2014년 전망,
금융 IT Innovation 컨퍼런스

금융권의 최적화된 고객정보 암호화 구현전략

2013. 12 노웅영

고객정보 암호화 왜 필요할까?

개인정보보호법 강화

2013년 8월 6일 일부개정
2014년 8월 7일부터 시행

1 주민등록번호 수집 법정주의

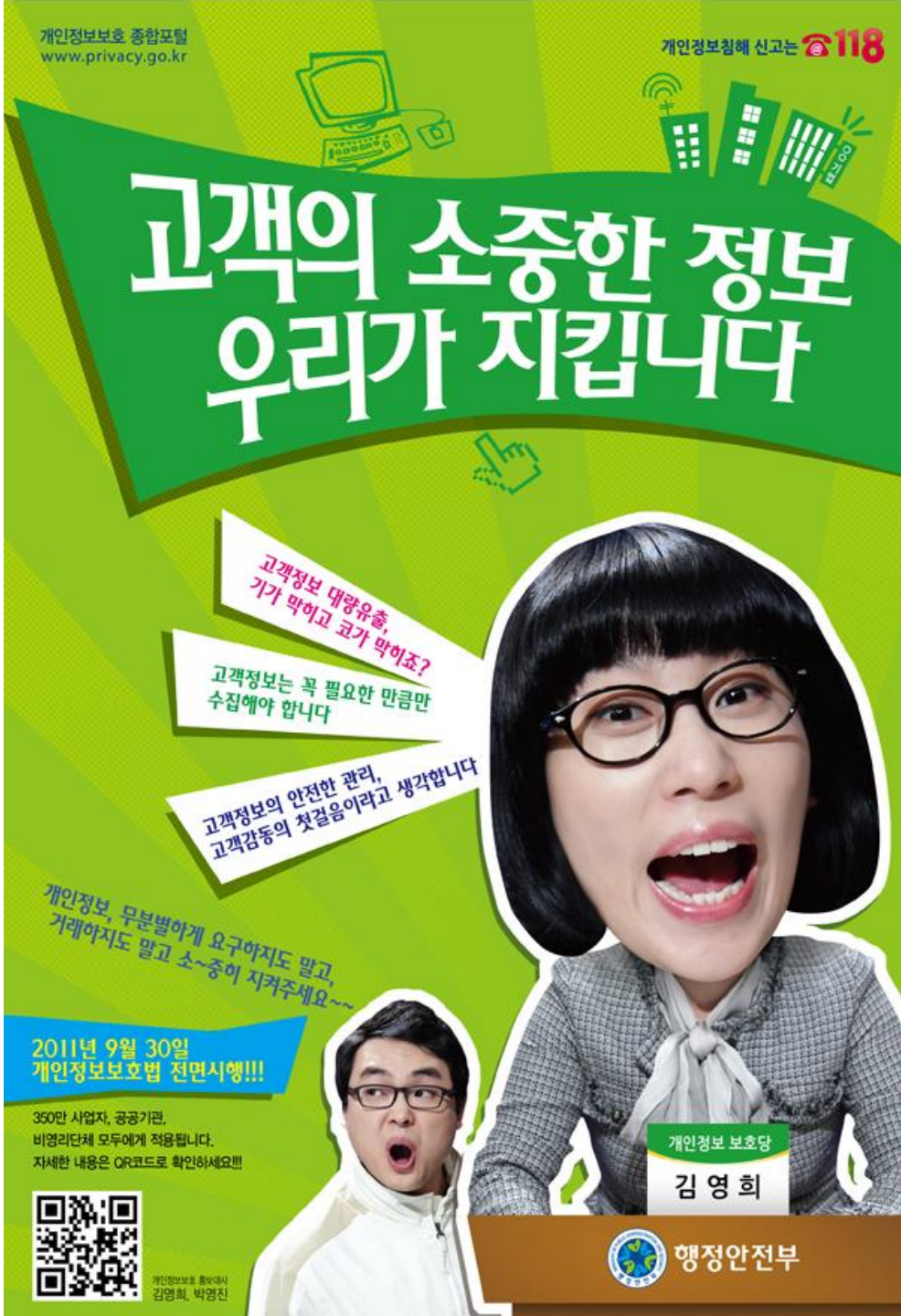
주민번호 수집을 원칙적으로 금지
기존에 수집한 주민번호 중 법령 근거가 없는 경우에는 법 시행 후 2년 이내에 파기

2 과징금 제도

주민번호 유출 시 5억원 이하의 과징금 부과
단, 안정성 확보조치를 모두 이행한 경우 제외

3 CEO 징계권고 제도

책임 있는 자에 해당 기관 대표자 및 책임 있는 임원이 포함 명확화



주요 개인정보 유출 사례

1 번 이상 개인정보 유출 경험

2 년 동안

6 천만 건 이상 유출

시기	해당기관	피해규모
2013년 2월	H사	148만 건
2013년 1월	S사	7,700만 건
2012년 7월	K사	870만 건
2012년 5월	E사	400만 건
2012년 3월	S사	20만 건
2011년 11월	N사	1,320만 건

배상소송
소송비용 발생
손해배상금 지급

기존 고객이탈
잠재고객 외면

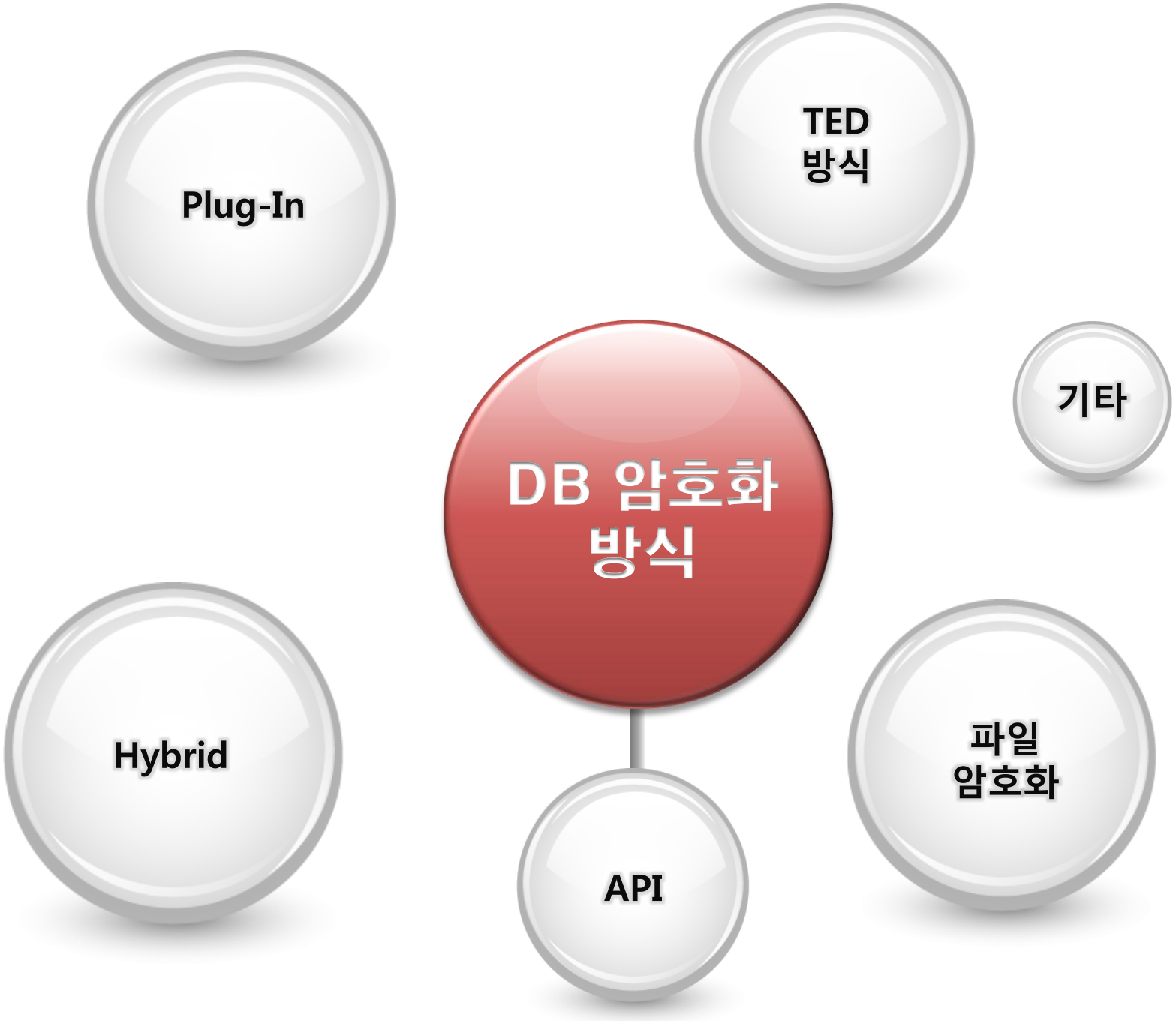
기업이미지 실추
매출 감소
가치 하락

경쟁사의 비방 대상

데이터베이스 보안 솔루션 방식별 비교

종류	개요	장점	단점
DB접근 제어 및 감사	▶사용자별 권한을 할당하고 SQL 제어를 통해 불법적인 접근을 차단 ▶접속 세션에 대한 모니터링 수행	▶부여된 접근권한에 따른 다양한 접근제어 가능 ▶해당 세션에 대한 모니터링 가능 ▶허가되지 않은 외부 침입에 대한 효과적 대응	**▶DB 자체 유출 후 해당 데이터 악용에 무방비**
DB 암호화	▶DB 테이블 암호화 ▶컬럼별로 암호화 적용 가능 ▶권한이 있는 자만 복호화 수행	▶DB 내의 데이터 파일 및 기타 물리적 방법으로 인한 불법 유출 방지 ▶DB 내 모든 데이터에 접근할 수 있는 DB 관리자로부터 민감한 정보 보호에 효과적	**▶기 구축된 DB에 적용 어려움** **▶인덱스 컬럼 암호화로 인한 성능저하 우려**

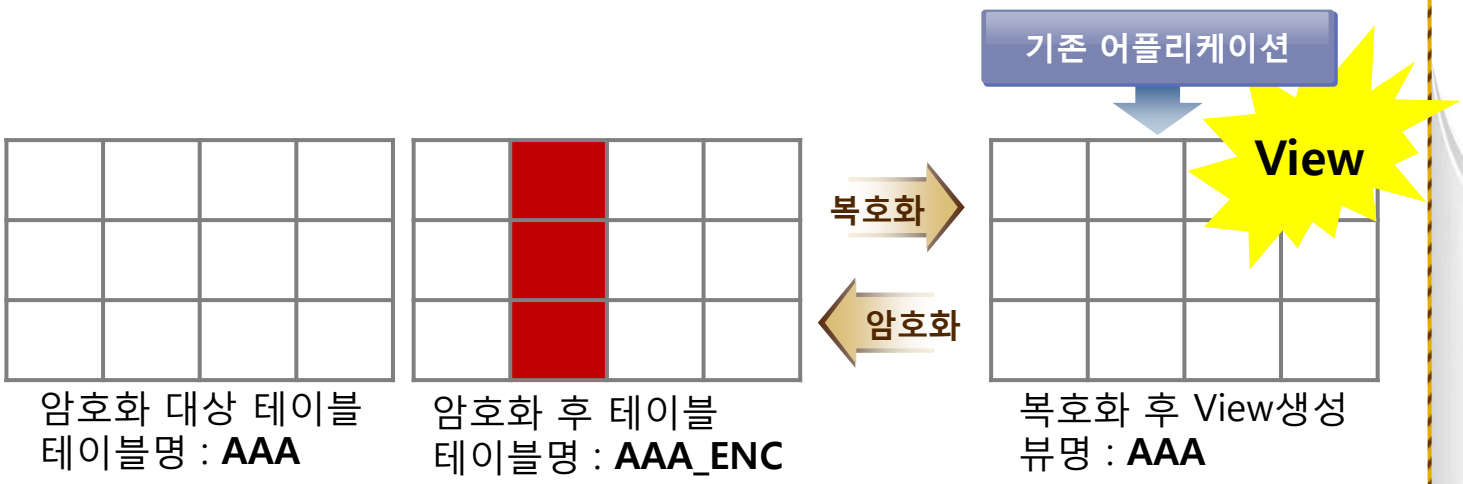
DB 암호화 기술


Plug-In
TED
방식
기타
DB 암호화
방식
Hybrid
파일
암호화
API

기존 DB 암호화 방식

구분	Plug-In 방식		API 방식
	View Trigger 방식	DB API 방식	Application API 방식
기준	DB 서버에서 암•복호화 수행 (암•복호화 모듈이 DB 서버에 Plug-In)		암•복호화가 DB 서버 이외에서 수행
설명	▶암호화 된 컬럼을 포함한 테이블에 대하여 평문 형태의 View를 만들어 활용 ▶응용 프로그램은 View를 참고, View 내용 변경 시 Trigger를 통하여 원본 테이블에 암호화 저장	▶DB 내의 외부함수 호출을 통한 암•복호화 수행	▶응용프로그램 내에서 암•복호화 수행
장점	▶구축 용이	▶Application API 효율성 유지	▶부하 분산
단점	▶성능이슈 발생	▶쿼리 수정	▶프로그램 수정

기존 DB 암호화 방식
- View Trigger 방식



- **View :** 임시적으로 만든 가상의 테이블
- **Trigger :** 해당 테이블의 특정 작업에 대하여 수행할 작업 정의
  예) 뷰 AAA에 대하여 값을 읽을려고 하면(SELECT) 테이블 AAA_ENC에서 복호화해서 보여줘라
  뷰 AAA에 값을 입력(INSERT/UPDATE)하면 암호화 해서 AAA_ENC에 저장하라

```
CREATE OR REPLACE FORCE VIEW  AAA ("COL1", ...) AS
  SELECT cast (DECRYPT(e.COL1) as VARCHAR2(20)) COL1, ... from  AAA_ENC e;
```

```
create or replace TRIGGER AAA_trg  INSTEAD OF INSERT ON AAA FOR EACH row
BEGIN
IF (inserting) THEN INSERT INTO AAA_ENC(COL1, ...)
                    VALUES (ENCRYPT(:new.COL1), ...);
```

DB 서버에서 암복호화

사용자 애플리케이션
수정 없음

View 생성으로 지나친
속도저하

기존 DB 암호화 방식
- DB API 방식

- DB 서버에서 암복호화
- DB 내의 외부함수 호출을 통한 암복호화 수행
- Application 쿼리 수정

DB API

기존 쿼리		암호화 수행 후
select id from table1	수동변환	select DECRYPT(id) from table;
insert into table1 value('7712121234567')		insert into table1 value(ENCRYPT('7712121234567'));

기존 DB 암호화 방식
- Application API 방식

- AP 서버에서 암복호화
- Application 내의 함수 호출을 통한 암복호화 수행
- Application 코드 수정

Application API

기존 프로그램

```
string a = '7712121234567';
insert into table1 value(a);

a= select id from table1;
Print (a);
```

수동변환

암호화 수행 후

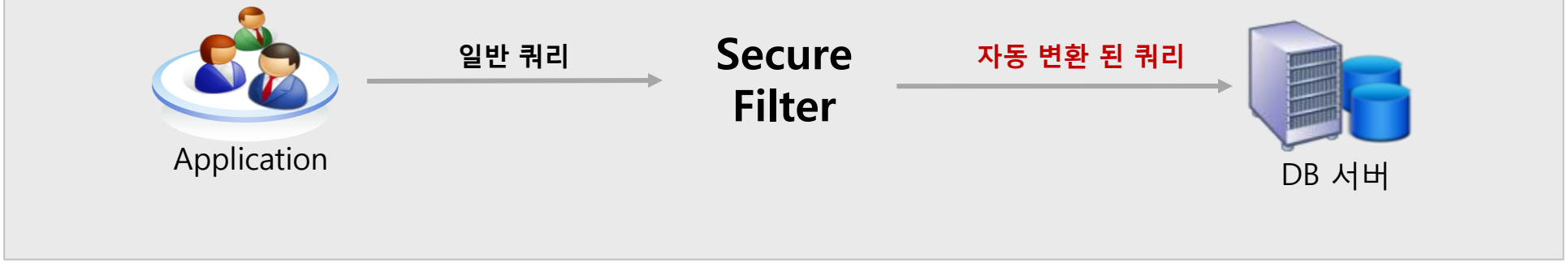
```
string a = ENCRYPT('7712121234567')
insert into table1 value(a);

a= select id from table1;
Print DECRYPT(a);
```

新 DB 암호화 방식
- Secure Filter 방식

DB 서버에서 암복호화
DB API 방식에서 Parser 기능을 결합한 방식

DB API 방식의 쿼리 수정을 Secure Filter에서 자동으로 변환



select id from table1

insert into table1
value('7712121234567')

자동변환

select DECRYPT(id) from table;

insert into table1
value(ENCRYPT('7712121234567'));

구축 난이도

Application API > DB API >> Secure Filter > View Trigger

성 능

(Application API = DB API = Secure Filter) > > View Trigger

금융권
고객정보 암호화
가능할까?


암호화 적용하면
시스템 속도가
느려진다는데..
법적으로 안 할
수도 없고..
장애가 발생하면
어떡하지??
엄청난 규모의
비용이 드는데..
아.. DB 암호화
어렵다..
적당한 솔루션이
있을까??

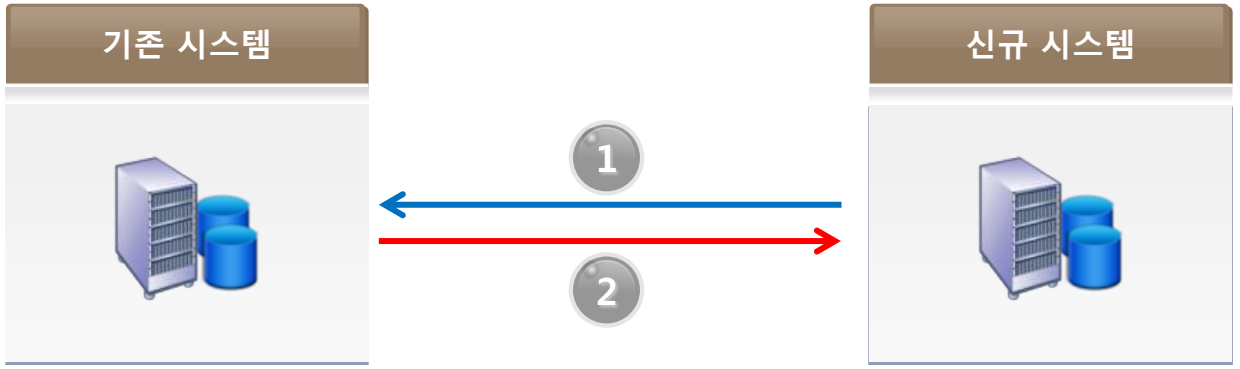
- 모델 정규화로 암호화 대상 최소화!
- API 방식 우선 적용!
- 기존 시스템과의 연동 방안 고민!

신규 시스템 구축 시

기존 시스템 적용 시

- Application 로직 변경 리스크!
- 영향도 분석에 따른 공수 및 방법론 고민!

암호화 전략 1
- 시스템간 연계를 가능케 하라!



Case 1

✓ **신규 시스템이 기존 시스템의 DB를 액세스 하는 경우**
- 암호화 기구축 시스템 : View 액세스 및 패키지 사용
- 암호화 비구축 시스템 : 문제 없음

Case 2

✓ **기존 시스템이 신규 시스템의 DB를 액세스 하는 경우**
- 신규시스템을 **View-Trigger 방식**으로 구성할 경우
  : 신규 시스템의 View 액세스 ➔ **신규 시스템 성능 이슈 발생**
- 신규시스템을 **API 방식**으로 구성할 경우
  : 신규 시스템의 API 사용 ➔ **기구축 / 비구축 시스템 변경 필요**

View-Trigger 사용 불가에 따른 대체 신기술 필요

영향도 최소화를 위한 자동 API 변환 기술 필요

암호화 전략 2

- 암호화 후 시스템 성능을 보장하라!

암호화 이전 평문	순서
520308-1284562	1
660902-1378136	2
781230-1875346	3
831025-2648515	4
930511-2846864	5

기타 알고리즘

전체 데이터를 각각 복호화 후

검색 후 출력하므로

시스템 부하 증가 문제점 노출

암호화 이후	순서
ei83zh-jx5s7w4l	4
wovui3-45dow5d	5
ci63jv-kvm14mw	3
0v4kjz-cp7q9ls	1
c4kehq-is57s87	5

암호화 이전 평문	순서
520308-1284562	1
660902-1378136	2
781230-1875346	3
831025-2648515	4
930511-2846864	5

OPE

(Order Preserving Encryption)

암호화 이후에도 순서 유지

암호화 이후	순서
10hw15-gaser85	1
156s5f-5s45sea	2
acgea4-peiz58s	3
ewb4b6-m5s8ge	4
fetg3-q8xdw45	5

암호화 전략 3

- 영향도 분석, 실시간 모니터링, 원활한 수행 보장하라!

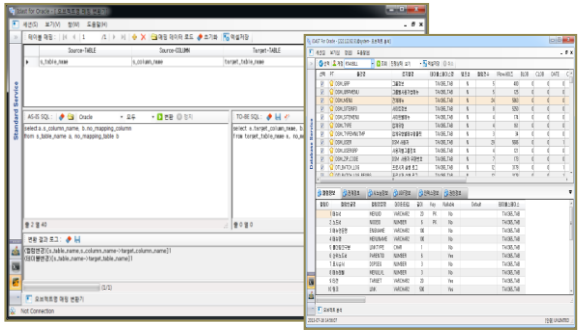
암호화 대상 추출?

Application과
암호화 대상과의
영향도 ?

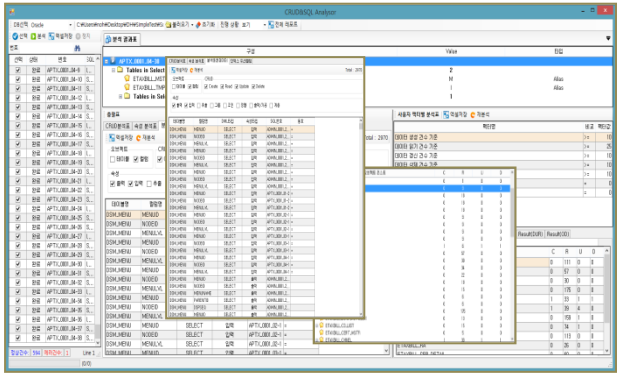
구축 시
초기 암호화 ?

모니터링 ?

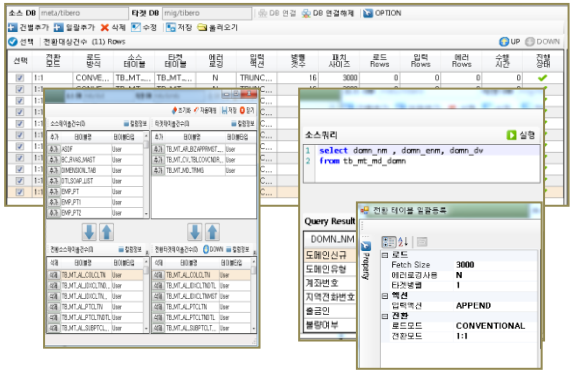
암호화 대상 추출 및 API 변환 툴



Application / SQL 영향도 분석



초기 암호화 수행 툴



성능 모니터링

금융권 고객정보암호화를 위한 제언

기존시스템의 변경 없이도
암호화 후 동일한 성능을 확보하고,
최적기간 내 암호화 수행을 위한 기술 필요

- Secure Filter를 활용한 쿼리 자동변환 기술
- 성능 이슈 해결을 위한 OPE 알고리즘 기술
- 기간내 수행을 위한 자동화 툴

Q&A

감사합니다