3. Problem Session
Cryptographic Hash Functions
(Summer Term 2014)

Bauhaus-Universität Weimar, Chair of Media Security
Prof. Dr. Stefan Lucks, Jakob Wenzel
URL: `http://www.uni-weimar.de/de/medien/professuren/mediensicherheit/teaching/`

Date: 06.05.2014 (15:15)

Task 1 (4 Credits) Indifferentiability
Let a two-level construction be defined as shown on Slide 42 of Section 2.1. Thus,

$$H_F(x) = F(h(x)).$$

We have shown that this construction is not indifferentiable in the Random Oracle Model (ROM), if h is a cryptographically secure one-way hash function (COWHF), and F is random function modelled as a random oracle.

1. Ask for $H(x) = y$.
2. Compute $h(x) = z$.
3. Ask for $F(z) = y'$.
4. If $y = y'$ output "real", else "random".

Next, consider this construction where h is not a COWHF but an invertible public permutation, e.g., the identity. Show that this new construction is secure in the indifferentiability model.

Task 2 (4 Credits) Structural Weakness
Let $H : \{0,1\}^* \rightarrow \{0,1\}^n$ be an iterated COWHF, e.g., based on the Merkle-Damgård structure. Further, consider a scenario where a client wants to store its data M on an external server. After a while the client wants to know, if its data where manipulated by the server. Thus, before sending the data to the server in the first place, it computes

$$h_i = H(C_i \mid\mid M)$$

for k distinct and secret challenges $C_1, \ldots, C_k$, and stores the corresponding hash values h_i on its own hard drive. Then, after sending the data M to the server, and later to prove the consistence of its data, it sends a challenge C_i to the server, getting h'_i as an answer. If $h_i = h'_i$, the client knows that the data was not be manipulated, since H is secure in the indifferentiability model, i.e., $\Pr[h_i = h'_i] = 1/2^n$.

Let the hash values h_i now be computed by

$$h_i = H(M \mid\mid C_i).$$

Show that this allows the server to easily betray the client.

Task 3 (4 Credits)
Consider a system which searches for preimages. You gain one digital coin for each preimage you found for a specific hash value $Y \in \{0,1\}^{56}$, where Y is the truncated output of an iterated hash function $H : \{0,1\}^* \to \{0,1\}^{128}$, e.g., based on the Merkle-Damgård structure, employing a cryptographically secure compression function $F : \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$. Given an adversary to this system with a **computational power of** $O(2^{70})$ operations. Find a strategy to maximize the amount of digital money this adversary can produce with its computational power. **The adversary needs at least $\mathbf{2^{20}}$ coins** to buy his new house.

Task 4 (6 Credits) Programming Task
In Task 4 of the second problem set the method for searching a near-collision (collision in the first k bytes) for SHA-512 was highly inefficient regarding to its memory usage. Now, we want to apply two alternative (and memory-efficient) approaches to search for a near-collision. Therefore, you should solve the following three tasks, using the programming language Python.

a) Implement the cycle-finding algorithms of Brent and Floyd (see Slide 50 of Section 3.1 and `http://en.wikipedia.org/wiki/Cycle_detection`) and search for a near-collision for SHA-512.

b) Measure the time required for finding a near-collision for both algorithms. Is one algorithm significantly faster than the other one? If so, give an explanation! *(Note that it makes only sense to compare the two algorithms for collisions with the same k.)*

c) Modify your solution from Task 4 (of the previous problem set) regarding to the usage of distinguished points (see Slide 51 of Section 3.1) and search for a near-collision for SHA-512. Thus, one does not store about $2^{n/2}$ hash values but only those which contain a certain pattern. A call to this program should like follows:
`./sha512_coll_dp.py <pattern>`.
An example call would look like:
`./sha512_coll_dp.py fefe`,
which indicates that only hash values are stored whose two least significant bytes are `0xfe`, each

Try to maximize the value k for both Task a) and Task c), i.e., try to find a collision for a maximal number of bytes. Send me the source code and the input messages which lead to the largest value of k via email. The group with the largest value of k gets a bag of gummy bears.