Cryptography

Lecture 14

Announcements

- HW 5 is up on the course webpage, due on Wednesday, 4/3.

Agenda

- Last time
  - Collision-Resistant Hash Functions (K/L 5.1)
  - MACs from CRHF (5.3)
    - Hash-and-Mac paradigm
    - Please read about HMAC on your own (K/L 5.3.2)
- This time
- Domain Extension
  - (Merkle-Damgard) (K/L 5.2)
  - Sponge Construction
  - New topic: Practical constructions
    - Stream Ciphers (K/L 6.1)

Domain Extension

The Merkle-Damgard Transform

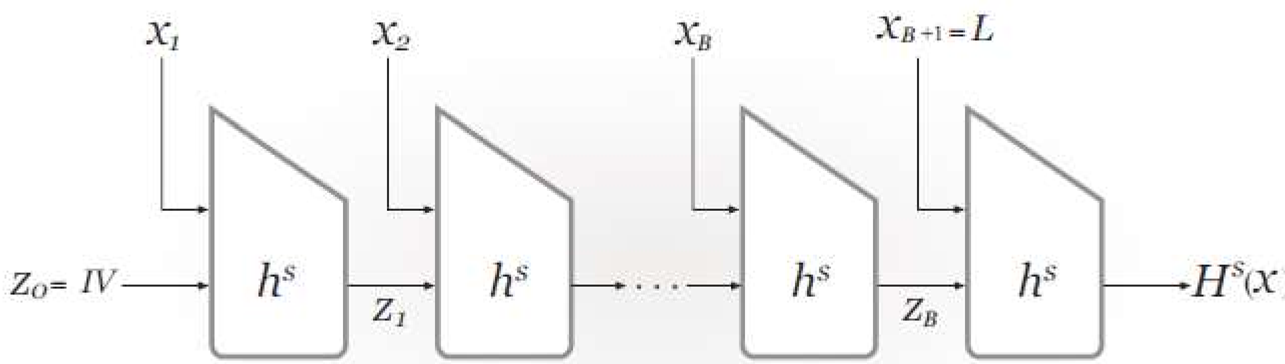


FIGURE 5.1: The Merkle-Damgård transform.

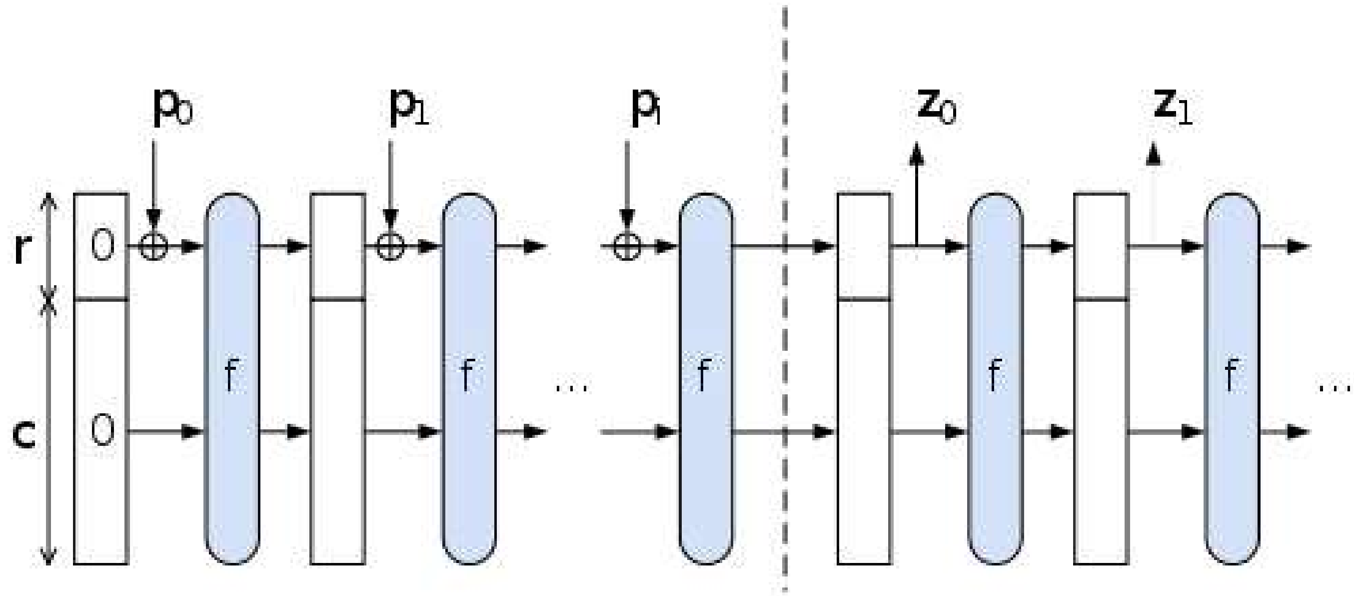
The Merkle-Damgard Transform

Let (Gen, h) be a fixed-length hash function for inputs of length $2n$ and with output length n. Construct hash function (Gen, H) as follows:

- Gen: remains unchanged
- H: on input a key s and a string $x \in \{0,1\}^*$ of length $L < 2^n$, do the following:
    1. Set $B := \left\lceil \frac{L}{n} \right\rceil$ (i.e., the number of blocks in x). Pad x with zeros so its length is a multiple of n. Parse the padded result as the sequence of n-bit blocks $x_1, \dots, x_B$. Set $x_{B+1} := L$, where L is encoded as an n-bit string.
    2. Set $z_0 := 0^n$. (This is also called the IV.)
    3. For $i = 1, \dots, B+1$, compute $z_i := h^s(z_{i-1} || x_i)$.
    4. Output z_{B+1}.

Security of Merkle-Damgard

Theorem: If (Gen, h) is collision resistant, then so is (Gen, H).



A sponge function is built from three components:

- a state memory, *S,* containing *b* bits,
- a public, truly random permutation *f*
- a padding function *P*

The state memory is divided into two sections:

- one of size *r* (the bitrate) and
- the other of size *c* (the capacity).

These sections are denoted *R* and *C* respectively.

The padding function appends enough bits to the input string so that the length of the padded input is a whole multiple of the bitrate, *r.* The padded input can thus be broken into *r*-bit blocks.

Operation

The sponge function operates as follows:

- The state *S* is initialized to zero
- The input string is padded. This means the input p is transformed into blocks of *r* bits using the padding function P.
- R is XORed with the first *r*-bit block of padded input
- S is replaced by *f*(S)
- R is XORed with the next *r*-bit block of padded input (if any)
- S is replaced by *f*(S)

…

The process is repeated until all the blocks of the padded input string are used up ("absorbed" in the sponge metaphor).

The sponge function output is now ready to be produced ("squeezed out") as follows:

- The R portion of the state memory is the first *r* bits of output
- If more output bits are desired, S is replaced by *f*(S)
- The R portion of the state memory is the next *r* bits of output

…

The process is repeated until the desired number of output bits are produced. If the output length is not a multiple of *r* bits, it will be truncated.

Sponge

- Is there a length-extension attack on Sponge?
- Plusses and minuses of Sponge versus Merkle Damgard.
- A Sponge-based hash known as Keccak was selected as SHA-3. Standardized by NIST in 2015.

LFSR

- Consists of an array of n registers $\vec{s} := s_{n-1}, \ldots, s_0$
- Feedback loop specified by a set of n feedback coefficients $\vec{c} := c_{n-1}, \ldots, c_0$.
- The size of the array is called the degree of the LFSR.
- Each register stores a single bit
- The state st of the LFSR at any point is the set of bits contained in the registers
- State of the LFSR is updated in a series of "clock ticks" by shifting the values to the right and setting the new value of the leftmost register.

LFSR

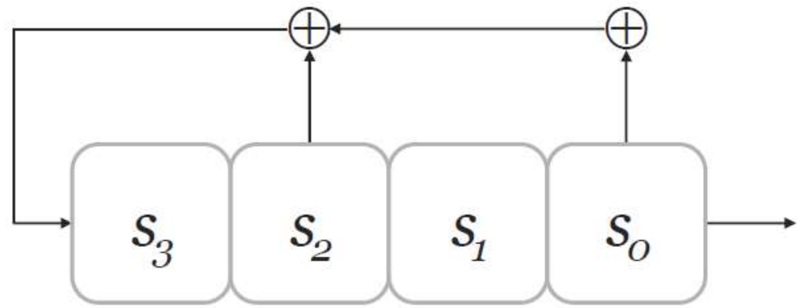


FIGURE 6.1: A linear feedback shift register.

If state in registers at time t is:

$$\vec{s}^{(t)} := s_3^{(t)}, s_2^{(t)}, s_1^{(t)}, s^{(t)}{}_0$$

Then state in registers at time $t+1$ is:

$$s_3{}^{(t+1)} = \langle \vec{c}, \vec{s}^{(t)} \rangle$$

$$s_2{}^{(t+1)} := s_3{}^{(t)}$$

$$s_1{}^{(t+1)} := s_2{}^{(t)}$$

$$s_0{}^{(t+1)} := s_1{}^{(t)}$$

Example

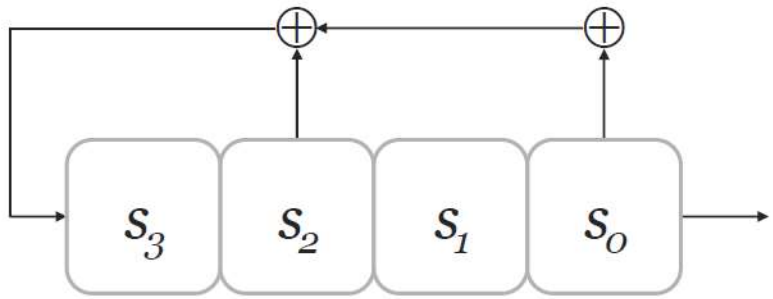


FIGURE 6.1: A linear feedback shift register.

Initial state:	0 0 1 1
	1 0 0 1
	1 1 0 0
	1 1 1 0
	1 1 1 1
	0 1 1 1
	0 0 1 1

- LFSR can cycle through at most 2^n states before repeating
- A maximum-length LFSR cycles through all 2^{n-1} non-zero states before repeating
- Depends only on feedback coefficients, not on initial state
- Maximum-length LFSR's can be constructed efficiently

Reconstruction Attacks

- LFSR are always insecure. We have the following generic attack:
- If state has n bits, then
  - First n output bits $y_0, \dots, y_{n-1}$ reveal initial state $s_0, \dots, s_{n-1}$
  - Can use next n output bits $y_n, \dots, y_{2n-1}$ to determine $c_0, \dots, c_{n-1}$ by setting up a system of n linear equations in n unknowns:

$$\begin{bmatrix} y_0 & y_1 & y_2 & y_3 \\ y_1 & y_2 & y_3 & y_4 \\ y_2 & y_3 & y_4 & y_5 \\ y_3 & y_4 & y_5 & y_6 \end{bmatrix} \times \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix}$$