

Security

Assignment 3, Wednesday 24 September 2008

Comments on the problematic issues

A cryptographic hash is a function h that assigns to a bit string another bit string (usually of fixed length, but this is not a formal requirement here), and has the following three properties:

- (E) h can be computed efficiently,
- (O) given a bit string y it is infeasible to determine a bit string x such that $h(x) = y$ (onewayness),
- (C) it is infeasible to determine two bit strings x, x' such that $h(x) = h(x')$ (collision resistance).

A Given a function H that meets (E), (O) and (C), determine for the following functions h which of the demands apply

- The function $h(x) = 1$: (E).
(O) fails because you can easily take SOME x such that $h(x) = 1$. Any x is such that $h(x) = 1$.
- The function $h(x) = x$: (E), (C).
(C) holds because h is an injection.
- The function $h(x_1 \cdots x_{2n}) = (x_1 \text{ XOR } x_{n+1}) \cdot (x_2 \text{ XOR } x_{n+2}) \cdots (x_n \text{ XOR } x_{2n})$: (E).
(O) fails, because given an output $y = (y_1, \dots, y_n)$, you can take $x = (y_1, \dots, y_n, 0, \dots, 0)$. It is easy to that $h(x) = y$.
(C) fails, because, given an output $x = (x_1, \dots, x_n, x_{n+1}, \dots, x_{2n})$, with $x_1 \neq x_{n+1}$, take, for instance, $x' = (x_{n+1}, \dots, x_{2n}, x_1, \dots, x_n)$. It is easy to see that $h(x) = h(x')$.
- The function $h(x_1 \cdots x_n) = H(x_1 \cdots x_{48}) \cdot x_{49} \cdots x_n$: (E).
(O) fails because given an output $y = (y_1, \dots, y_k, x_{49} \cdots x_n)$, by the finite search (up to 2^{48} binary combinations to check) we can find x_1, \dots, x_{48} , s.t. $H(x_1, \dots, x_{48}) = y_1, \dots, y_k$. Take then $x = (x_1, \dots, x_{48}, x_{49} \cdots x_n)$. It is easy to see that $h(x) = y$.
(C) fails, if H is not an injection, by a similar reason. Let $x = (x_1, \dots, x_{48}, x_{49} \cdots x_n)$. By the finite search you find x'_1, \dots, x'_{48} , such that $H(x_1, \dots, x_{48}) = H(x'_1, \dots, x'_{48})$ and $(x_1, \dots, x_{48}) \neq (x'_1, \dots, x'_{48})$. Take $x' = (x'_1, \dots, x'_{48}, x_{49} \cdots x_n)$. It is easy to see that $h(x) = h(x')$ and $x \neq x'$.
(C) holds if H is an injection, but this is a rare case. So, if you have answered just (E), we count it as the correct answer.
- The function $h(x_1 \cdots x_{2n}) = H(x_1 \cdots x_n) \cdot x_{n+1} \cdots x_{2n}$: (E), (O), (C).
Unlike the previous comment, the length of the prefix here, n , grows proportionally to the length of the input word $x_1 \cdots x_{2n}$. So, here the fact that H satisfies (E), (O) and (C) indeed helps.

1. One way to construct a one-time pad uses a hash function h and starts with an initial key K and appends iterated applications of h :

$$K \cdot h(K) \cdot h(h(K)) \cdot h(h(h(K))) \cdots$$

- Which of the three properties (E), (O), (C) of h is used?

(E) to make the whole pad computable, more or less effectively.

(O) to make restoring K unfeasible.

(C) to prevent cycling in the pad (cyclic pads are easily reconstructible).

Some of you did not answer the question that was about the properties of h . Instead you studied the properties of *pads* computed in such way. But this was not the question. Moreover, when we speak about *pads*, we do not discuss (E), (O) or (C). These properties are applicable to *hashes*.

- Is this way to construct a one-time pad secure? Motivate your answer.

No, it is not. Once an attacker guesses correctly some $h(\dots(h(K))\dots)$, (s)he can easily obtain the rest of the pad and thus the rest of the message.