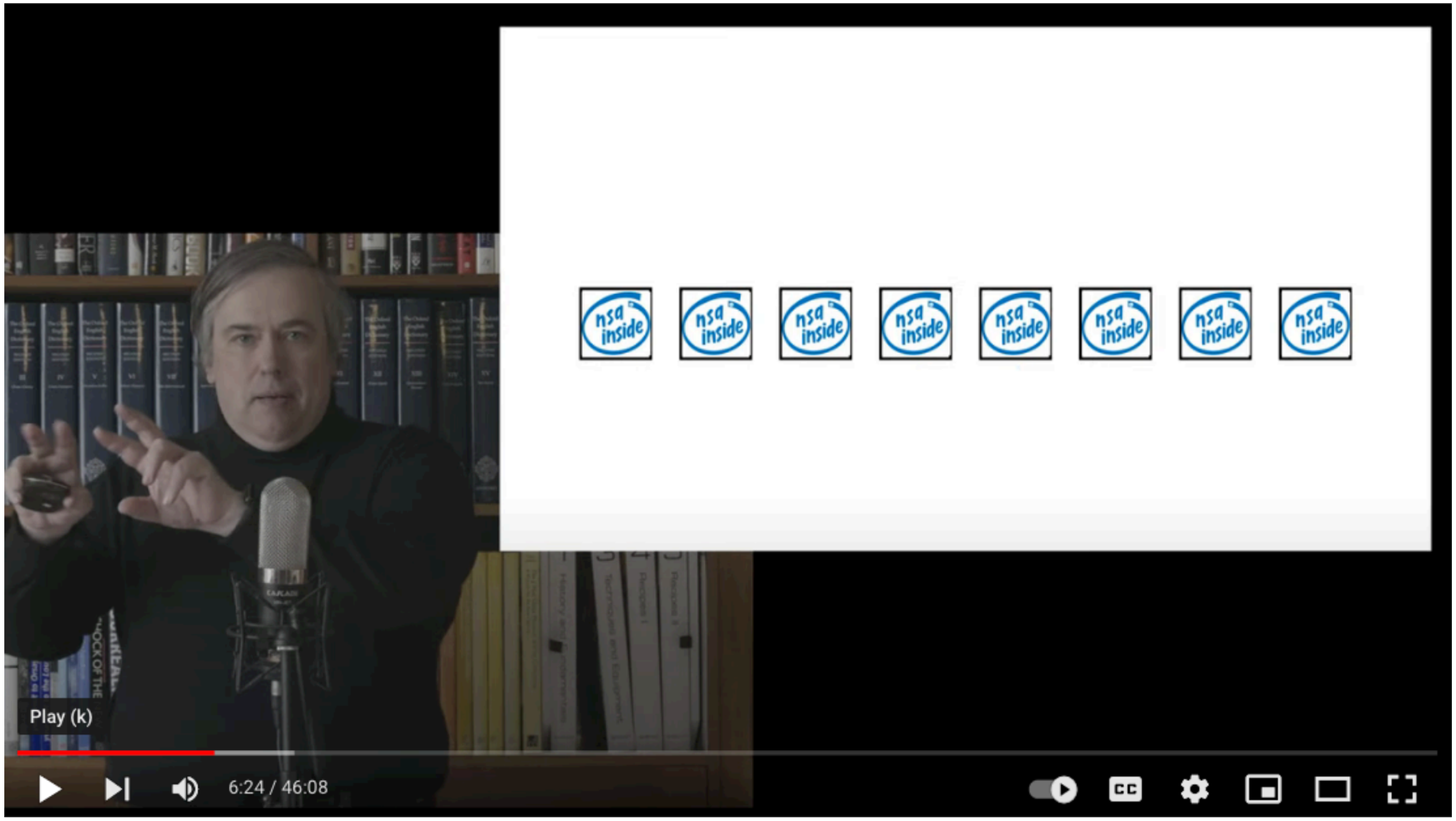
NSA



- 3:13 - 8:40
- https://www.youtube.com/watch?v=tPEi0mnUY_o

DES and the NSA

- The NSA first weakened DES
  - Shortening the 64-bit key in the original "Lucifer" system from IBM to 56 bits
- The NSA also strengthened it
  - Improving the "S-Box" to resist differential cryptanalysis, a technique that was secret at the time
- This created a NOBUS system

NOBUS

- The NSA wants us to use cryptography that is
  - Weak enough so the NSA can break it
  - Strong enough so no one else can break it
- "Nobody But Us"
- This depends on the NSA keeping secrets

Edward Snowden

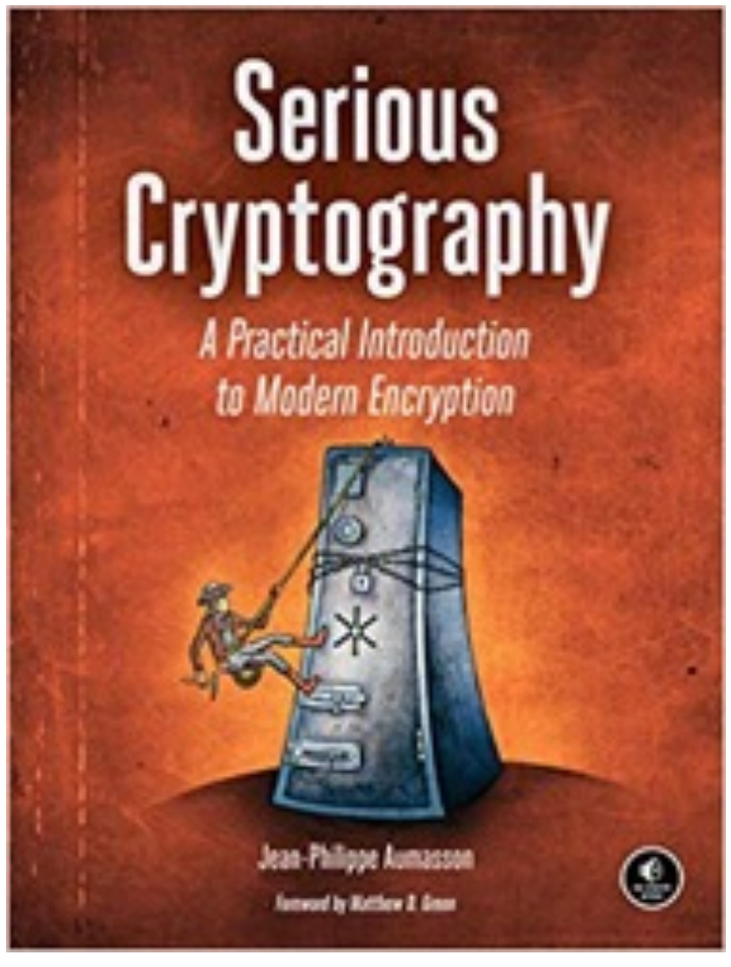
- Leaked NSA secrets in 2013
- https://www.youtube.com/watch?v=0hLjuVyIIrs

WannaCry

- Ransomware that did great damage in 2017
- Spread through NSA exploit "EternalBlue"
- Stolen by The Shadow Brokers in 2016
  - Attributed to either another NSA insider or the Russians

CNIT 141

Cryptography for Computer Networks



5. Stream Ciphers

Updated 9-21-22

Topics

- How Stream Ciphers Work
- Hardware-Oriented Stream Ciphers
- Software-Oriented Stream Ciphers
- How Things Can Go Wrong

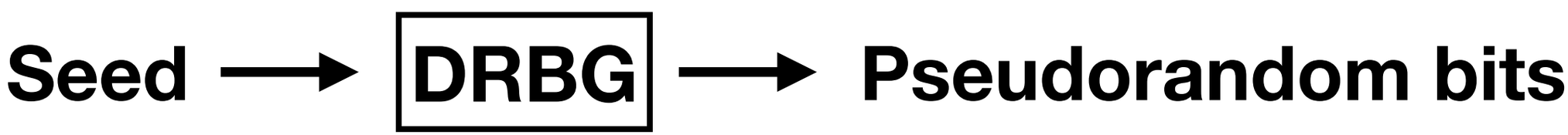
How Stream Ciphers Work

Block v. Stream

- Both block ciphers and stream ciphers are *symmetric*
  - Secret key shared by sender & receiver
  - Block ciphers mix key with blocks of plaintext to produce blocks of ciphertext
  - Stream ciphers use key to produce a pseudorandom stream of bits
    - Then XOR the stream with the plaintext

DRBG

- Deterministic Random Bit Generators
  - Produce a bitstream from a *seed*
- Anyone who knows the seed can reproduce the bits



Key and Nonce

- Stream ciphers use ***key*** and ***nonce***
- The ***key*** is used for many messages, and is secret
- The ***nonce*** changes for each message
  - Not secret--sent with message in plaintext

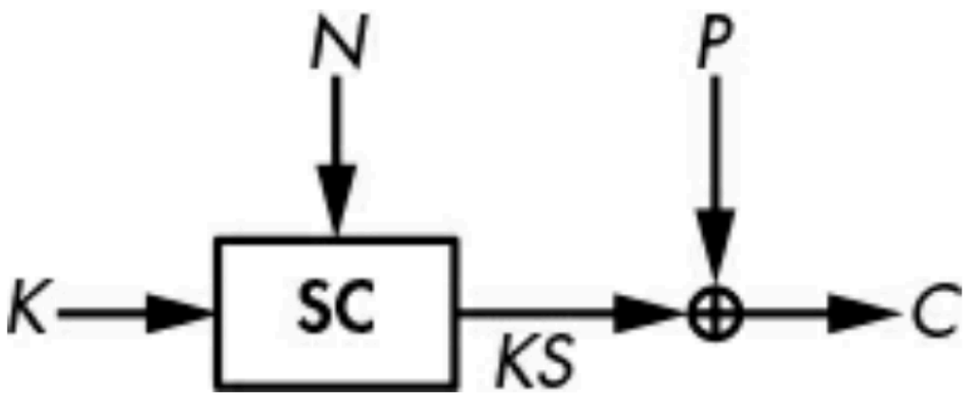
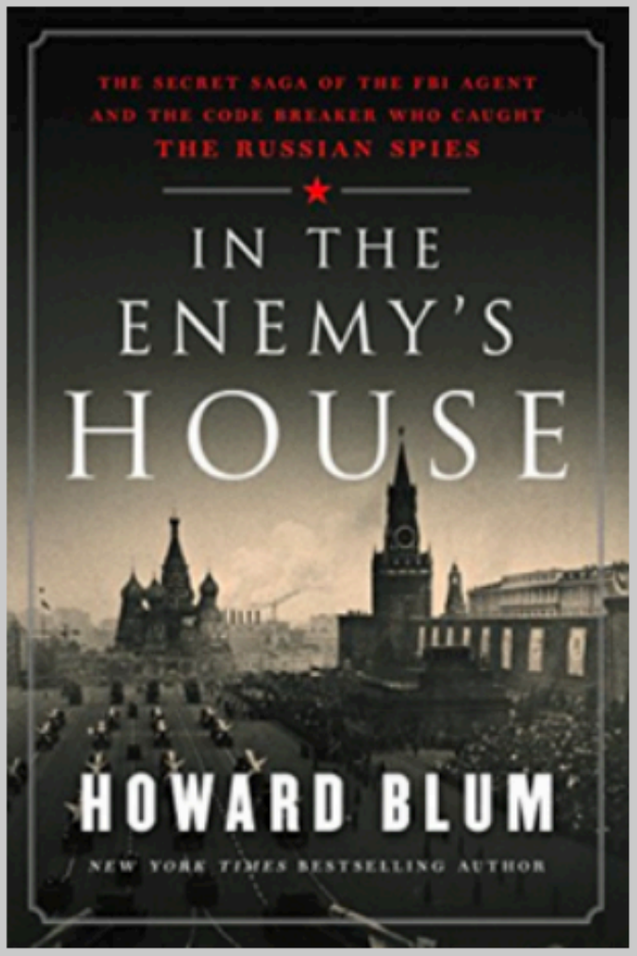


Figure 5-1: How stream ciphers encrypt, taking a secret key, K, *and a public nonce,* N

Nonce Re-Use

- If the key and nonce are re-used for a second message, the system is insecure
- Several simple attacks can find the key
- Russia made this error during WW II
- Microsoft made this error in Microsoft Office encryption before 2007
  - Links Ch 5a, 5b, 5c

C 105: Two-Time Pad (20 pts + 65 extra)



Background

One-time pads are very secure, but not if you use the pad more than once. Russia made this error during World War II and it led to complete compromise of their encryption system, as detailed in this book.

What You Need

Any computer with Python 3.

Using a One-Time Pad

Let's use this keystring:

```
Unbreakable awesome secret secure system
```

Stateful and Counter-Based Stream Ciphers

- Two types of stream ciphers
  - *Stateful ciphers*
    - Secret internal state
  - *Counter-Based ciphers*
    - A counter instead of a state

Stateful Stream Cipher

- ***State*** is initialized with ***key*** and ***nonce***
- Update function updates the state and produces more keystream bits

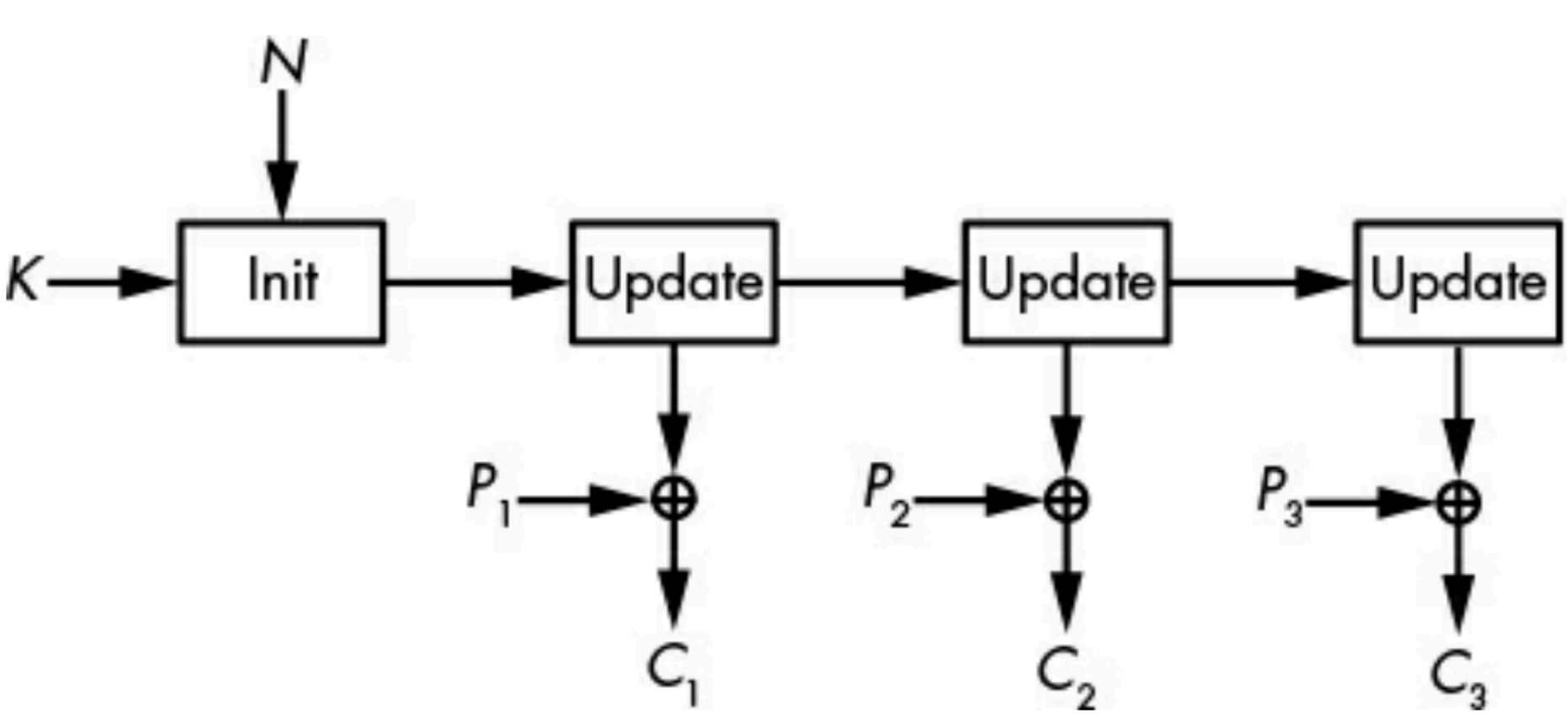


Figure 5-2: The stateful stream cipher

Counter-Based Stream Cipher

- Counter replaces state

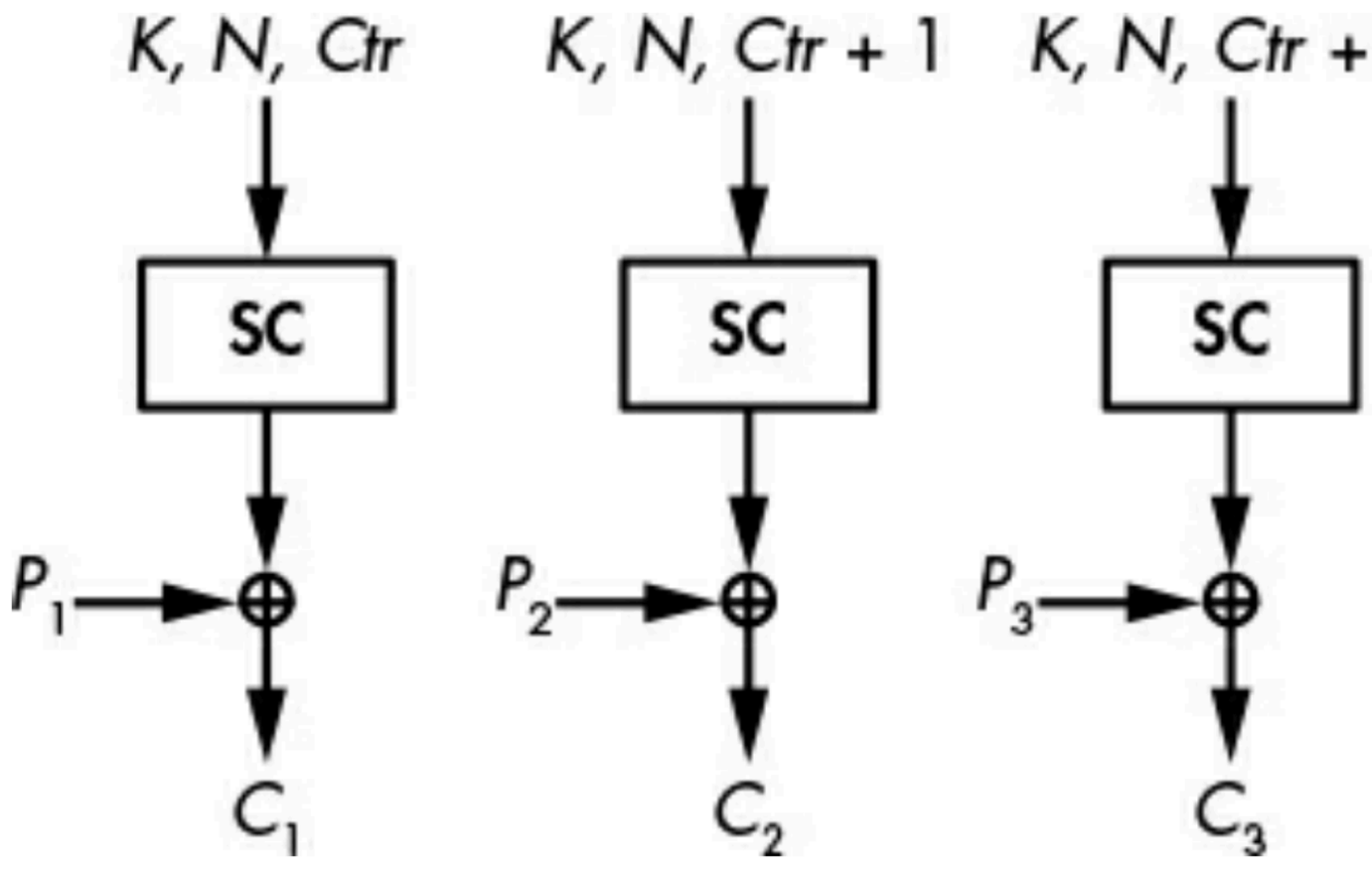


Figure 5-3: The counter-based stream cipher

Hardware-Oriented Stream Ciphers

Dedicated Hardware

- Electronic device that implements cryptographic algorithm
  - Can't be used for anything else
- Application-Specific Integrated Circuits (ASIC)
- Programmable Logic Devices (PLD)
- Field-Programmable Gate Arrays (FPGA)

Hardware v. Software

- Software implementations
  - Uses microprocessor instructions
  - Uses generic operations like ADD and SUB
  - Processes bytes, 32-bit words, or 64-bit words
- Hardware
  - Works with bits

Cost

- The first stream ciphers required less memory and fewer logic gates than block ciphers
- Cheaper to implement in hardware
- Stream cipher required less than 1000 gate-equivalents
- Software-oriented block ciphers required at least 10,000 gate-equivalents

DES v AES

- DES was optimized for hardware
  - In the 1970s, most target applications were hardware implementations
  - S-boxes are small and fast in hardware
  - Inefficient in software
- AES deals with bytes
  - More efficient than DES in software

- ASIC - fastest, costs $2,000 - $5,000
  - 100x faster than a CPU for Bitcoin
- GPU - not as fast but cheaper
- CPU - too slow to keep up
  - Link Ch 5d

Shift Register

- Left-shift operation **<<1**
- Starting with a register of 255
- **11111111 << 1** outputs **1**
- **11111110 << 1** outputs **1**
- **11111100 << 1** outputs **1**
- **11111000 << 1** outputs **1**

Shift Register

- Left-shift operation **<<1**
- Starting with a register of 170
- **10101010 << 1** outputs **1**
- **01010100 << 1** outputs **0**
- **10101000 << 1** outputs **1**
- **01010000 << 1** outputs **0**

Feedback Shift Register

- A register **R** of bits with an update ***feedback*** function **f** that produces a pseudorandom bit
- State stored in the register
- Each ***update*** uses the ***feedback*** function to change the state and output one bit

$$R_{i+1} = (R_t << 1) | \mathbf{f}(R_t)$$

| is the logical OR operator and << is the shift operator

4-Bit Example

- **R** has 4 bits; **f** XORs all four bits
  - **1100** << **f**(1100) = 0
  - **1000** << **f**(1000) = 1
  - **0001** << **f**(0001) = 1
  - **0011** << **f**(0011) = 0
  - **0110** << **f**(0110) = 0
  - **1100** ***it repeats from here on***

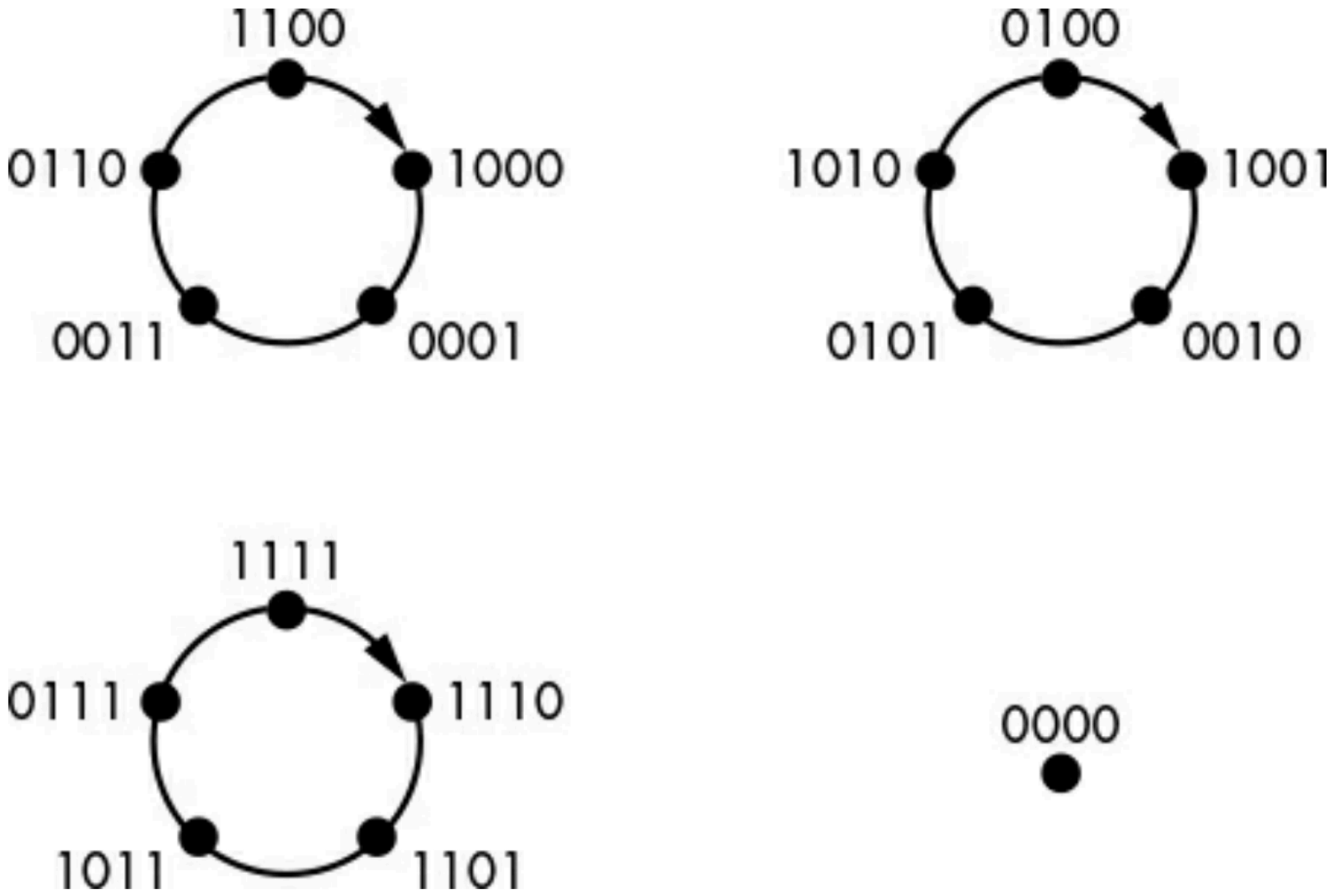


Figure 5-4: Cycles of the FSR whose feedback function XORs the 4 bits together

Linear Feedback Shift Register (LFSR)

- **f** is the XOR of some bits in the state
- The 4-bit example is an LFSR with a period of 5
- The longest possible period of an LFSR is
  - $2^n - 1$ (n is number of bits in R)
  - Because the all-zero register always repeats itself

Another 4-Bit LFSR

- **f** XORs only bits 1, 3, and 4 from the right
  - 1101 (where the ones are)

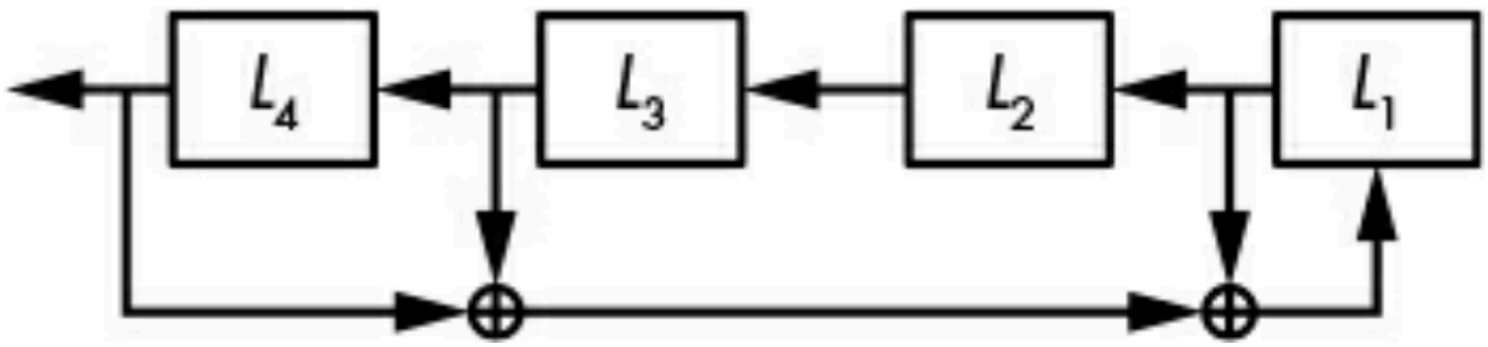


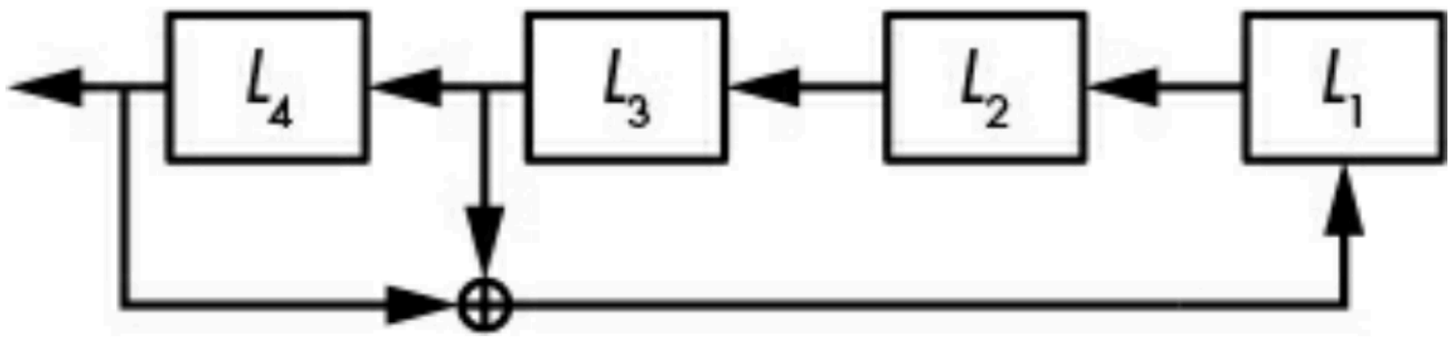
Figure 5-5: An LFSR with the feedback polynomial $1 + X + X^3 + X^4$

Another 4-Bit LFSR

- **R** has 4 bits; **f** XORs bits 1, 3, and 4
  - **0001** << **f**(0001) = 1
  - **0011** << **f**(0011) = 1
  - **0111** << **f**(0111) = 0
  - **1110** << **f**(1110) = 0
  - **1100** << **f**(1100) = 0
  - **1000** << **f**(1000) = 1
  - **0001**
- Period is 6 (error in textbook, see errata)

LFSR with Maximal Period

- **R** has 4 bits; **f** XORs bits 3 and 4
  - **0001** << **f**(0001) = 0
  - **0010** << **f**(0010) = 0
  - **0100** << **f**(0100) = 1
  - **1001** << **f**(1001) = 1



LFSR with Maximal Period

0 0 0 1	0 0 1 1	0 1 0 1	1 1 1 0
0 0 1 0	0 1 1 0	1 0 1 1	1 1 0 0
0 1 0 0	1 1 0 1	0 1 1 1	1 0 0 0
1 0 0 1	1 0 1 0	1 1 1 1	0 0 0 1

LFSRs Are Insecure

- If **R** has **n** bits
- An attacker only needs **n** output bits to recover the initial state
- Using the "Berlekamp-Massey" algorithm
  - To solve the equation defining **f**

Filtered LFSR

- Pass output bits through a nonlinear function **g**
  - Uses AND or OR along with XOR, such as
    - **(L_1 AND L_2) XOR (L_3 AND L_4)**

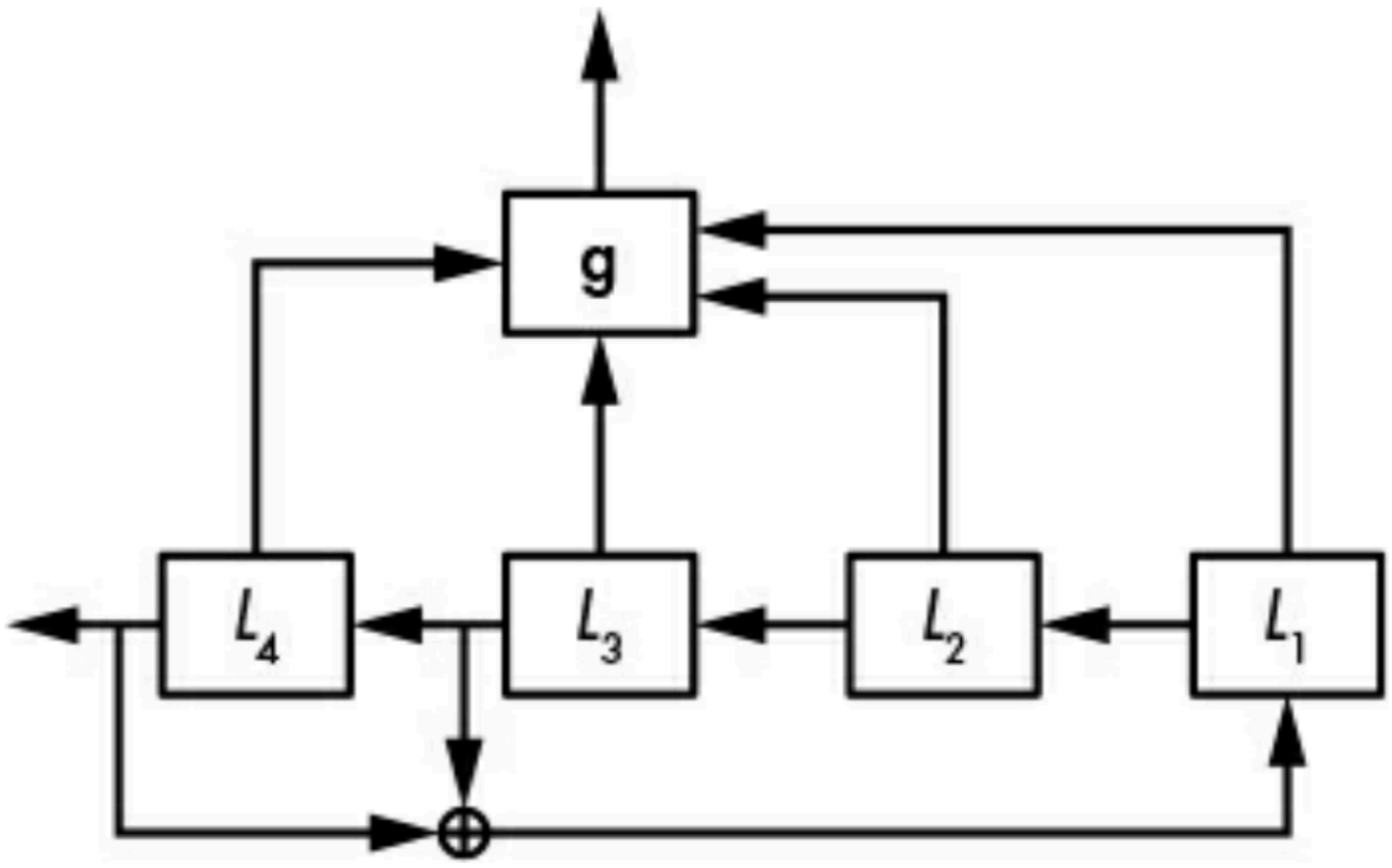


Figure 5-7: A filtered LFSR

Filtered LFSRs are Still Weak

- **g** function hides the linearity of the system
- Better than LFSRs but still breakable with more complex attacks
  - *Algebraic attacks* solving the nonlinear equations
  - *Cube attacks* using derivatives
  - *Fast correlation attacks* exploit nonlinear functions that behave like linear ones

Nonlinear FSRs

- Replace **f** with a nonlinear function like

  L_1 XOR L_2 XOR (L_1 AND L_2) XOR (L_3 AND L_4)

- Makes it very complex to attack or analyze
- No way to calculate the period or to know if it's maximal
- Combining LFSR and NFSR can ensure maximal period and cryptographic strength

Grain-128a

- 128-bit LFSR, 128-bit NFSR, filter function **h**

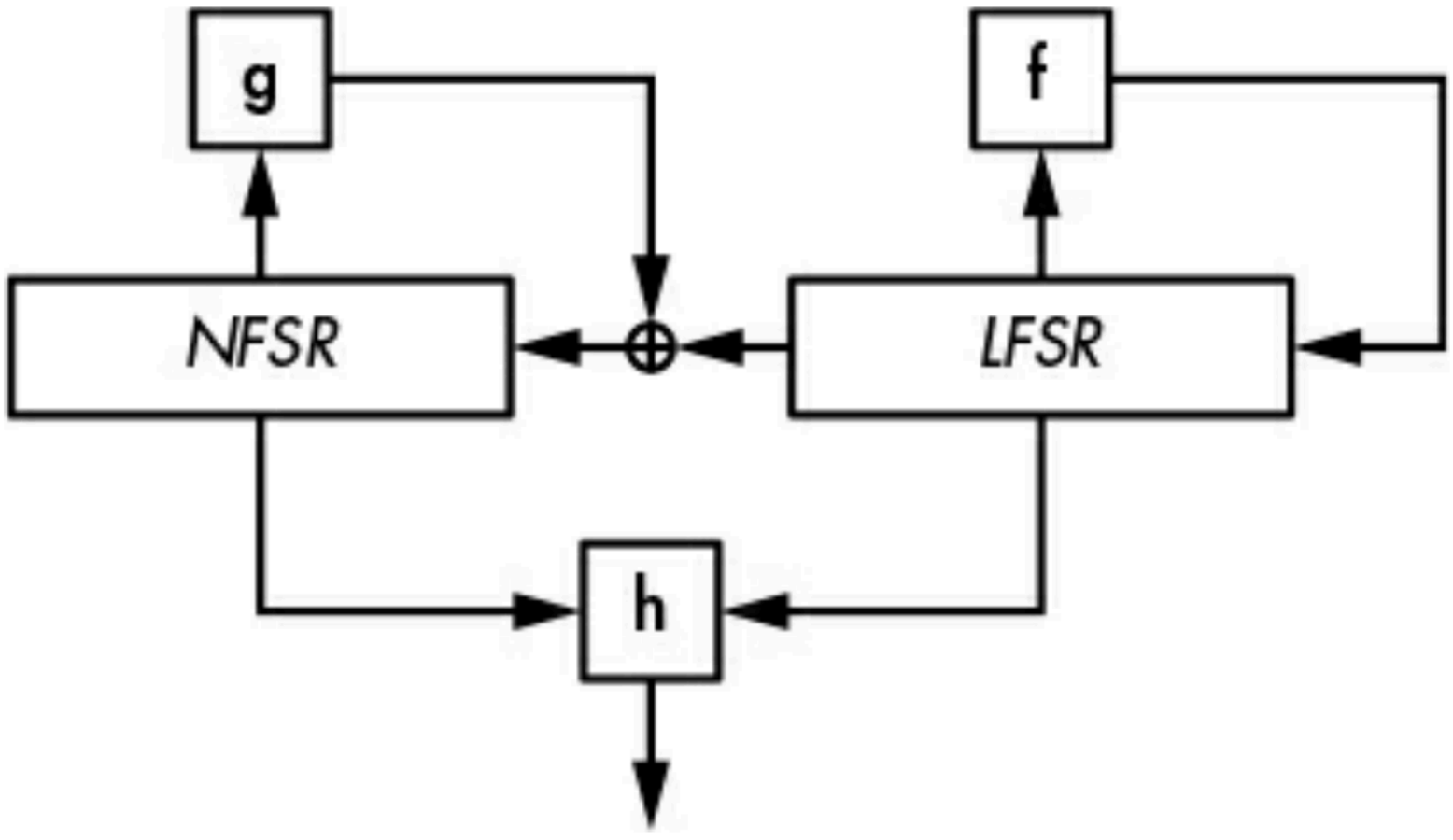


Figure 5-8: The mechanism of Grain-128a, with a 128-bit NFSR and a 128-bit LFSR

Grain-128a

- The LFSR has period $2^{128} - 1$
- The NSFR and **h** provide cryptographic strength
- Grain-128a takes a 128-bit **key** and a 96-bit **nonce**
  - Initial state of NSFR is **key**
  - Initial state of LFSR is **nonce,** 31x**0**, **1**

Grain-128a

- LFSR feedback function
  - + is XOR, N_iN_j means AND

$$\mathbf{f}(L) = L_{32} + L_{47} + L_{58} + L_{90} + L_{121} + L_{128}$$

- NSFR feedback function

$$\mathbf{g}(N) = N_{32} + N_{37} + N_{72} + N_{102} + N_{128} + N_{44}N_{60} + N_{61}N_{125} + N_{63}N_{67} + N_{69}N_{101}$$
$$+ N_{80}N_{88} + N_{110}N_{111} + N_{115}N_{117} + N_{46}N_{50}N_{58} + N_{103}N_{104}N_{106} + N_{33}N_{35}N_{36}N_{40}$$

Security of Grain-128a

- Grain-128 was proposed in 2006
  - Shown to be susceptible to side-channel fault attacks
- Grain-128a proposed in 2011
  - There is no known attack against it yet
- Used in proprietary industrial systems

A5/1

- Stream cipher used to encrypt voice communications in 2G mobile standard
- Created in 1987, but kept secret
- Some NATO signal intelligence agencies wanted to keep GSM encryption weak in the 1980s
- **A5/2** was a deliberately weakened version for certain export regions

A5/1

- A5/1's design was leaked in 1994 and it was reverse engineered in 1999
- In 2014, 7.2 billion GSM customers relied on A5/1 to protect their phone conversations
- A5/1 has serious security weaknesses
- The NSA can routinely decrypt A5/1 messages

Three LFSRs

- The three LFSRs have a total of 64 bits
- They use these functions

$$1 + X^{14} + X^{17} + X^{18} + X^{19}$$
$$1 + X^{21} + X^{22}$$
$$1 + X^{8} + X^{21} + X^{22} + X^{23}$$

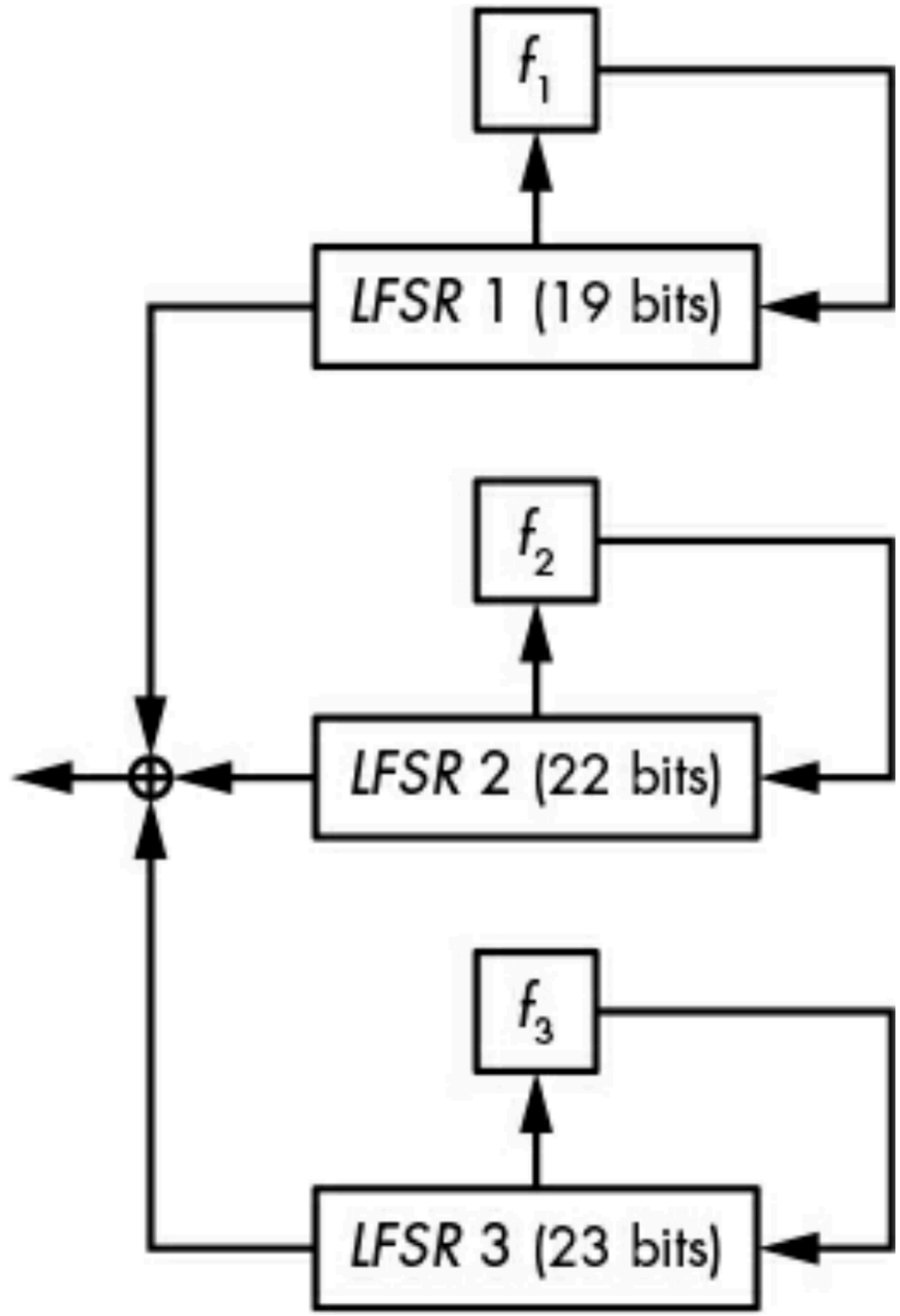


Figure 5-9: The A5/1 cipher

Updating

- Instead of updating all three LFSRs at each clock cycle
- Uses three "clocking bits" to determine how many to update

1. **Checks the value of the ninth bit of LFSR 1, the 11th bit of LFSR 2, and the 11th bit of LFSR 3, called the *clocking bits*. Of those three bits, either all have the same value (1 or 0) or exactly two have the same value.**
2. **Clocks the registers whose clocking bits are equal to the majority value, 0 or 1. Either two or three LFSRs are clocked at each update.**

Brute Force Attack

- Try all 64-bit keys
- 2^{64} operations
- There's no defense against this except using a longer key
- But there are faster attacks against A5/1

Attacks on A5/1

- Subtle attacks
  - Exploit simplicity to reduce the problem
- Brutal attacks
  - Exploit short key:
    64-bit key and 22-bit nonce

Subtle Attacks

- Brute-force internal states, one by one
- Takes 2^{52} operations, much less than 2^{64}
- Works but it's slow; taking hours even on a cluster of dedicated hardware devices

```
For all 2^19 values of LFSR 1's initial state
    For all 2^22 values of LFSR 2's initial state
        For all 2^11 values of LFSR 3's clocking bit during the first 11 clocks
            Reconstruct LFSR 3's initial state
            Test whether guess is correct; if yes, return; else continue
```

Brutal Attacks

- Time-Memory Trade-Off
- Internal state has 64 bits
- Precalculate something and make a big table

Codebook Attack

- Pre-calculate output for all 2^{64} possible keys
- Simply look up the key
- But this requires 2^{64} bytes of RAM
- 256 exabytes; 256 million Terabytes
- Not practical

TMTO Attack

- Make a smaller look-up table with part of the calculation
- Perform more computation during the online phase of the attack
- In 2010, researchers took two months to precalculate a 2 TB table
  - Using GPUs running 100,000 instances of A5/1 in parallel
- Enabled decryption in near-realtime

A5/1 is Obsolete

- A5/1 was abandoned for 3G and 4G networks
- 3G uses the KASUMI block cipher
- People are concerned that the NSA has a backdoor into it
  - Link Ch 5i

5a

Software-Oriented Stream Ciphers

Efficiency

- Software stream ciphers work with 32-bit or 64-bit words
  - Instead of individual bits
- More efficient on modern CPUs

Padding Oracles

- Stream ciphers have become more popular
- Because block ciphers have suffered from padding oracle attacks (in CBC mode)

RC4

- Designed in 1987 by Ron Rivest
- Reverse-engineered and leaked in 1994
- Was very widely used until 2015
- Used in WEP and TLS

IETF takes rifle off wall, grabs RC4 cipher's collar, goes behind shed

Vulnerable cipher is about to go to crypto heaven

By Richard Chirgwin 1 Dec 2014 at 02:29 11 SHARE ▼

Microsoft kills RC4 crypto

By Darren Pauli 10 Aug 2016 at 06:04 5 SHARE ▼

Microsoft has removed support from RC4, an ancient and easy-to-evade cipher, from its browsers.

How RC4 Works

- Internal state is 256 bytes, starting at
  - S[0]=0; S[1]=1; S[2]=2, ... S[255]=255
  - All it does is shuffle those values around like a deck of cards
  - No XOR or OR or AND

Key Schedule

- Initializes array from an n-byte key
- Result looks random but depends on the key

```
j = 0
# set S to the array S[0] = 0, S[1] = 1, . . . , S[255] = 255
S = range(256)
# iterate over i from 0 to 255
for i in range(256):
    # compute the sum of v
    j = (j + S[i] + K[i % n]) % 256
    # swap S[i] and S[j]
    S[i], S[j] = S[j], S[i]
```

Getting the Keystream

- To encode plaintext P that is m bytes long
- It generates M bytes of keystream like this

```
i = 0
j = 0
for b in range(m):
    i = (i + 1) % 256
    j = (j + S[i]) % 256
    S[i], S[j] = S[j], S[i]
    KS[b] = S[(S[i] + S[j]) % 256]
```

RC4 in WEP

- WEP encrypts payload data in 802.11 wireless frames
- All payloads use the same key, but a different nonce
- But RC4 doesn't support a nonce!
- So WEP treats the first 24 bits of the key as a nonce

Nonce Collisions

- A 24-bit nonce is too short
- After 2^{12} packets (4096), a duplicate nonce will likely be found
- Encrypted with identical keystreams

Nonce Exposure

- The nonce is sent in cleartext
- But that means the first 3 bytes of the key are also known
- This opens the door to deducing more bytes of the key
- Especially since wireless frames are very repetitive, so much of the plaintext is known

WEP Insecurity

- WEP was ratified in 1999
- WEP attacks appeared in 2001
- Many stronger attacks came later
- WEP is totally broken

RC4 in TLS

- TLS doesn't try to add a nonce
- It uses a unique 128-bit session key
- The flaw is intrinsic to RC4: statistical bias or non-randomness
- The second keystream byte is zero 1/128 of the time rather than 1/256

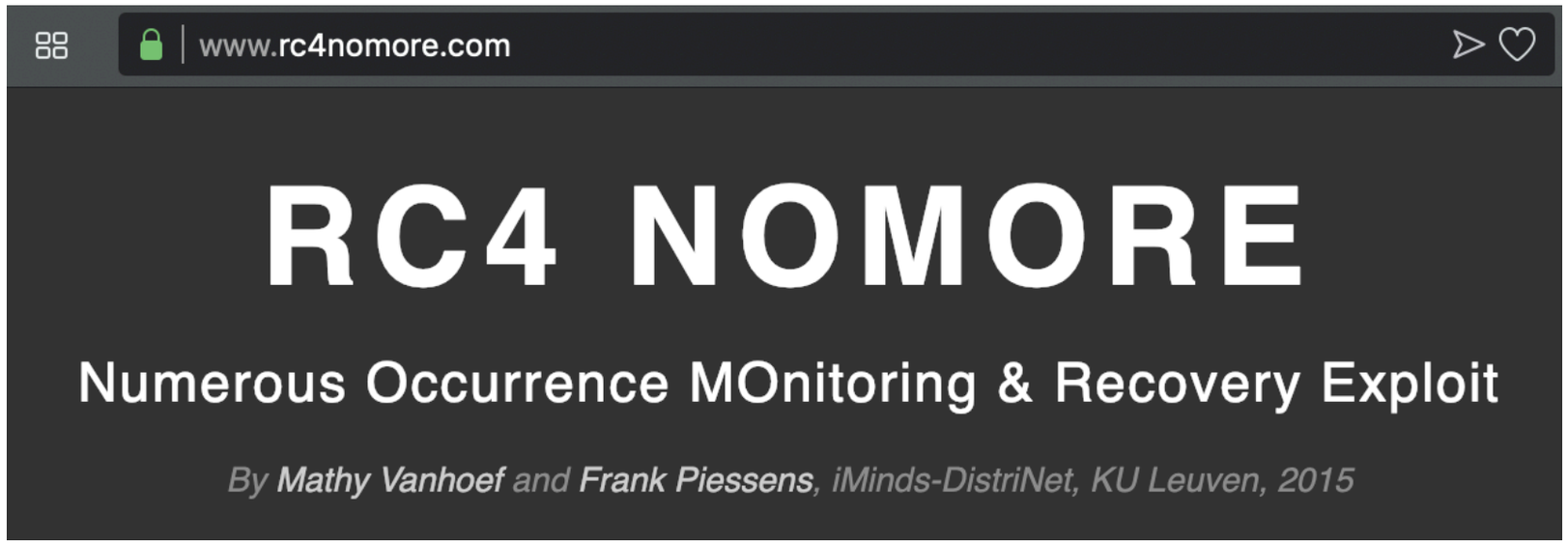
Exploits

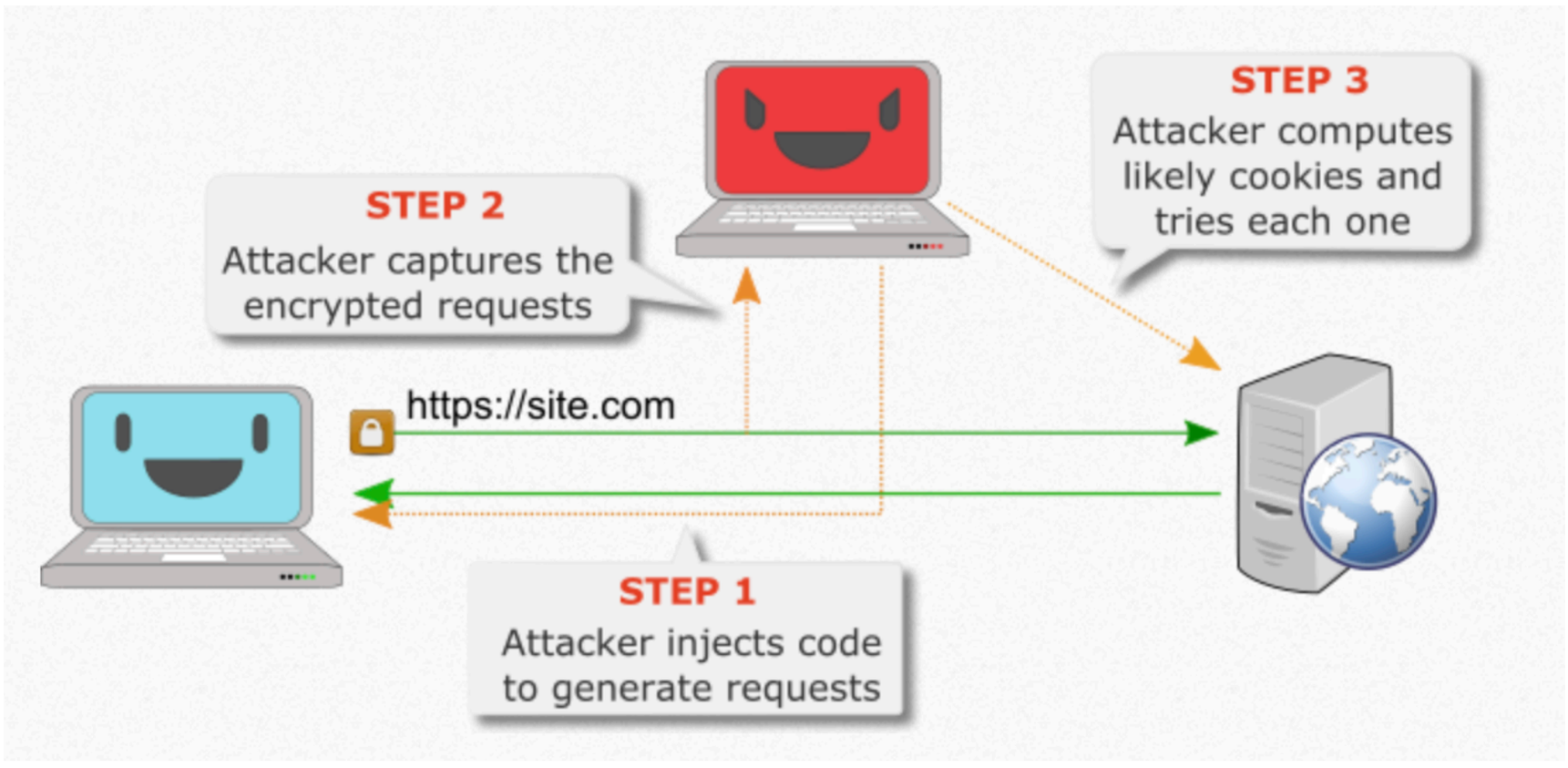
- Public exploits emerged in 2011
- NSA allegedly exploited them much earlier
- All the first 256 bytes are biased

Weakest Attack

- Simply collect many ciphertexts from the same plaintext, with different keys
- First keystream bytes are more likely to be zero than any other value
- So the ciphertext bytes are most likely to equal the plaintext byte
- Plaintext can be directly recovered without the key!
- But it takes millions of ciphertexts

RC4 Attacks


www.rc4nomore.com
RC4 NOMORE
Numerous Occurrence MOnitoring & Recovery Exploit
By Mathy Vanhoef and Frank Piessens, iMinds-DistriNet, KU Leuven, 2015


STEP 2
Attacker captures the encrypted requests
STEP 3
Attacker computes likely cookies and tries each one
https://site.com
STEP 1
Attacker injects code to generate requests

RC4 NOMORE

- Attacks TLS using RC4 (HTTPS)
- Can decrypt an encrypted cookie value
- In 75 hours
- Making 4450 requests per second
  - Link Ch 5l

Salsa20

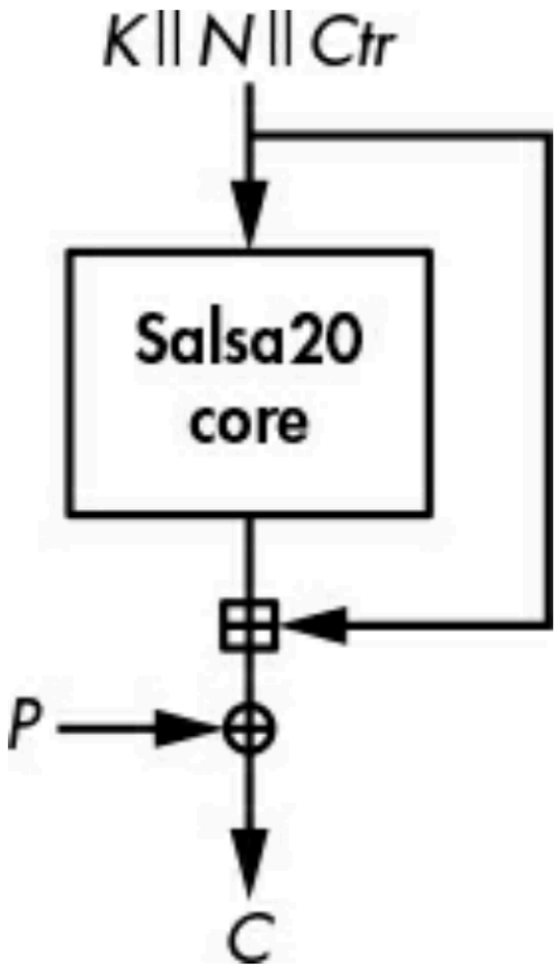
- Counter-based stream cipher
- Approved by eStream, an European competition to find secure ciphers (link Ch 5k)

eSTREAM portfolio [edit]

As of September 2011 the following ciphers make up the eSTREAM portfolio:[2]

Profile 1 (software)	Profile 2 (hardware)
HC-128 [1]	Grain [2]
Rabbit [3]	MICKEY [4]
Salsa20/12 [5]	Trivium [6]
SOSEMANUK [7]	

Salsa20 Encryption



- 512-bit blocks of plaintext
- A counter increments for each block
- Each block encrypted with a **key**, **nonce**, and **counter**
- Adds original secret state to output
- XORs that with the plaintext

Figure 5-10: Salsa20's encryption scheme for a 512-bit plaintext block

Quarter-Round Function

- Transforms four 32-bit words a, b, c, d
- <<< is wordwise left-rotation by the specified number of bits, which can be from 1 to 31

$$b = b \oplus \left[(a + d) <<< 7 \right]$$
$$c = c \oplus \left[(b + a) <<< 9 \right]$$
$$d = d \oplus \left[(c + b) <<< 13 \right]$$
$$a = a \oplus \left[(d + c) <<< 18 \right]$$

Bitwise Rotation

$$0\text{x}01234567 <<< 8 = 0\text{x}23456701$$

$$0\text{x}01234567 <<< 16 = 0\text{x}45670123$$

$$0\text{x}01234567 <<< 22 = 0\text{x}59c048d1$$

Salsa20's 512-bit State

- A 4x4 array of 32-bit words
- Initial state
  - 8 key words (256 bits)
  - 2 nonce words (64 bits)
  - 2 counter words
  - 4 fixed constant words

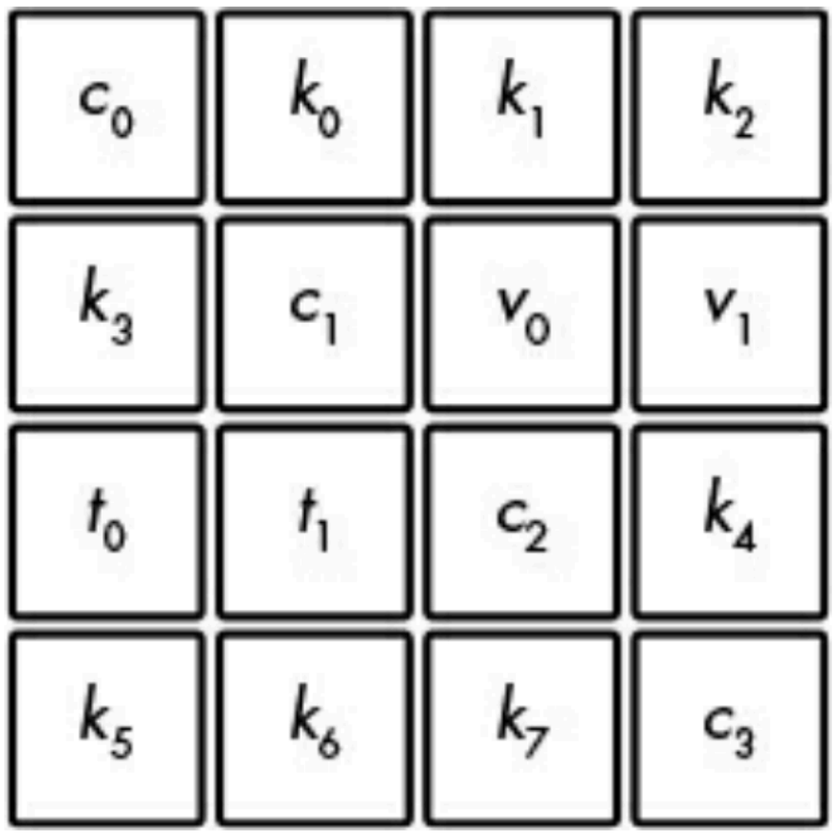


Figure 5-11: The initialization of Salsa20's state

Column and Rows

The column-round transforms the four columns like so:

$$\mathbf{QR}(x_0, x_1, x_2, x_3)$$
$$\mathbf{QR}(x_5, x_6, x_7, x_4)$$
$$\mathbf{QR}(x_{10}, x_{11}, x_8, x_9)$$
$$\mathbf{QR}(x_{15}, x_{12}, x_{13}, x_{14})$$

The row-round transforms the rows by doing the following:

$$\mathbf{QR}(x_0, x_1, x_2, x_3)$$
$$\mathbf{QR}(x_5, x_6, x_7, x_4)$$
$$\mathbf{QR}(x_{10}, x_{11}, x_8, x_9)$$
$$\mathbf{QR}(x_{15}, x_{12}, x_{13}, x_{14})$$

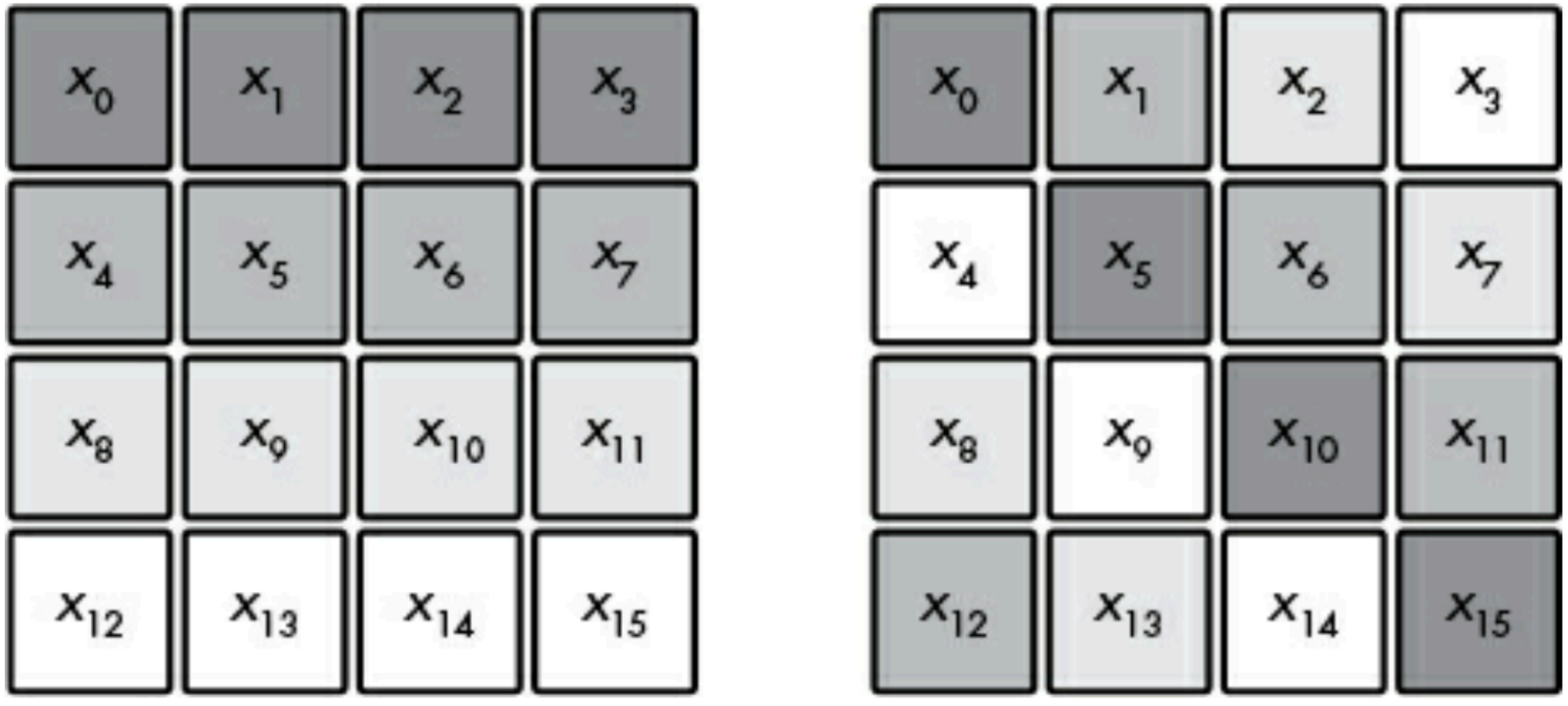


*Figure 5-12: Columns and rows transformed by Salsa20's quarter-round (**QR**) function*

Security of Salsa20

- Salsa20 uses 20 rounds
  - The Salsa20/12 variant uses only 12 rounds to make it faster
  - Versions with 8 or fewer rounds have some weak published attacks
  - No known attacks against versions with more rounds
    - Link Ch 5l

How Things Can Go Wrong

Nonce Reuse

- If the same key and nonce are used for two plaintexts
  - They are encrypted with identical keystreams
  - XOR the ciphertexts together and the keystream vanishes
  - You have the XOR of the two plaintexts
- Microsoft Office did this; and WEP did it by accident

Broken RC4 Implementation

- This code occasionally sets an internal state byte to zero
- Slowly destroys the state and stops encrypting

```
#define TOBYTE(x) (x) & 255
#define SWAP(x,y) do { x^=y; y^=x; x^=y; } while (0)

static unsigned char A[256];
static int i=0, j=0;

void init(char *passphrase) {
    int passlen = strlen(passphrase);
    for (i=0; i<256; i++)
        A[i] = i;
    for (i=0; i<256; i++) {
        j = TOBYTE(j + A[TOBYTE(i)] + passphrase[j % passlen]);
        SWAP(A[TOBYTE(i)], A[j]);
    }
    i = 0; j = 0;
}

unsigned char encrypt_one_byte(unsigned char c) {
    int k;
    i = TOBYTE(i+1);
    j = TOBYTE(j + A[i]);
    SWAP(A[i], A[j]);
    k = TOBYTE(A[i] + A[j]);
    return c ^ A[k];
}
```

Weak Ciphers Baked into Hardware

- Satphone standards used deliberately insecure "export" ciphers similar to A5/2
- Adopted by most commercial vendors in the early 2000s
- Link Ch 5l

Satellite Phone Encryption Calls Can be Cracked in Fractions of a Second

July 10, 2017 Swati Khandelwal

5b