국산 형태보존암호 FEA

김경호

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

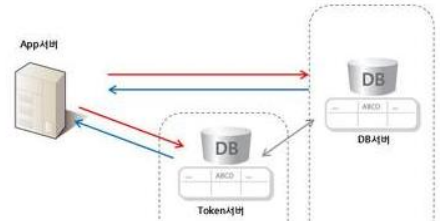
CryptoCraft LAB

1. 형태보존암호란?

- 형태보존암호
  - 평문과 암호문의 형태가 동일한 암호
    - Ex) 192.168.0.1 -> 456.332.1.235
    - 입력이 10진수 13자리 -> 출력도 10진수 13자리
    - 입력이 16진수 7자리 -> 출력도 16진수 7자리
  - 블록 사이즈가 고정된 블록 암호와 다르게 입력값과 출력값 크기 동일
  - 기존 블록 암호에 비해 메모리 공간 효율성 증가
    - 블록 사이즈에 따른 Padding이 없기 때문

1. 형태보존암호란

- 형태보존암호의 장점 (vs 블록암호)
  - 형태의 유지
    - 주민등록번호 및 신용카드 정보를 암호화 하는 경우 형태 보존의 장점
    - 블록암호의 경우 암호문이 ASCII 코드로 표현이 불가능 (0x30 ~ 0x39)
    - 블록암호는 정해진 블록 사이즈만큼 Padding 필요
  - 안정적인 토큰화 가능 (Tokenization)
    - Tokenization
      - 임의 생성 값으로 민감한 정보를 대체하는 기술(LINUX Password)
      - 중요 데이터는 따로 보관하고 토큰만 대조
    - 기존의 PRNG를 이용한 토큰화에 비해 보안성 증가 (Key 사용)
    - FPE는 복호화가 가능한 토큰이기 때문에 따로 보관 X

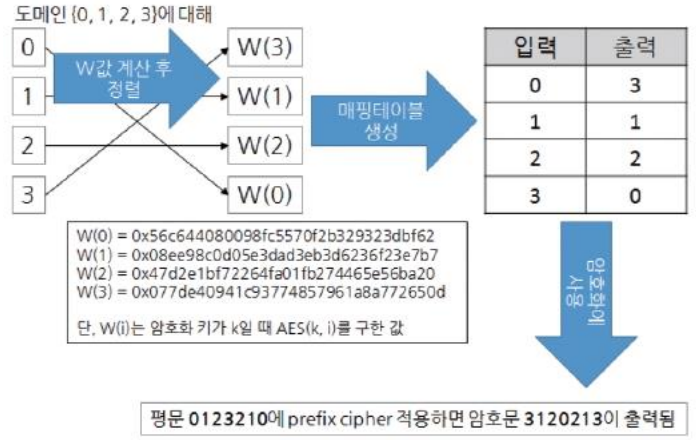


Tokenization 적용 방식	Token서버 보안 수준	DB서버 보안 수준	전체 시스템의 보안 수준
일반적인 Tokenization 방식 (SPIN 등)	안전 (Strong)	매우 취약 (Very Weak)	매우 취약 (Very Weak)
FPE를 적용한 Tokenization 방식	안전 (Strong)	안전 (Strong)	안전 (Strong)

1. 형태보존암호란

- 형태보존암호의 종류

  - Prefix cipher
    - 기존의 블록암호를 이용하여 암호화
    - 암호문을 오름차순 정렬 후 복호화 (결과를 테이블에 저장)
    - 장점 -> 짧은 길이의 데이터의 경우 효율적으로 사용
    - 단점 -> 길이가 길어지면 테이블 크기가 커짐(사용 불가)

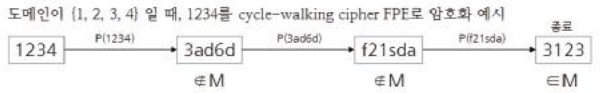


평문 **0123210**에 prefix cipher 적용하면 암호문 **3120213**이 출력됨

  - Cycle-walking cipher
    - 기존의 블록암호를 이용하여 암호화
    - 원본과 동일한 형태가 나올 때까지 알고리즘 반복
    - 장점 -> 테이블을 저장할 필요가 없음
    - 단점 -> 정확한 암호화 시간을 알 수 없음(언제 나올지 모르기 때문)

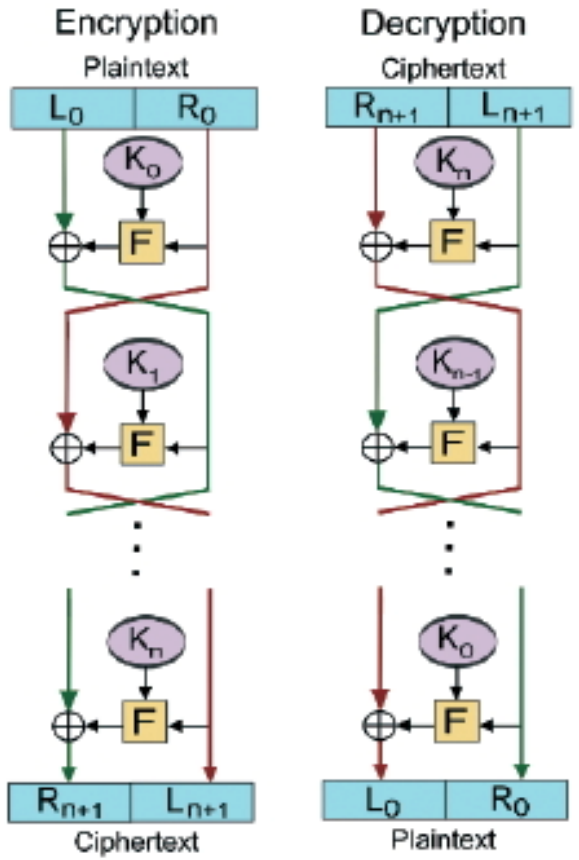
```
CycleWalkingFPE(x) {
  if P(x)가 M의 원소이면
    return P(x)
  else
    return CycleWalkingFPE(P(x))
}
```

x : 평문
P : AES 등과 같은 블록암호화 알고리즘
M : 도메인 원소들로 부터 순열 조합 가능한 집합



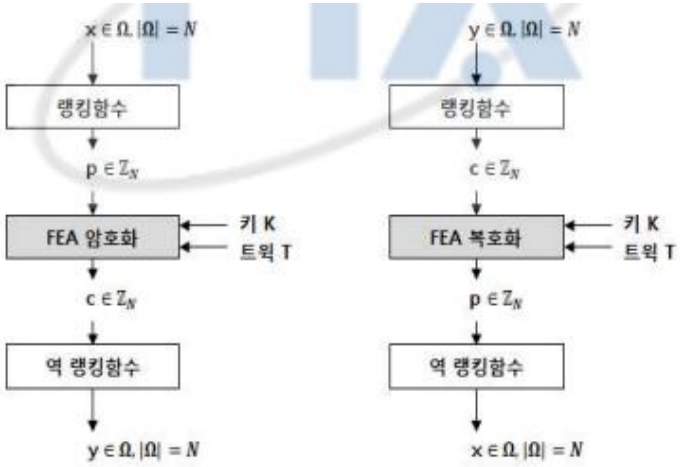
1. 형태보존암호란

- 형태보존암호의 종류
  - Feistel network cipher
    - Feistel 구조로 라운드마다 정해진 F() 함수를 이용하여 암호화
      - F() 함수의 보안성 = 암호의 보안성
    - 최근 FPE에서 가장 많이 사용되는 방식
    - 장점 -> 크기에 따라 성능 저하가 없음
    - NIST 표준 FPE인 FF1, FF3도 Feistel 구조 사용
    - 국산 FPE인 FEA도 Feistel 구조 사용



2. FEA(Format-Preserving Encryption)

- FEA (Format-Preserving Encryption)
  - Feistel 구조를 이용한 국산 FPE 알고리즘
  - 기존 블록암호를 사용하는 다른 알고리즘과 다르게 정해진 F() 함수 사용
  - Tweak을 이용하여 보안성 강화
    - Ex) 동일한 평문 -> 다른 암호문 출력
    - 비밀 데이터를 제외한 다른 데이터를 Tweak으로 사용
  - 암호화 이전에 랭킹함수를 통해 자연수로 변환
    - 기존 평문의 형태에서 자연수로 변환
    - 암호화 이후에 다시 기존 형태로 변환



2. FEA(Format-Preserving Encryption)

- FEA의 파라미터 정리

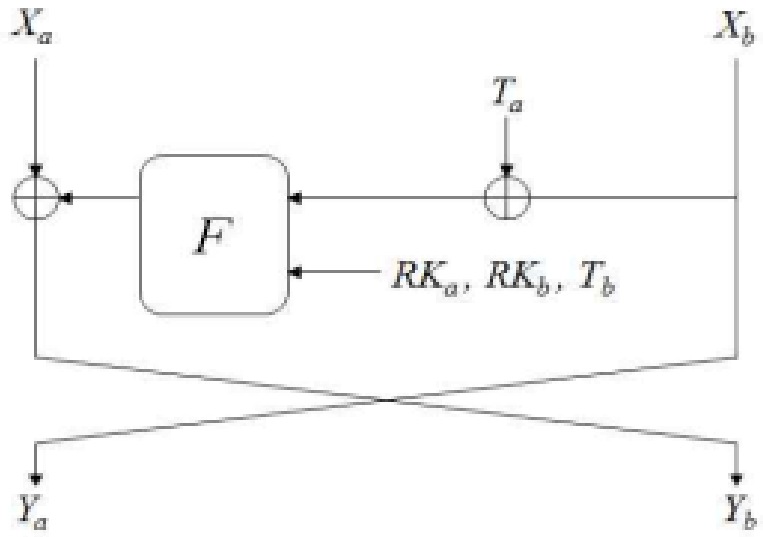
파라미터	내 용
알고리듬 타입	1 또는 2의 값을 가진다. 타입 1인 FEA-1은 제1형 TBC에 기반하여 정의되고, 타입 2인 FEA-2는 제2형 TBC에 기반하여 정의된다.
키 길이	128, 192, 또는 256 비트의 값을 가지며 목표하는 안전성에 따라 설정한다.
메시지 집합의 크기 N	2^8 이상, 2^{128} 이하의 정수
메시지의 최대 비트 길이 n	$\lceil \log_2(N) \rceil$
트윅 비트 길이	알고리듬 타입이 1인 경우는 128 − n, 알고리듬 타입이 2인 경우는 128로 정의된다.

파라미터	내 용
알고리듬 타입	FEA의 타입에 따라 1(FEA-1) 또는 2(FEA-2)로 설정된다.
키 길이	FEA의 키 길이와 동일하게 설정된다.
블록 비트 길이	n은 FEA와 동일하게 설정된다. $n_1 = \lceil n/2 \rceil$, $n_2 = \lfloor n/2 \rfloor$로 설정된다.
트윅 비트 길이	FEA와 동일하게 제1형은 128 − n, 제2형은 128로 설정된다.
라운드 수 r	알고리듬 타입과 키 길이에 따라 다음과 같이 정의된다.

타입 \ 키 길이	128	192	256
제1형	12	14	16
제2형	18	21	24

2. FEA(Format-Preserving Encryption)

- FEA 암호화 과정



1. 함수명 : TBC.Enc()

2. 입력 : 평문 X, 라운드 키 RK_a^i, RK_b^i, 라운드 트윅 T_a^i, T_b^i,

3. 처리 과정

$(X_a^1, X_b^1) \leftarrow \text{split}_E(X)$;
for i = 1 to r do
$X_a^{i+1} \leftarrow X_b^i$;
if i is odd then
$X_b^{i+1} \leftarrow X_a^i \oplus F_o(X_b^i \oplus T_a^i, RK_a^i, RK_b^i, T_b^i)$;
else
$X_b^{i+1} \leftarrow X_a^i \oplus F_e(X_b^i \oplus T_a^i, RK_a^i, RK_b^i, T_b^i)$;
end if
end for
$(Y_a, Y_b) \leftarrow \text{swap}(X_a^{r+1}, X_b^{r+1})$;
$Y \leftarrow \text{cat}(Y_a, Y_b)$;

4. 출력 : 암호문 Y

2. FEA(Format-Preserving Encryption)

- 치환 계층(SBL), 확산 계층(DL)

  - SBL()
    - AES Subbytes 연산과 비슷한 SBOX 참조 연산
    - 기약 다항식 $t^8 + t^4 + t^3 + t^2 + 1$를 이용하여 계산된 SBOX 사용
    - 입력 값이 0x1a -> 출력 값은 0xc3

  - DL()
    - 8 x 8 행렬 M과 입력 값 1 x 8 행렬의 곱셈 연산
    - 기약 다항식 $t^8 + t^6 + t^5 + t^4 + 1$을 이용한 곱셈 연산
    - 암호화 과정에서 가장 많은 시간 소요 예상

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	62	31	70	8e	bc	30	9c	78	e0	5c	ce	bb	42	ac	b8	df
1	29	e7	86	5f	ee	ba	3f	87	c0	36	c3	14	7c	ec	73	da
2	57	72	f6	77	98	3b	c5	c4	4c	52	81	20	15	97	26	fc
3	8b	3c	af	6e	c8	7e	f0	40	24	a1	b1	54	ff	ad	51	bd
4	c1	13	41	b5	6b	94	63	d6	de	6f	89	d2	a9	d4	17	38
5	a5	f2	e3	db	47	66	ed	cb	4e	d5	05	60	8c	06	92	a3
6	be	68	56	a7	80	32	fa	6c	8f	88	d9	50	0a	21	3d	75
7	71	01	e5	7a	c6	b9	82	64	d1	00	7d	2b	a0	1a	5e	f5
8	35	90	2f	2a	83	49	5a	a8	d8	8d	46	96	dc	b0	c9	dd
9	cd	65	44	c7	43	67	55	eb	e1	9d	34	74	b3	4a	ca	d7
a	79	bf	f7	99	6a	2d	ef	85	e2	5d	fe	11	0f	19	cc	e4
b	58	09	8a	1b	6d	91	9f	4b	61	2c	2e	cf	27	10	18	b7
c	1d	0c	9b	39	7f	d3	84	a4	f9	76	33	f4	f3	d0	07	0e
d	22	1f	fd	25	12	08	1e	4d	b6	b4	53	37	e8	b2	9e	93
e	02	e9	f1	3a	0b	fb	45	69	ea	f8	c2	1c	04	59	03	48
f	16	a2	4f	3e	9a	23	aa	ae	5b	e6	95	ab	7b	0d	28	a6

2. FEA(Format-Preserving Encryption)

- Key schedule
  - 키 길이에 따라 정해진 라운드의 라운드 키를 생성
  - 암호화 Type과 키 길이에 따라 정해진 라운드 상수 이용
  - Key.Init()
    - Key가 128 -> $K_a \| K_b = K$, $K_c = K_d = 0$
    - Key가 192 -> $K_a \| K_b \| K_c = K$, $K_d = 0$
    - Key가 256 -> $K_a \| K_b \| K_c \| K_d = K$

i	키 길이		
	128	192	256
1	71366FBD8EEF2E7D	D2F928B5C6C08B51	8F1C67DA8E609269
2	9063FF208A85D13F	4CBE190CDCC2962C	9B705F1835E0CDDC
3	FDB54B3C9A86CB08	D0A2A85F772C8A07	6BF524A08A50A621
4	F2EA772BE55E4DE0	E3FB1D49F5932802	6B3C821900ADAB39
5	7C8814F95B9F8D0B	047117EEE8007DFE	1F0EB84F4DE6881C
6	EB21FBFFCCBB8DF5	4390E40073A64C7D	887FBA6319CBF504
7		EE9FAB45168DDADC	51547790DD0B8145
8			AD7C1F118CA88090

i	키 길이		
	128	192	256
1	C9E3B39803F2F6AF	A4198D55053B7CB5	93C7673007E5ED5E
2	40F343267298B62D	BE1442D9B7E08DF0	81E6864CE5316C5B
3	8A0D175B8BAAFA2B	3D97EEEA5149358C	141A2EB71755F457
4	E7B876206DEBAC98	AA9782D20CC69850	CF70EC40DBD75930
5	559552FB4AFA1B10	5071F733039A8ED5	AB2AA5F695F43621
6	ED2EAE35C1382144	625C15071EA7BCA1	DA5D5C6B82704288
7	27573B291169B825	CF37D8F11024C664	4EAE765222D3704A
8	3E96CA16224AE8C5	86D094E21E74D0A5	7D2D942C4495D18A
9	1ACBDA11317C387E	47DF6E91FC91754B	3597B42262F870FD
10		1F0B2F23B88200E7	73D53787626CC076
11		29816E82B43E6464	4ADF41D8ECAFEE96
12			E59D0F633ACA9195

Init(K; K_a, K_b, K_c, K_d);
for i = 1 to $\lceil r/2 \rceil$ do
  $X \leftarrow K_a \oplus K_c \oplus RC_{type,|K|,i}$;
  $X \leftarrow SBL(X)$;
  $X \leftarrow DL(X)$;
  $Y \leftarrow K_b \oplus K_d \oplus n \oplus X$;
  $Y \leftarrow SBL(Y)$;
  $Y \leftarrow DL(Y)$;
  $X \leftarrow X \oplus Y$;
  $K_a \leftarrow K_a \oplus X$; $K_b \leftarrow K_b \oplus Y$;
  $K_c \leftarrow K_c \oplus X$; $K_d \leftarrow K_d \oplus Y$;
  $K_c \leftarrow K_c \oplus K_d$; $K_d \leftarrow K_d \oplus K_c$;
  $RK_a^{2i-1} \leftarrow K_a$; $RK_b^{2i-1} \leftarrow K_b$;
  $RK_a^{2i} \leftarrow K_c$; $RK_b^{2i} \leftarrow K_d$;
end for

2. FEA(Format-Preserving Encryption)

- Tweak schedule
  - FEA는 암호화 타입에 따라 트윅 길이가 다름
    - FEA-1 -> (128 – 블록 사이즈) bit
    - FEA-2 -> 128bit

if type = 1 then
  $T_L \leftarrow T[0 : 64 - n_2 - 1]$; $T_R \leftarrow T[64 - n_2 : 128 - n - 1]$;
  for i = 1 to r do
    $T_a^i \leftarrow 0$;
    if i is odd then
      $T_b^i \leftarrow T_L$;
    else
      $T_b^i \leftarrow T_R$;
    end if
  end for
else if type = 2 then
  $T_L \leftarrow T[0 : 63]$; $T_R \leftarrow T[64 : 127]$;
  for i = 1 to r do
    if $(i \equiv 1 \bmod 3)$ then
      $T_a^i \| T_b^i \leftarrow 0$;
    else if $(i \equiv 2 \bmod 3)$ then
      $T_a^i \| T_b^i \leftarrow T_L$;
    else
      $T_a^i \| T_b^i \leftarrow T_R$;
    end if
  end for
end if

2. FEA(Format-Preserving Encryption)

- 라운드 함수 F()
  - 홀수 라운드에서 m1 = n2, m2 = n1
  - 짝수 라운드에서 m1 = n1, m2 = n2

블록 비트 길이	n은 FEA와 동일하게 설정된다. $n_1 = \lceil n/2 \rceil$, $n_2 = \lfloor n/2 \rfloor$로 설정된다.

1. 함수명 : $F_o()$, $F_e()$
2. 입력 : $X_b \oplus T_a$ (m_1 비트), RK_a (64 비트), RK_b (64 비트), T_b ($(64 - m_1)$ 비트)
3. 처리 과정
   - $Y \leftarrow (X_b \oplus T_a) \| T_b$;
   - $Y \leftarrow Y \oplus RK_a$;
   - $Y \leftarrow SBL(Y)$;
   - $Y \leftarrow DL(Y)$;
   - $Y \leftarrow Y \oplus RK_b$;
   - $Y \leftarrow SBL(Y)$;
   - $Y \leftarrow DL(Y)$;
   - $Z \leftarrow Y[0 : (m_2 - 1)]$;
4. 출력 : Z (m_2 비트)

3. 제안 기법

- FEA 알고리즘에서 소요시간이 가장 큰 DL 함수 최적화
  - 반복적인 8bit X 8bit 곱셈
    - 기존 연구 논문[1] 단점 보완
      - Timing Attack 가능
    - PAGE와 유사한 기법의 곱셈기 vs 단순 8bit MUL 연산
  - Modulo reduction 필요
    - 사전테이블을 이용한 Modulo reduction

```
uint8_t GF_mul(u8 a, u8 b)
{
    uint8_t result = 0, t;
    while (a != 0)
    {
        if ((a & 1) != 0)
            result ^= b;
        t = (b & 0x80);
        b <<= 1;
        if (t != 0)
            b ^= 0x71;
        a >>= 1;
    }
    return result;
}
```

3. 제안 기법

- 소프트웨어 최적화
  - SBL() 연산의 SBOX 참조 연산에서 Index 최적화를 이용한 연산 감소
    - SBOX 주소 -> 0x0100
  - 암호화 전반적으로 이루어지는 비트 Split 연산을 어셈블리로 효율적 연산
- 부채널 공격 및 마스킹 연산 추가
  - 마스킹 연산 및 부채널 대응성을 가지는 곱셈기로 교체

4. 결론

- 소프트웨어 최적화를 통한 암호화 연산 시간 감소
- 부채널 공격 및 대응성
- FEA 뿐만 아니라 FF1, FF3와 같은 다양한 구현 결과 필요
- 기존 FF1, FF3의 새로운 취약점 연구 및 적용 가능성 파악
- 다양한 플랫폼에서 적용

Q & A