Crypto - HW 5

Hagai Ben Yehuda, ID num: 305237000
Jonathan Bauch, ID num: 204761233

1 1

1.a

Indeed, suppose we have the signature σ_m of $m \in \{1, ..., n\}$, then we can obtain the signature of any $k \in \{1, ..., n\}$ that satisfies $k < m$ by simply setting $\sigma_k = f^{m-k}(\sigma_m)$, then we have that

$$f^k(\sigma_k) = f^k(f^{m-k}(\sigma_m)) = f^m(\sigma_m) = y$$

With the last equality is due to the assumption that σ_m is the correct signature for m. Thus this is not a one time secure signature scheme.

1.b

First we show that f^k is also a permutation, we do so using induction:
For the base case where $k = 1$ this is true from the definition of f. Now assume that f^{k-1} is a permutation, let $x \in \{0,1\}^n$ then x has a source under f (as f is a permutation), namely there exists $y \in \{0,1\}^n$ such that $f(y) = x$, from the induction assumption we know that f^{k-1} is a permutation and thus y has a source $z \in \{0,1\}^n$ such that $f^{k-1}(z) = y$, thus $f(f^{k-1}(z)) = f(y) = x$, thus x has a source under f^k. Since $\{0,1\}^n$ is finite and since we have showed that f^k is on-to, we have that f is also one to one, Hence f is a permutation. Now assume that f^k is not one-way then there is a polynomial time algorithm A that satisfies:

$$\Pr_{x \leftarrow \{0,1\}^n}(A(f^k(x)) = x) > \epsilon$$

We define an algorithm A' that on input $y = f(x)$ calculates $f^{k-1}(y)$ and feeds it to A. A' is polynomial since A and k are polynomial, also note that:

$$\Pr_{x \leftarrow \{0,1\}^n}(A'(f(x)) = x) = \Pr_{x \leftarrow \{0,1\}^n}(A(f^k(x) = x) > \epsilon$$

Leading the a contradiction to the assumption that f is a OWP, hence no such A exists and f^k is also a OWP.

1.c

Indeed, assuming that there is some polynomial algorithm A such that for some $m \in \{1, ...n\}$, $\sigma_m = f^{n-m}(m)$ and $m' > m$, A satisfies: Indeed, assuming that there is some polynomial algorithm A such that for some $m \in \{1, ...n\}$, $\sigma_m = f^{n-m}(m)$ and $m' > m$, A satisfies:

$$\Pr_{x \leftarrow \{0,1\}^n}(A(\sigma_m) = \sigma(m') = f^{n-m'}(x)) > \epsilon$$

We construct a polynomial algorithm A' that inverts f with the same probability, on input $f(w)$ A' will do the following:

- Set $k = m' - m$.
- Set $\sigma_m = f^{k-1}(f(w)) = f^k(w)$.
- Execute $A(\sigma_m)$ and return its result.

Then

$$\begin{aligned}\Pr_{w\leftarrow\{0,1\}^n}(A'(f(w)) = w) &= \Pr_{x\leftarrow\{0,1\}^n}(A'(f^{n-m}(x)) = w)\\ &= \Pr_{x\leftarrow\{0,1\}^n}(A(\sigma_m) = w)\\ &= \Pr_{x\leftarrow\{0,1\}^n}(A(\sigma_m) = f^{-k}(\sigma_m)))\\ &= \Pr_{x\leftarrow\{0,1\}^n}(A(\sigma_m) = \sigma_{m'}) > \epsilon\end{aligned}$$

The first equality is due to the fact that given a random w, the probability for any $x \leftarrow \{0,1\}^n$ to be its k'th source is equal for every x as we assume that f was chosen uniformly form the random permutation functions. This is obviously a contradiction to the assumption that f is a OWF, showing no such algorithm A exists.

1.d

//////////// need to do this!!!!!!!!!! ////////////

2

Let $\tilde{f}$ be some one way function, and define f such that $f(xl) = f(x0)$ (with l being a single bit). Suppose that f is not a one way function, then there is an algorithm A such that

$$\Pr_{x\leftarrow\{0,1\}^n}(A(f(x)) = x) > \epsilon$$

Construct an algorithm A' that on input $\tilde{f}(x)$ executes A to receive yl and returns $y0$. Then we have:

$$\begin{aligned}\Pr_{x\leftarrow\{0,1\}^n}(A'(\tilde{f}(x)) = x) &\geq \frac{1}{2}\Pr_{x\leftarrow\{0,1\}^{n-1}}(A'(\tilde{f}(x0)) = x0)\\ &= \frac{1}{2}\Pr_{x\leftarrow\{0,1\}^n}(A(f(x)) = x)\\ &> \frac{\epsilon}{2}\end{aligned}$$

The equality (in the second line) is correct because A' will produce a valid result whenever A is a able to find the source of a message in $\{0,1\}^n$ because if it finds a source then we know that zeroing the last bit of the result will be the source of the original function. Now that we have constructed f and have shown that it is a one way function, then if we use Lamports scheme with f and have a valid signature $(x_{m_1,1}, ..., x_{m_n,n})$ of the message $(m_1, ..., m_2)$ then we know that if $x_{m_1,1} = xl$ then $(x\bar{l}, ..., x_{m_n,n})$ is also a signature for the same message from the construction of f, since $f(xl) = f(x\bar{l})$, showing that using f Lamports scheme is not a strong one time signature scheme.

3

Assume we are given such an algorithm A that breaks (ϵ, t)-existential-unforgeability of the scheme, we construct A' that breaks $(\frac{\epsilon}{t+1}, 1)$-existential- unforgeability of the underlying one-time scheme as follows:

- Draw $r \in \{1, ..., t\}$ uniformly.
- If $r > 1$
  - Execute A while simulating the oracle for the first $r-1$ messages (generating our own keys).
  - For the $r-1$'th message A asks for, m_{r-1}, sign (m_{r-1}, pk) with pk being the oracle's public key (which we are trying to attack) and send the result to A.
  - For the next message A asks for, m_r, generate the keys (pk_r, sk_r), ask the oracle to sign (pk_r, m_r)
- If $r = 1$
  - Publish that our public key is pk (the public key that belongs to the oracle we are trying to break).
  - For the first message A asks for, m_1 generate keys (pk_2, sk_2) and ask the oracle to sign (pk_2, m_1)
- Continue the process of simulating the oracle normally.
- A returns a trust chain of messages that is different from the one we have provided to it. (We can assume that A's output is of the form $((m_1, pk_2, \sigma_1), ..., (m_n, pk_{n+1}, \sigma_n))$) If the chain A returned differs from the chain we have created in the $r+1$'th index (even if $r = t$ that mean there is an extra message in the chain returned by A), return $(m_{r+1}, pk_{r+2}, \sigma_{r+1})$.

First we note that A' runs roughly in the same time as A since it only runs A, generates keys and queries the oracle once. As stated, note that A' only queries the oracle once so we only need to prove that A' can forge a new message with probability $> \frac{\epsilon}{t+1}$. Note that A is able to forge a signature with probability $> \epsilon$ which mean that with probability $> \epsilon$ at the end of A we will have a valid trust chain which is different form the one we have supplied A with. note that the chain must differ at atleast one place (form the first to one after the last) thus the probability that the chain remains unchanged up to the r'th index and then there is a difference in the $r+1$'th place is $\frac{1}{t+1}$ (because the difference must start somewhere and we have chosen r uniformly) in which case if A indeed forged a valid chain, σ_{r+1} is a signature of (m_{r+1}, pk_{r+2}) signed using sk_r (because that is the private key matching the public key signed by the previous message, which we assume was unchanged) which is the oracles private key and (m_{r+1}, pk_{r+2}) is a previously unsigned message. Thus in this case (which happens with probability $> \frac{\epsilon}{t+1}$) A' breaks the underling signature scheme, showing it is not $(\frac{\epsilon}{t+1}, 1)$-existential- unforgeability as requested.

4

4.a

Indeed, assume that $y \notin QR$ then there are two options:

- If in the first step P sent z such that $z = r^2$ (i.e. z is a quadratic residue) then with probability $\frac{1}{2}$- V will choose $b = 1$ in step 2 in which case there is no a_1 that satisfies $a_1^2 = zy$ because if there was such an a_1 then $y = (a_1 z^{-1})^2$ which contradicts the assumption that $y \notin QR$. Hence with probability at least $\frac{1}{2}$ - V rejects.
- If in the first step P sends z such that z is not a quadratic residue then if P chooses $b = 0$ in the second step then P cannot send r in the third step (because no such r exists by definition, regardless of P's computational limitations). Hence with probability at least $\frac{1}{2}$ - V rejects.

Thus regardless of what P sends in the first step there is a probability of at least $\frac{1}{2}$ that V will reject if $y \notin QR$ (meaning it will accept with probability $\leq \frac{1}{2}$). As requested.

4.b

Indeed, we show a simulator that operates on input (y, b):

- If $b = 0$:
  - Draw $r \in {\mathbb{Z}^*}_N$ and set $\tilde{z} = r^2$
  - Set $\tilde{a_0} = r$.
- If $b = 1$
  - Draw $\tilde{a_1} \in {\mathbb{Z}^*}_N$ uniformly.
  - Set $\tilde{z} = \tilde{a_1}^2 y^{-1} \pmod N$

Then the output of this simulator is Indeed indistinguishable from a true execution of the protocol since if $b = 0$ then we send values that would have made the verifier accept (we send a quadratic residue and then its root) and if $b = 1$ we send $\tilde{z} = \tilde{a_1}^2 y^{-1} \pmod N$ and thus $\tilde{z}y = \tilde{a_1}^2 \pmod N$, hence the verifier would accept. Since by assumption quadratic residues are polynomially indistinguishable form non quadratic residues $\tilde{z}$ is indistinguishable form a uniformly drawn quadratic residue and thus the view is indistinguishable from a successful transaction between the prover and verifier.

5

Let T be a set of t indices in $\{1, \dots, n\}$. $\{f(k)\}_{k \in T}$ are the secret-shares corresponding to T. Let f be a polynomial. Define the partial functions:

$$f_i(x) = y_i \prod_{k \in T \setminus \{i\}} \frac{x - k}{i - k}$$

Then as shown in class:

$$f(x) = \sum_{i \in T} f_i(x)$$

Therefore:

$$\begin{aligned} f(0) &= \sum_{i \in T} f_i(0) \\ &= \sum_{i \in T} y_i \prod_{k \in T \setminus \{i\}} \frac{k}{k - i} \end{aligned}$$

Therefore if we define:

$$b_i = \prod_{k \in T \setminus \{i\}} \frac{k}{k - i}$$

We get:

$$S = f(0) = \sum_{i \in T} b_i y_i$$

Using above formula for $\{b_i\}$, we demonstrate in sage secret reconstruction for various cases as defined in the exercise.

Listing 1: Q5 Code

```
t = 3
n = 6
p = 11

F = Integers(p)
R.<x> = F[]

T1 = {F(1), F(2), F(4)}
T2 = {F(1), F(2), F(5)}

def Q5():
    f, g = get_random_polynomials()
    print 'f:', f
    print 'g:', g
    check(f, T1)
    check(f, T2)
    check(g, T1)
    check(g, T2)

def get_random_polynomials():
    f = R.random_element(degree=(-1, t - 1))
    g = f + randint(1, p - 1)
    return f, g

def check(f, T):
    shares = get_shares(f)
    T_shares = [shares[k - 1] for k in T]
    coefs = get_coefs(T)
    real_secret = f(0)
    calculated_secret = calc_secret(T_shares, coefs)
    print '*' * 20
    print 'shares:', shares
    print 'T_shares:', T_shares
    print 'coefs:', coefs
    print 'real_secret:', real_secret
    print 'calculated_secret:', calculated_secret
    print 'success?', real_secret == calculated_secret

def get_shares(f):
    return [f(i) for i in range(1, n + 1)]

def get_coefs(T):
    return [get_coef(T, i) for i in T]

def get_coef(T, i):
    return prod([k / (k - i) for k in T - {i}])

def calc_secret(shares, coefs):
    return sum([s * c for s, c in zip(shares, coefs)])
```

```
if __name__ == '__main__':
    Q5()
```

Listing 2: Q1 Output

```
f: 3*x^2 + 7*x + 3
g: 3*x^2 + 7*x + 7
*******************
shares: [2, 7, 7, 2, 3, 10]
T shares: [2, 7, 2]
coefs: [10, 9, 4]
real secret: 3
calculated secret: 3
success? True
*******************
shares: [2, 7, 7, 2, 3, 10]
T shares: [2, 7, 3]
coefs: [8, 2, 2]
real secret: 3
calculated secret: 3
success? True
*******************
shares: [6, 0, 0, 6, 7, 3]
T shares: [6, 0, 6]
coefs: [10, 9, 4]
real secret: 7
calculated secret: 7
success? True
*******************
shares: [6, 0, 0, 6, 7, 3]
T shares: [6, 0, 7]
coefs: [8, 2, 2]
real secret: 7
calculated secret: 7
success? True
```

6

6.a

6.b