CO487 Assignment 4

Daniel Burstyn (20206120)

April 7, 2009

1. **RSA Signatures**

   a) It is immediately obvious that this scheme is existentially forgeable. The message $m = 1$ has the signature $s = 1$.

   b) It is very easy to retrieve the private key d with a chosen-message attack, if you choose the message $m = 2$. After obtaining the signature, the attacker has to compute successive powers of 2 (mod n) until the signature is reached. This is very efficient to do since computing the next power of 2 involves only a bit-shift operation, and possibly a modulo operation. The number of iterations it took to reach the signature is the private key d, and we have completely broken the scheme.

2. **Chinese Remainder Theorem**
   After step (i) we will have 4 integers: s, t, m, n such that $sm + tn = 1$. Now observe the following:

$$\begin{aligned}
sm + tn &= 1 \\
sm + tn &\equiv 1 \bmod m \\
tn &\equiv 1 \bmod m \\
atn &\equiv a \bmod m \\
atn + km &\equiv a \bmod m \\
atn + bsm &\equiv a \bmod m
\end{aligned}$$

   By the same logic, we can also show that $atn + bsm \equiv b \bmod n$. So if $x = atn + bsm$ then $x \equiv a \bmod m$ and $x \equiv b \bmod n$ therefore $x \equiv atn + bsm \bmod mn$. Thus, I have shown that the algorithm is correct.

3. **Error attack on the RSA signature scheme**

   a) In order to prove the correctness, we can show that $m^d \equiv s_p \bmod p$, and that $m^d \equiv s_q \bmod q$. With this we can say that $s = m^d$ is the unique solution mod pq. Thus if we come across an s via the algorithm discussed, then by the uniqueness, it must be $m^d \bmod n$ which of course is the RSA signature. So, below we show that $m^d \equiv s_p \bmod p$:

      First, let us represent d as $q(p-1)+r$ where q, r are the quotient and remainder when d is divided by $p-1$, respectively. Now observe that $d_p = r$.

$$\begin{aligned}
m^d &\equiv m^d \bmod p \\
&\equiv m^{q(p-1)+r} \bmod p
\end{aligned}$$

$$\begin{aligned}
&\equiv m^{q(p-1)}m^r \bmod p \\
&\equiv m^r \bmod p \text{ (Recall by FLT } m^{p-1} \equiv 1) \\
m^d &\equiv m^{d_p} \bmod p \\
m^d &\equiv s_p \bmod p
\end{aligned}$$

By the same logic, we can also show that $m^d \equiv s_q \bmod q$, and thus by my arguments above I have proven the correctness of the algorithm.

b) First recall that repeated square multiplcation has a worst-case running time of $O(k^3)$ where k is the bitlength of the modulus. Now, since $n = pq$, if i, j, k are the bitlengths of p, q, n respectively, then $k \approx i + j$. So computing m^d directly takes $O(k^3)$. The proposed algorithm involves computing s_p, s_q and also finding s. Computing s_p and s_q will take $O(i^3) + O(j^3)$ and finding s can be done via the algorithm in 2. through EEA and one modulo operation—a total of $\approx O(i^2)$ if we assume $p > q$ WLOG. Since we already have $O(i^3)$ we can discard this term as negligible.

Now the issue is which is faster: $O(k^3)$ or $O(i^3)+O(j^3)$. Well since $k = i+j$ we can see that by the binomial expansion, $O(k^3) > O(i^3) + O(j^3)$. Although they have the same assymptotic running time, the constant factor difference makes this proposed method faster. If $i \approx j$, which is usually true, then we get that this new algorithm is faster by a factor of about 4.

c) We know that the smart card is producing s' in the form $s' = s'_p xq + s_q yp \bmod n$ (see question 2) where s'_p is the erroneous s_p. We also know that the intended signature on M is $m \equiv s^e \bmod n \equiv (s_p xq + s_q yp)^e \bmod n$.
Observe that in the binomial expansion of this, we get $(s_p xq)^e + (s_q yp)^e +$ a number of terms, of which all have at least one p and one q and are all thus divisible by n. Therefore we get that $m \equiv (s_p xq)^e + (s_q yp)^e \bmod n$.
So, we have the following:

$$\begin{aligned}
s' &\equiv s'_p xq + s_q yp \bmod n \\
s'^e &\equiv (s'_p xq)^e + (s_q yp)^e \bmod n \text{ (For the same reason as above)} \\
m &\equiv (s_p xq)^e + (s_q yp)^e \bmod n \text{ (As we just got above)} \\
m - s'^e &\equiv (s_p xq)^e + (s_q yp)^e - (s'_p xq)^e - (s_q yp)^e \bmod n \\
&\equiv x^e q^e (s_p - s'_p)^e \bmod n
\end{aligned}$$

Now, we can find $m - s'^e$ efficiently since it only requires a hash and one repeated square multiply. Now observe that $\gcd(m - s'^e, n) = q$. Thus we have found q, and p can subsequently be found very easily as $p = n/q$.
There are still of course the issues where $x = p$ or $s_p - s'_p = p$. These, however, cannot occur. First $x = p$ is not possible due to the properties of the EEA, and secondly since s_p is computed mod p, if $s_p - s'_p = p$ then that would imply that $s_p = s'_p$ which cannot be true since we are given the fact that $s'_p \neq s_p$.

d) There are a number of ways to prevent such an attack. By directly computing the signature we avoid this particular problem. By verifying the signature before transmitting we gain a lot of extra security. The smart card itself could be designed to better resist this attack. The most feasible and efficient solution, I feel, would be verification before transmitting since it requires only a small, efficient, software change.

4. **Discrete Logarithms**
In this problem, we have $p = 569, g = 256, q = 71$. So, for the algorithm, $m = 9, g^{-m} =$

372 mod p.

For the first step, we compute the table:

j	$g^j \bmod p$
0	1
2	101
3	251
1	256
5	315
6	411
7	520
4	528
8	543

Now, we compute $h(g^{-m})^i$ for successive is until we come across a value in our table. For $i = 3$ we find the value 528, and therefore $j = 4$. $a = mi + j = 9 * 3 + 4 = 31$ and we have found $a = \log_g 327 = 31$.

5. **Poor random number generator in DSA**

Let $k = k_0^b$ be the k used for the first message, and let $r = (g^k \bmod p) \bmod q$ be the r used for the first message. Furthermore, let $k_1, k_2, k_3, r_1, r_2, r_3$ represent the ks and rs used in the three messages. Finally, let s_1, s_2, s_3 be the signatures on M, M', M'', which have SHA-1 hashes of m, m', and m''. Now observe that by the definition of this scheme:

$$\begin{array}{ll} k_1 = k & r_1 = r \\ k_2 = kk_0 & r_2 = r^{k_0} \\ k_3 = kk_0^2 & r_3 = r^{2k_0} \end{array}$$

We know from DSA that $s_1 = k_1^{-1}(m+ar_1)$, which implies that $k_1 = s_1^{-1}(m+ar_1)$. Similarily, we find that $k_2 = s_2^{-1}(m' + ar_2)$, and from above we know that $k_2 = k_1 k_0$. Therefore, we can plug in, and get that $k_0 s_1^{-1}(m + ar_1) = s_2^{-1}(m' + ar_2)$.

If we follow the same logic, we can also find that $k_0 s_2^{-1}(m' + ar_2) = s_3^{-1}(m'' + ar_3)$. Now we can compute m' and m'' since they are simple hashes, and each of r_i are given to us. This gives us two equations, in which the only unkowns are k_0 and a. Since we have two equations and two unknowns, we are able to solve, and thus find a.

This can be done efficiently since this process only involves finding multiplicative inverses (efficient) and solving a system of equations (also efficient).

6. **Elliptic curve computations**

a) $E(\mathbb{Z}_{11}) = \{(0,4), (0,7), (3,4), (3,7), (4,0), (8,4), (8,7), (9,2), (9,9), \infty\}$

b) $\#E(\mathbb{Z}_{11}) = 10$

i) $(0,7) + (3,4)$: First, $\lambda = \frac{4-7}{3-0} = -1$.

$$\begin{aligned} P + Q &= (\lambda^2 - x_1 - x_2, -y_1 + \lambda(x_1 - x_3)) \\ &= (1 - 0 - 3, -7 - (0 - x_3)) \\ &= (-2, -9) \\ &= (9, 2) \end{aligned}$$

ii) $(3,4) + (3,7)$: Immediately notice that $Q = -R$, and so $Q + R = \infty$.

iii) $(3,7)+(3,7)$: First, $\lambda = \frac{3(3)^2+a}{2*7} = \frac{7}{3} = 6$.

$$\begin{aligned} R+R &= (\lambda^2 - 2x_1, -y_1 + \lambda(x_1 - x_3)) \\ &= (3 - 2*3, -7 + 6(3 - x_3)) \\ &= (-3, -6x_3) \\ &= (8,7) \end{aligned}$$

iv) $(4,0)+(4,0)$: Immediately notice that $S=-S$ since $y=0$, so once again $2S = S+S = \infty$