CS/ECE407 Cryptography – Homework 4 Solutions
Hash Functions and Public Key Cryptography

Due: Thursday, April 16, 3:30PM CT

Remember, you are free to collaborate with up to one classmate. See the course webpage for more details. You are expected to write out and submit your own solutions! Your collaboration is for discussing problems at a high level, not plagiarizing answers.

Your solutions should be typed or carefully handwritten. We provide a LaTeX template for this document, if you would like to use it as a starting point. **If we cannot read your handwritten answers, they will not receive credit.**

Typed solutions should be submitted through Gradescope (see course webpage). Hand-written solutions should be scanned and turned in through Gradescope.

Definition 1 (Collision Resistance)**.** A hash function H with seeds from S is **collision resistant** if for any polytime adversary $\mathcal{A}$, the following probability is negligible:

$$\Pr\left[H(s, x_0) = H(s, x_1) \;\middle|\; \begin{array}{l} s \leftarrow S(1^\lambda) \\ (x_0, x_1) \leftarrow \mathcal{A}(1^\lambda, s) \end{array} \right] < \text{negl}(\lambda)$$

Definition 2 (Discrete Logarithm Hardness)**.** Let $\mathbb{G}_\lambda$ denote a family of cyclic groups with respective generators g_λ. We say that the discrete logarithm problem for this family is hard if for any polytime adversary $\mathcal{A}$, the following probability is negligible:

$$\Pr\left[x = y \;\middle|\; \begin{array}{l} q = |\mathbb{G}_\lambda| \\ x \leftarrow \mathbb{Z}_q \\ y \leftarrow \mathcal{A}(1^\lambda, g_\lambda^x) \end{array} \right] < \text{negl}(\lambda)$$

Informally, we sample a random exponent x, give g^x to the adversary, and the adversary wins if it outputs x.

Problem 1 (Hash Functions)**.** Let $h : \{0,1\}^\lambda \times \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ be a collision resistant hash function whose seeds are uniformly sampled from $\{0,1\}^\lambda$. Consider the following function $H : \{0,1\}^\lambda \times \{0,1\}^{3\lambda} \to \{0,1\}^\lambda$ which splits input x into three length-λ strings, then calls h twice:

$$\begin{array}{l} H(s, x): \\ \quad (x_0, x_1, x_2) = \mathsf{thirds}(x) \\ \quad \mathsf{return}\ h(s, h(s, x_0 \mid x_1) \mid x_2) \end{array}$$

1. **(2 points)** H is also collision resistant. Give a security reduction that proves this.
2. **(2 points)** H is well-defined only for inputs x of length exactly 3λ. Use h to construct a hash function H' that is well-defined for any input x that has length *strictly less* than 3λ. Give a security reduction that proves your H' is collision resistant.
3. **(2 points)** Now, use h to construct a collision-resistant function H'' that is well-defined for inputs x of *any* polynomial length. Give one or two sentences that informally argument why your H'' is collision resistant; you do not need to prove this formally.

Solution 1.

1. Let's start by recalling what it means for H to be collision resistant:

$$\Pr\left[x \neq y \text{ and } H(s, x) = H(s, y) \;\middle|\; \begin{array}{l} s \leftarrow S(1^\lambda) \\ (x, y) \leftarrow \mathcal{A}(1^\lambda, s) \end{array} \right] < \text{negl}(\lambda)$$

We can inline the definition of H inside the probability statement:

$$\Pr\left[\begin{array}{l} x \neq y \text{ and } h(s, h(s, x_0 \mid x_1) \mid x_2) = h(s, h(s, y_0 \mid y_1) \mid y_2) \end{array} \;\middle|\; \begin{array}{l} s \leftarrow S(1^\lambda) \\ (x, y) \leftarrow \mathcal{A}(1^\lambda, s) \\ (x_0, x_1, x_2) = \mathsf{thirds}(x) \\ (y_0, y_1, y_2) = \mathsf{thirds}(y) \end{array}\right]$$

For convenience, let's give names to some intermediate variables:

$$X_L = h(s, x_0 \mid x_1) \qquad Y_L = h(s, y_0 \mid y_1) \qquad X = X_L \mid x_2 \qquad Y = Y_L \mid y_2$$

Now, our proof will proceed by case analysis on the relationship between X and Y. We will show that now matter how X and Y are related, the adversary has negligible probability of demonstrating a collision, by reduction to the security of h.

- We start by considering if $X \neq Y$. Namely, we want to analyze the probability that $X \neq Y$ and yet $h(s, X) = h(s, Y)$. Or, a bit more formally, we are analyzing the following probability:

  $$\Pr\left[X \neq Y \text{ and } h(s, X) = h(s, Y) \;\middle|\; \begin{array}{l} s \leftarrow S(1^\lambda) \\ (X, Y) \leftarrow \mathcal{B}(1^\lambda, s) \end{array}\right]$$

  Where $\mathcal{B}$ is a reduction to $\mathcal{A}$, as follows:

  $$\begin{array}{l} \mathcal{B}(1^\lambda, s): \\ \quad (x, y) \leftarrow \mathcal{A}(1^\lambda, s) \\ \quad (x_0, x_1, x_2) = \mathsf{thirds}(x) \\ \quad (y_0, y_1, y_2) = \mathsf{thirds}(y) \\ \quad X = h(s, x_0 \mid x_1) \mid x_2 \\ \quad Y = h(s, y_0 \mid y_1) \mid y_2 \\ \quad \mathsf{return}\ (X, Y) \end{array}$$

  But the probability is negligible by definition of collision resistance for h.

- Now, consider $X = Y$. But remember that $\mathcal{A}$ is only considered to have found a collision if it finds $x \neq y$ such that $H(s, x) = H(s, y)$. Let $x_{01} = x_0 \mid x_1$ and let $y_{01} = y_0 \mid y_1$. Examining the definition of X, Y, we can see that it must be that the following quantities are equal:

  $$h(s, x_{01}) = h(s, y_{01})$$

  Namely, the probability we are analyzing is bounded by the following:

  $$\Pr\left[x_{01} \neq y_{01} \text{ and } h(s, x_{01}) = h(s, y_{01}) \;\middle|\; \begin{array}{l} s \leftarrow S(1^\lambda) \\ (x_{01}, y_{01}) \leftarrow \mathcal{C}(1^\lambda, s) \end{array}\right]$$

  Where $\mathcal{C}$ is a reduction to $\mathcal{A}$, as follows:

  $$\begin{array}{l} \mathcal{C}(1^\lambda, s): \\ \quad (x, y) \leftarrow \mathcal{A}(1^\lambda, s) \\ \quad (x_0, x_1, x_2) = \mathsf{thirds}(x) \\ \quad (y_0, y_1, y_2) = \mathsf{thirds}(y) \\ \quad x_{01} = x_0 \mid x_1 \\ \quad y_{01} = y_0 \mid y_1 \\ \quad \mathsf{return}\ (x_{01}, y_{01}) \end{array}$$

But, again, by definition of collision resistance, this probability is negligible.

So in sum, there are only two ways to induce a collision in H: either induce a collision on the inner hash evaluation $X_L = Y_L$, or induce a collision on the outer hash evaluation $h(s, X_L \mid x_2) = h(s, Y_L \mid y_2)$. In either case, the collision resistance of h means that an adversary can produce such a collision with at most negligible probability. Thus, the probability a poly-bounded adversary can find a collision in H is bounded by the sum of the probabilities of the subcases, which is also negligible. Thus, H is collision resistant.

2.

$$
\begin{aligned}
&H'(s, x): \\
&\quad x' = \mathsf{pad}(x) \\
&\quad \mathsf{return}\ H(s, x')
\end{aligned}
$$

Here, pad is a function that pads a single 1 to its input, followed by a string of zeros, until the output has length 3λ. The crucial point is that pad is injective; namely, every possible input has a distinct output string. Hence, finding a collision on the inputs involves finding a collision on the corresponding input to H. But H is collision resistant, so the reduction is, in fact, immediate. Formally, assuming we have an adversary $\mathcal{A}$ that finds collisions in H', we can construct an adversary $\mathcal{B}$ that finds collisions in H:

$$
\begin{aligned}
&\mathcal{B}(1^\lambda, s): \\
&\quad (x, y) \leftarrow \mathcal{A}(1^\lambda, s) \\
&\quad \mathsf{return}\ (\mathsf{pad}(x), \mathsf{pad}(y))
\end{aligned}
$$

Since pad is injective, $x \neq y$ implies $\mathsf{pad}(x) \neq \mathsf{pad}(Y)$.

3. Here's one H'' that works. Note that there are many valid ways to construct H''. Here, pad appends a single one, then appends zeros until the input is of length a multiple of λ. blocks splits x into n chunks, each of length λ bits.

$$
\begin{aligned}
&H''(s, x): \\
&\quad x' = \mathsf{pad}(x) \\
&\quad (x_0, ..., x_{n-1}) = \mathsf{blocks}(x') \\
&\quad out := 0 \\
&\quad \mathsf{for}\ i \in \{0, ..., n-1\}: \\
&\quad\quad out := h(s, out \mid x_i) \\
&\quad \mathsf{return}\ h(s, out \mid n)
\end{aligned}
$$

Intuitively, this hash function works because finding a collision requires finding a "sub-collision" in some call to h. If no such sub-collision is found, the output will not collide. Note that there is an important detail here: we include the length n (interpreted as a string of length λ) in the final hash. This prevents the adversary from forcing collisions between (1) a long string x and (2) a hash of the prefix of x concatenated with the suffix of x.

Problem 2 (Discrete Logarithms and Collision Resistance)**.** Let $\mathbb{G}_\lambda, g_\lambda$ denote a family of discrete-log-hard groups (see Definition 2). Now, consider the following hash function definition. First, its

seeds are drawn as follows:

$$S(1^\lambda) = \left\{ (g,h) \;\middle|\; \begin{array}{l} q \leftarrow |\mathbb{G}_\lambda| \\ g \leftarrow g_\lambda \\ x \leftarrow \mathbb{Z}_q \\ h \leftarrow g^x \end{array} \right\}$$

That is, we choose a group depending on λ, then sample a random group element h. Now, the hash function H takes a seed (g,h) and two integers (a,b) and outputs a single group element as follows:

$$H((g,h),(a,b)) = g^a h^b$$

- **(3 points)** Give a security reduction showing that if the discrete logarithm problem is hard for groups $\mathbb{G}_\lambda$, then H is a collision resistant hash function. Hint: *Remember, this means that if H is* not *collision resistant—i.e., if there exists an efficient adversary that finds collisions with noticeable probability—then there should also be an adversary that computes discretes logarithms.*

Solution 2. We can solve this quite directly via the contrapositive. Namely, if H is not collision resistant, then we can solve discrete log. Let A be an adversary that wins the collision resistance game, i.e. that with non-negligible probability produces a pair $(a_0,b_0) \neq (a_1,b_1)$ that collide. We can construct an adversary $\mathcal{B}$ that computes discrete logs:

$$\begin{array}{l} \mathcal{B}(1^\lambda, g_\lambda^x): \\ \quad g = g_\lambda \\ \quad h = g_\lambda^x \\ \quad s = (g,h) \\ \quad ((a_0,b_0),(a_1,b_1)) \leftarrow \mathcal{A}(1^\lambda, s) \\ \quad \mathsf{return}\ (a_0 - a_1)(b_1 - b_0)^{-1} \bmod q \end{array}$$

Essentially, $\mathcal{B}$ just prepares a hash seed as described by the hash function specification, then gives it to $\mathcal{A}$. Then $\mathcal{B}$ performs simple modular arithmetic to find the discrete logarithm. The reason this works is as follows. Suppose $\mathcal{A}$ indeed produces a non-trivial collision $((a_0,b_0),(a_1,b_1))$. Then, we know:

$$g^{a_0} h^{b_0} = g^{a_1} h^{b_1}$$

And because $h = g^x$, this means:

$$g^{a_0 + b_0 x} = g^{a_1 + b_1 x}$$

By properties of cyclic groups, this must mean:

$$a_0 + b_0 x = a_1 + b_1 x \mod q$$

And hence:

$$x = (a_0 - a_1)(b_1 - b_0)^{-1} \mod q$$

There is one additional detail that should be considered: Can $\mathcal{A}$ find a non-trivial collision, but where $b_0 = b_1$? However, this is impossible. Namely, as argued above, we know that for a collision we have:

$$g^{a_0+b_0x} = g^{a_1+b_1x}$$

And if, $b_0 = b_1$, then we additionally know the following holds:

$$g^{a_0} = g^{a_1}$$

But since we are working in a cyclic group, this implies that $a_0 = a_1$. But that would would mean that both $b_0 = b_1$, and $a_0 = a_1$. In other words, this is a trivial collision, and $\mathcal{A}$ is only considered to have won if it finds a non-trivial collision.

Problem 3 (Public Key Encryption)**.**

1. **(2 points)** It is crucial to use randomized encryption when using public keys. Suppose that we use a deterministic public-key encryption algorithm $Enc(pk, m)$. Describe an adversary that can decrypt a ciphertext from any such scheme in time that scales linearly in $|\mathcal{M}|$ where $\mathcal{M}$ is the scheme's message space.

2. **(3 points)** Consider the following attempt at key exchange:

   (a) Alice samples random key $k \leftarrow \{0,1\}^\lambda$ and random mask $r \leftarrow \{0,1\}^\lambda$. Alice sends $s = k \oplus r$ to Bob.

   (b) Bob samples $t \leftarrow \{0,1\}^\lambda$ and sends $u = t \oplus s$ to Alice.

   (c) Alice computes $w = u \oplus r$ and sends w to Bob.

   (d) Alice outputs k; Bob outputs $w \oplus t$.

   Is the protocol correct? Namely do Alice and Bob compute the same key? Is the protocol secure? Why/why not?

Solution 3.

1. Since the adversary is allowed to spend time scaling in the message space, it can simply try to encrypt each message in the message space. Specifically, the following adversary will work:

$$\begin{array}{l}\mathcal{A}(pk, c): \\ \quad \textsf{for each } m \in \mathcal{M}: \\ \quad\quad \textsf{if } Enc(pk, m) = c: \\ \quad\quad\quad \textsf{return } m\end{array}$$

   Because the encryption scheme is deterministic, if c is indeed an encryption of m, then the condition $Enc(pk, m) = c$ will be true, and the adversary will correctly output m.

2. **Yes**, the protocol is correct:

$$\begin{aligned} w \oplus t &= (u \oplus r) \oplus t && \text{Defn. } w \\ &= ((t \oplus s) \oplus r) \oplus t && \text{Defn. } u \\ &= ((t \oplus (k \oplus r)) \oplus r) \oplus t && \text{Defn. } s \\ &= k && \text{Associativity, Commutativity, Involutivity of } \oplus \end{aligned}$$

So Alice and Bob both output k, a shared key. **No**, the protocol is not secure. Any eavesdropper observes messages s, u, w. From these three messages, the eavesdropper can reconstruct the secret k in polynomial time:

$$\begin{aligned} s \oplus u \oplus w &= s \oplus u \oplus (u \oplus r) && \text{Defn. } w \\ &= s \oplus r \\ &= (k \oplus r) \oplus r && \text{Defn. } s \\ &= k \end{aligned}$$