Cryptography

Lecture 15

Announcements

- Midterm--Median grade 61 (+10 point curve)
- Extra Credit Opportunities
- Current Events and Scholarly Paper

Agenda

- This time
- Domain Extension
  - (Merkle-Damgard) (K/L 5.2) (Review)
  - Sponge Construction
  - New topic: Practical constructions
    - Stream Ciphers (K/L 6.1)

The Merkle-Damgard Transform

SHA-1

SHA-2

K

m_2 m_B length of key plus msg.

Digital Sig

not post-quantum

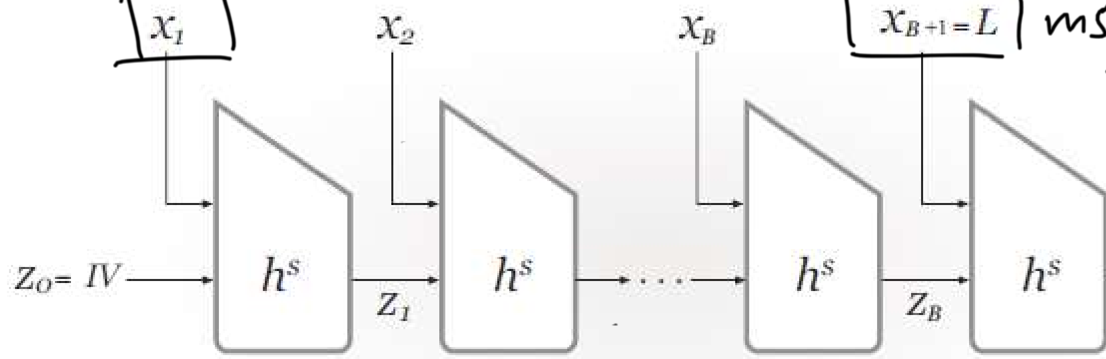


FIGURE 5.1: The Merkle-Damgård transform.

Proof of Work

No way to add transactions to Blockchain w/o doing $\sim 2^t$ amount of work

✓ Blockchain / Bitcoin

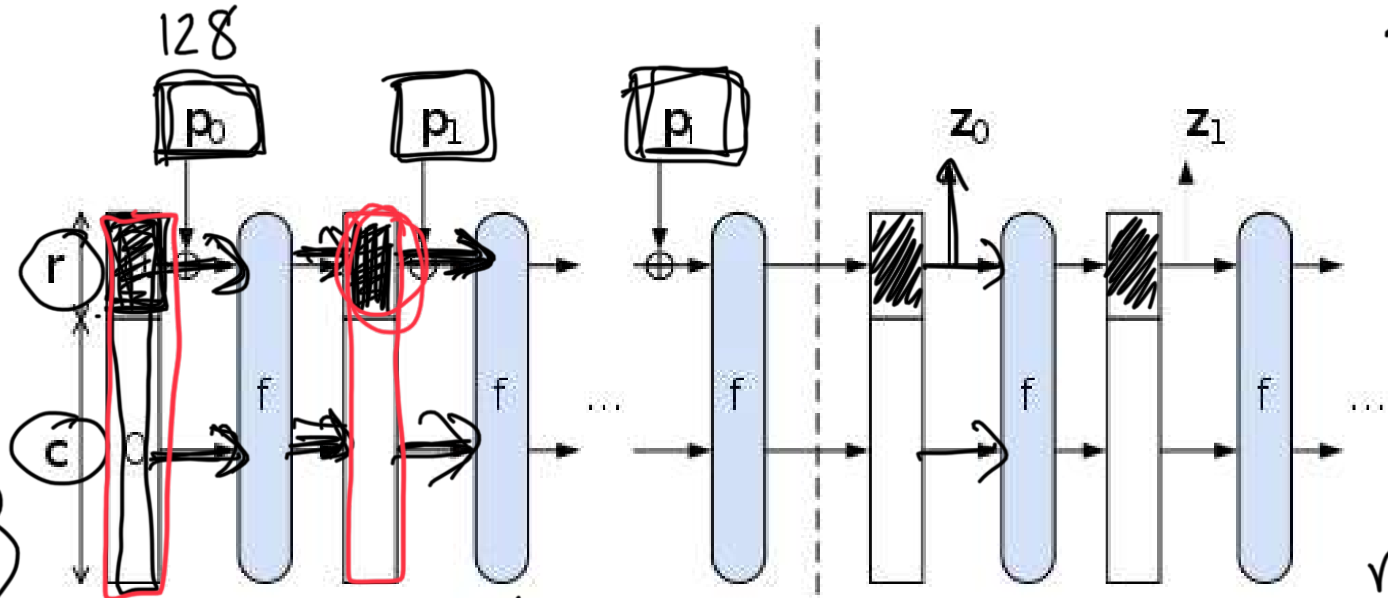
✓ Length Extension Attack

M-D doesn't behave like a truly random function

Sponge Construction (behaves like a truly random function)

SHA-3

Keccak



secret state

absorb

↳ truly random permutation

squeeze

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.

Tradeoffs:

Sponge requires "ideal" perm even for collision resistance

If you're willing to assume "ideal" perm then you get a stronger guarantee

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.

$\boxed{\text{LFSR}}$ Linear Feedback Shift Register

candidate construction Stream Cipher = PRG.

- 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

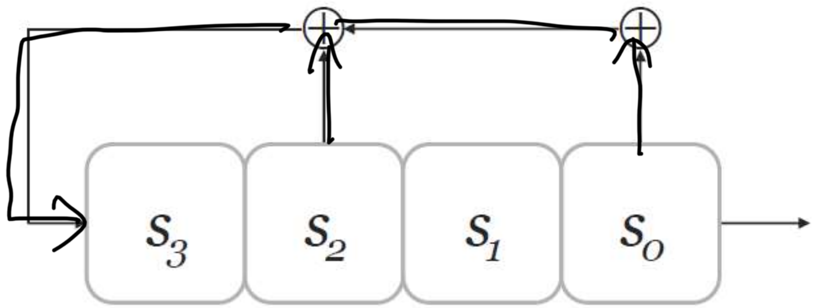


FIGURE 6.1: A linear feedback shift register.

$s_3^{(0)}\ s_2^{(0)}\ s_1^{(0)}\ s_0^{(0)}$

c_3	c_2	c_1	c_0
0	1	0	1

If state in registers at time t is:

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

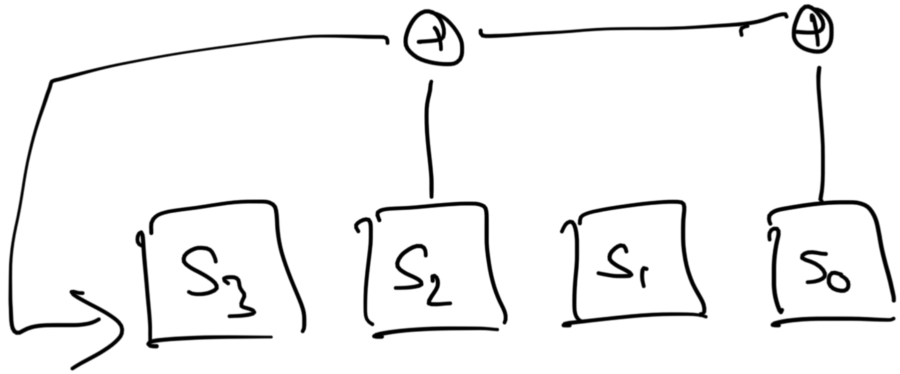
Then state in registers at time $t+1$ is:

$$\boxed{s_3{}^{(t+1)} = \langle \vec{c}, \vec{s}^{(t)} \rangle} \bmod 2\ .$$

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

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

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



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

Example

feedback is a non-linear func of current state

Add non-linearity

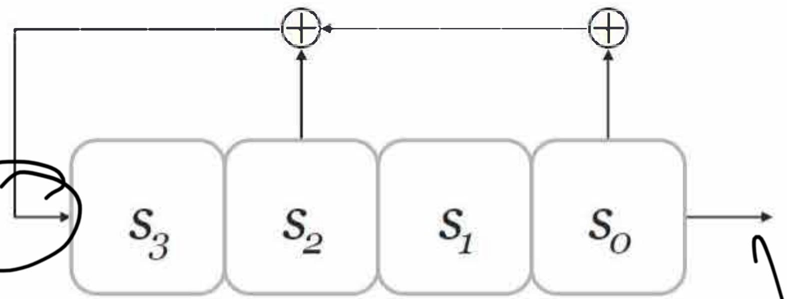


FIGURE 6.1: A linear feedback shift register.

output is a non-linear function of current state

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

Fully break LFSR due to linearity.

First n outputs $y_0 \dots y_{n-1}$ give me the initial state.

$$\underbrace{\begin{array}{cccc} \multicolumn{4}{c}{\overbrace{y_{n-1} \quad \cdots \quad y_1 \quad y_0}^{\text{initial state.}}} \\ y_n & y_{n-1} & \cdots & y_1 \\ \vdots & & & \\ y_{2n-2} & \cdots & \cdots & y_{n-1} \end{array}}_{\text{Known}} \underbrace{\begin{bmatrix} c_{n-1} \\ \vdots \\ \vdots \\ c_0 \end{bmatrix}}_{\substack{\text{var's} \\ \text{feedback coeff}}} = \underbrace{\begin{matrix} y_n \\ y_{n+1} \\ \vdots \\ y_{2n-1} \end{matrix}}_{\text{Known}}$$

Can be set up after seeing $2n$ outputs of LFSR

Once I solve for $c_{n-1} \dots c_0$ can predict the next output.

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:

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