CO487 Assignment 3

Daniel Burstyn (20206120)

Feb 27, 2009

1. **MAC schemes derrived from block ciphers**

   i) This MAC scheme is **insecure**.
   First recall that a MAC scheme is insecure if it is vulnerable to a chosen-message attack. Now I will show that this scheme is vulnerable to such an attack.

   Observe that to compute the tag of any arbitrary message $m = m_1 m_2 \dots m_t$ an attacker can do the following:
   - Obtain the message tag pair $(m_1, MAC_k(m_1))$ via chosen-message attack.
   - From the scheme, we know that $MAC_k(m_1) = c_t = c_0 \oplus m_1 \oplus E_k(m_1)$
   - So, compute $E_k(m_1) = MAC_k(m_1) \oplus m_1$
   - Now repeat this process to determine $E_k(m_2), \dots, E_k(m_t)$
   - It is now very easy to compute $MAC_k(m)$ since we know all of m_i and $E_k(m_i)$

   Thus this MAC scheme is insecure.

   ii) This MAC scheme is also **insecure**.
   An attacker can obtain the MAC tag for a 1 block message $m = m_1$ to receive (m, c_0, c_t). However, observe that $c_t = E_k(c_0 \oplus m)$. Also observe, that if we pick $m' = c_0$, and $c_0' = m$ then $c_t' = E_k(c_0' \oplus m') = E_k(m \oplus c_0) = c_t$. Thus (c_0, m, c_t) is also valid, and we have proved that the scheme is insecure.

   iii) This MAC scheme is also **insecure**.
   Since H is a hash function with 80 bit hashes, then we can use the generic attack for finding collisions (slide 130) to find a collision in H in $\approx 2^{40}$ steps, which is feasible. Suppose the messages m and m' are the collision, such that $H(m) = H(m')$. We can choose the message m to obtain $MAC_k(m)$. Again, observe that $MAC_k(m) = E_k(H(m)) = E_k(H(m')) = MAC_k(m')$ and we have thus computed the tag of a new message, and have therefore shown that the MAC scheme is insecure.

   iv) This MAC scheme is also **insecure**.
   First we notice that for one block messages, this scheme is almost identical to the scheme used in (ii), except that the resulting c_t is hashed with H, so we will reuse the attack from (ii). From (ii) we know that if (m, c_0, c_t) is valid, then (c_0, m, c_t) is valid. Since c_t remains the same, then for this scheme, h will remain the same. So, if we obtain (m, c_0, h) from a chosen-message attack, then (c_0, m, h) is also valid, and we have once again proved the scheme is insecure.

2. **Another MAC scheme derrived from a block cipher**

   First, let us define $M_i' = M_i$ with the last block of zeros, and the last one removed. That way by using the message M_i', the first step of the encryption will result in a one block modified message M_i. Also, for convenience let us define $M_0 = 1000 \dots 0$ — the one block message of 1 one and 127 zeroes.

Now, let's begin by obtaining the MAC tags of $M'_1, M'_2, \ldots, M'_l$. Which from the scheme are equal to $c_1, c_2, \ldots, c_l$ respectively.

Next, we will obtain b by obtaining the tag to the message $M_1M'_1$. We see that $MAC_{(k,b)}(M_1M'_1) = c_1(1+b) \bmod p$. Now, since p is prime, we can divide by c_1 (Recall that modulo division is multiplication by an inverse which ALWAYS exists as long as the modulus is prime.) Therefore we can obtain b after a trivial subtraction by 1.

The last thing we need to find is $c_0 = AES_k(M_0)$. To do this we obtain the tag for the message M_1 (Not M'_1). The padding step of the scheme will result in the modified message M_1M_0, and the tag itself will be $c_1 + c_0 b \bmod p$. We can subtract c_1, and now divide by b to get c_0.

We now have all the components we need to compute the MAC of M. Observe that $MAC_{(k,b)}(M) = (c_1 + c_2b + c_3b^2 + \ldots + c_lb^{l-1} + c_0b^l) \bmod p$, and that we know all the variables and can thus compute the MAC.

Last we must consider whether this computation is **efficient**. The computations done are addition, subtraction, multiplication (by inverses), and exponentiation. Addition, subtraction and multiplication are trivially efficient, and we know from class that modular exponentiation is also effienct. Therefore it has been shown that an attacker can **effienctly** compute he MAC tag of M.

3. **RSA computations**

a) We must determine Alice's private key from her public key by factoring n:
- With a calculator, we quickly find that $1073 = 29 * 37$.
- We then compute $\phi = (p-1)(q-1) = 28 * 36 = 1008$.
- Since $ed \equiv 1 \bmod \phi$, we can use the extended euclidian algorithm to determine d. We can use the EEA to find the multiplicative inverse of e in the integers mod ϕ. Let's call the inverse i. Now $ed \equiv 1 \bmod \phi \Rightarrow ied \equiv i \bmod \phi \Rightarrow d \equiv i \bmod \phi$. Since $d < \phi$ we can conclude that $d = i$.
- We will use the EEA to find x and y in $\phi x + ey = 1$ since e and ϕ are coprime, and in fact y is the multiplicative inverse we seek.

x	y	$ex + \phi y$	quotient
1	0	1008	
0	1	715	1
1	-1	293	2
-2	3	129	2
5	-7	35	3
-17	24	24	1
22	-31	11	2
-61	86	2	5
327	-461	1	

- Using the EEA we find that $y = d = -461 \pmod{\phi} \equiv 547 \pmod{\phi}$ and thus we have obtained Alice's private key.

b) To encrypt $m = 3$, we begin by obtaining $(n, e) = (1073, 715)$ from the assignment. We now need to compute $c = m^e \bmod n$. We can use the repeated exponentiation process to do this computation effeciently, although doing it by hand can be painstaking, so we will do the computation in Maple instead. We find that $c = 548$.

4. **Computing GCD without long division**

a) Compute GCD using GCDEasy:

Step	a	b	e
Begin	308	440	1
While a, b both even...	154	220	2
...	77	110	4
While $a \neq 0$ {			
While b is even...	77	55	4
$a \geq b$ so $a = a - b$	22	55	4
goto start of loop			
While a is even...	11	55	4
$b > a$ so $b = b - a$	11	44	4
goto start of loop			
While b is even...	11	11	4
$a \geq b$ so $a = a - b$	0	11	4
}			

The result is $e * b = 44$.

b) The first step of GCDeasy is while a and b are both even, we divide them by 2. Since a and b are both integers, they can be prime factored. The loop effectively is while there is a 2 in the prime factorization, remove a 2. It is obvious to see that eventually, all the 2s will be gone, since both numbers are finite, and the loop will end, and we will move on to the next step.

For the next step, at the beginning of each loop, we reduce a and b so that they are both odd. Now observe that the if statement will either reduce a or reduce b, and that a cannot go below 0, and that b cannot go below 1.

We can see that at each iteration, either a or b must decrease, that neither can increase at any point, and that neither can go below 0. By this logic, we can conclude that either a or b will eventually reach their minimum values. If a reaches 0, then the loop obviously terminates. If b reaches 1, then we can easily see that a will continue to decrease until it reaches 0, and the loop will still terminate.

Therefore this procedure always terminates.

c) First lets simplify a bit by considering the first loop, and last step. Note that at the end of the first loop, e is the greatest common divisor or a and b that is a multiple of 2. Thus we can see that $\gcd(\frac{a}{e}, \frac{b}{e}) * e = \gcd(a, b)$. Now we only need to worry about the correctness of the inner loop. For convenience let $a' = \frac{a}{e}$ and $b' = \frac{b}{e}$.

Let us now say that $\hat{a}$ and $\hat{b}$ refer to the running values of the variables in the procedure. At the top of the loop, we know of course that $\gcd(a', b') = \gcd(\hat{a}, \hat{b})$. This is our loop invariant. we know that we have already taken out all the 2s from their gcd, so by dividing by 2 here again, we can still say that our invariant is true. Now the if statement either does $\hat{a} = \hat{a} - \hat{b}$ or $\hat{b} = \hat{b} - \hat{a}$ depending on which is larger. However, from the assignment, we know that $\gcd(a, b) = \gcd(a, b - a)$, and that similarily $\gcd(a, b) = \gcd(a - b, b)$, again depending on which is larger. So, once again we can conclude that our invariant is still true. Now that we have reached the end of the loop, and the invariant still holds true, we can conclude that after the loop finishes (which it does as proved in (b)) that $\gcd(a', b') = \gcd(\hat{a}, \hat{b})$. Since $\hat{a} = 0$, we can conclude that $gcd(\hat{a}, \hat{b}) = \hat{b}$.

Now that we have shown that inner loop is correct, and that the initial loop, and the final multiplication by e are correct if and only if the inner loop is correct, we can conclude that the procedure in its entirety is correct.

d) First, we observe that each division is a bitshift (one bit operation) and each subtraction is single bit operation, so the running time of GCDeasy is the number of divisions plus the number of subtractions.

In the worst case, a is $\{1\}^k = 2^k - 1$ (and is odd), and to minimize the number of subtractions, we want b to be one in the inner loop, so the worst case is if b is $\{1\}\{0\}^{k-1} = 2^{k-1}$. Note that both numbers have the same bitlength (k) and that it doesn't matter which is a or which is b so I picked them this way arbitrarily.

In the first step, a is odd, so we immediately move on to the next step. Here b is even, so we reduce it to 1 in $k-1$ divisions (since it was 2^{k-1}). Notice now that $a = \{1\}^k$ and that $b = 1$. After the if statement, a is reduced by one and is now $\{1\}^{k-1}\{0\}$ for a total of k bit operations so far.

Now, b will remain at 1 for the remainder of the process. We observe that at each iteration a is now even, so it is bitshifted by one, and then again the if statement is reduced by 1 and now a looks like $\{1\}^{k-2}\{0\}$. This will repeat until $a = 0$ which means two bit operations times $k-1$ bits. This gives a grand total of $3k-2$ bit operations, and thus the running time for GCDeasy is $O(k)$.

e) From (d) we found that the running time for GCDeasy is $O(k)$. We can see that a k-bit number is at most $2^k - 1$, and so, if the running time of GCDeasy is $O(k)$, then the running time is $O(log_2(a))$ where a is the larger of the two numbers. We know that log time is polynomial-time, so **yes**, GCDeasy is a polynomial-time algorithm.