메모리 효율적인 TMTO 암호 해독 방법

정회원 김 영 식*, 임 대 운**°

Memory-Efficient Time-Memory Trade-Off Cryptanalysis

Young-Sik Kim*, Dae-Woon Lim**° *Regular Members*

요 약

Hellman에 의해서 처음 제시된 TMTO (time memory trade-off) 암호 해독 방법은 블록 암호, 스트림 암호, 그리고 해쉬 함수와 같은 일반적인 암호 시스템에 광범위하게 적용된다. 이 논문에서는 TMTO 암호 해독 방법에서 선계산 단계에서 테이블을 저장하기 위해 필요한 저장 공간을 감소시킬 수 있는 방법을 제안한다. 시작점을 의사 난수 수열군 통해서 생성하고 시작점의 실제 값 대신 난수 수열군에서 몇 번째로 취한 값인지에 대한 색인을 저장하는 방식으로 시작점을 저장하기 위해 필요한 메모리 용량을 줄일 수 있다. 이 논문에서는 이 방법을 사용하면 키의 길이가 126 비트일 경우에 시작점을 저장하기 위해 필요한 메모리의 양을 10% 이하로 줄이는 것도 가능하다는 것을 보일 것이다. 이에 대한 비용으로 온라인 단계에서의 탐색 시간이 조금 더 늘어나지만 메모리가 시간에 비해서 더 비싼 자원이기 때문에 필요한 메모리 용량이 감소하게 되면 공격의 실현가능성이 더욱 커지게 된다.

Key Words : Cryptanalysis, distinguished point, Hellman table, rainbow table, time-memory trade-off

ABSTRACT

Time-memory trade-off (TMTO) cryptanalysis proposed by Hellman can be applied for the various crypto-systems such as block ciphers, stream ciphers, and hash functions. In this paper, we propose a novel method to reduce memory size for storing TMTO tables. The starting points in a TMTO table can be substituted by the indices of n-bit samples from a sequence in a family of pseudo-random sequences with good cross-correlation, which results in the reduction of memory size for the starting points. By using this method, it is possible to reduce the memory size by the factor of 1/10 at the cost of the slightly increasing of operation time in the online phase. Because the memory is considered as more expensive resource than the time, the TMTO cryptanalysis will be more feasible for many real crypto systems.

Ⅰ. 서 론

임의의 주어진 암호 시스템을 해독하기 위한 가장 기본적인 공격 방법은 전수 검사를 통해서 주어진 암호문에 대해서 모든 가능한 키를 사용해서 복호를 해 보는 방법이다. 하지만 이 경우 공격에 걸리는 시간 복잡도가 키의 길이가 n일 때 $O(n)$로 주어진다. 또 다른 반대의 극단으로 사전에 미리 알려진 평문에 대한 모든 가능한 키와 암호문 쌍을 얻어서 하나의 표에 저장하는 방법이 있다. 이 경우

※ 이 논문은 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임 (KRF-2007-331-D00373). 또한 이 연구의 주요 계산은 한국과학기술정보연구원(KISTI)이 무상으로 제공한 슈퍼 컴퓨팅 자원을 이용하여 수행되었음.

* 삼성전자, System LSI 사업부 (mypurist@gmail.com)
** 동국대학교 정보통신공학과 (daewoonlim@gmail.com)(° : 교신저자)

논문번호 : KICS2008-08-335, 접수일자 : 2008년 8월 5일, 최종논문접수일자 : 2008년 11월 12일

실제로 알려진 평문에 대응하는 암호문이 주어지면 단순히 표를 한 번 참고하는 것만으로도 해독이 가능하지만 키와 암호문의 모든 쌍을 저장하기 위한 방대한 양의 메모리 공간이 필요하다.

Hellman은 이러한 두 가지 극단적인 방법에 대한 절충으로서 시간과 메모리의 상반 관계에 의한 공격을 제안하였다[1]. Hellman은 블록 암호인 DES에 이 공격을 적용시켰는데 후에 Babbage[2]와 Gollic[3]은 내부 상태만의 함수로 외부 출력이 결정되는 스트림 암호의 특성을 사용해서 여러 데이터를 동시에 사용해서 TMTO (time-memory trade-off) 공격을 스트림 암호에 더 효과적으로 적용할 수 있음을 보였다. 그 이후에 Biryokov와 Shamir는 Hellman의 방법과 Babbage- Golic의 방법을 결합시킨 새로운 공격 방법을 제안하였다[5].

Rivest는 TMTO 공격에 대한 최적화의 일환으로 오늘날 널리 이용되는 DP (distinguished point) 방법을 제안하였다[4],[7]. DP 방법에서는 테이블에 저장되는 끝점이 언제나 일정한 규칙을 갖기 때문에 미리 계산된 테이블을 검색하는 과정을 크게 줄일 수 있다. 한편 Oechslin[6]은 작은 테이블을 여러 개 생성하는 Hellman의 방법을 변형해서 커다란 하나의 테이블로 대체할 수 있는 방법을 제안하였고 이런 테이블을 Rainbow 테이블이라 불렀다. 이 방법을 사용하면 Hellman에 비해서 테이블 검색 시간이 절반으로 줄어들게 된다. 최근에는 Hong[8] 등이 DP 방법을 변형해서 끝점에 시작점의 정보를 저장하는 VDP (variants of distinguished point) 방법을 제안하기도 하였다.

이 논문에서는 시작점을 저장하는 정보를 간단한 값으로 치환하는 방식으로 필요한 사용량을 줄이는 방법을 제안한다. 일반적으로 메모리는 시간에 비해 좀 더 비싼 자원으로 여겨진다. 다시 말해 메모리를 증가시키려면 해당하는 하드웨어를 늘려야 하지만 시간을 증가시키려면 해당 프로그램을 조금 더 오래 돌리는 것으로 충분하다. 이러한 상황에서 메모리의 사용을 효율화 하게 되면 TMTO 공격의 실현 가능성을 그만큼 높일 수가 있게 된다. 이 방법을 Hellman의 방법에 적용시키면 시작점을 저장하는 메모리의 공간을 10% 이하로 낮출 수가 있게 되고 시작점과 끝점을 다 합쳐서 전체적으로 메모리 공간이 50% 가깝게 줄어들게 된다. 하지만 이때의 실제 연산 시간은 전체 시간에 비해서 대략 3% 정도 더 늘어나게 된다.

이 논문은 다음과 같이 구성된다. 제 Ⅱ장에서는 TMTO 공격에 대한 기본적인 개념 및 알려진 공격 방법을 소개한다. 제 Ⅲ장에서는 테이블을 저장하는데 필요한 메모리 공간을 줄이는 새로운 방법에 대해서 설명한다. 제 Ⅳ장에서는 새로운 방법의 성능을 비교할 것이다. 끝으로 제 Ⅴ장에서는 결론을 내리고 마칠 것이다.

Ⅱ. 기존의 TMTO 방법

TMTO에서는 실제 암호 해독이 일어나기 전인 선계산(pre-computation) 단계에서 사전에 미리 알려진 평문을 사용해서 키 공간상의 점 사이의 연속적인 관계를 찾는 과정을 거친다. 이를 위해 고정된 평문 m을 가정한다. 그리고 이 m에 대해서 키 k에서 암호문 c로의 함수 $E_k(m)$을 가정한다. 여기서 c는 다시 축약 함수 R_i에 대입되어 c에서 키 공간상의 한 점으로 사상된다. 즉 다음과 같은 관계가 성립한다.

$$f_i(k) = R_i(E_k(m))$$

이런 $f_i(k)$에 다시 f_i를 반복해서 $t-1$번 적용하면 길이가 t인 하나의 체인(chain)을 얻을 수 있다. 여기서 f_i는 일방향 함수로 결국 TMTO는 암호문 c로부터 일방향 함수 f_i의 역 이미지인 k를 찾는 문제라 할 수 있다.

2.1 Hellman의 방법

Hellman[1]으로부터 시작된 모든 TMTO 암호 해독 법은 두 가지 단계로 구분된다. 하나는 공격을 위한 테이블을 미리 계산하는 선계산 단계이고 또 하나는 실제로 주어진 암호문의 키를 찾는 과정인 온라인(online) 단계이다. Hellman의 선계산 단계에서는 m개의 시작점을 균일하게(uniformly) 분포하는 무작위 값으로 정한다. 이런 m 개의 시작점에 f_i를 반복적으로 t번 적용하게 되면 그림 1에서처럼 길이가 $t-1$인 체인이 총 m 개가 만들어진다. 여기서 t번 째 값을 끝점이라 하고 중간 값들을 모두 생략하고 시작점(SP)과 끝점(EP)만을 저장한 Hellman 테이블을 만든다. 이 때 온라인 단계에서 검색의 편이를 위해서 끝점을 기준으로 오름차순으로 정렬을 해서 저장한다.

만일 함수 f_i에 서로 다른 시작점들을 대입했을 때 나오는 중간 값이 모두 다르다고 가정하면 f_i로 만들어지는 하나의 테이블은 총 mt 개의 원소를 포

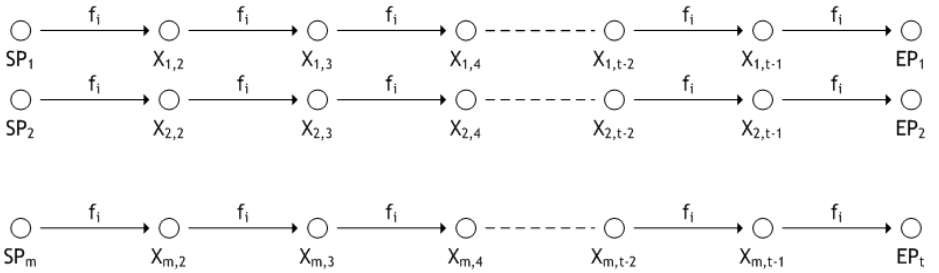

그림 1. i번째 Hellman 테이블의 생성

함하게 된다. 전체 원소의 개수 N은 생일 역설(birthday paradox)로부터 $mt^2 = N$의 관계를 갖는다. 따라서 하나의 Hellman 테이블은 전체 N 개의 키 중에서 $1/t$ 개만을 커버하고 있다. 확률을 높이기 위해 만일 t개의 서로 다른 축약 함수 R_i $(1 \le i \le t)$를 사용해서 테이블을 만든다면 키 공간을 더 많이 커버할 수 있는 테이블들을 얻을 수 있다.

물론 테이블 속에는 체인 속에 사이클이 발생하거나 서로 다른 두 체인에 대해서 병합(merge)이 발생할 수 있기 때문에 동일한 값이 여러 번 반복해서 나올 수도 있다.

온라인 단계에서는 평문 m과 대응되는 암호문 c가 주어진다. 이 c를 통해 $Y_{i,t} = R_t(c)$를 얻는다. 이 값을 끝점과 비교해 보아서 $Y_{i,t} = f_i^{t-1}(SP_i)$인 끝점을 발견한다면 시작점 SP_i로부터 f_i를 $t-2$번 적용해서 얻는 $f_i^{t-2}(SP_i)$가 키가 될 확률이 존재한다. 암호학에서 이것은 선택 평문 공격(chosen plaintext attack)으로 알려져 있다. 만일 같은 점이 없다면 $Y_{i,t-1} = f_i(Y_{i,t})$를 계산해서 다시 끝점과 비교해 본다. 같은 점을 찾을 때까지 이와 같은 과정을 모든 t 개의 테이블에 대해서 t 번씩 반복한다.

2.2 Distinguished Point (DP)

Rivest는 특정한 성질을 만족하는 점들만을 끝점으로 선택함으로써 온라인 단계에서 테이블을 검색하는 횟수 자체를 줄일 수 있는 방법을 제안하였다[4],[7]. 처음에 일정한 길이 d 개의 연속된 0이 나오는 값이 일반적으로 많이 쓰이는 특정한 규칙이다. DP 방법에서는 이러한 성질을 만족하는 끝점이 나올 때까지 f_i를 적용해서 Hellman 테이블을 구성한다. 따라서 체인의 길이는 t로 고정된 것이 아니라 가변적이다. 실제 온라인 단계에서는 매번 $Y_{i,j}$를 테이블과 비교해 보는 것이 아니라 DP가 나올 때까지 $Y_{i,j}$에 f_i를 적용하다가 DP가 나올 때만 테이블을 비교해 보게 된다.

DP의 또 다른 장점은 병합을 제거할 수가 있다는 것이다[4]. 테이블을 만든 후 끝점을 기준으로 정렬을 할 때, 끝점이 동일한 체인이 있다면 병합이 일어난 것으로 보고서 길이가 긴 체인을 남기고 나머지를 제거한다. 이 때 체인의 평균 길이와 최대 및 최소 길이가 고정되어 있어서 이 값을 만족시키지 못하는 체인은 버려지게 된다. 체인의 평균 길이를 맞추기 위해서 평균보다 더 긴 길이를 가지는 c체인이 존재하므로 실제 온라인 시간은 대략 최대 체인 길이 $t_{\max}$에 테이블의 개수 t를 곱한 값이 된다.

2.3 Rainbow 테이블

Oechslin이 제안한 방법으로 하나의 테이블의 열마다 서로 다른 f_i를 적용해서 병합이 일어날 가능성을 제거하고 있다[6].

Hellman의 경우는 하나의 테이블을 만들기 위해서 f_i를 사용하고, 이 때 R_i를 약간씩 변형시킨 총 t개 f_i를 이용해서 서로 다른 테이블들을 생성하였다. 따라서 전체 테이블의 행의 개수를 모두 더하면 mt개의 행이 존재한다. 그러나 Rainbow 테이블에서는 하나의 테이블의 행의 개수가 mt로서 각각의 열은 서로 다른 $t-1$ 개의 함수 f_i $(1 \le i \le t-1)$를 가지고서 생성한다. 매 열마다 f_i가 다르기 때문에 같은 입력에 대해서도 서로 다른 값이 나올 확률이 매우 크다. 따라서 병합이 일어날 확률은 매우 작다.

실제로 병합이 일어나는 경우는 서로 다른 두 개의 체인에서 중간에 같은 열에서 우연히 같은 출력이 나오는 경우이다. 하지만 정렬 과정에서 끝점이 동일한 값들을 하나만 남기고 모두 제거하면 병합을 제거할 수 있다. 물론 제거된 횟수만큼 새로운 시작점에서 새로운 체인을 만들어야 한다.

Hellman 테이블에서는 전체 검색을 위해서 f_i 함

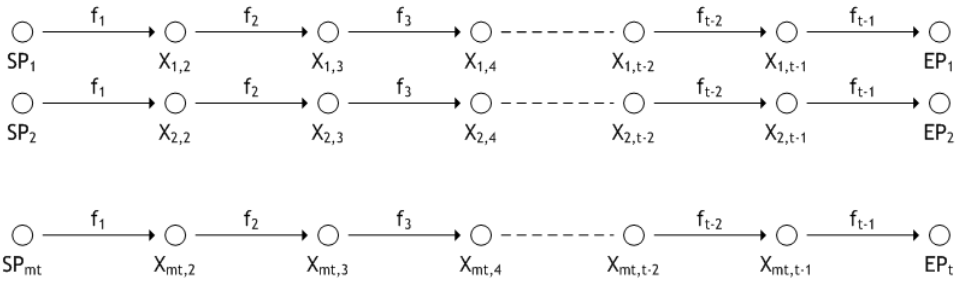

그림 2. Rainbow 테이블의 생성

수들을 최대 t^2 번 적용한다. 하지만 Rainbow 테이블에서는 하나의 테이블을 검색하기 위해서 총 $t(t+1)/2$번 f_i 함수들을 적용한다. 따라서 큰 t에 대해서 $t^2 \gg t$이므로 대략 $t^2/2$번 f_i 함수들을 적용하는 것이 된다. Rainbow 테이블은 f_i 함수와 관련된 연산을 절반만 적용하고서도 동일한 키 공간을 검색할 수가 있다.

2.4 Variants of Distinguished Point (VDP)

VDP 는 DP의 일종으로 시작점의 정보를 이용해서 DP를 정의함으로써 메모리를 효율화하는 방식이다[8]. 이 방법에서는 앞에서와는 달리 시작점을 난수가 아니라 체인이 위치하게 될 테이블의 색인과 그 테이블 내에서의 체인의 일련번호로 정의한다. 예를 들어 색인이 i인 테이블 내에서 j 번째 체인의 시작점은 $(i \| j)$로 정의된다. 그런 후에 끝점이 j 값으로 시작 될 때를 DP로 정의하면, i 번째 테이블 내에서는 끝점만 보고서도 시작점을 알 수가 있다. 또한 끝점들이 시작점의 오름차순 색인으로 시작되기 때문에 별도로 정렬을 하지 않아도 된다. 더 나아가 시작점 자체를 저장할 필요가 없고 끝점에서도 시작점의 색인에 해당하는 비트 정보는 따로 저장할 필요가 없다.

하지만 DP와 달리 검출할 수 없는 병합이 발생한다. 예를 들어 $i+1$번째 체인에 EP_i가 들어 있는 $C_1 \rightarrow C_2 \rightarrow EP_i \rightarrow C_3 \rightarrow \cdots \rightarrow EP_{i+1}$과 같은 예에서는 EP_i와 관련된 일부 키가 EP_{i+1}의 체인 속에 포함되어 있음을 볼 수 있다. 또한 테이블의 색인과 체인의 위치를 시작점으로 사용하였기 때문에 DP를 찾지 못하는 경우에는 해당 체인이 공백으로 남을 수도 있다. 그리고 기본적으로 DP방식에서처럼 온라인 단계에서의 시간이 체인의 최대 길이 $t_{\max}$와 t의 곱으로 표현되는데 $t_{\max} > t$이기 때문에 검색 시간이 증가한다. 또한 DP와는 달리 모든 $Y_{i,j}$가 DP가 된다.

Ⅲ. 시작점 치환 기법

만일 키의 길이가 n 비트라고 하면 하나의 시작점과 끝점으로 이루어진 체인을 저장하기 위해서는 총 $2n$ 비트의 메모리가 필요하다.

이 장에서는 시작점을 저장하기 위한 정보의 양을 의사 난수 수열 군을 사용해서 줄이는 방법을 제안할 것이다. 이를 위해 먼저 시작점을 난수가 아니라 단순히 일련번호를 사용하더라도 메모리 크기가 줄어들 수 있음을 지적할 것이다.

3.1 시작점 색인

먼저 Hellman의 방법에 적용했을 경우를 살펴보면 일반적으로 테이블을 생성하게 되는 함수 f_i는 공격하고자 하는 암호 시스템에 대응되는 함수이고 잘 설계된 암호 시스템은 보통 좋은 의사 난수 발생기(pseudo random number generator, PRNG)이기 때문에 임의의 입력에 대해서 의사 난수 출력을 발생시킨다. 따라서 시작점이 굳이 난수가 아니더라도 f_i를 여러 번 반복하면서 생기는 중간 값과 마지막 값은 의사 난수가 된다.

하나의 Hellman 테이블에 포함된 체인의 총 개수는 m 개로 시작점을 저장하기 위해서 총 mn 비트의 메모리가 필요하다. 만일 시작점이 단순히 0부터 $m-1$에 대응되는 이진수로 표현된 색인으로 구성되어 있다면 이 경우에 $\lceil \log_2 m \rceil$ 만큼의 하위 비트에만 실제 데이터가 저장되고 $n - \lceil \log_2 m \rceil$의 값은 항상 0으로 남는다. 여기서 $\lceil x \rceil$는 x보다 크거나 같은 최소의 정수이다. 만일 Hellman 테이블에서 일반적으로 사용되는 값인 $m = 2^{n/3}$를 사용하면 이러한 시작점을 저장하기 위해 필요한 메모리는 $mn/3$로 필요한 메모리의 양이 1/3로 줄어들었다. 전체적으로는 $2nm$ 비트에서 $4mn/3$ 비트로 2/3으로 줄어들었다.

하지만 Rainbow 테이블에서는 행의 개수가 mt 개인 하나의 큰 테이블을 사용하므로, 예를 들어 $m=t=2^{n/3}$이라면 시작점을 저장하기 위한 메모리가 총 $2mn/3$ 비트가 필요하다. 전체적으로는 $5mn/3$ 비트가 필요하므로 전체적으로 메모리양이 5/6으로 줄어들었다.

3.2 의사 난수 수열군

LFSR은 간단한 구성으로 좋은 난수 특성을 보여주는 수열을 만들어 낸다. 그러나 주어진 LFSR로부터는 주기가 2^n-1인 단 하나의 수열이 만들어질 뿐이다. 이러한 LFSR의 구성을 바꿈으로써 좋은 난수 특성을 갖는 서로 다른 수열들을 여러 개 만들어 낼 수가 있다. 이런 수열군들 중에서 오래 전부터 널리 알려진 것으로 Gold 수열군[10]이 있다. 이 논문에서는 예시를 위해 구체적으로 Gold 수열군을 사용해서 초기값을 생성할 것이지만 다른 수열군을 사용하더라도 아무런 문제가 없다.

Gold 수열군은 두 개의 LFSR의 합으로 구성된다. 하나의 LFSR에 대해서 또 다른 LFSR은 $q=2^k+1$ 또는 $2^{2k}-2^k+1$로 데시메이션(decimation)한 값을 출력한다. 여기서 k는 n이 홀수일 때는 $\gcd(n,k)=1$이고 $n\equiv 2 \bmod 4$ 일 때는 $\gcd(n,k)=2$인 값이다. 따라서 서로 다른 두 개의 수열을 동시에 만들어서 더할 때 위상의 차이에 따라서 서로 다른 수열이 만들어진다. 따라서 Gold 수열군에는 총 2^n+1개의 서로 다른 수열이 존재한다. (이 중에서 두 개는 각각의 LFSR 수열이다.) 이렇게 생성된 수열은 n이 홀수일 때는 서로 다른 수열의 상호 상관값이 위상 차이에 따라서 $\{\pm 2^{(n+1)/2}-1,-1\}$에서 값을 갖고, $n\equiv 2 \bmod 4$ 일 때는 $\{\pm 2^{(n+2)/2}-1,-1\}$에서 값을 갖는다. 이러한 상호 상관 특성은 서로 다른 수열에서 연속된 n 비트의 값이 동일한 값이 나오는 확률이 매우 작도록 만들어 준다.

3.3 의사 난수 수열에서 생성된 시작점

Gold 수열은 두 개의 n 비트 LFSR로 구성되어 있다. 따라서 초기 값도 $2n$ 비트가 필요하다. 먼저 의사 난수 수열을 생성하는 n 비트의 초기값 IV를 결정한다. 그 후 IV로부터 si만큼 이동 시킨 후에 n비트의 레지스터에 저장된 값을 색인 0, 그 이후로 s만큼 이동 시킨 값을 색인 i $(1\le i\le 2^{n/3-r}-1)$로 나타낸다. 그런 후에 주기가 2^n-1인 총 2^r개의 의사 난수 수열로부터 n비트씩 새로운 난수를 총 $2^{n/3-r}$개 발생시켜서 시작점으로 사용한다. 사용한

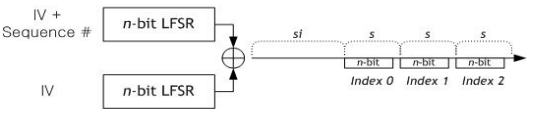

그림 3. Gold 수열에서 TMTO 테이블의 시작점 생성.

시작점은 0부터 $n/3-r-1$까지의 정수로 나타낸다. Gold 수열군에 속하는 각각의 의사 난수 수열은 두 개의 LFSR 중 하나의 초기값 IV에 수열의 고유 숫자를 더한 값을 수열의 초기값으로 사용함으로써 선택이 가능하다.

이때 si와 s 값은 이렇게 선택되는 수열들이 이전에 나온 샘플과 충분히 다른 값이 나오도록 선택해야 한다. LFSR의 특성상 $s\ge n$이면 서로 다른 값이 나온다. 수열군 내의 수열들 사이에는 상호 상관관계가 작기 때문에 충분히 큰 n에 대해서 서로 다른 수열들 사이에서 같은 n 비트 샘플이 나올 확률도 매우 작다.

3.4 Hellman 테이블에 적용

1) 선 계산 단계: 총 t 개의 Hellman 테이블에는 중복되는 끝점이 발생 시 새로운 체인을 생성하기 위한 여분의 수열까지 포함해서 총 $2^r+l_{\max}$개의 수열이 필요하다. 여기서 $l_{\max}$는 2^r보다 작은 정수로 선택한다. 각각의 수열로부터는 $2^{n/3-r}$ 개의 색인을 선택하게 된다. 여기서 발생된 시작점을 사용해서 기존과 마찬가지로 m 개의 체인을 만든 후에 실제 시작점 대신에 수열에서 취한 샘플의 $n/3-r$ 비트의 색인을 저장한다. 서로 다른 테이블에는 서로 다른 수열을 적용할 수 있는데 각각의 테이블에 적용된 수열의 고유 번호 (Sequence #)의 범위를 테이블 색인과 함께 저장한다.
   테이블이 완성 되면 끝점을 기준으로 오름차순으로 정렬한다. 끝점에 동일한 값이 존재하면 그 중에서 하나만 남기고 나머지는 버린다. 그리고 버린 개수만큼 새로운 체인을 여분의 수열을 사용해서 생성한다.
2) 온라인 단계: $Y_{i,j}$를 끝점과 차례로 비교한다. 만일 일치하는 값이 있다면 시작점에는 샘플에 대한 색인만 저장되어 있기 때문에 해당 색인을 총 $2^r+l_{\max}$ 개의 수열 모두에 적용시켜서 얻은 모든 시작점을 사용해서 비교를 해 보아야 한다. 이 때 추가적인 시간적 비용이 발생한다.

3.5 Rainbow 테이블에 적용

Rainbow 테이블도 각각의 열을 생성할 때 서로

다른 함수 f_i를 사용하고 총 mt 개의 체인을 한 번에 만들어 낸다는 점만 Hellman 테이블과 다를 뿐이기 때문에 시작점을 동일한 방법으로 생성하면 마찬가지로 시작점을 저장하는데 필요한 정보량을 축소시킬 수 있다. 그러나 $2^r > t$이면 온라인 단계에서 발생하는 추가적인 시간 비용이 기존의 시간 비용을 초과해 버리기 때문에, $t \gg 2^r$로 만들어야 한다. 또한 행의 개수가 mt 개이므로 시작점의 색인을 저장하기 위해 필요한 메모리가 Hellman 테이블에 비해서 두 배 정도 더 늘어난다.

3.6 오경보 (False Alarm)의 영향

R_i로 인해서 $Y_{i,j}$와 끝점이 일치하더라도 실제로 그 값이 항상 키가 되는 것은 아니다. 이런 경우를 오경보라고 한다. 실제 Hellman의 방법이나 DP, 그리고 Rainbow 방법은 모두 온라인 시간의 상당 부분을 오경보를 해결하는데 사용한다. Avoine 등[9]은 Hellman/Rainbow 테이블을 완전히 다시 계산하지 않고서도 오경보를 검출할 수 있는 check point방법을 제안하였다. 여기에서 제안하는 방법도 테이블을 검색하는 과정은 기존과 동일하기 때문에 check point 방법으로 오경보 문제를 완화시킬 수가 있다.

Ⅳ. 성능 비교

LFSR을 기반으로 하는 수열군을 사용해서 시작점을 만들어 내는 과정은 하드웨어적으로나 소프트웨어적으로 매우 빠르게 이루어지기 때문에 어려운 일이 아니다. 다만 일치점이 있을 때 $2^r + l_{\max}$개의 수열들에 대해서 실제로 어떤 수열에서 나온 색인인지를 하나씩 확인을 해야 하기 때문에 온라인 단계에서 시작 복잡도 T가 증가하게 된다. 여기서는 Hellman의 경우와 Rainbow의 경우에 시간 복잡도 T가 늘어나는 정도와 메모리의 감소 비율을 추정해 본다.

4.1 Hellman 방식에서의 성능 비교

Hellman 의 방법에서 T는 최악의 경우로 $t^2 + t$이다. 여기에 총 $2^r + l_{\max}$ 개의 수열에 대해서 마지막 검색을 반복하는 데 따른 비용을 추가하면 최악의 경우 $t^2 + (2^r + l_{\max} + 1)t$ 번 연산을 수행하게 된다. Hellman에서 $t = 2^{n/3}$일 때 만일 r이 $n/3$보다 충분히 작으면 이러한 추가적인 연산은 전체 연산 시간에 큰 영향을 주지 않는다. 표 1은 키의 길이가 126 비트일 때 기존의 T와 T'를 비교한 결과를 보여준다.

표 1. Hellman 테이블에서 $n = 126$일 때 시간 비용 비교.

r	$n/3 - r$	T	T'	증가비율
26	16	2^{84}	$2^{84.0000}$	0.002%
27	15	2^{84}	$2^{84.0000}$	0.003%
28	14	2^{84}	$2^{84.0001}$	0.006%
29	13	2^{84}	$2^{84.0002}$	0.012%
30	12	2^{84}	$2^{84.0004}$	0.024%
31	11	2^{84}	$2^{84.0007}$	0.049%
32	10	2^{84}	$2^{84.0014}$	0.098%
33	9	2^{84}	$2^{84.0028}$	0.195%
34	8	2^{84}	$2^{84.0056}$	0.391%
35	7	2^{84}	$2^{84.0112}$	0.781%
36	6	2^{84}	$2^{84.0224}$	1.563%
37	5	2^{84}	$2^{84.0444}$	3.125%

표 1에서 볼 수 있는 것처럼 $n/3 = 42$일 때 $r = 37$일 때에도 시간적인 비용은 3%정도만 늘어날 뿐이다. $r = 35$일 때에도 시간적 비용은 1%미만으로 늘어난다.

반대로 이 방법을 통해서 얻는 이득을 생각해 보자. 시작점과 끝점은 하나를 저장하는데 각각 n 비트의 메모리가 필요하다. 하지만 일반적으로 Hellman의 방법에서 $m = 2^{n/3}$이므로 수열군의 색인을 이용함으로써 실제로 저장해야 할 하나의 시작점의 크기는 $n/3 - r$ 비트이다. 따라서 전체 시작점을 저장하는데 필요한 메모리양은 $(n/3 - r)m$ 비트이다. III.1절에서처럼 단순히 시작점의 색인을 저장하면 1/3만큼 필요한 메모리가 감소했는데 수열군의 색인을 사용함으로써 추가적으로 필요한 메모리를 더 줄일 수 있었다. 그림 4는 Hellman 테이블에서의 메모리 감소 비율을 나타낸다. 여기서 SEQ는 수열군을 이용한 방법을 의미하고 IND는 단순히 색인만을 사용했을 때의 값이다. TOT는 전체 메모리의 감소 비율이고 STA는 시작점만의 감소 비율이다.

예를 들어 $n = 126$비트일 때 메모리 감소율은 $(42 - r)/126$인데 $r = 35$일 때는 시작점을 저장하기 위해서 기존대비 5.6% 정도의 메모리만이 필요할 뿐이다. 이때 시간적으로는 0.8% 정도의 연산이 더 필요한 수준이다. 물론 끝점을 저장하기 위한 메모리는 그대로이기 때문에 시작점을 저장하기 위한 메모리가 매우 작아지게 되면 전체적으로 메모리는 기존 대비 53%정도만 필요하게 된다.

표 2. Rainbow 테이블에서 $n=126$일 때 시간 비용 비교.

r	$n/3-r-1$	T	T'	증가비율
26	15	2^{83}	$2^{83.0000}$	0.003%
27	14	2^{83}	$2^{83.0001}$	0.006%
28	13	2^{83}	$2^{83.0002}$	0.012%
29	12	2^{83}	$2^{83.0004}$	0.024%
30	11	2^{83}	$2^{83.0007}$	0.049%
31	10	2^{83}	$2^{83.0014}$	0.098%
32	9	2^{83}	$2^{83.0028}$	0.195%
33	8	2^{83}	$2^{83.0056}$	0.391%
34	7	2^{83}	$2^{83.0112}$	0.781%
35	6	2^{83}	$2^{83.0224}$	1.563%
36	5	2^{83}	$2^{83.0444}$	3.125%

4.2 Rainbow 방식에서의 성능 비교

여기에서도 최대 $(2^r+l_{\max})t$ 만큼의 시간 비용이 더 증가하게 된다. 하지만 $t \gg 2^r$이면 추가적인 연산이 전체 연산 복잡도에 미치는 영향은 무시할 수 있을 정도로 작아진다. Rainbow 테이블에서 전체 연산량은 $t(t+1)/2+(2^r+l_{\max})t$에서 충분히 큰 t에 대해서 대략적으로 $t^2/2+2^rt$로 볼 수 있다. 즉, $t=2^{n/3}$일 때 $2^rt(2^{n/3-r-1}+1)$로 볼 수 있다. 키의 길이가 $n=126$일 때 표 2에서 시간 비용의 증가 비율을 볼 수 있다.

표 2에서 보면 $r=35$일 때 1.6% 정도의 추가 연산이 필요하고 이것은 Hellman의 경우보다 대략 2배정도 더 많은 비용에 해당한다. Hellman에 비해서 Rainbow 테이블의 연산 복잡도가 절반 정도이지만 추가되는 비용은 사실상 두 가지 방법이 모두 거의 동일하기 때문에 Rainbow 테이블에서의 부담이 두 배 정도 더 늘어난다.

반면에 $r=35$인 경우 시작점을 저장하기 위해 필요한 메모리는 기존 대비 $(84-r)/126$으로 약 38.9% 감소했다. Hellman의 경우는 5.6%였던 것과 비교하면 감소율은 많이 떨어진다. 하지만 단순히 색인만 저장하는 경우에도 66.7% 정도의 비율인 것을 고려할 때 수열군을 사용하게 되면 1.6%정도의 시간적 비용으로 효율적인 메모리 사용이 가능해진다.

4.3 분석 및 토의

한 테이블에 포함된 체인의 길이는 t로 일정하다. 이 논문에서는 미리 정해진 수열군을 사용함으로써 저장해야 할 색인의 길이를 줄일 수 있고, 수열군에서 생성된 난수를 사용해서 서로 다른 시작점들을 만들어 낼 수가 있음을 보였다.

Hellman은 시작점을 난수로 사용할 것을 제안하였지만 일반적으로 암호 시스템은 좋은 PRNG라고 가정할 수 있기 때문에 여기서는 간단한 의사 난수 수열로 시작점 생성을 대체하였다. 따라서 수열 생성법이 알려져 있을 때 $m=2^{n/3}$인 경우 일련번호를 저장하는 형태의 시작점의 Shannon 엔트로피는 원래 크기의 1/3정도로 줄어든다. 여기에서 추가로 수열 자체에 대한 정보를 버림으로써 하나의 체인 당 r비트의 정보를 더 줄일 수 있다. 실제 검색 단계에서는 버린 r비트 정보를 찾기 위해서 $2^r+l_{\max}$의 반복이 발생했지만 이러한 반복은 실제로 $Y_{i,j}$와 끝점 사이에서 일치점이 발생할 때만 일어나기 때문에 전체적인 영향은 매우 적다.

그러나 일반적으로 메모리의 비용은 같은 양의 시간보다 더 비싼 것으로 여겨진다. 예를 들어 메모리를 절약할 수 있다면 시간적인 비용을 어느 정도 만회하는 것이 현실적으로 가능하다. 키의 길이가 증가하면 필요한 메모리의 크기는 지수함수적으로 증가한다. 이 때 충분히 큰 n에 대해 상당량의 하드웨어 비용은 테이블을 저장하기 위한 RAM이나 HDD 또는 DVD와 같은 메모리 매체에 들어가게 되는데 만일 저장할 데이터가 절반으로 줄어든다면 메모리를 구매하기 위한 단가가 절반으로 줄어들게 된다. 메모리를 절약한 비용으로 또 다른 프로세서를 확보한다면 테이블 검색을 동시에 수행할 수가 있으므로 시간상의 비용을 충분히 만회할 수 있다.

VDP의 경우도 메모리의 효율적인 사용이 가능하도록 만들지만 가변적인 체인 길이에 따른 비용에 더하여 체인마다 서로 다른 DP 가 사용되는데 따르는 비용까지 감안하면 온라인 단계에서 테이블 검색 횟수나 f_i 함수의 적용 횟수는 Hellman이나 Rainbow의 방법에 비해서 더 늘어나게 된다.

다음 표는 제안된 방법의 비트 레벨 TMTO 공격 방법과 관련된 변수들을 Hellman과 Rainbow의 것과 비교하고 있다.

표 3. 제안된 방법의 비트 레벨 TMTO 성능

	M	T	Lookup	Bit-Level Trade-off
H	mtw	t^2	t^2	$TM^2=N^2$
R	mtw	$t^2/2$	t	$TM^2=2N^2$
H+SEQ	mtw_r^H	t^2+2^rt	t^2	$TM^2=v_H^2N^2/v_1$
R+SEQ	mtw_r^R	$t^2/2+2^rt$	t	$TM^2=v_R^2N^2/v_2$

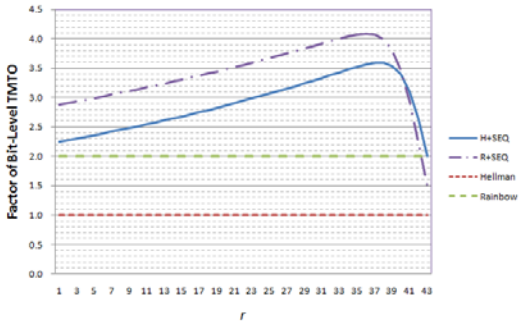

그림 4. 제안된 방법의 비트 레벨 상반 관계 비교.

표에서 H는 Hellman의 방법을 그리고 R은 Rainbow 방법을 의미한다. 여기서 $w = v_H w_r^H = v_R w_r^R$이고 $t^2 + 2^r t = v_1 t^2$, $t^2/2 + 2^r t = v_2 t^2$이다. 시간적인 비용에서 작은 r에 대해서 t^2 내지는 $t^2/2$로 근사가 되므로 제안된 방법은 Hellman 테이블에 적용했을 때 비트 레벨 상반 관계 측면에서 $9/4N^2 < TM^2 < 4N^2$가 되고 Rainbow 테이블에 적용하면 $(6/5)^2 N^2 < TM^2 < 9/4N^2$이 된다. 하지만 상한의 경우 작은 r이라는 전제와 모순이 되기 때문에 실제로는 그림 4와 같은 형태의 Trade-off 곡선을 갖게 된다. 그림 4는 $TM^2 = cN^2$에서 r에 따른 c의 값을 그래프로 나타낸 것이다.

이 때, 앞서 설명한 바와 같이 그림 4에서 충분히 r이 작을 때에는 필요한 메모리의 감소에 대한 이득이 더 크다가 r이 $n/3$에 근접하면서 비트 레벨 상반 관계 인자 c가 급격히 떨어지게 된다.

V. 결 론

이 논문에서는 TMTO 암호 해독 방법에서 실제 테이블을 저장하는데 필요한 메모리의 크기를 줄이는 방법을 제안하였다. 기존의 연구에서는 일반적으로 테이블을 저장하기 위해서 (시작점, 끝점) 쌍의 각각의 워드를 저장한다고 가정하고 있지만 실제로 워드 크기에 대해서는 자세히 고려하지 않았다. 이 논문에서는 시작점을 다른 값으로 치환함으로써 실제로 저장해야 할 워드의 비트 수를 키의 길이 n보다 훨씬 더 줄일 수 있음을 보였다.

시작점은 의사 난수 수열군으로부터 생성되는데 수열군에서 생성된 실제 의사 난수 값이 저장되는 것이 아니라 초기 값에서 몇 번째 취해진 값인지에 대한 색인을 대신 저장함으로써 저장해야 할 데이터의 크기를 감소시키고 있다. 하나의 테이블을 생성하는데 관련된 수열들은 순차적인 방식으로 미리 정해지기 때문에 시작점에 관련된 색인을 알게 되면 실제 어떤 시작점을 사용했는지를 알기 위해서는 해당 테이블에 관련된 모든 수열들을 다시 검사해 봐야 한다. 이것은 TMTO의 개념이 시작점에 한해서 이중으로 적용된 것으로 볼 수 있다.

향 후 시작점에 대한 정보를 수열군의 색인으로 대치하는 현재의 방법을 Hellman와 Rainbow의 방법을 기반으로 DP 나 VDP를 함께 적용시킨 상태에 어떻게 적용시킬지에 대한 연구를 진행할 예정이다. 특히 실제 실현 가능한 예로부터 실제적인 모의실험을 진행할 예정이다.

참 고 문 헌

[1] M. E. Hellman, “A cryptanalytic time- memory trade-off,” *IEEE. Trans. Inf. Theory*, Vol.IT-26, No.4, pp.401-406, July 1980.

[2] S. Babbage, “A space/time tradeoff in exhaustive search attacks on stream ciphers,” in *Proc. IEE European Convention on Security and Detection*, No.408, May 1995.

[3] J. Golic, “Cryptanalysis of alleged A5 stream cipher,” *EUROCRYPT’97*, *LNCS 1233*, pp.239-255, 1997.

[4] J. Borst, B. Preneel, J. Vandewalle, “On the time-memory tradeoff between exhaustive key search and table pre- computation,” in *Proc. 19th Symp. Inf. Theory in the Benelux*, WIC, 1998, pp.111-118.

[5] A. Biryukov and A. Shamir, “Cryptanalytic time/memory/data tradeoffs for stream ciphers,” *ASIACRYPT 2000*, *LNCS 1976*, pp.1-13, 2000.

[6] P. Oechslin, “Making a faster cryptanalytic time-memory trade-off,” *CRYPTO 2003*, *LNCS 2729*, pp.617-630, Aug. 2003.

[7] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat, “A time-memory tradeoff using distinguished points: New analysis & FPGA results,” *CHES 2002*, *LNCS 2523*, pp.593-609, 2003.

[8] J. Hong, K. C. Jeong, E. Y. Kwon, I.-S. Lee, and D. Ma, “Variants of the distinguished point method for cryptanalytic time memory trade-offs,” *ISPEC 2008*, *LNCS 4991*,

pp.131-145, 2008.

[9] G. Avoine, P. Junod, and P. Oechslin, "Time-memory trade-offs: False alarm detection using checkpoints," *INDOCRYPT 2005*, *LNCS 3797*, pp.183-196, 2005.

[10] R. Gold, "Maximal recursive sequences with 3-valued recursive cross-correlation functions," *IEEE Trans. Inf. Theory*, Vol.14, No.1, pp.154-156, Jan. 1968.

김 영 식 (Young-Sik Kim) 정회원

2001년 2월 서울대학교 전기공학부 공학사
2003년 2월 서울대학교 전기·컴퓨터공학부 석사
2007년 2월 서울대학교 전기·컴퓨터공학부 박사
2007년 3월~현재 삼성전자
<관심분야> 암호학, 시퀀스, 오류정정부호, 디지털 통신

임 대 운 (Dae-Woon Lim) 정회원

1994년 2월 한국과학기술원 전기및전자공학과 학사
1997년 2월 한국과학기술원 전기및전자공학과 석사
2006년 8월 서울대학교 전기·컴퓨터공학부 박사
1995년 9월~2002년 8월 LS산전(주) 중앙 연구소 선임 연구원
2006년 9월~현재 동국대학교 IT학부 조교수
<관심분야> OFDM, 부호 이론, 시공간 부호