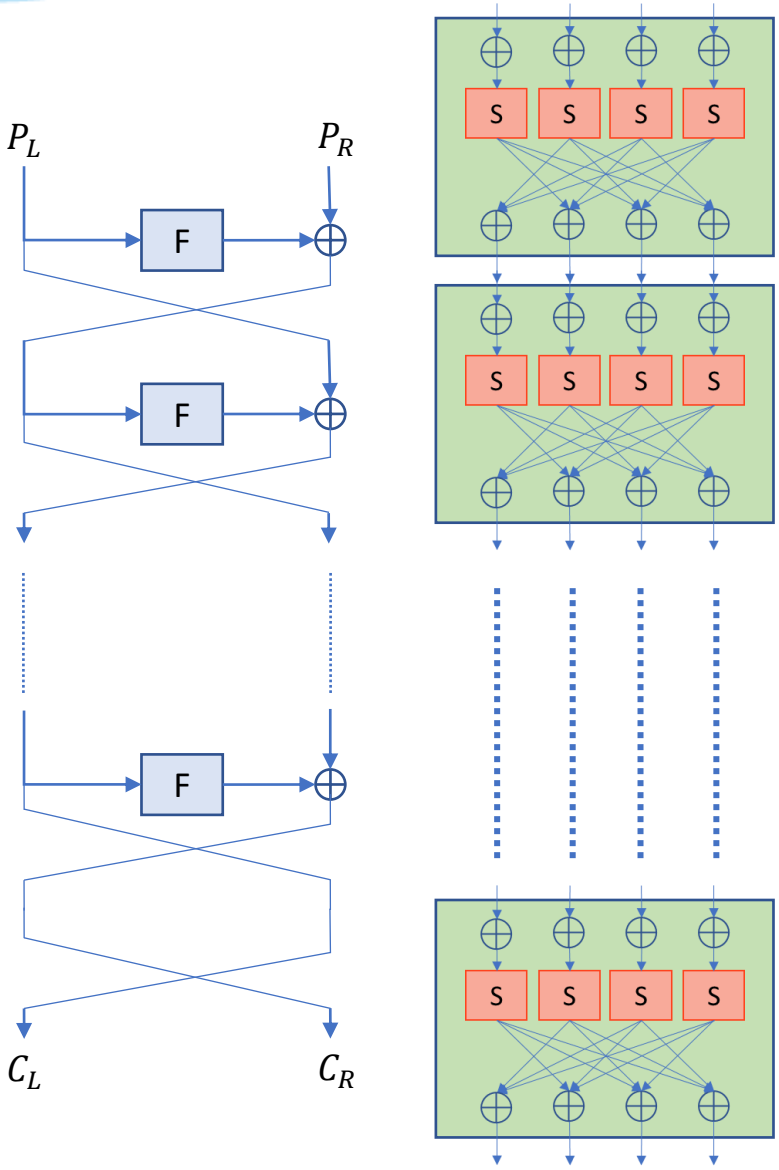
Cryptanalysis
(암호분석)

Chapter 4 – Part 2

2020.4

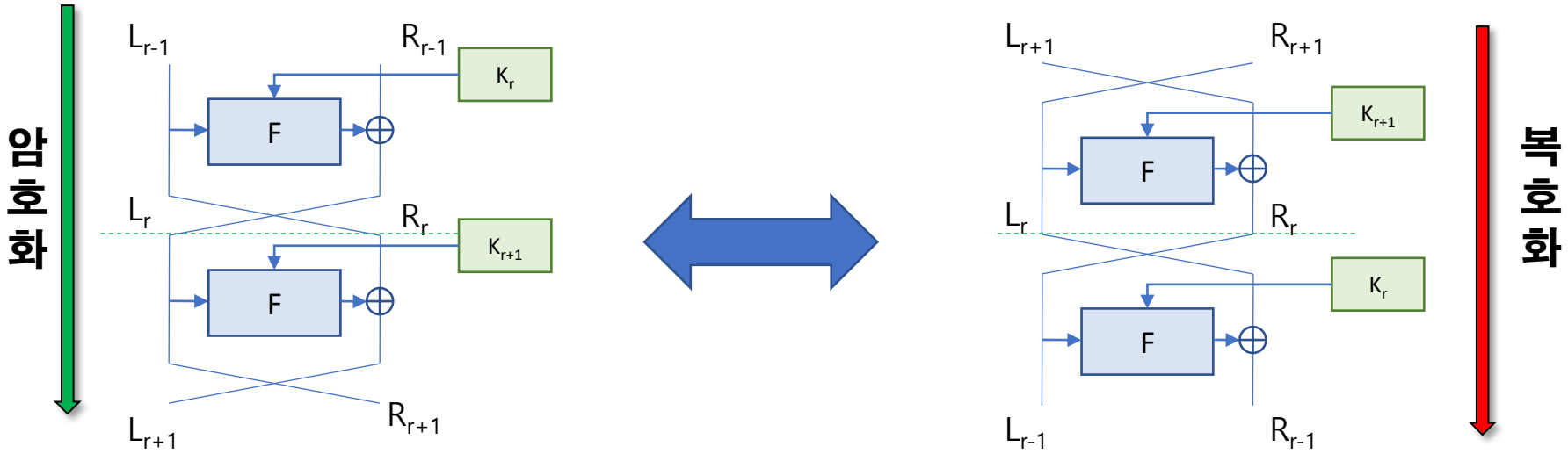
Contents

- Feistel Structure
- Toy Cipher TF20



Feistel Structure

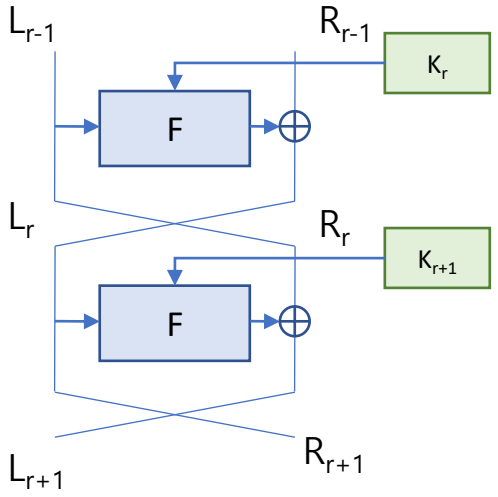
- 라운드 함수 F를 이용한 Feistel 구조
  - 암호화: $R_i = L_{i-1}, L_i = R_{i-1} \oplus F_K(L_{i-1})$
  - 복호화: $L_{i-1} = R_i, R_{i-1} = L_i \oplus F_K(R_i)$



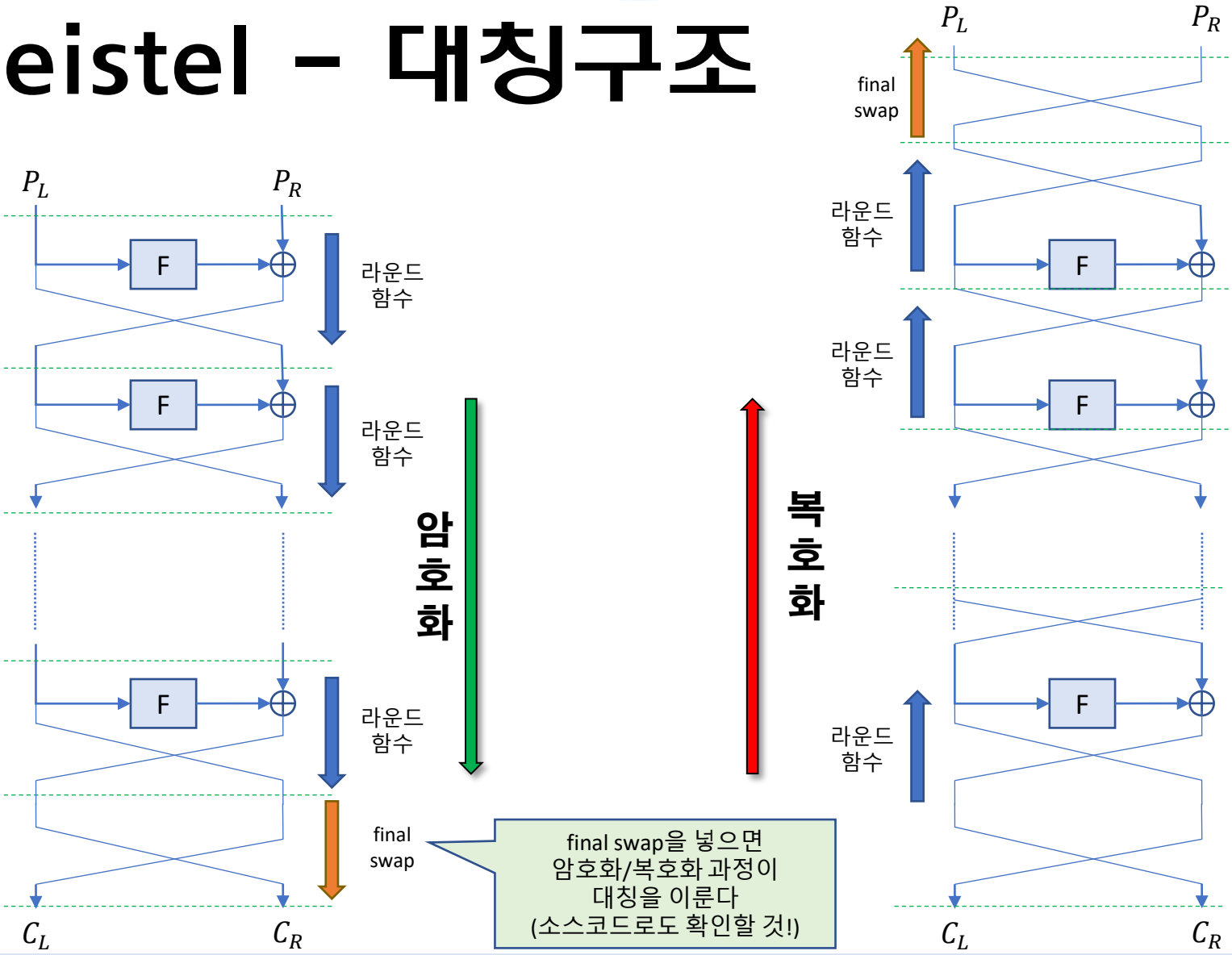
Feistel 구조 – 라운드 함수

- 라운드 함수 $F: \{0,1\}^n \rightarrow \{0,1\}^n$
  - 블록 크기: $2n$
  - 블록 구조: (L, R), 블록 크기: 라운드 함수 입출력의 두 배

- Feistel 라운드 함수의 특징
  - SPN보다 간단한 라운드 함수
    → 경량 구현에 적합함
    → SPN보다 많은 라운드를 사용함
  - 복호화에 역함수를 사용하지 않음
    → 라운드 함수가 일대일 대응일 필요 없음
    → 복호화 알고리즘의 구현이 간단함

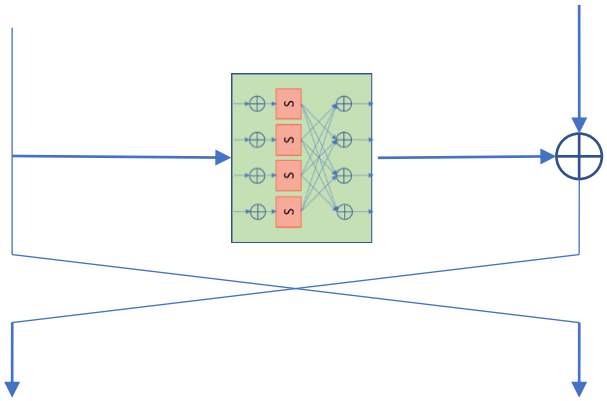


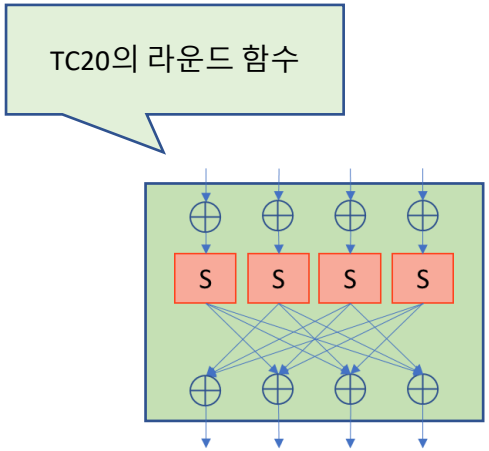
Feistel – 대칭구조



Toy Cipher – TF20

- 블록암호 TF20 디자인
  - 블록 크기: 64비트
  - 암호키: 32비트 (키 스케줄 없음)
  - 16 round Feistel structure
  - 라운드 함수: TC20의 라운드 함수

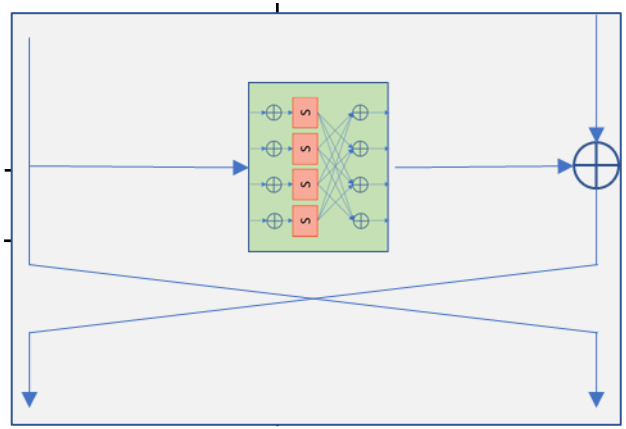




TF20 – 라운드 암호화 구현

라운드 암호화

```
#--- (L, R) --> (L', R')=( F(L,rkey)^R, L )
def Enc_Round(inL, inR, rkey):
    outL = outR = [0, 0, 0, 0]
    outR = inL
    outF = Round_F(inL, rkey)
    for i in range(len(outF)):
        outL[i] = outF[i] ^ inR[i]

    return outL, outR
```



라운드 복호화

```
#--- (L', R') --> (L, R)=( R', F(R',rkey)^L' )
def Dec_Round(inL, inR, rkey):
    outL = outR = [0, 0, 0, 0]
    outL = inR
    outF = Round_F(inR, rkey)
    for i in range(len(outF)):
        outR[i] = outF[i] ^ inL[i]
    return outL, outR
```

TF20 – 암호화 구현 (방법1)

- 16라운드 암호화 과정

```
def TF20_Enc(input_state, key):
    state = input_state
    stateL = state[:4]
    stateR = state[4:]

    numRound = 16 # 라운드 수

    for i in range(0, numRound):
        stateL, stateR = Enc_Round(stateL, stateR, key)

    state = stateL + stateR
    return state
```

입력 평문(리스트)을
왼쪽, 오른쪽 4바이트씩 나눈다.

왼쪽, 오른쪽 리스트를 합쳐
하나의 리스트로 만든다.

TF20 – 복호화 구현 (방법1)

- 16라운드 복호화 과정

```
def TF20_Dec(input_state, key):
    state = input_state
    stateL = state[:4]
    stateR = state[4:]

    numRound = 16 # 라운드 수

    for i in range(0, numRound):
        stateL, stateR = Dec_Round(stateL, stateR, key)

    state = stateL + stateR
    return state
```

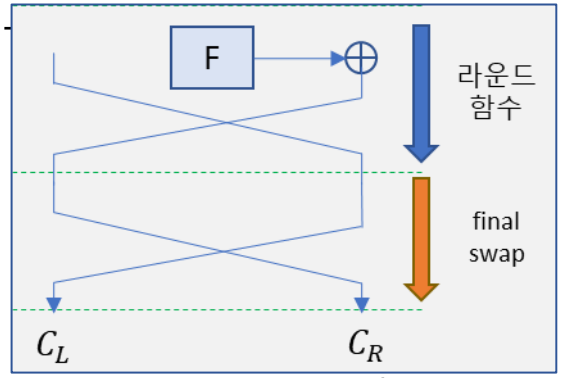
호출하는
라운드 함수만 달라진다.
(라운드 함수의 구조는 거의 같다)

TF20 – 암호화 구현 (방법2)

엄밀히 말하면
(방법1)과 (방법2)의
알고리즘은 다른 것이다.

- Final swap을 넣어 대칭구조로 변경한 TF20

```
def TF20_Enc(input_state, key):
    state = input_state
    stateL = state[:4]
    stateR = state[4:]

    numRound = 16 # 라운드 수
    for i in range(0, numRound):
        stateL, stateR = Enc_Round(stateL, stateR, key)

    #- final swap
    stateL, stateR = stateR, stateL

    state = stateL + stateR
    return state
```



마지막 라운드 후
좌우를 바꾼다.

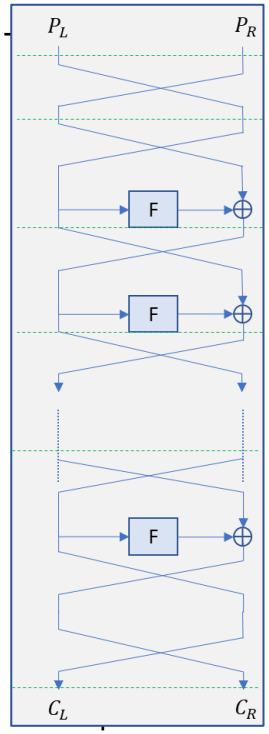
TF20 – 복호화 구현 (방법2)

- Final swap을 넣어 대칭구조로 변경한 TF20

```
def TF20_Dec(input_state, key):
    state = input_state
    stateL = state[:4]
    stateR = state[4:]

    numRound = 16 # 라운드 수
    for i in range(0, numRound):
        stateL, stateR = Enc_Round(stateL, stateR, key)

    #- final swap
    stateL, stateR = stateR, stateL

    state = stateL + stateR
    return state
```

함수이름만 다르고 내용은 TF20_Enc()와 같다.

암호화 함수와 완전히 동일하다!!!



TF20 – 암복호화 예제

```
message = 'ABCDEFGH'
    key = [0, 1, 2, 3]
    input_state = [ ord(ch) for ch in message ]

    output_state = [ item for item in TF20_Enc(input_state, key)  ]

    print('message =', message)
    print('input plaintext =', input_state)
    print('output ciphertext =', output_state)

    dec_state = [ item for item in TF20_Dec(output_state, key)  ]

    print('decrypted state =', dec_state)
    byte1 = bytes(dec_state)
    str1 = byte1.decode('utf8')
    print('decrypted state =', str1)
```

```
intput = [65, 66, 67, 68] [69, 70, 71, 72]
message = ABCDEFGH
input plaintext = [65, 66, 67, 68, 69, 70, 71, 72]
output ciphertext = [12, 128, 14, 94, 112, 136, 114, 220]

decrypted state = [65, 66, 67, 68, 69, 70, 71, 72]
decrypted state = ABCDEFGH
```

TF20 – Review

- Toy Cipher TF20 구조
  - TC20의 라운드 함수를 재활용한 구조
  - 블록 크기: TC20(32비트) → TF20(64비트)

- TF20의 장점과 단점
  - 암호화 과정과 복호화 과정이 동일함
    → 라운드 함수의 암호화 과정만 구현하면 됨
  - 키 스케줄을 적용한다면, 라운드 함수에 입력되는 라운드 키만 순서를 바꾸면 됨
  - 암호키(32비트)가 블록 크기보다 작아 안전성이 낮음
  - 암호화 시작과 끝에 화이트닝(암호키 적용)이 결핍됨