Problem 1: Fun with Hash Functions

a. We wish to construct an $H_a$ that is collision-resistant such that $H'_a$ (which drops the LSB of $H_a$) is not.

First, we fix two distinct strings $x_1, x_2$. Define $h_1 h_2 \ldots h_k := H(x_1)$, where $h_i$ is the $i$th bit of $H(x_1)$. We will define our first attempt of a hash function

$$H'_a(x) = \begin{cases} h_1 \circ_h 2 \circ \cdots \circ h_{k-1} \circ \overline{h_k}, & \text{if } x = x_2 \\ H(x) & \text{otherwise.} \end{cases}$$

In other words, $H'_a$ outputs $H(x_1)$ with its last bit flipped on input $x_2$; otherwise, it simply outputs $H(x)$.

We can see that this is a step in the right direction: $H'_a$ seems intuitively collision-resistant, since we are only changing the output at one point, and it is true that if we drop the LSB then we can find a collision at $x_1, x_2$. But we find that there is a problem – while it isn’t obvious how to find a collision for $H'_a$, it is difficult to prove that we cannot. If our adversary returns a collision $(m_1, x_2)$ for $H'_a$, then we do not know how to find a collision for $H$; since $H'_a(x_2)$ is hardcoded and does not tell us anything about a particular input to $H$. This can break our entire reduction: the adversary knows $x_2$, so it is perfectly allowed to focus exclusively on finding a collision there, and thus $H'_a$ may not be collision-resistant even if $H$ is.

So let us try again. Define

$$H_a(x) = \begin{cases} h_1 \circ_h 2 \circ \cdots \circ h_{k-1} \circ \overline{h_k}, & \text{if } x = x_2 \\ H(x) & \text{if } H(x) \neq H_a(x_2) \\ H(x_2) & \text{if } H(x) = H_a(x_2) \end{cases}$$

In other words, $H_a$ outputs $H(x_1)$ with its last bit flipped on input $x_2$. Otherwise, on input $x$, $H_a$ calculates $H(x)$ and checks if it is equal to $H_a(x_2)$: if it is, it outputs $H(x_2)$, otherwise it outputs $H(x)$.

Assume for the sake of contradiction that $H_a$ is not collision-resistant, i.e. there exists an adversary $A$ which can find a collision with non-negligible probability. Let us define an adversary $B$ which finds a collision for $H$ with non-negligible probability.

Note first that, by construction, $H_a$ outputs $H_a(x_2)$ at exactly one input, so there does not exist a collision for $H_a$ where one of the inputs is $x_2$. Let $B$ run $A$, finding $a \neq b$ such that $H_a(a) = H_a(b)$. There are then three cases:

Case 1 $H_a(a) = H_a(b) = H(x_2)$ and $H(a) = H(b) = H_a(x_2)$. $B$ outputs $(a, b)$.

Case 2 $H_a(a) = H_a(b) = H(x_2)$ and WLOG $H(a) \neq H_a(x_2)$. Then by construction, $H_a(a) = H(a)$, so $B$ outputs $(a, x_2)$.

Case 3 $H_a(a) \neq H(x_2)$ and $H_a(b) \neq H(x_2)$. Then we know that $H_a(a) = H(a)$ and $H_a(b) = b$, so $B$ outputs $(a, b)$.
b. Assume for the sake of contradiction that one of $H_1, H_2$ is collision-resistant (WLOG, $H_1$), but $H_b = (H_1(m), H_2(m))$ is not collision resistant. Since $H_b$ is not collision-resistant, there exists an adversary $A$ that can find a collision with non-negligible probability. Let us define an adversary $B$ that finds a collision in $H_1$ with that same probability.

$B$ runs $A$, getting a collision $m_1 \neq m_2$ such that $H_b(m_1) = (x, y) = H_b(m_2)$. By definition of $H_b$,

$$x = H_1(m_1) = H_2(m_2)$$

$$y = H_2(m_1) = H_2(m_2),$$

so $B$ outputs $(m_1, m_2)$ as well, and is correct just as often as $A$. Thus, $H_b$ is collision resistant when either of $H_1, H_2$ is.

c. Let $H_1$ be a CRHF but $H_2$ be the constant function $H_2(x) = 0^k$. Then $H_c$ is not collision resistant, as it is the constant function $H_c(x) = H_1(0^k)$. If we switch the definitions of $H_1, H_2$, we get that $H_c(x) = 0^k$. So even if one of $H_1, H_2$ is a CRHF, there exist choices for the other function that make $H_c$ not a CRHF.

d. Proof. Assume for the sake of contradiction that $F_d$ is not a PRF. In other words, there exists an adversary $A$ which can distinguish between $F_d$ and a random function with non-negligible probability $\epsilon(k)$. Let us define an adversary $B$ which can distinguish between $F$ and a random $R$ with non-negligible probability.

Our adversary $B$ is given access to oracle $O_B$, which is either $F(k, \cdot)$ or a truly random function $R$. $B(1^k)$ runs $A(1^k)$. If $A$ queries $x_i$, then $B$ responds with $O_B(H(x_i))$. $B$ then outputs whatever $A$ outputs.

If $O_B$ was $F$, then $B$ will act just like $F_d$ on $A$’s oracle queries, so $B$ will be correct with the same probability as $A$. We now need to show that, if $O_B$ is $R : \{0, 1\}^k \mapsto \{0, 1\}^k$, then $A$ will act as if it was given oracle access to a random function $\{0, 1\}^* \mapsto \{0, 1\}^k$. Since $H$ is collision-resistant, we know that $A$ will only select $x_i \neq x_j$ such that $H(x_i) = H(x_j)$ with negligible probability. Since $R$ is random, $A$ gains no information about $H(x_i)$ from $R(H(x_i))$, so $A$ only has an additional advantage when it does happen to find a collision for $H$.

Thus, $B$ is correct with probability $\epsilon(k) - \nu(k)$ for negligible probability $\nu$, and thus $F$ is not a PRF. But this contradicts our definition of $F$, so $F_d$ must be a PRF.

**Problem 2: Broken Signatures**

a. We want to show that if factoring is hard, it is also hard to make the scheme target-message forgeable using a public-key only attack. We are, of course, going to prove this using the contrapositive: if there exists an adversary $A$ who can forge a signature for a target message with a PK-only attack, we can construct a $B$ that factors $n$ with non-negligible probability. Assume that we have some $A$ such that

$$\Pr [PK \leftarrow Gen(1^k); m \leftarrow QR_n \cup QNR_n; b \leftarrow A(PK, m) : \text{Verify}(m, b) = \text{True}] = \epsilon(k)$$

for some non-negligible $\epsilon(\cdot)$. Our algorithm $B$ picks some random $r$ in $\mathbb{Z}_n^*$ and computes $x \equiv r^2 \pmod{n}$. With probability 1/2, it sets $x := -x$ (so that it sometimes gives $A$ inputs from $QNR_n$). $B$ then gives $x$ to $A$ (along with the PK, which in this case is just $n$), and takes the $b$ that $A$ returns. If $b^2 \equiv \pm x \pmod{n}$ but $b \neq \pm r \pmod{n}$, we have seen that $B$ can factor $n$ (see next paragraph for a reminder how). Otherwise, $B$ outputs “fail.”
If we have \( b^2 \equiv r^2 \pmod{n} \), we know that \( b^2 - r^2 = kn \) for \( k \in \mathbb{Z} \). Then \((b - r)(b + r) = kn\), which is the same as saying that \( n \mid (b - r)(b + r) \). Because \( b \neq r \pmod{n} \) we know that \( n \nmid b - r \), and because \( b \neq -r \pmod{n} \), we know that \( n \nmid b + r \). Then there must exist some non-trivial divisor of \( n \), call it \( p \), such that \( p \mid (b + r) \) and \( p \mid n \). So \( p = \gcd(n, b + r) \), and once we have this value we have \( q \) and \( n \) is completely factored. So our algorithm works to factor \( n \) because it is sufficient to find two square roots with this property.

Since \( \mathcal{A} \) is outputting a random square root of \( \pm x \) and there are four squares roots modulo \( n \), \( \mathcal{B} \) has an \( \epsilon/2 \) probability of getting something useful, which is non-negligible and so we are done.

b. For this attack, our adversary should pick the signature \( \sigma \) first. Then, it can simply compute \( \sigma^2 \pmod{n} \) to get something that is in the message space (in fact, specifically in \( QR_n \)), and has \( \sigma \) as its corresponding signature.

c. Our adversary first picks a random \( x \) in \( \mathbb{Z}_n^* \), then computes \( m = x^2 \) and gives this to the signer.

We get back some square root \( \sigma \) and check to see if \( \sigma \neq \pm x \pmod{n} \). Since there are four square roots of \( m \) modulo \( n \), we have probability \( \frac{1}{2} \) of this condition being met. As we have seen in Part A, once we have two elements \( \sigma \) and \( x \) such that \( \sigma^2 \equiv x^2 \pmod{n} \) but \( \sigma \neq \pm x \pmod{n} \), we can find a non-trivial divisor of \( n \) and therefore factor it completely.

Note that, while in part a., we had

\[
\begin{align*}
g^r_1 g^r_2 &= g^{r_1} g^{r_2} \pmod{q} \\
g^{m - m_i} &= g^{r_i - r} \pmod{q} \\
g^{2(m - m_i)} &= g^{2(r_i - r)} \pmod{q} \\
g^{2(2m - m_i)} &= g^{2(2r_i - 2r)} \pmod{q} \\
2(m - m_i) &\equiv 2\alpha(r_i - r) \pmod{\phi(q)}
\end{align*}
\]

Note that, because \( q \) is a safe prime, \( \phi(q) = q - 1 = 2p \) for some prime \( p \). This means that:

\[
\begin{align*}
2(m - m_i) &\equiv 2\alpha(r_i - r) \pmod{2p} \\
2(m - m_i) &= 2\alpha(r_i - r) + 2kp \\
(m - m_i) &= \alpha(r_i - r) + kp \\
(m - m_i) &\equiv \alpha(r_i - r) \pmod{p}
\end{align*}
\]

This means that we can calculate

\[
\alpha \equiv (m - m_i)(r_i - r)^{-1} \pmod{p},
\]

**Problem 3: GHR signature**

a. Here, we need to construct an adversary \( \mathcal{B} \) that is given a prime modulus \( q \), \( g \) and \( y \) such that \( g^\alpha \equiv y \pmod{q} \) for some \( \alpha \). \( \mathcal{B} \) must be able to find \( \alpha \) with some non-negligible probability. To give input to \( \mathcal{A} \), \( \mathcal{B} \) needs to be able to set up a public key \( PK \).

To do this, \( \mathcal{B} \) picks random \( k \)-bit primes \( p_1 \) and \( p_2 \) and sets \( n = p_1 p_2 \); then, \( \mathcal{B} \) picks \( s \leftarrow \mathbb{Z}_n^* \) and \( pk \) to be some key for a hash function. Finally, \( \mathcal{B} \) then sets \( g_1 = g^2 \) and \( g_2 = g^{y^2} \) and sends \( \mathcal{A} \) the public key \( PK = (n, s, q, pk, g_1, g_2) \). Since \( \mathcal{B} \) knows the factorization of \( n \), \( \mathcal{B} \) is able to answer \( \mathcal{A} \)'s signing queries just as the real signer would. So they follow the protocol completely normally for messages \( m_1 \) through \( m_\ell \). \( \mathcal{B} \) will also store each of the \( \ell \) queries and \( \ell \) responses \( (m_i, (\sigma_i, r_i)) \).

With non-negligible probability \( \epsilon(k) \), \( \mathcal{A} \) will produce a successful forgery \((m^*, (\sigma^*, r^*))\) such that \( g_1^{m^*} g_2^{r^*} = g_1^{m_i} g_2^{r_i} \) for some \( i \). \( \mathcal{B} \) can then search through the stored table of queries and find the \( i \) that satisfies that equation. After finding \( i \), \( \mathcal{B} \) knows that:

\[
\begin{align*}
g_1^{m^*} g_2^{r^*} &\equiv g_1^{m_i} g_2^{r_i} \pmod{q} \\
g_1^{m^* - m_i} &\equiv g_2^{r_i - r^*} \pmod{q} \\
g^{2(m^* - m_i)} &\equiv g^{2(r_i - r^*)} \pmod{q} \\
g^{2(2m^* - m_i)} &\equiv g^{2(2r_i - 2r^*)} \pmod{q} \\
2(m^* - m_i) &\equiv 2\alpha(r_i - r) \pmod{\phi(q)}
\end{align*}
\]

HW 9-3
because \( r_i \neq r^* \) (since, if they were equal mod \( p \), then \( m_i = m^* \)), and so \( r_i - r^* \) has an inverse mod \( p \). Therefore, \( B \) can correctly compute \( \alpha' = \alpha \pmod{p} \). All \( B \) needs to do now is figure out the value of \( \alpha \pmod{2p} \).

We know that either

\[
\alpha \equiv \alpha' \pmod{2p}, \quad \text{or} \quad \alpha \equiv p + \alpha' \pmod{2p}.
\]

We can then test both of these numbers to figure out which one is the discrete log of \( y \) base \( g \), by computing \( g^{\alpha'} \) and \( g^{p+\alpha'} \) and comparing both values to \( y \). We then output the correct value.

This means that \( B \) succeeds with the same probability that \( A \) does, and so (in this case) \( B \) also has non-negligible probability \( \epsilon(k) \).

b. Now, \( B \) gets a value \( pk \) and needs to find \( x \) and \( x' \) such that \( x \neq x' \) and \( h_{pk}(x) = h_{pk}(x') \) for the hash function corresponding to the key\( pk \). Here, our \( B \) picks \( p_1, p_2, q \) as safe primes, and sets \( n = p_1 p_2 \). \( B \) then picks \( g_1 \) and \( g_2 \) as random generators for \( QR_q \), with \( g_1 \neq g_2 \). Finally, \( B \) publishes \( PK = (n, s, q, pk, g_1, g_2) \). Again, since \( B \) knows the factorization of \( n \), it can respond to signature queries normally.

\( B \) will (after answering all signature queries) get back a successful forgery \((m^*, (\sigma^*, r^*))\) such that \( g_1^m g_2^{r^*} \neq g_1^m g_2^{r^*} \), but \( H_{pk}(m^*) = H_{pk}(m_i) \) for some \( i \). We simply set

\[
x = g_1^m g_2^{r^*}, \quad \text{and} \quad x' = g_1^m g_2^{r^*},
\]

and we have found a collision in \( h_{pk} \). Thus, \( B \) succeeds whenever \( A \) succeeds.

c. Say that this signature scheme is existentially forgeable. This means that there exists a PPT adversary \( A \) that can, with non-negligible probability, output a successful forgery \((m^*, (\sigma^*, r^*))\) after querying a signing oracle a polynomial number of times on messages \( m_1, \ldots, m_\ell \), where \( m^* \neq m_i \) for all \( i \).

How can this happen? This can be divided into three cases:

**Case 1** The verifier accepts the signature, and for all \( i \in \{1, \ldots, \ell\} \), \( h_{pk}(g_1^{m^*} g_2^{r^*}) \neq h_{pk}(g_1^{m_i} g_2^{r_i}) \).

But as seen in class, this would break the RSA assumption.

**Case 2** There exists some \( i \) such that \( h_{pk}(g_1^{m^*} g_2^{r^*}) = h_{pk}(g_1^{m_i} g_2^{r_i}) \), but \( g_1^{m^*} g_2^{r^*} \neq g_1^{m_i} g_2^{r_i} \). Then, as proved in part b, this would break the collision-resistance of \( h \).

**Case 3** There exists some \( i \) such that \( g_1^{m^*} g_2^{r^*} = g_1^{m_i} g_2^{r_i} \) (and therefore \( h_{pk}(g_1^{m^*} g_2^{r^*}) = h_{pk}(g_1^{m_i} g_2^{r_i}) \)).

Then because we know that \( m^* \neq m_i \), part a shows that this case would mean that the discrete logarithm assumption would not hold.

If the signature scheme is existentially forgeable, then one of these cases must occur with non-negligible probability, so one of our three assumptions must not hold. Therefore, we have shown that if the discrete log assumption, the strong RSA assumption, and \( h \)'s collision resistance hold, then the signature scheme is not existentially forgeable, which is what we wanted.

**Problem 4: Digital Signatures**

Let \( KeyGen \) be the key generation algorithm of a digital signature scheme. WLOG, assume that \( KeyGen(1^k) \) uses at most \( k \) random bits. Let \( KeyGen_r \) denote running \( KeyGen \) with randomness \( r \). Note that \( KeyGen_r \) is then a deterministic algorithm.
Define a function \( f \) as follows: On input \( r \) of length \( k \), run \( \text{KeyGen}_r(1^k) \) to obtain \((pk, sk)\), and output \( pk \). We will show that if \( f \) is not one-way, then we can construct an algorithm that takes \( pk \) as input and outputs \( sk \) such that \((pk, sk)\) is in the range of \( \text{KeyGen} \), which contradicts the security of the digital signature scheme.

**Proof.** If \( f \) is not one-way, then there exists a PPT \( A \) and a non-negligible \( \epsilon \) such that

\[
\Pr[r \leftarrow \{0, 1\}^k; r' \leftarrow A(1^k, f(r)) : f(r') = f(r)] = \epsilon(k).
\]

Define a PPT \( B \) as follows: On input \((1^k, pk)\), run \( A(1^k, pk) \) to obtain \( r' \). Run \( \text{KeyGen}_r(1^k) \) to obtain \((pk', sk')\). Output \( sk' \).

Let us analyze \( B \)'s probability of success. By our problem statement, we have that

\[
\Pr[\text{Success}] = \Pr[(pk, sk) \leftarrow \text{KeyGen}(1^k); sk' \leftarrow B(1^k, pk) : (pk, sk') \in \text{Range}(\text{KeyGen}(1^k))].
\]

Then by our construction of \( B \), we can expand this to

\[
\Pr[(pk, sk) \leftarrow \text{KeyGen}(1^k); r' \leftarrow A(1^k, pk); (pk', sk') \leftarrow \text{KeyGen}_r(1^k) : (pk, sk') \in \text{Range}(\text{KeyGen}(1^k))].
\]

Since \( \text{KeyGen} \) uses each \( k \)-bit string as its randomness with uniform probability, we can choose and fix that randomness without altering the behavior of the algorithm.

\[
\Pr[\text{Success}] = \Pr[r \leftarrow \{0, 1\}^k; (pk, sk) \leftarrow \text{KeyGen}_r(1^k); r' \leftarrow A(1^k, pk); (pk', sk') \leftarrow \text{KeyGen}_r(1^k) : (pk, sk') \in \text{Range}(\text{KeyGen}(1^k))].
\]

But the usage of \( \text{KeyGen}_r \) in this expression is precisely our definition of \( f \)! Thus, we simplify \( B \)'s probability of success to

\[
\Pr[r \leftarrow \{0, 1\}^k; r' \leftarrow A(1^k, f(r)); (pk', sk') \leftarrow \text{KeyGen}_r(1^k) : (pk, sk') \in \text{Range}(\text{KeyGen}(1^k))].
\]

It is difficult to discuss the range of a general algorithm directly, so let us lower bound our probability in order to simplify.

\[
\Pr[\text{Success}] \geq \Pr[r \leftarrow \{0, 1\}^k; r' \leftarrow A(1^k, f(r)); (pk', sk') \leftarrow \text{KeyGen}_r(1^k) : (pk, sk') \in \text{Range}(\text{KeyGen}(1^k)) \land pk = pk'].
\]

Since \( pk = pk' \) entails that \((pk, sk') \in \text{Range}(\text{KeyGen}(1^k))\), we can reduce this to

\[
\Pr[\text{Success}] \geq \Pr[r \leftarrow \{0, 1\}^k; r' \leftarrow A(1^k, f(r)); (pk', sk') \leftarrow \text{KeyGen}_r(1^k) : pk = pk'].
\]

We once again notice that \( f \) is being used here implicitly, and find

\[
\Pr[\text{Success}] \geq \Pr[r \leftarrow \{0, 1\}^k; r' \leftarrow A(1^k, f(r)) : f(r) = f(r')] \geq \epsilon(k).
\]

Thus, \( B \) succeeds with non-negligible probability at recovering a viable secret key from the public key, and the digital signature scheme is not secure. \( \square \)