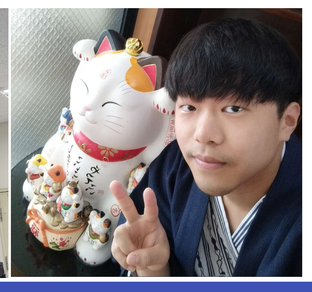
의사랜덤함수(PRF)의 정의와 시각화에 대한 연구

- 랜덤성의 복잡성과 우아함-



지용현, 김동현

국민대학교 과학기술대학 정보보안암호수학과

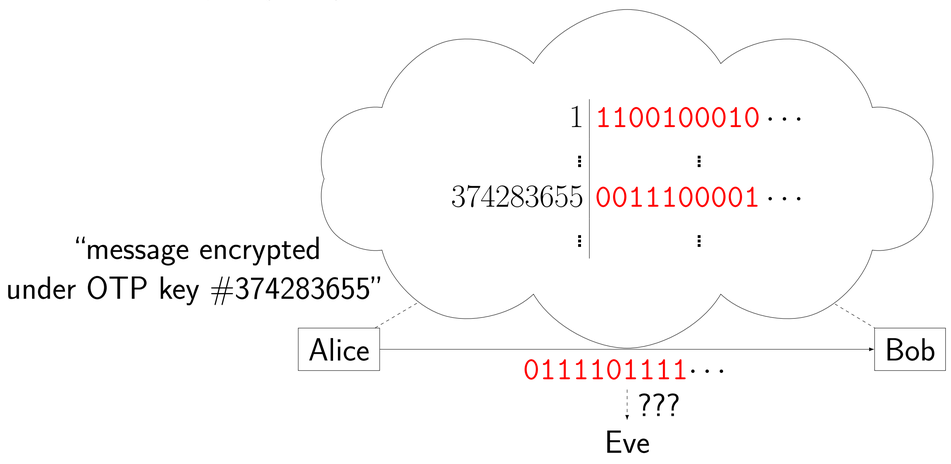
hacker3740@gmail.com, eastchord0729@gmail.com

소개

현대 암호학에서 블록 암호는 안전한 의사 랜덤 치환 (Pseudo Random Permutation, PRP)임을 기대합니다. 블록 암호의 안전성을 논하기 위해서는 안전한 의사 랜덤 치환 (PRP)과 의사 랜덤 함수 (Pseudo Random Function, PRF)의 이해가 필요합니다. 본 연구에서는 안전한 PRF의 정의를 의사 코드 형식으로 표현하고, 랜덤 함수 (Random Function, RF)의 현실적인 모델링 방식을 소개한 후 안전하지 않은 PRF를 비교 분석합니다.

PRF? 너 누구니?

Alice와 Bob이 무한한 공유된 난수 테이블을 가질 수 있다면, 이를 λ-bit 크기의 조각으로 나누어 각각을 one-time pad (OTP) 방식을 사용할 수 있습니다.



Alice는 특정 번호의 키를 사용해 메시지를 암호화하고, "Bob! 이 메시지는 #374283655 키를 사용해 암호화했어"라고 알립니다. Bob은 목록에서 #374283655 위치를 찾아 메시지를 해독합니다. Alice가 키를 반복해서 사용하지 않는다면, 공유된 난수 테이블에 접근할 수 없는 도청자 Eve는 암호화된 메시지에 대해 아무 것도 알 수 없습니다.

그러나 무한한 공유된 난수 테이블을 상상하는 것은 *비현실적*입니다. 그러나 실제로는 지수승 만큼의 많은 난수도 마치 무한한 것처럼 유용할 수 있습니다. 예를 들어 '2λ' 만큼의 암호 키를 가진 표를 공유한다면, 필요한 모든 메시지를 암호화하는 데 충분합니다.

의사 랜덤 함수(Pseudo-random Function, PRF)는 Alice와 Bob이 실제로 거대한 난수 테이블을 공유하는 효과를 낼 수 있게 하는 도구입니다. 무한 난수 테이블과 의사 랜덤 함수를 다음과 같이 묘사할 수 있습니다.

무한 난수테이블과 의사 난수 함수를 다음과 같이 묘사할 수 있습니다.

for $x \in \{0,1\}^{\text{Input}}$
$T[x] \leftarrow \{0,1\}^{\text{Out}}$

Lookup $(x \in \{0,1\}^{\text{Input}})$:
return $T[x]$

Array T

Index $x_{\text{Input}-1}\cdots x_1x_0$	Output $T[x]$	Binary String Mapping
000	$T[000]$	1100100010···
001	$T[001]$	0110000101···
⋮	⋮	

$k \leftarrow \{0,1\}^{\lambda}$

Lookup $(x \in \{0,1\}^{\text{Input}})$:
return $F(k,x)$

Pseudorandom Function F

Input $(x \in \{0,1\}^{\text{Input}})$
Seed $(k \in \{0,1\}^{\lambda})$
→ Output $(\{0,1\}^{\text{Out}})$

위 그림에서 묘사된 무한 난수 테이블의 경우 구현 측면에서 바라보았을 때 $2^{\text{Out}\cdot 2^{\text{Idx}}}$ 크기의 테이블을 가지고 있어야 합니다. 시뮬레이션을 해보기 위하여 다음과 같이 On-The-Fly 방식으로 테이블을 구현합니다.

$T := \{\}$
Lookup $(x \in \{0,1\}^{\text{Input}})$:
if $T[x]$ undefined:
$T[x] \leftarrow \{0,1\}^{\text{Out}}$
return $T[x]$

```python
class RandomFunction:
    def init(self, y_space):
        self.y_space = y_space
        self.results = {}  # to store results

    def lookup(self, x):
        if x not in self.results:
            self.results[x] = np.random.choice(self.y_space)
        return self.results[x]
```

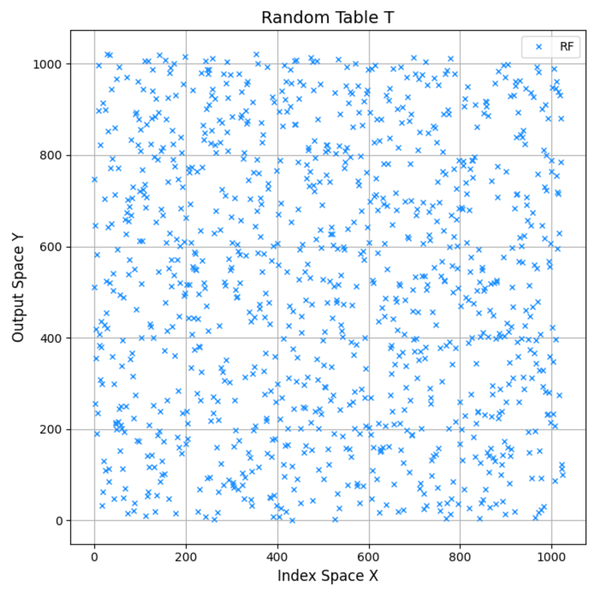
랜덤성 모듈화

$\texttt{Input} = 10 = \texttt{Out}$ 인 상황에서 다음과 같은 결과를 얻습니다.

```python
input = np.arange(2**10)
out = np.arange(2**10)

rf_ex = RandomFunction(out)
rf_out = np.array([rf_ex(x)
  for x in input])

```



위 그림의 왼쪽의 RF output에 저장된 값들을 오른쪽과 같이 값들을 시각화하여 분석한 결과 랜덤성이 나타나 보이는 것으로 확인하였습니다.

나쁜(X) PRF의 예시

위 그림과 같이 랜덤한 것처럼 보이는 의사 랜덤 함수 F을 어떻게 설계해야 할까요? 다음과 같은 접근을 생각해보겠습니다. 단, 여기서 $G(k)$는 안전한 의사 난수 생성기 (PRG)로 가정합니다.

$F(k,x)$:
return $G(k) \oplus x$

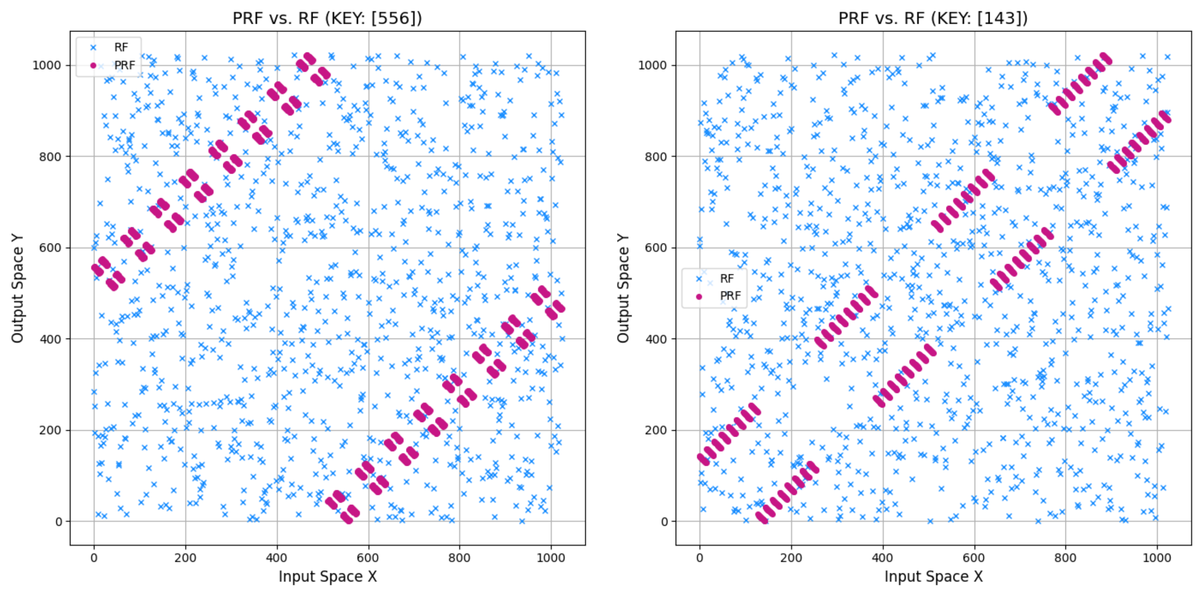
```python
key = np.random.randint(0, 2**10, size=1)

def prf(x, k):
    x_k = np.bitwise_xor(x, k)
    return x_k

input = np.arange(2**10)

prf_out = np.array([prf(x, key) for x in
  input])
```

위 그림의 오른쪽의 PRF output에 저장된 값들을 오른쪽과 같이 값들을 시각화하여 분석한 결과는 다음과 같습니다.



랜덤성을 보이는 RF와 달리 값들이 일정한 패턴까지 보이는 것을 확인하였습니다. 이러한 PRF의 설계는 RF와 구분되는 확실한 특성을 보이므로 PRF로 사용할 수 없습니다.

결론

본 연구에서 안전한 의사 랜덤 함수 (PRF)를 정의하기 위한 시각화 관점에서의 접근을 시도하였습니다. 연구에서는 랜덤 함수 (RF)와 의사 랜덤 함수 (PRF)를 의사 코드로 모델링하고, RF의 난수 테이블의 크기를 고려하여 On-The-Fly 방식으로 시뮬레이션하였습니다. 모듈화한 RF의 출력 분포를 분석한 결과, 무작위성이 확인되었습니다. 반면에 설계가 미흡한 PRF는 일정한 패턴을 보임으로써, 이러한 패턴이 암호 시스템의 유효성을 저해할 수 있음을 드러냈습니다. RF의 무작위 분포와 결함이 있는 PRF의 패턴화된 출력 사이의 시각적 차이는 암호 알고리즘 개발에서 철저한 설계와 테스팅의 중요성을 강조합니다. 이 연구를 바탕으로 앞으로 블록 암호의 설계에 있어 핵심적인 구성요소인 의사 랜덤 치환 (Pseudo-Random Permutation, PRP)의 시각화에 관한 연구를 진행할 것입니다.

참고 문헌

[1] M. Rosulek, *The Joy of Cryptography*, [Online]. Available: https://joyofcryptography.com
[2] N. P. Smart, *Cryptography Made Simple*. 1st ed. Springer International Publishing, 2016.
[3] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. 2nd ed. Chapman and Hall/CRC, 2014.