Required homework (Jan 29; due Feb 12)

Instructions. Turn your answers to this homework in, on paper, at the beginning of class on February 12. You are welcome to discuss the problems in groups, but your final write-up must be your own.

When asked to break a cryptosystem, your answer should include (1) a brief description of how to attack the scheme, and (2) a summary of the complexity of the attack (in number of known or chosen plaintexts, and in workfactor if relevant). You can make my life easier by circling your summary of the complexity of the attack. You don't need to micro-optimize the complexity of your attack (I don't care about the difference between $O(n)$ and $O(n \log n)$), but try to avoid being grossly and unnecessarily inefficiently (e.g., $O(2^n)$ vs. $O(2^{n/2})$, or using chosen plaintexts where the same number of known plaintexts suffice).

1 Length-doubling hashing

I have a good[1] hash function $h : \{0,1\}^{64} \to \{0,1\}^{64}$, and I want to double it to get a 128-bit hash function. My brilliant idea is to define the function $R : \{0,1\}^{128} \to \{0,1\}^{128}$ given by

$$R(w,x,y,z) = \langle h(w,y), h(x,z) \rangle$$

and then iterate this function some number of times. I want you to analyze the security of this idea.

My hope is that H will be a good hash function. In other words, H should have no identifying characteristics that wouldn't appear in a random function on 128 bits. If you can find any violation of this, it counts as an attack. For instance, if you can show how to find an input m such that $H(m)$ is zero in the first 80 bits using only 2^{32} steps of computation, this is a successfull attack on H, since for a random function you'd typically need about 2^{80} trials to find such an output. Similarly, if you can find a collision in much less than 2^{64} time, that's another attack, because a truly random function wouldn't yield a collision until you do close to 2^{64} computations. And so on. Feel free to be creative.

For each of the following, find the best attack you can.

1. $H = R$ (one round).

2. $H = R \circ R$, i.e., $H(m) = R(R(m))$ (two rounds).

3. $H = R \circ R \circ R$ (three rounds).

4. $H = R \circ R \circ R \circ R$ (four rounds).

Clarification (Feb 1). *It has since been pointed out to me that this problem is, well, somewhat easier than I intended. That's ok—I won't change the problem, and you should still solve the original problem—but if you want, you may optionally and for fun think about what to do if we instead change the problem to add a bijectivity constraint: namely, we say that h behaves like a random bijective 64-bit function, and H is intended to behave like a random bijective 128-bit function.*

[1] In other words, h behaves roughly like a function chosen uniformly at random from the set of all 64-to-64-bit functions. Of course, since h is a hash function, it is publicly computable: anyone can easily compute $h(x)$ given x.

2 Randomized encryption

Consider the following way of building a keyed, randomized 1-bit-to-2-bit encoder. Look at the 4 possible 2-bit strings. We'll call these our codewords. We pick two of these to represent the message bit '0', and the remaining two to represent the message bit '1'. The secret key k is the association between 2-bit codewords and 1-bit messages. To encode a message bit x, we flip a fair coin and randomly pick one of the two codewords associated with x. We'll let $E_k(x)$ denote the encoding of bit x under key k using this scheme (beware that E_k is not a proper function but rather a randomized algorithm). Also, write $D_k(y)$ for the deterministic, keyed decoding scheme ($D_k : \{0,1\}^2 \to \{0,1\}$ is indeed a true function).

We can extend this to encode arbitrary-length messages by just encoding each bit of the message separately, using an independently chosen key for each bit of the message. In other words, if we have a n-bit message $x = \langle x_1, \ldots, x_n \rangle$, we define

$$E_k(x) = \langle E_{k_1}(x_1), \ldots, E_{k_n}(x_n) \rangle$$

where $k = \langle k_1, \ldots, k_n \rangle$.

This encoding can also be iterated an arbitrary number of times. For instance, we can compute $E_{k'}(E_k(x))$, which gives a keyed, randomized encoding that expands its input by a factor of four. Iterating it m times will expand the input by a factor of 2^m.

Analyze the five-fold composition of this scheme, using independent keys in each layer (so that we use $1+2+4+8+16 = 31$ keys in all when encoding a one-bit message). The keyspace is obviously very large. But how secure is it?

3 Design your own pseudorandom permutation

I want you to design your own 64-bit reversible keyed hash function. Here is a chance to exercise your creativity.

In other words, your goal is to design a function $F : \mathsf{K} \to \{0,1\}^{64} \to \{0,1\}^{64}$. For some key space K, and for each key $k \in \mathsf{K}$, the function $F_k : \{0,1\}^{64} \to \{0,1\}^{64}$ should be invertible. (In other words, given y, we can compute x such that $F_k(x) = y$; we write $x = F_k^{-1}(y)$.)

You must ensure that one can compute both F_k and F_k^{-1} efficiently given k. I'm going to impose a performance limit: given some simple RISC instruction set with basic operations like addition, xor, multiplication, shift, load, store, branch and so on, your scheme should not take much more than about 1000 instructions to compute F_k or F_k^{-1}. You don't have to implement your scheme, nor describe how to do so in your answer; I'm merely giving this constraint to rule out horribly inefficient solutions.

Your scheme should be secure against all attacks of complexity 2^{64} or less. Against attackers who don't know the key k, F_k should behave pseudorandomly, and it should be impossible to find any patterns in F_k. In other words, it should behave like a function chosen uniformly at random from the set of all invertible functions mapping 64 bits to 64 bits (or, at least, no attacker should be able to find any identifying patterns that would distinguish F_k from such a truly random invertible function).

The critical constraint is that I want your design to be as simple and elegant as you can make it. Your answer must fit on a single side of a single sheet of paper, without using tiny fonts. The model is that I want something that can be projected on the screen and give a readable description. You can use pseudo-code. You can use a picture. You can use something else.

Make sure you put your answer to this question on a separate piece of paper (put your name on it).

Ideally, your design will be simple enough to memorize after looking at it for a few minutes, and then allow me to think about attacks on it for a few hours as I sit in the hot tub. I will grade your proposal on two aspects: (1) simplicity, clarity, and creativity; and (2) lack of any blindingly obvious security flaws.