버킷과 블룸필터를 혼합한 민감한 데이터 보안

유천영* · 김지홍**

The Secure Algorithm on the Sensitive data using Bloom filter and bucket method

Choun-young Yu* · Ji-hong Kim**

이 논문은 교육과학기술부의 지원에 의해 수행된 연구임(과제번호 2010-0023648)

요 약

최근 개인정보 유출 사건과 관련된 사고가 사회적 이슈가 되고 있다. 민감한 개인정보를 암호화하여 데이터를 보호한다면 정보 유출이 확실히 줄어들 것이다. 본 논문에서는 데이터 보호를 위한 기존의 연구 방법을 분석하고, 블룸필터 방식에 버킷을 혼합한 복합적인 방법으로 민감한 정보를 암호화하기 위하여 제안하였다. 가장 많이 사용되는 튜플 암호화 방식에 버킷인덱스 방식을 적용한 방식이다. 이 방식은 버킷정보로 인하여 데이터의 분포가 노출된다는 단점을 가진다. 이러한 버킷인덱스 방식의 데이터 노출 특성을 방지하기 위하여, 본 논문에서는 버킷인덱스와 블룸필터를 적용한 복합적인 암호화방식을 제안한다. 제안된 방식의 특징은 데이터의 노출을 방지할 뿐 아니라 데이터베이스의 검색성능을 향상시킨다.

ABSTRACT

Recently privacy breaches has been an social issues. If we should encrypt the sensitive information in order to protect the database, the leakage of the personal sensitive data will be reduced for sure. In this paper, we analyzed the existing protection algorithms to protect the personal sensitive data and proposed the combined method using the bucket index method and the bloom filters. Bucket index method applied on tuples data encryption method is the most widely used algorithm. But this method has the disadvantages of the data exposure because of the bucket index value presented. So we proposed the combined data encryption method using bucket index and the bloom filter. Features of the proposed scheme are the improved search performance of data as well as the protection of the data exposure.

키워드

데이터베이스, 암호화, 복호화, 긍정오류

Keywords

database, encrypt, decrypt, false positive

* 정회원 : 세명대학교 정보통신학부

** 정회원 : 세명대학교 정보통신학부 (교신저자, jhkim@semyung.ac.kr)

접수일자 : 2012. 03. 13

심사완료일자 : 2012. 03. 30

Open Access http://dx.doi.org/10.6109/jkiice.2012.16.5.939

Ⅰ. 서 론

최근 데이터베이스의 규모가 점차 대형화되고 있는 반면에 유출 사고도 크게 나타나고 있다. 예로써, 2008년 L사의 1,700만 고객정보유출 및 최근까지의 유출사고가 있었고, 최근에는 운전면허 발급업무 마비 사고가 있었는데 증명사진 데이터베이스의 서버 용량 초과가 문제가 되었다. 이처럼 고객정보유출과 용량 과부하의 문제점으로 인하여 데이터베이스의 관리만을 외부 DB 전문 업체에 의뢰하고 대기업이나 컨텐츠 제공 업체는 클라이언트와의 서비스에만 집중하려는 경향이 늘어나고 있다. 그림 1은 DAS(Database As a Service)인 외부 DB 전문 업체와의 협력 시스템인 일반적인 구조이다[1].

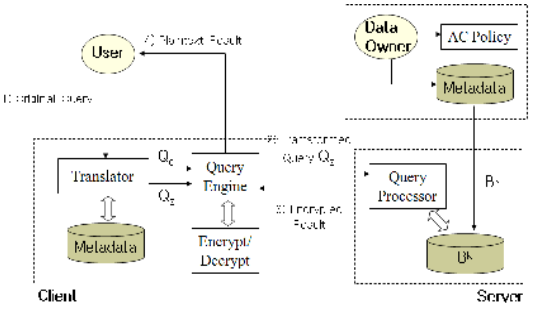

그림 1. 암호/복호 모듈을 사용한 DAS 시스템
Fig. 1 DAS system using the encryption and decryption modules

그림1과 같이, 클라이언트에는 암/복호 기능만을, 외부 DB 관리 서버에는 암호화된 데이터만을 보관한다. 데이터베이스를 암호화함으로서 외부 DB 관리자로 부터의 위험과 데이터 분실 위험에 대비할 수 있으며, 또한 클라이언트와 서버 간에도 암호화된 데이터만이 있기 때문에 노출 및 불법 도청의 위험도 방지할 수 있다[2]. 데이터베이스 암호화 방식으로는 레코드 단위, 튜플 단위, 속성 단위의 암호화 방식으로 구분할 수 있다. 레코드단위의 암호화 방식의 경우 DB 서버와 클라이언트간의 통신량이 증가하고, 클라이언트에서 복호해야 할 데이터양도 크게 증가한다. 속성단위의 암호화방식은 민감한 데이터만을 암호화 하는 장점이 있는 반면에 암호화를 하기 위해 너무 많은 패딩이 필요한 단점이 있다. 튜플 방식의 암호화 방식은 해당 튜플 전체를 복호하고, 다시 필요한 부분만을 선택하여야 한다는 단점이 있기 때문에 버킷 등을 이용한 보완방법이 많이 사용된다[2,3,4].

본 논문은 다음과 같이 구성된다. 2장에서는 데이터 암호화를 위한 최근 연구인 버킷기반 (bucket based Index) 방식, Fragmentation Based(분단 기반) 방식, 블룸필터(bloom filter) 방식을 알아보고, 3장에서는 보다 효율적인 구조인 민감한 정보의 보안을 위한 방법을 제시한다. 개인 데이터 중에서 가장 민감한 데이터(주민등록번호)만을 제안 한다. 고객들의 출생년도를 일정한 범위로 버킷을 설정한다. 즉, 숫자 데이터의 범위를 동일한 크기로 분할하여, 버킷 값을 생성하고, 이를 해싱하여 블룸필터에 적용하는 방식이다. 4장에서는 제안한 방법에 대한 실험 및 결과를 분석 한다. 제안한 방법인 버킷(Bucket) 방식과 블룸필터(Bloom filter) 방식을 결합한 방식의 안전성을 증명한다. 실험은 8등급, 4등급, 2등급 각각으로 적용하여 동일값과 범위값을 실험하였다. 마지막으로 5장에서 결론으로 마무리 한다.

Ⅱ. 최근 연구 동향

hacigumus[2]등이 제안한 버킷기반 (bucket based Index) 방식은 튜플 전체를 암호화하고, 각 필드의 속성 도메인을 버킷 단위로 분류하여 처리하는 기법이다. 레코드는 암호화시켜 E_tuple 필드에 저장하고, 다른 필드들에는 동일한 크기를 가지는 숫자데이터의 임의 구간을 버킷으로 생성되는 값으로 저장되고 검색에는 이 버킷 값을 비교하여 데이터를 검색한다. 예로는 0~9까지의 버킷은 “α”이고, 10~19까지의 버킷은 “β”이다. 이러한 버킷 값들을 각각의 버킷 필드에 저장하고, 검색 시에는 버킷 값을 비교하여 원하는 결과를 추출한다.

스탠포드 대학의 G. Agarwal에 의해 제안된 Fragmentation Based(분단 기반) 방식[4]은 아웃소싱 데이터베이스에 대한 보안방법으로 튜플에 저장된 속성들을 특성에 의해 서로 분리된 서버들에 저장하는 방식

으로, 두 개의 서버 간에는 일종의 장벽이 설치되어 서로 통신할 수 없도록 구성된다.

블룸필터(bloom filter) 방식인 키워드 검색이 있는데, 이 방식은 m 비트로 구성된 필터를 사용하며, n개의 키워드 요소에 대하여 k 개의 해쉬 함수 결과에 따라 해당 비트를 "1"로 설정하는 방식이다. 예를들면, "Apple"이라는 데이터는 E-tuple 필드에 저장되고, 그 해쉬 결과가 {29, 79, 118}일 때에 Bloom 필드의 해쉬 결과 위치에는 "1"로 세팅되어 저장된다. 검색시에는 검색하려는 데이터의 해쉬 결과와 Bloom 필드의 "1"로 세팅된 결과가 동일하다면 "Apple"이라는 데이터를 얻게 된다. 블룸필터 방식은 검색하려는 키워드의 해쉬값이 데이터베이스와 일치하는지의 여부로 존재를 확인하기 때문에, 여러 개의 키워드 요소에 의해 해쉬 값이 동일하게 설정될 수 있으므로 긍정 오류(false positive)를 가질 수 있다. 만약, "Banana"와 "Orange" 데이터가 E-tuple에 저장되고, 각각의 해쉬 결과가 {17, 79, 120}과 {58, 94, 118}로 세팅되었다면, "Apple"과 "Banana", "Orange"가 DB에 저장되어있고, 검색하려는 데이터가 "Grape"이며, 해쉬 결과가 {29, 94, 120}일 때, 데이터베이스에는 "Grape" 데이터가 존재하지 않더라도 해쉬 결과의 위치가 이미 "1"로 세팅되어있어 존재하고 있다는 것으로 인식된다. 이러한 결과를 긍정 오류라고 한다. 블룸필터는 비록 긍정 오류가 있어도 1차 검색 후, 2차 복호화 과정을 거쳐서 올바른 결과 값을 찾을 수 있는 방식이다[5].

Ⅲ. 민감한 정보 DB 보안 방법

3.1. 민감한 정보 DB 보안 방법 제안

본 논문에서는 외부 DB 서버에 저장되는 고객들의 개인 정보 데이터 중에서 가장 민감한 숫자 데이터(주민등록번호) 만을 제안하고자 한다. 고객들의 출생년도를 일정한 범위로 분리되는 버킷을 설정할 때 먼저 구간별로 버킷을 설정한다. 즉, 숫자 데이터의 범위를 동일한 크기로 분할하여, 버킷 값을 생성하고, 이를 해싱하여 블룸필터에 적용하는 방식이다. 그림 2는 각 영역에서의 값에 대한 버킷 값의 생성의 예이다.

α	β	γ	δ	ε	ζ	η	θ
1940~1949	1950~1959	1960~1969	1970~1979	1980~1989	1990~1999	2000~2009	2010~2019

그림 2. 출생년도에 대한 버킷구분 예
Fig. 2 The bucket classification for Year of Birth

예를 들어, 1976년도의 데이터를 저장할 때에 버킷값은 "δ" 이고, 버킷 값의 해싱 결과가 {2, 80, 211}로 나왔을 때, 1976년도의 데이터는 E_Tuple 필드에 암호화되어 저장되고, Bloom 필드에 해싱 값의 위치 비트를 "1"로 저장한다. 그리고 검색 시에는 1976년도를 포함하는 버킷값 "δ"을 해싱 하여 나온 결과를 Bloom 필드와 AND 연산으로 "1"이 되는지를 검사하여 원하는 결과를 추출 후 복호화 하는 방법이다.

3.2. 실험 데이터베이스 샘플 및 블록도

그림 3은 실험을 위한 평문 데이터베이스이며, 이와 동일 시 되는 데이터를 버킷 방식과 제안하는 방식의 데이터베이스에도 적용한다. 또한 버킷방식과 제안 방식의 암호화 과정(E_Tuple 필드)은 AES 암호 알고리즘을 사용하였다.

name	jumin_1	jumin_2	address_1	address_2	blood	height	weight
장규영	500818	1010043	제주도	서귀포	B	189	33
정영현	721111	1010088	충청북도	충주	O	192	34
김병영	941213	1010133	서울	강남	A	194	35
박은용	600826	1010193	전라남도	광주	O	195	88
장영성	411227	1010269	충청북도	충주	AB	188	55
차남남	740223	1010274	충청남도	대전	AB	150	79
최혜홍	450618	1010284	경상남도	진주	AB	186	37
정병대	640229	1010314	제주도	서귀포	A	191	56

그림 3. 평문 데이터베이스
Fig. 3 The plaintext database

그림 4는 버킷만을 적용한 데이터베이스 블록도이고, 그림 5는 그림 3과 동일 시 되는 데이터를 전체 튜플을 암호화한 암호화 값과 필드별로 분류된 해당 버킷을 적용한 데이터베이스를 보여주고 있다.

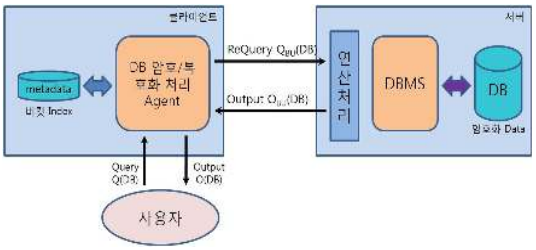

그림 4. 버킷 인덱스 방식 블록도
Fig. 4 The block diagram of the bucket index method

E_tuple	n_id	j1_id	j2_id	a1_id	a2_id	b_id	h_id	w_id
ZVlKTV6vQN2Y+QpQ1CHjRwSEi0d3i6pu7upkISOSA...	6	B	B2	W	W	3	a	d
FlajlzoarXnUxC4XcSGUvnYKSwli3WI5wYbG1YYb8A...	6	B	B2	T	T	11	b	b
ahVWy4OO6HRJnL8mI49iF7oM47+ltRAdBWi9yK5o6...	2	A	A2	P	P	11	d	c
dhF0N4eaT20jsf/HWe8iMdd1m6Q8FRlqCvn5229pq...	7	B	B2	W	W	6	e	a
t8U8SM0/yWNBislZCABg5EBpf5cGZyD+oinqpkj3TN...	2	A	A2	O	O	5	d	e
hTMU2wbwICMuxyM7u6HSKyL4gqq3toboKVCR3qT...	5	B	B2	R	R	5	f	f
RkdMM6DfL9mFezXup3kK2efZJkID4+P/jiO4TBxLgl...	6	B	B2	O	O	5	a	a
Wr2jdmV0uBAfeZYxoQ1gqcx3usgcozd3aPlYSxoOLu...	6	A	A2	Y	Y	11	g	d
/m8mJFQnW429aqCLVLE8Np4lo4X06jiXgBwzqPS6...	7	B	B2	Q	Q	11	b	e
clWRoZYlpAHI6D6Gp5v/Aiszf4KOju/xlysM4Ws00Tx...	6	B	B2	I	I	6	a	b
QoYLA2R7YoCx9R51jb4VX7SVxQkHsJo1pPr9qC6ia...	8	A	A2	E	E	6	c	c
FzP8k+kfE7tyaqLeGNkUWldJBCGiHMtYeBS3lGvvO...	5	B	B2	I	I	6	e	e

그림 5. 암호문과 버킷 적용 데이터베이스
Fig 5. The encrypted data and the bucket indexed data base

그림 6은 본 논문의 제안 방식으로 블룸필터와 버킷을 같이 적용한 블록도와 그림 7은 그림 6에서 민감한 숫자 데이터의 버킷 인덱스 값을 해싱한 결과를 블룸 필드에 추가 삽입되는 예시 블록도이다. 그림 8은 그림 3과 동일 시 되는 데이터베이스를 전체 튜플을 암호화 한 값과 블룸 필터를 적용한 데이터베이스를 보여주고 있다.

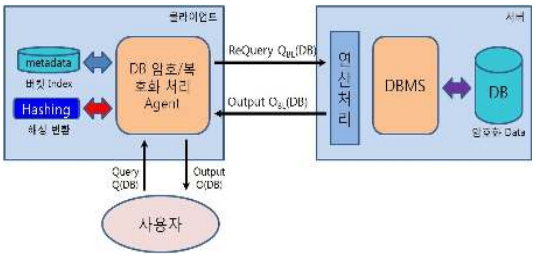

그림 6. 블룸필터와 버킷 적용 방식 블록도
Fig. 6 The block diagram of the bucket indexed method using bloom filter

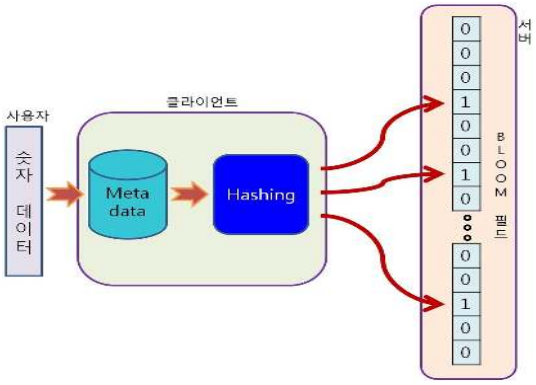

그림 7. 제안 방식
Fig. 7 The proposed algorithm

E_tuple	bloom
ZVlKTV6vQN2Y+QpQ1CHjRwSEi0d3i6pu7upkISOS...	00000000001000000000000000000010010000000010100001...
FlajlzoarXnUxC4XcSGUvnYKSwli3WI5wYbG1YYb8...	00000000000000000000000000000011000000000001000001...
ahVWy4OO6HRJnL8mI49iF7oM47+ltRAdBWi9yK5o...	00000000000000000000000000000010000000000010000...
dhF0N4eaT20jsf/HWe8iMdd1m6Q8FRlqCvn5229pq...	00000000001000000000000000000010000000000001000001...
t8U8SM0/yWNBislZCABg5EBpf5cGZyD+oinqpkj3TN...	00000000010000000000010000000010000000000010000...
hTMU2wbwICMuxyM7u6HSKyL4gqq3toboKVCR3qT...	00000000000000000000010000010000000000001001000...
RkdMM6DfL9mFezXup3kK2efZJkID4+P/jiO4TBxLgl...	00000000011000000000001000001000000000001000001...
Wr2jdmV0uBAfeZYxoQ1gqcx3usgcozd3aPlYSxoOLu...	00000000000000000000010000000000000000000010001...
/m8mJFQnW429aqCLVLE8Np4lo4X06jiXgBwzqPS6...	00000000000000000000000000000011000000000001000000...
clWRoZYlpAHI6D6Gp5v/Aiszf4KOju/xlysM4Ws00Tx...	00000000001000000000000000000011000000000001000001...
QoYLA2R7YoCx9R51jb4VX7SVxQkHsJo1pPr9qC6ia...	00000000000010000000000000000000000000000001000001...
FzP8k+kfE7tyaqLeGNkUWldJBCGiHMtYeBS3lGvvO...	00000000000000000000001000000010000000000001000001...
2cfLU/orgOgWPhXoDj/SsdnKabiYx6b8uD0t1UR48...	00000000001000000000001000000010000000000010100001...
lQ1DgINf413cs301lTi8WFh82D3eHaKcwV63yjfnW+l...	00000000000000000000001000000000000000000001001...

그림 8. 제안 방식에서의 암호문 데이터베이스
Fig. 8 The encrypted data base constructed using the proposed method

그림 8의 내용은 3.1 절에서 기술했듯이 해당 범위의 버킷 값을 해싱한 결과가 저장되었고, Bloom 필드는 256 비트를 배열에 문자형으로 저장되었다. 저장방식은 해쉬 알고리즘인 MD5, SHA-1, SHA256 의 3종류를 사용해서 원본의 각 필드에 해당하는 버킷 값을 해싱한 3개씩의 해쉬 값을 Bloom 필드에 적용하여 저장 되었다.

Ⅳ. 실험 및 결과

4.1. 데이터베이스 보안 실험

본 논문에서는 민감한 숫자 범위형 데이터 검색 방식만을 제안하고 있기 때문에 고객정보 데이터 중에서 주민등록 번호에 대한 표 1 과 같은 버킷 데이터 예를 이용하고, 쿼리문에도 이용하고자 한다. 본 논문에서의 실험은 10만명의 데이터베이스로 8등급, 4등급, 2등급 각각

으로 적용하여 동일값과 범위값을 실험하였다.

표 1. 등급별로 적용된 버킷 데이터
Table. 1 The bucket index according to by grade class

버킷적용(8등급)		버킷적용(4등급)		버킷적용(2등급)	
출생년도	j_id	출생년도	j_id	출생년도	j_id
1940~1949	α	1940~1959	α	1940~1979	α
1950~1959	β				
1960~1969	ɣ	1960~1979	β		
1970~1979	δ				
1980~1989	ε	1980~1999	ɣ	1980~2019	β
1990~1999	ζ				
2000~2009	η	2000~2019	δ		
2010~2019	Θ				

실험에 앞서 먼저 1988년생만을 검색하는 평문 DB에 동일값 쿼리는, " SELECT * FROM Company WHERE Jumin like '88%' " 가 된다. 이 쿼리를 버킷만을 적용한 암호화 DB에 동일한 쿼리는 " SELECT E_Tuple FROM Company WHERE j_id = 'ε' " 가 되는데 결국에는 범위형 검색과 같아서 해당 범위를 먼저 가져와 복호화한 배열에 다시 동일값 검색을 하여 추출한다. 논문에서 제안하는 버킷과 블룸을 적용한 데이터베이스에서 동일값 검색 시에 먼저 "1988"년생의 버킷 값은 'ε'이고, 이 값의 해싱 결과 값{ 17, 89, 169 }이 생성 되었을 때 쿼리문은 "SELECT E_Tuple FROM Company WHERE (CONVERT(bit, SUBSTRING([bloom], 17,1)) & 1) = 1 AND (CONVERT(bit, SUBSTRING([bloom], 89,1)) & 1) = 1 AND (CONVERT(bit, SUBSTRING([bloom], 169,1)) & 1) = 1 " 이 된다. 여기까지 각각의 데이터베이스에서 동일값 검색의 쿼리문을 확인하였고 검색 시간 결과를 표 2 에서 확인할 수 있다.

표 2. 동일값 데이터 검색 결과
Table. 2 The same value data search results

결과 / 적용구분	동일값 검색 평균 시간	비고
평문	00.0036001	
버킷	00.0356020	
버킷 + 블룸	00.0392026	제안방법

이제 본 논문에서 제안하는 민감한 데이터인 숫자 범위형 검색의 쿼리를 확인한다. 범위 검색을 8등급, 4등급, 2등급별로 구분하여 각각의 DB를 만들고 테스트를 하였다. 평문 데이터베이스에서 범위 검색방식을 먼저 보면, 1980년도에서 1989년도까지의 데이터를 검색하는 쿼리를 보낼 때 " SELECT * FROM Company WHERE Jumin like '8%' " 가 된다. 이 쿼리를 그림 5처럼 버킷 인덱스를 적용한 암호화 데이터베이스(8등급)에 동일한 범위를 검색하게 된다면 "SELECT E_Tuple FROM Company WHERE j_id = 'ε' " 인 쿼리가 서버에 전송된다. 4등급으로 적용된 데이터베이스에 사용되는 쿼리로는 "SELECT E_Tuple FROM Company WHERE j_id = 'ɣ'" 이고, 2등급으로 적용된 데이터베이스에 사용되는 쿼리로는 "SELECT E_Tuple FROM Company WHERE j_id = 'β' " 로 전송이 된다. 블룸 필터는 한 개의 데이터(키워드)만으로 검색하고자 경우에는 데이터에 생성되는 3개의 비트 위치가 일치해야 원하는 데이터를 얻게 된다. 그래서 버킷과 블룸을 혼합하여 사용한다면 민감한 정보의 유추를 막을 수 있다.

본 논문에서 제안하는 방식은 블룸 필터와 버킷을 결합하여 적용한 데이터베이스 검색이다. 검색하려는 키워드의 버킷값을 해싱하여 블룸필드에 삽입된 비트와 비교한다. 이 방식으로 버킷을 8등급으로 가정 했을 때 위와 동일한 검색은 버킷 결과 값이 "ε"이고, 해싱 결과 값이 { 3, 13, 135 }로 생성되었을 때에 쿼리문으로는 "SELECT E_Tuple FROM Company WHERE (CONVERT(bit, SUBSTRING([bloom], 3,1)) & 1) = 1 AND (CONVERT(bit, SUBSTRING([bloom], 13,1)) & 1) = 1 AND (CONVERT(bit, SUBSTRING([bloom], 135,1)) & 1) = 1 " 이 된다. 이렇게 생성된 해쉬 값과 Bloom 필드의 배열 값이 AND 연산으로 3개의 "1"이 만들어진다면 해당되는 E_Tuple 값을 클라이언트의 복호화를 거쳐서 사용자에게 원하는 데이터를 전송시킨다. 한 개라도 "0" 이 된다면 데이터가 없다는 뜻이다. 이 쿼리문을 보면 해당 비트 위치만이 노출이 되기 때문에 공격자는 그 비트가 무엇을 의미하는지를 전혀 모르기 때문에 데이터 유출 위험이 없다.

4.2. 데이터베이스 보안 실험 결과

본 논문에서는 제안한 고객의 민감한 데이터 보안을 위한 검색 실험을 하였다. 실험에 사용된 환경으로는 컨

텐츠 업체에서는 Windows 7 Home Premium K 의 운영체제와 Visual C# 2010을 데이터 검색용으로 사용하였고, 외부 DB 관리 업체(DAS)에서는 Windows 7 Home Premium K 인 운영체제와 MS SQL Server 2008을 DB 서버로 사용하였다. 평문 데이터베이스와 버킷 적용 데이터베이스, 제안하는 블룸과 버킷을 혼합 적용한 데이터베이스 순으로 실험 결과를 확인한다. 먼저 평문 데이터베이스에서 “1980~1989”까지의 검색 결과를 그림 9에서 확인할 수 있다.

그림 9. 평문 DB 검색 결과
Fig. 9 Data search results with the plantext DB

그림 10 은 버킷 인덱스 방식을 사용한 DB의 검색 결과이다. “1980~1989”까지의 검색을 3가지 경우(2등급, 4등급, 8등급)를 적용하여 실험하였다. 등급별로 적용한 데이터베이스의 검색 결과 중에서 8등급으로 적용된 결과를 확인할 수 있다. 버킷 인덱스 방식은 데이터베이스 보안도 되면서 컨텐츠 업체의 DB 관리도 가능하다. 하지만 버킷값이 그대로 DB에 기록이 되어있기 때문에 수만번의 호출에 따른 빈도수와 표 1 같은 데이터와의 비교로 원본의 추측 값을 얻을 가능성이 있다.

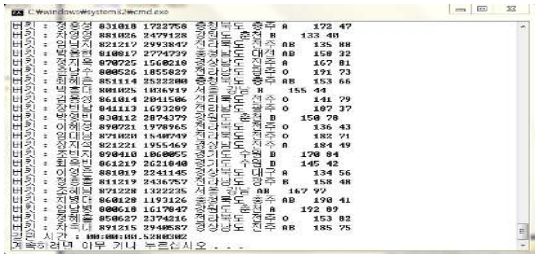

그림 10. 버킷 인덱스 방식 검색결과
Fig. 10 Data search results with the bucket indexd method

그림 11 은 본 논문에서 제안하는 블룸 필터와 버킷을 혼합 적용한 데이터베이스의 검색 결과이다. 버킷과 동일한 등급을 적용하여 실험하였다. 각각의 데이터베이스별 성능 평가 결과를 표 3에 정리하였다.

그림 11. 제안 방식 검색 결과
Fig. 11 Data search results using the proposed method

표 3. 등급별 성능 평가 비교
Table. 3 The performance comparison according to class level

등급	DataBase	버킷	버킷 범위	검색 범위	False Positive (%)
2	평문	X	X	1980 ~ 1989	True Positive
	암호+버킷	’β’	1980 ~ 2019		24.16 %
	암호+블룸				
4	평문	X	X		True Positive
	암호+버킷	’ɣ’	1980 ~ 1999		15.82 %
	암호+블룸				
8	평문	X	X		True Positive
	암호+버킷	’ε’	1980 ~ 1989		7.49 %
	암호+블룸				

각각의 데이터베이스 쿼리 실험 응답 시간의 등급별 결과를 표 4, 표 5, 표 6에서 확인 할 수 있다.

표 4. 2 등급 버킷 데이터 검색 결과
Table. 4 2 level partitioned bucket data search result

결과 / 적용구분	숫자 범위형 검색 평균 시간	비고
평문	00.1160070	
버킷	01.0438598	
버킷+블룸	01.0716612	제안방식

표 5. 4 등급 버킷 데이터 검색 결과
Table. 5 4 level partitioned bucket data search result

결과 / 적용구분	숫자 범위형 검색 평균 시간	비고
평문	00.1192074	
버킷	00.8358466	
버킷+블룸	00.9192531	제안방식

표 6. 8 등급 버킷 데이터 검색 결과
Table. 6 8 level partitioned bucket data search result

결과 / 적용구분	숫자 범위형 검색 평균 시간	비고
평문	00.1168066	
버킷	00.6320366	
버킷+블룸	00.7974453	제안방식

실험 결과로 확인할 수 있듯이 제안하는 방식이 시간이 조금 더 많이 걸린다. 하지만, 버킷을 적용한 보안 방법에서는 버킷의 단위를 줄이면 줄일수록 원본 데이터 추측 가능성이 더욱 더 커지고 False Positive 확률이 커지지만, 본 논문에서 제안하는 방법은 버킷 단위를 훨씬 더 줄이더라도 데이터의 추측 가능성이 전혀 없고, False Positive 확률이 적어서 데이터 보안에 더 효율적임을 확인 할 수 있다.

V. 결 론

본 논문에서는 버킷방식과 블룸필터 방식을 결합한 민감한 데이터 보안을 위해 검색방법을 제안하고 실험하였다. 평문 검색방식의 경우 해당 테이블에 존재하는 데이터 값 그대로 쿼리하기 때문에 쉽게 노출이 된다. 버킷 인덱스 검색 방식의 경우는 원본의 내용을 알 수는 없지만 버킷 값이 DB에 존재하고 있어 표1과 같은 데이터와 수 만 번의 호출에 의한 빈도수를 비교하여 원본의 추측 값을 얻을 수가 있다. 예로서, 4장의 평문 검색 방식은 원본의 내용을 쿼리를 하게 되고, 버킷 검색 방식은 쿼리문에서 'ε'의 버킷 값이 버킷 테이블에 그대로 존재하기 때문에 이 버킷 값의 호출 빈도수에 의한 원본 결과 값을 추측할 가능성이 있다.

본 논문에서 제안하는 블룸 필터와 버킷을 결합한 방식은, 민감한 숫자 데이터에 동일한 크기의 버킷을 적용하고, 적용한 버킷 값을 해싱한 결과를 Bloom 필드와 비교하는 특성으로 DB 원본이나, 쿼리 에도 버킷의 해싱 결과만이 노출이 된다. 이것만으로는 원본의 의미를 공격자는 전혀 알지 못하기 때문에 데이터의 노출이 불가능하다. 또한, 버킷을 적용한 방식에서는 버킷의 단위를 줄이면 줄일수록 원본 추측 가능성이 커지고 False Positive 확률이 커지지만, 제안하는 방식에서는 버킷의 단위를 줄이더라도 원본 추측 가능성이 전혀 없고, False Positive 확률이 적어서 보다 나은 데이터 보안 방안을 제공한다.

향후에는 본 논문의 결과로서 제시된 데이터베이스 보안 방법을 숫자형 데이터베이스 뿐 아니라, 문장형 데이터를 포함하는 모든 필드에 버킷과 블룸 필터를 적용하기 위한 연구를 진행할 예정이다. 또한 본 논문의 결과는 각종 민감 데이터를 보관하고 있는 데이터베이스의 보안과 검색 성능을 향상시키기 위한 분야에 활용될 수 있을 것으로 기대된다.

참고문헌

[1] Hakan Hacigûmûs, Bala Iyer, and Shrad Mehrotra, "Providing database as a service", In Proc, of the 18th International Conference on Data Engineering, San Jose, California, USA, IEEE Computer Society, pp.21-28, 2002.

[2] Hakan Hacigûmûs, Bala Iyer, Chen Li, and Shrad Mehrotra, "Executing SQL over encrypted data in the database service provider model", In Proc. of the ACM SIGMOD, pp.216-227, 2002.

[3] Hakan Hacigümüs, Bala Iyer, and Shrad Mehrotra, "Efficient exeurtion of aggregation queries over encrypted relational databases", In Lee, J., li, J. Wudhang, K., and Lee, D., Eds., Proc. of the 9th International Conference on Database Systems for Advanced Applications, Volume 2973 of Lecture Notes in Computer Science, Jeju Island, Korea, Springer, pp.123-132, 2004.

[4] Aggarwal, Gagan and Bawa, Mayank and Ganesan, Prasanna, "Two can keep a secret: a distributed architecture for secure database services", In Proc. of the Second Biannual Conference on Innovative Data Systems Research (CIDR 2005), Asilomar, CA, pp. 34-42, 2005.

[5] Andrei Broder, Michael Mitzenmacher, "Network Applications of Bloom Filters:", A Survey. Internet Math. Volume 1, Number 4, pp,89-95, 2003.

저자소개

유천영(Choun-young Yu)

2005년 2월 : 세명대학교 컴퓨터과학과 졸업
2007년 8월 : 세명대학교 전자계산 교육대학원 석사
2008년 3월～현재 : 세명대학교 전산정보 일반대학원 박사수료
※관심분야 : 정보보호, 데이터베이스 보안, 네트워크 보안

김지홍(Ji-hong Kim)

1982년 2월 : 한양대학교 전자공학과 학사
1984년 2월 : 한양대학교 대학원 전자통신공학과 석사
1996년 2월 : 한양대학교 대학원 전자통신공학과 박사
1991년 3월 ～ 현재 : 세명대학교 정보통신학부 교수
※관심분야 : 네트워크 및 정보보호, 의료정보 데이터베이스 보안