Problem 1. (a) We have $e_{eve}d_{eve} = 1 \pmod{\varphi(N)}$. Thus, $\varphi(N)$ divides $e_{eve}d_{eve} - 1$.

(b) First, suppose $e_{bob}$ is relatively prime to $V$. Using the extended Euclidean algorithm, Eve can find an integer $d$ such that $d = e_{bob}^{-1} \pmod{V}$, where $V = e_{eve}d_{eve} - 1$ is the multiple of $\varphi(N)$ from part (a). Since $\varphi(N)$ divides $V$, we have that $e_{bob}d = 1 \pmod{\varphi(N)}$, and so 
\[ e^d = (x^e)^d = x^{e_{bob}d} \pmod{\varphi(N)} = x \pmod{V}. \]

Suppose $a = \gcd(e_{bob}, V) \neq 1$. Additionally, write $V = k \cdot \varphi(N)$, where for some integer $k$. Since $\gcd(e_{bob}, \varphi(N)) = 1$, we have that $\gcd(e_{bob}, V) = \gcd(e_{bob}, k)$.

Thus, $V/a = \varphi(N) \cdot k/a$ is still a multiple of $\varphi(N)$. We can replace $V$ with $V/a$, and repeat this process until we obtain a value $V$ where $\gcd(V, e_{bob}) = 1$. Then, we can directly apply the algorithm for the case when $\gcd(V, \varphi(N)) = 1$.

Important Note: It is not always the case that $\gcd(e_{bob}, k/a) = 1$. For a counter-example, take $e_{bob} = 3$ and $k = 9$ (in which case $a = \gcd(e_{bob}, k) = 3$). Note though that we will only repeat this process logarithmically many times (each iteration, we reduce $V$ by a factor of two) before obtaining a multiple of $\varphi(N)$ that is relatively prime to $e_{bob}$.

Problem 2. Let $t = |X|/B$ as in the hint. We describe our procedure for constructing the table $T$. First, we mark all the elements in $X$ as unprocessed. Then, while there exists an unprocessed element $z \in X$, we repeatedly apply $f$ to $z$ until we have a complete cycle $(z, f(z), f^2(z), \ldots, f^n(z))$. In particular, this means that $f^{(n+1)}(z) = z$. The cycle exists because $f$ is a permutation on a finite range. We mark all elements $(z, f(z), \ldots, f^n(z))$ as processed. If $n \geq t$, we add the ordered-tuple $(z, f^t(z), f^{2t}(z), \ldots, f^{(n/t)}(z))$ to $T$. By construction, there are at most $B = |X|/t$ entries in our table.\footnote{This could be off by a factor of two but if instead, we store every 2$t$ elements in each cycle, inversion still runs in $O(t)$ time using a table with at most $B$ entries. For simplicity, we will present the algorithm without this extra caveat.}

Next, we describe how to invert an input $y \in X$ in time $O(t)$. Given an element $y$, we repeatedly apply $f$ to $y$ until we obtain an element $f^{(i)}(y)$ that is contained in $T$, or is equal to $y$. There are two possibilities:

- Suppose $y$ is an element in a cycle with length less than $t$. Then, for some $1 \leq i < t$, $f^{(i)}(y) = y$. The pre-image of $y$ is thus $f^{(i-1)}(y)$, which we have computed with $i - 1 < t$ invocations of $f$.

- Suppose $y$ is an element in a cycle with length at least $t$. In our construction of $T$, we have stored every $t$th element in this cycle. Thus, for some $0 \leq i < t$, $f^{(i)}(y) \in T$. Take the element in $T$ that immediately precede $f^{(i)}(y)$. Denote it $x$. By construction of the table, we have that $f^{(i)}(x) = f^{(i)}(y)$, and so, we have $f^{(t-i-1)}(x)$ is the pre-image of $y$. Thus, we can invert $y$ with exactly $t - 1$ invocations of $f$.

Thus, we have demonstrated that we can invert $f$ on any input using at most $t = O(|X|/B)$ invocations of $f$.

Problem 3. (a) We first show that $g$ and $h$ generate the same subgroup $G$ of order $q$. Briefly, let $\gamma$ be a generator of $\mathbb{Z}_p^*$, and let $g = \gamma^a$ and $h = \gamma^b$. Let $c = \gcd(a, b)$. By the extended Euclidean algorithm, $c = ax + by$
for some $x, y$, so that $\eta = \gamma^x$ is in the subgroup $G$ generated by $g, h$. But by the definition of gcd, $a/c$ and $b/c$ are integers, so that $g = \eta^{a/c}$ and $h = \eta^{b/c}$ are generated by $\eta$. Therefore every element of $G$ is generated by $\eta$. But $\eta$ has order $q$, because $g$ and $h$ do, and so $|G| = q$. We conclude that $g$ and $h$ both generate $G$.

To prove the main claim, let $G$ be the subgroup generated by $g$ and $h$. Since $b = g^x h^y$, $b \in G$. Now, if we fix $b$, then for every $x \in \mathbb{Z}_q$, there is a unique $y \in G$ such that $g^x y = b$, namely $y = bg^{-x}$. Finally, since $h$ generates $G$, this means that there is a unique $r \in \mathbb{Z}_q$ such that $h^r = y$. Thus, we conclude that for a fixed commitment $b$, for every value $x \in \mathbb{Z}_q$, there exists a unique randomizer $r \in \mathbb{Z}_q$ such that $g^x h^r = b$.

Thus, the commitment scheme is perfectly hiding.

(b) Suppose that given $g$ and $h$, Alice can construct a commitment $b$ that she can open as $x$ and also as $x'$. That is, she can produce $x \neq x', r, r'$ such that

$$g^x h^r = b = g^{x'} h^{r'}$$

Dividing through, $g^{x-x'} = h^{r-r'}$. Since $x \neq x'$, $g^{x-x'} \neq 1$ and so $r \neq r'$. Then, since $q$ is prime, we can compute $(r' - r)^{-1} \pmod q$. Raising both sides of the above equation to this power, we have

$$g^{(x-x')(r'-r)^{-1}} = h,$$

so we see that $(x - x')(r' - r)^{-1} \in \mathbb{Z}_q$ is the discrete log of $h$ base $g$.

**Problem 4.** (a) Let $(x_1, y_1)$ and $(x_2, y_2)$ be the collision given by $A$ on inputs $u, v$. Thus, $x_1^u \cdot y_1^v = x_2^u \cdot y_2^v \pmod n$. Or equivalently, $(x_1/x_2)^u = y_1^v \cdot y_2^{-v} \pmod n$. Note that $1/x_2$ is well defined as $x_2 \in \mathbb{Z}_n^*$. As $0 \leq y_1, y_2 < e$, it implies $|y_1 - y_2| < e$. Next, if $y_1 = y_2$, then we get $x_1 = x_2 \pmod n$, which implies $x_1 = x_2$. The last fact follows from the fact that the RSA function with exponent $e$, RSA$(x) = x^e \pmod n$ is a permutation. If $y_1 = y_2$, then $x_1 = x_2$ and the tuples given are not a valid collision. Thus $y_1 \neq y_2$, or equivalently, $|y_1 - y_2| > 0$. Thus, algorithm $B$ simply runs algorithm $A$ on $u, n$ and then outputs $a = x_1 x_2^{-1} \pmod n$ and $b = y_1 - y_2$.

(b) Using the hint, consider $h, s, e, t$ such that $hs + et = 1$. Consider $\alpha = a^s u^t$. We get $a^b = a^s u^{bt}$. However, $u^b = a^e$, which implies $u^{bt} = a^{et}$. Thus $a^b = a^{et} = a^1 = a$. Hence $\alpha = a^{1/b}$. Note that all operations are performed in $\mathbb{Z}_n$.

(c) One possible collision is $(u, e)$ and $(u^2, 0)$. $H(u, e) = u^e \cdot u^e = u^{2e} = (u^2)^e \cdot u^0 = H(u^2, 0)$.

**Problem 5.** (a) With overwhelming probability, $x \neq 0$, so it suffices to just consider cases where $x \neq 0$. The signature is computed as

$$s = (y - m) \cdot x^{-1} \pmod q$$

where $x^{-1} \in \mathbb{Z}_q$. To verify that this works, observe that

$$g^m h^s = g^m h^{(y-m)x^{-1}} = g^m g^{(y-m)x^{-1}} = g^m g^{y-m} = g^y = u.$$
5. If $\mathcal{A}$ outputs a forgery $(m^*, s^*)$, $\mathcal{B}$ computes and outputs $x = (m - m^*)(s^* - s)^{-1} \in \mathbb{Z}_q$.

We claim that $\mathcal{B}$ solves the discrete log in $\mathbb{G}$ with advantage $\varepsilon$. First, we argue that $\mathcal{B}$ correctly simulates the view for adversary $\mathcal{A}$. In the real scheme, the components $h, u$ in the public key $(g, h, u)$ are uniform and independent in $\mathbb{G}$. This is also the case in the public key $\mathcal{B}$ constructs for $\mathcal{A}$: the challenge $h$ from the discrete log challenger is uniform in $\mathbb{G}$ and the value $s$ is chosen uniformly and independently at random from $\mathbb{Z}_q$, so $g^{m^*}h^s$ is uniform in $\mathbb{G}$ since $h$ is a generator of $\mathbb{G}$. By construction, the signature $s$ that the adversary receives is properly distributed (for each message $m$, there is only one valid signature $s$). Thus the view $\mathcal{B}$ simulates for $\mathcal{A}$ is identically distributed as the view $\mathcal{A}$ expects when interacting with a challenger for the weak one-time signature game. Thus, with probability $\varepsilon$, $(m^*, s^*)$ is a valid forgery, which means that $m \neq m^* \in \mathbb{Z}_q$ and moreover,

$$g^{m^*}h^s = u = g^{m^*}h^{s^*}.$$  

Since $m \neq m^* \in \mathbb{Z}_q$, it follows that $s \neq s^* \in \mathbb{Z}_q$. But then, as in Problem 3, we have that

$$g^{(m-m^*)(s^*-s)^{-1}} = h,$$

or equivalently, $(m - m^*)(s^* - s)^{-1}$ is the discrete log of $h$ base $g$. We conclude that $\mathcal{B}$ wins the discrete log game in $\mathbb{G}$ with advantage $\varepsilon$.

(c) Given two message/signature pairs $(m_0, s_0), (m_1, s_1)$ with $m_0 \neq m_0$, compute $x = (m_0 - m_1)(s_1 - s_0)^{-1} \in \mathbb{Z}_q$. A similar calculation as in part (b) shows that $g^x = h$. Then compute $y = m_0 + s_0x$. Notice that $g^y = g^{m_0}(g^x)^{s_0} = u$. Therefore, we have recovered the secret key $(x, y)$. Thus, we can now sign arbitrary messages of our choosing.

(d) For the extra credit, let $\mathcal{A}$ be an adversary for the one-time signature scheme. Let $\varepsilon$ be the advantage of $\mathcal{A}$. We construct the following adversary $\mathcal{B}$ for solving the discrete log in $\mathbb{G}$:

1. The discrete log challenger sends $\mathcal{B}$ a tuple $(g, h)$ where $h$ is random in $\mathbb{G}$.

2. $\mathcal{B}$ chooses random exponents $a_0, a_1, b_0, b_1, c_0, c_1 \overset{\$}{\leftarrow} \mathbb{Z}_q$. It then computes $h_0 \leftarrow g^{a_0}h^{a_1}$, $h_1 \leftarrow g^{b_0}h^{b_1}$, and $u \leftarrow g^{c_0}h^{c_1}$.

3. $\mathcal{B}$ starts running $\mathcal{A}$ and gives it the public key $(g, h_0, h_1, u)$.

4. When $\mathcal{A}$ issues a signing request for a message $m$, $\mathcal{B}$ computes values $s_0, s_1 \in \mathbb{Z}_q$ such that $c_0 = m + a_0s_0 + b_0s_1$ and $c_1 = a_1s_0 + b_1s_1$. Adversary $\mathcal{B}$ sends the signature $(s_0, s_1)$ to $\mathcal{A}$.

5. If adversary $\mathcal{A}$ outputs a forgery $(m^*, s^*)$ for some message $m \neq m^*$, then $\mathcal{B}$ outputs the value of $x \in \mathbb{Z}_q$ that satisfies

$$m^* - m = a_0(s_0^* - s_0) + b_0(s_1^* - s_1) = x[a_1(s_0^* - s_0^*) + b_1(s_1^* - s_1^*)]$$

We claim that $\mathcal{B}$ simulates the correct view for adversary $\mathcal{A}$. First, since the exponents $a_0, a_1, b_0, b_1, c_0, c_1$ are all chosen uniformly and independently in $\mathbb{Z}_q$, the public key is properly distributed. Next, we note that $(s_0, s_1)$ is a valid signature on the adversary’s message $m$:

$$g^{m^*}h_0^{s_0}h_1^{s_1} = g^{m+a_0s_0+b_0s_1}h_0^{a_1s_0+b_1s_1} = g^{c_0}h^{c_1}.$$  

Moreover, the values $s_0, s_1$ exist with overwhelming probability since the exponents $a_0, a_1, b_0, b_1, c_0, c_1$ are drawn uniformly at random from $\mathbb{Z}_q$ (and can be computed efficiently since we are solving a linear system in $\mathbb{Z}_q$). Finally, the fact that $s_0$ is uniform in $\mathbb{Z}_q$ follows from the fact that $c_0, c_1$ are uniformly drawn from $\mathbb{Z}_q$. We conclude that $\mathcal{B}$ properly simulates the view for $\mathcal{A}$, and so, with probability $\varepsilon$, $\mathcal{A}$ outputs a pair $(m^*, s^*)$ such that $m \neq m^*$ and $s^* = (s_0^*, s_1^*)$ is a valid signature on $m^*$. By definition then, $g^{m^*}h_0^{s_0^*}h_1^{s_1^*} = g^{m}h_0^{s_0}h_1^{s_1}$.

Writing $h = g^x$, and using the fact that $g$ is a generator of $\mathbb{Z}_q$, this means that

$$m^* - m = a_0(s_0^* - s_0) + b_0(s_1^* - s_1) = x[a_1(s_0^* - s_0^*) + b_1(s_1^* - s_1^*)] \pmod{q}.$$  

3
As long as \( a_1(s_0 - s_0^*) + b_1(s_1 - s_1^*) \neq 0 \) and \( a_0(s_0^* - s_0) + b_0(s_1^* - s_1) \neq m - m^* \), then \( \mathcal{B} \) can solve for \( x \in \mathbb{Z}_q \) efficiently, and thus win the discrete log game. We show that this holds with overwhelming probability. Consider the probability that \( a_1(s_0 - s_0^*) + b_1(s_1 - s_1^*) = 0 \). First, it cannot be the case that \( s_0 = s_0^* \) and \( s_1 = s_1^* \) and \((s_0^*, s_1^*)\) remains a valid signature on \( m^* \neq m \). Therefore, at least one of \( s_0 - s_0^* \) and \( s_1 - s_1^* \) is non-zero. Taken over the randomness in \( a_1, b_1 \), with overwhelming probability \( a_1(s_0 - s_0^*) + b_1(s_1 - s_1^*) \neq 0 \) (note that this holds even conditioned on the adversary’s knowledge of the public key \((g,h_0,h_1,u)\) and the tuple \((m,s_0,s_1)\)). A similar argument shows that with overwhelming probability \( a_0(s_0^* - s_0) + b_0(s_1^* - s_1) \neq m - m^* \), and so we conclude that with probability negligibly close to \( \varepsilon \), adversary \( \mathcal{B} \) wins the discrete log game in \( G \).

**Problem 6.** Given an 80-byte (640 bit) prefix \( B \), we show how to construct a 256-byte (2048 bit) value \( B^* \) such that the first 80 bytes of \( B^* \) is equal to \( B \), and where \( B^* \) is a perfect cube over the integers. Using the procedure described in the problem, this enables an adversary to forge signatures on arbitrary messages of its choosing. We construct the value \( B^* \) as follows:

1. Let \( y = 2^{1408}B + 2^{1408} - 1 \). Since \( B \) is a 640-bit value, \( y \) is a 2048-bit value whose first 640 bits are equal to \( B \) and whose last 1408 bits are all 1’s. In other words, \( y \) is the largest 2048-bit value whose prefix is \( B \).

2. Let \( x = \lfloor 3\sqrt[3]{y} \rfloor \) be the floor of the cube root of \( y \) (computed over the real numbers). Note that this can be computed efficiently (for instance, using binary search).

3. Output \( B^* = x^3 \).

By construction, \( B^* \) is a perfect cube over the integers. It suffices to show that the first 640-bits of \( B^* \) equals \( B \). This is equivalent to showing that \( 2^{1408}B \leq B^* \leq y \), since all values in the interval \([2^{1408}B, y]\) share the same 640 most-significant bits. By construction, \( x^3 \leq y < (x + 1)^3 \). We can give a crude upper bound on the difference between \((x + 1)^3\) and \( x^3 \):

\[
(x + 1)^3 - x^3 = 3x^2 + 3x + 1 \leq 8x^2 \leq 8y^{2/3} \leq 2^{3+1366} = 2^{1369}.
\]

Since \( x^3 \leq y < (x + 1)^3 \), it follows that \( y - x^3 < (x + 1)^3 - x^3 \leq 2^{1369} < 2^{1408} \). We conclude that \( 2^{1408}B \leq x^3 \leq y \). Since \( B^* = x^3 \), the claim follows.