cap A'(h(x)) \neq x) = \\
& \Pr_{x \leftarrow U_n} (A \text{ finds an inverse for } h(x)) - \Pr_{x \leftarrow U_n} (A \text{ finds an inverse for } h(x) \cap A(h(x)) = x) = \\
& \Pr_{x \leftarrow U_n} (A \text{ finds an inverse for } h(x)) - \Pr_{x \leftarrow U_n} (A(h(x)) = x) = \\
& \Pr_{x \leftarrow U_n} (A \text{ finds an inverse for } h(x)) - \Pr_{x \leftarrow U_n} (A' \text{ returns the same element in both indices}) = \\
& \Pr_{x \leftarrow U_n} (A \text{ finds an inverse for } h(x)) * (1 - \frac{1}{2^n}) \geq \epsilon(1 - \frac{1}{2^n})
\end{aligned}$$

Hence  $A'$  is a PPT algorithm as requested.

### 3.c

We construct  $A'$  as before and provide a new analysis (as some of the constraints regarding  $h$  and the origin size have changed)

$$\Pr_{x \leftarrow U_n} (A' \text{ produces a valid collision}) = \Pr_{x \leftarrow U_n} (A \text{ finds inverse}) - \Pr_{x \leftarrow U_n} (A(h(x)) = x)$$

note that:

$$\begin{aligned}
& \Pr_{x \leftarrow U_n} (A(h(x)) = x) = \\
& \Pr_{x \leftarrow U_n} (A(h(x)) = x \cap \exists x' \in \{0, 1\}^n x' \neq x \text{ s.t. } h(x) = h(x')) + \\
& \Pr_{x \leftarrow U_n} (A(h(x)) = x \cap \forall x' \in \{0, 1\}^n h(x') = h(x) \rightarrow x = x')
\end{aligned}$$

but we have that

$$\Pr_{x \leftarrow U_n} (\forall x' \in \{0, 1\}^n h(x') = h(x) \rightarrow x = x') \leq \frac{2^n - 1}{2^{n+s}} \leq \frac{1}{2^s}$$

since there are at most  $2^n$  possible values for  $h$  (the number of elements in the range). thus

$$\Pr_{x \leftarrow U_n} (A(h(x)) = x \cap \forall x' \in \{0, 1\}^n h(x') = h(x) \rightarrow x = x') \leq \frac{1}{2^s}$$

in addition since in the second case there are at least two elements that collide we have that

$$\Pr_{x \leftarrow U_n} (A(h(x)) = x \cap \exists x' \in \{0, 1\}^n x' \neq x \text{ s.t. } h(x) = h(x')) \leq \frac{\Pr_{x \leftarrow U_n} (h(A(h(x))) = h(x))}{2}$$

Therefore we obtain:

$$\Pr_{x \leftarrow U_n} (A' \text{ produces a valid collision}) \geq \Pr_{x \leftarrow U_n} (A \text{ finds inverse}) - \frac{1}{2^s} - \frac{\Pr_{x \leftarrow U_n} (A \text{ finds inverse})}{2} \geq \frac{\epsilon}{2} - \frac{1}{2^s}$$

as requested.

## 4

### 4.a

Let  $m \in \{0, 1\}^{351}$  then  $H(m) = h(0^{160} || m || 0) = H(m || 0)$  this is true since when calculating the hash of a message it is padded with zeros to be of a length that is a multiply of 352.

## 4.b

To solve this problem we pad the message with zeros to have a size that is a multiple of 352 and then add another block that is the number of zero bits we have added to the last block of the original message (if no bits were added to the last block then a block of zeros is added). Note that this makes the mapping from a message to its padded form one to one (the original message can be determined from the padded message). A general idea for the equivalence between the security of the proposed scheme and  $h$ : assume that  $H$  is not secure, then we can find in polynomial time two messages  $m_1, m_2 \in \{0, 1\}^n$  such that  $m_1 \neq m_2$  and  $H(m_1) = H(m_2)$ . Now we inspect the process of calculating  $H$  for both messages, since the output of both calculations is identical, but the input is different (as we have shown the padding is one to one) there must be a point where the input to both invocations of  $h$  is different but the output is identical (as the final output is identical) meaning that we can find using this pair a pair of messages that form a collision in  $h$  in contradiction to  $h$ 's security.

## 5

We construct an adversary  $A$  that on input  $(c, pk)$  performs the following:

- Set  $m'$  to be the next message by lexicographical order (if the message space is not bounded we can also think of a lexicographical order).
- Get the next element by lexicographic order and define it to be  $r$  (next as in after the previous value of  $r$ , in practice  $m = r$  this separation is meant to increase the readability of the next step).
- Check for all messages with lexicographic order upto  $m'$  if  $Enc_{pk,r}(l) = c$  (where  $l$  is a message with lexicographic order smaller than  $m'$ , we go over all of them) if  $l$  satisfies this condition return  $l$ , check if  $m'$  satisfies  $Enc_{pk,r'}(m') = c$  for any  $r'$  with lexicographical order smaller or equal to  $r$  if  $m'$  satisfies the condition for some  $r'$  return  $m'$  else go back to the previous state.

Note that since  $Enc_{pk,r}$  is an encryption scheme it is one-to-one (otherwise it would not be possible to decipher messages), thus if we find  $m'$  and that satisfies for some  $r'$ :  $Enc_{pk,r'}(m') = c = Enc_{pk,r}(m)$  it must also satisfy  $m' = m$ . Since we are guaranteed that  $c$  has an origin under  $Enc_{pk,r}$  (we know that  $c$  was generated from encrypting  $m$  with some  $r$  using this scheme),  $A$  will reach it in finite time because the message  $m$  has a finite lexicographical order and so does the random element  $r$  thus since we go over all possible pairs, we will find a matching pair. Because no other pair can satisfy  $Enc_{pk,r'}(m') = c$  ( $Enc_{pk,r}$  is one-to-one),  $A$  will not stop before reaching the pair, hence  $A$  will return  $m$  with probability 1.

## 6

As hinted, we set  $Enc_{pk}(r, b) = (F_{pk}(r), B(r) \oplus b)$ , and set  $Dec_{sk}(r', b') = Inv_{sk}(r'), B(Inv_{sk}(r')) \oplus b'$ .  
**Correctness:**

$$\begin{aligned}
 Dec_{sk}(Enc_{pk}(r, b)) &= Dec_{sk}(F_{pk}(r), B(r) \oplus b) \\
 &= (Inv_{sk}(F_{pk}(r)), B(Inv_{sk}(F_{pk}(r))) \oplus B(r) \oplus b) \\
 &= (r, B(r) \oplus B(r) \oplus b) \\
 &= (r, 0 \oplus b) \\
 &= (r, b)
 \end{aligned}$$

**Semantically-Secure:** Assume that this is not a  $2\epsilon$  semantically-secure encryption, then there exists a PPT algorithm  $A$  such that:

$$\Pr[A(F_{pk}, B_k(r) \oplus 0) = 1] - \Pr[A(F_{pk}, B_k(r) \oplus 1) = 1] \geq 2\epsilon$$

Which equivalently means:

$$\Pr[A(F_{pk}, B_k(r)) = 1] - \Pr[A(F_{pk}, \overline{B_k(r)}) = 1] \geq 2\epsilon$$

Thus:

$$\Pr[A(F_{pk}, \overline{B_k(r)}) = 1] \leq \Pr[A(F_{pk}, B_k(r)) = 1] - 2\epsilon$$

Note that:

$$\Pr_{u \leftarrow U_1}[A(F_{pk}(r), u) = 1] = \frac{1}{2} \Pr[A(F_{pk}, B_k(r)) = 1] + \frac{1}{2} \Pr[A(F_{pk}, \overline{B_k(r)}) = 1]$$

Hence using the the last inequality we get:

$$\begin{aligned} \Pr_{u \leftarrow U_1}[A(F_{pk}(r), u) = 1] &= \frac{1}{2}(\Pr[A(F_{pk}, B_k(r)) = 1] + \Pr[A(F_{pk}, \overline{B_k(r)}) = 1]) \\ &\leq \frac{1}{2}(\Pr[A(F_{pk}, B_k(r)) = 1] + \Pr[A(F_{pk}, B_k(r)) = 1] - 2\epsilon) \\ &= \Pr[A(F_{pk}, B_k(r)) = 1] - \epsilon \end{aligned}$$

By rearranging the terms, we obtain:

$$\Pr[A(F_{pk}, B_k(r)) = 1] - \Pr_{u \leftarrow U_1}[A(F_{pk}(r), u) = 1] \geq \epsilon$$

Note that we can set  $A'(pk, F_{pk}(r), B(r)) = A(F_{pk}(r), B(r))$  and thus  $A'$  contradicts the fact that  $B(r)$  is an  $\epsilon HCB$  for  $F$ . Hence our encryption scheme is in-fact  $2\epsilon$  semantically-secure.

## 7

Listing 1: Q7 Code

```
import random

def Q7a():
    print 'Q7a'
    for m in [35, 37, 38]:
        check_group(m)
        print('-', *16)

def check_group(m):
    Zm = Integers(m)
    G = [x for x in Zm if is_unit(x, m)]
    G_with_orders = {x: multiplicative_order(x) for x in G}
    max_order = max(G_with_orders.values())
    max_order_elements = [x for x, x_order in G_with_orders.items() if x_order == max_order]
    print 'm:', m
    is_cyclic = (max_order == len(G))
    print 'G_order:', len(G)
    print 'max_order_of_elements_in_G:', max_order
    print 'is_G_cyclic?', is_cyclic
    print 'G_elements_of_maximum_order:', max_order_elements
```

```

def Q7b():
    print 'Q7b'
    m = 2**107 - 1
    N = 100000
    Zm = Integers(m)
    random_units = get_random_units(m, N)
    generators = [x for x in random_units if is_multiplicative_generator(x, m)]
    A = len(generators)
    print 'N: ', N
    print 'A: ', A
    print 'first_10_generators:'
    for i, g in enumerate(generators[:10]):
        print '____' + str(g)
    print 'sampled_generators_ratio:', float(A) / N
    print 'real_generators_ratio:', float(euler_phi(m - 1)) / m

def get_random_units(m, count):
    return [get_random_unit(m) for _ in range(count)]

def get_random_unit(m):
    while True:
        r = random.randint(1, m - 1)
        if is_unit(r, m):
            return r

def is_multiplicative_generator(g, k):
    phi_k = euler_phi(k)
    return all(pow(g, phi_k//p, k) != 1 for p, _ in phi_k.factor())

# Note: Theses functions are already implemented in sage,
#       but it wasn't clear if we are allowed to use them.
def is_unit(g, m):
    # Check if g is invertible in Zm
    return gcd(g, m) == 1

def multiplicative_order(y):
    order = 1
    x = y
    while x != 1:
        x *= y
        order += 1
    return order

if __name__ == '__main__':
    Q7a()
    Q7b()

```

Listing 2: Q7 Output

```

Q7a
m: 35
G order: 24
max order of elements in G: 12

```

```
is G cyclic? False
G elements of maximum order: [2, 3, 12, 17, 18, 23, 32, 33]
```

---

```
m: 37
G order: 36
max order of elements in G: 36
is G cyclic? True
G elements of maximum order: [2, 5, 13, 15, 17, 18, 19, 20, 22, 24, 32, 35]
```

---

```
m: 38
G order: 18
max order of elements in G: 18
is G cyclic? True
G elements of maximum order: [3, 13, 15, 21, 33, 29]
```

---

```
Q7b
N: 100000
A: 33209
first 10 generators:
  66857716506250319367553264301193
  32776249515633484594257352388433
  90283088453948613309802956552831
  103701705485172287585538502325344
  84065116369294643431138850252003
  80185259793528183472008725361755
  19719375452390168077799296829009
  67997902305654377490822770728753
  45286240638232656939088920158247
  59960276008596838400897452282289
sampled generators ratio: 0.33209
real generators ratio   : 0.330161384116
```

**7.b:** the sampled generators ratio ( $A/N$ ) is 0.332, which is very close to the real ratio of generators  $\phi(\phi(2^{107} - 1))/\phi(2^{107} - 1) \approx 0.330$ . So we would say that the estimate is pretty good.

Listing 3: Q8 Code

```

R.<x> = QQ[]
f = x^4 + x^3 + x^2 + x + 1
K.<a> = GF(3**4, name='a', modulus=f(x))

def Q8():
    print 'f(x)_____:', f
    print 'f(x)_factors_:', f.factor_mod(3)
    print 'f_irreducible?', len(f.factor_mod(3)) == 1
    units = [g for g in K if is_unit(g)]
    generators = {g for g in units if is_generator(g)}
    print 'multiplicative_generators_count:', len(generators)
    print 'multiplicative_generators_____:'
    for g in generators:
        print '____' + str(g)

def is_generator(y):
    return multiplicative_order(y) == 3**4 - 1

# Note: Theses functions are already implemented in sage,
#       but it wasn't clear if we are allowed to use them.
def is_unit(g):
    # Check if g is invertible in K
    return f.gcd(R(g)) == 1

def multiplicative_order(y):
    order = 1
    x = y
    while x != 1:
        x *= y
        order += 1
    return order

if __name__ == '__main__':
    Q8()

```

Listing 4: Q8 Output

```

f(x)          : x^4 + x^3 + x^2 + x + 1
f(x) factors  : x^4 + x^3 + x^2 + x + 1
f irreducible? True
multiplicative generators count: 32
multiplicative generators      :
    a + 2
    2*a + 1
    a^2 + 2
    a^2 + 2*a
    a^2 + 2*a + 2
    2*a^2 + 1
    2*a^2 + a

```



$$\begin{aligned}
&2*a^2 + a + 1 \\
&a^3 + 2 \\
&a^3 + 2*a \\
&a^3 + 2*a + 1 \\
&a^3 + a^2 + a + 2 \\
&a^3 + a^2 + 2*a \\
&a^3 + a^2 + 2*a + 2 \\
&a^3 + 2*a^2 + 1 \\
&a^3 + 2*a^2 + 2 \\
&a^3 + 2*a^2 + a + 1 \\
&a^3 + 2*a^2 + a + 2 \\
&a^3 + 2*a^2 + 2*a \\
&a^3 + 2*a^2 + 2*a + 2 \\
&2*a^3 + 1 \\
&2*a^3 + a \\
&2*a^3 + a + 2 \\
&2*a^3 + a^2 + 1 \\
&2*a^3 + a^2 + 2 \\
&2*a^3 + a^2 + a \\
&2*a^3 + a^2 + a + 1 \\
&2*a^3 + a^2 + 2*a + 1 \\
&2*a^3 + a^2 + 2*a + 2 \\
&2*a^3 + 2*a^2 + a \\
&2*a^3 + 2*a^2 + a + 1 \\
&2*a^3 + 2*a^2 + 2*a + 1
\end{aligned}$$