Massachusetts Institute of Technology
6.857: Network and Computer Security
Professor Ron Rivest

Handout 7
March 12, 2012
Due: March 19, 2012

Problem Set 3

This problem set is due *online,* at `https://courses.csail.mit.edu/6.857/` on *Monday, March 19* by **11:59 PM**. Please note that no late submissions will be accepted.

You are to work on this problem set with your assigned group of three or four people, given at `http://courses.csail.mit.edu/6.857/2012/ps3groups.html`. If your name is missing, please email `6.857-tas@mit.edu`. Be sure that all group members can explain the solutions. See Handout 1 (*Course Information*) for our policy on collaboration.

Homework must be submitted electronically! Submissions should be in pdf form, with the document named by each team member's last name appended. Each problem answer must appear on a separate page. Mark the top of each page with your group member names, the course number (6.857), the problem set number and question, and the date. We have provided templates for LaTeX and Microsoft Word on the course website (see the *Resources* page).

Grading: All problems are worth 10 points.

With the authors' permission, we will distribute our favorite solution to each problem as the "official" solution—this is your chance to become famous! If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this in the email used to turn in the submission.

Problem 3-1. SHA-256 Collisions

Define h to be the SHA-256 hash function, and let h_n denote h with its output truncated to n bits (for any n, $1 \leq n \leq 256$).

You are asked to find the largest n you can for which you can demonstrate a collision for h_n.

In Python, $h_n(x)$ can be defined:

```
import hashlib

def h(n,x):
    """
    here n in a positive integer not more than 256, and
    x is a byte string
    returns first n bits of sha256(x).
    """
    assert 0 < n <= 256
    y = hashlib.sha256(x).digest()
    bytes_wanted = (n+7)/8
    bits_wanted = n - 8*(bytes_wanted-1)    # in last byte
    mask = ((1 << bits_wanted)-1)
    z = y[:bytes_wanted-1] + chr(ord(y[bytes_wanted-1]) & mask)
    return z
```

Your implementation should use only a small amount of memory (not proportional to the number of calls to h your program makes). You can use variants of the "Pollard-rho" algorithm or Floyd's "two-finger" algorithm (or any other low-memory algorithm you like).

Estimate the amount of time your implementation would take to find a collision for a hash function with a d-bit output, for $d = 64, 80, 128, 160$,and 256.

Turn in the largest collision you found, and also your code as part of the pdf document. (The verbatim command is useful for putting code in a latex document.) Your collision should be different than that turned in by any other team!

Problem 3-2. Merkle-Damgård Transform - from Katz and Lindell 4.15

The following problem is taken from Katz and Lindell, Problem 4.15 parts (a), (b), and (c) on page 157.

As Professor Rivest mentioned in class, a formal definition of hash functions includes a *family* of hash functions indexed by a *key* s. A randomized algorithm Gen is used to generate a seed s for a hash function, and then h^s is used to denote the hash function indexed by seed s.

The Merkle-Damgård transform converts any fixed-length hash function (which takes inputs of a fixed-length to some output length) into a full fledged hash function (which can take inputs of arbitrary length to the given output length), while maintaining collision resistance properties. This transform is widely used in practice.

Though this transform works for fixed-length hash functions that shrink by any amount, even just a single bit, for simplicity we will give the construction for a function that shrinks its input length by a factor of 2. Let (Gen, h) be a fixed-length collision resistant hash function for inputs of length $2\ell(n)$ and with output length $\ell(n)$ - $h := \{0,1\}^s \times \{0,1\}^{2\ell(n)} \to \{0,1\}^{\ell(n)}$. The Merkle-Damgård transform constructs a variable-length hash function (Gen, H) with $H := \{0,1\}^s \times \{0,1\}^* \to \{0,1\}^{\ell(n)}$ as follows:

- Gen remains unchanged
- H: on input a key s and a string $x \in \{0,1\}^*$ of length $L < 2^{\ell(n)}$, do the following (set $\ell = \ell(n)$ in what follows):
  1. Set $B := \lceil \frac{L}{\ell} \rceil$ (i.e., the number of blocks in x). Pad x with zeroes if necessary so its length is becomes a multiple of ℓ. Parse the padded results as the sequence of ℓ-bit blocks $x_1, \ldots, x_B$. Set $x_{B+1} := L$, where L is encoded using exactly ℓ bits.
  2. Set $z_0 = 0^\ell$.
  3. For $i = 1, \ldots, B+1$, compute $z_i := h^s(z_{i-1}||x_i)$.
  4. Output z_{B+1}.

For each of the following modifications to the Merkle-Damgård transform, determine whether the result is collision resistant or not. If yes, provide a proof; if not, demonstrate an attack.

(a) Modify the construction so that the input length is not included at all (i.e., output z_B and not $z_{B+1} = h^s(z_B||L)$).

(b) Modify the construction so that instead of outputting $z = h^s(z_B||L)$, the algorithm ouputs $z_B||L$.

(c) Instead of using a fixed IV, choose $IV \leftarrow \{0,1\}^l$ and define $z_0 := IV$. Then, set the output to be $IV||h^s(Z_B||L)$.