3BC와 F2mECC를 이용한 MDIT(Mobile Digital Investment Trust) 에이전트 설계 및 보안 강화

A MDIT(Mobile Digital Investment Trust) Agent design and security enhancement using 3BC and F2mECC

정 은 희* 이 병 관**
Eun-Hee Jeong Byung-Kwan Lee

요 약

본 논문은 모바일 전자 상거래 환경에서 투자 신탁을 위한 MDIT(Mobile Digital Investment Trust) 에이전트 설계를 제안하고, 또한 이 MDIT에서 모바일 전자 상거래 환경이 가지고 있는 기억 장치 용량, CPU 처리 속도 문제, 그리고 보안 강화 문제점을 해결하기 위해, 비밀키 암호 알고리즘인 3BC(Bit, Byte and Block Cypher)와 공개키 알고리즘인 F2mECC 설계를 제안한다. 특히, MDIT는 모바일 전자상거래에 투자 신탁의 개념을 도입한 은행 업무 보안 프로토콜로서 공개키 알고리즘인 ECC를 이용해 공유 비밀키를 생성한 후, 이 값을 공유 비밀키를 대칭키로 사용하는 블록 암호화 기법인 3BC에 사용하여, 보안 기능을 강화시키고, 처리속도를 감소시킨 MDIT 보안 에이전트를 구축하였다.

Abstract

This paper propose not only MDIT(Mobile Digital Investment Trust) agent design for Trust Investment under Mobile E-commerce environment, but also the symmetric key algorithm 3BC(Bit, Byte and Block Cypher) and the public encryption algorithm F2mECC for solving the problems of memory capacity, CPU processing time, and security that mobile environment has. In Particular, the MDIT Security Agent is the banking security project that introduces the concept of investment trust in mobile e-commerce. This mobile security protocol creates a shared secrete key using F2mECC and then it's value is used for 3BC that is block encryption technique. The security and the processing speed of MDIT agent are enhanced using 3BC and F2mECC.

☞ Keyword : SSL, DES, ECC, MDIT, EC-Sign, 3BC

1. 서 론

현재 국내 모바일 서비스 환경을 고려해 볼 때, 모바일 단말기(휴대폰, PDA)를 통한 금융 업무의 수행이 증가하고 있으며, 높은 모바일 폰 보급률과 인터넷 금융의 성공적인 정착, 그리고 국내 소비자들이 이동성과 즉시성 가치를 중시여기는 금융문화 특징 등으로 모바일 금융 서비스의 전망이 밝다고 할 수 있다. 하지만, 모바일을 통한 금융 업무 수행의 불편함 때문에 고객의 수용도가 낮으며 서비스의 확산에 선행되어야 할 기술적인 장애인 보안이 극복되지 못하고 있다[1,2].

본 논문에서 제안하는 MDIT 에이전트는 기술적인 장애인 보안을 해결하고, 소액지불뿐만 아니라, 계좌 생성 및 계좌 이체를 할 수 있으며, 투자 개념을 도입한 모바일 환경의 은행업무 프로젝트로서, 보안에는 공개키 보안을 강화한 F2mECC와 블록 암호화 기법인 3BC(Bit, Byte, Block)를 이용한다. MDIT 에이전트는 크게 세 가지 부분으로 나누어 볼 수 있는데, 첫째 디지털 투자 신탁 시스템 내에서 계좌 생성과 거래를 익명화 하였고, 둘째 고객이 인터넷을 통해 안전하게 접근

* 정 회 원 : 삼척대학교 경제통상학과 전임강사
jeh@samcheok.ac.kr(제 1저자)
** 종신회원 : 관동대학교 멀티미디어공학전공 교수
bklee@kwandong.ac.kr(공동저자)
[2004/10/01 투고 - 2004/10/18 심사 - 2004/12/13 심사완료]

하여 화폐를 전송하는 것을 허용하였으며, 셋째 고객이 예금한 자금으로 가장 적합한 곳을 검색하여 투자를 하여 이윤을 남기는 투자 신탁이다.

본 논문의 구성은 제 2장에서 관련연구에 대해 설명하고 제 3장에선 MDIT 에이전트 설계에 대해서 설명하였다. 그리고 제 4장에서 MDIT 에이전트 보안 설계에 대해서 설명하였고, 제 5장에서 시뮬레이션 결과에 대해 설명하였으며, 마지막으로 제 6장에서 결론을 맺는다.

2. 관련연구

2.1 F2mECC(Elliptic Curve Cryptography)

ECC는 1985년 N.Kobitz와 V.Miller에 의해 제안되었다[3]. RSA와 이산 대수처럼 일반적으로 사용되는 공개키 보안 시스템과 비교해 볼 때, ECC는 좀 더 작은 키 크기만으로도 다른 공개키 시스템과 동일한 수준의 안전도를 유지할 수 있는 장점으로 인해 IC 카드와 같은 메모리와 처리 능력이 제한된 하드웨어에도 이식이 가능하다. 또한, 동일한 유한체 연산을 사용하면서도 다른 타원곡선을 선택할 수 있어서 추가적인 보안이 가능하기 때문에 고수준의 안전도를 유지하기 위한 차세대 암호 알고리즘으로 각광 받고 있다[4-6]. 타원곡선 암호화 시스템에선 타원곡선 E의 원소인 P, Q가 주어졌을 때, 타원곡선의 암·복호화는 $Q = aP$ 일 때 a를 계산해 내는 것이 어려운 이산 대수 문제를 이용한다. 따라서, 타원곡선 암호화 시스템은 타원곡선 상에서 점 P를 a번 더하는 계산이 주를 이룬다.

본 논문에서 사용된 타원 곡선 E는 표수가 2인 $y^2 + xy = x^3 + ax^2 + b,\ a, b \in F_2^m$이다. 체(field) F_2^m의 원소들은 m비트 스트링으로 표현하며, F_2^m의 기술방법은 다항식 기저(polynomial basis) 혹은 최적 정규 기저(optimal normal basis)를 사용하는데, 본 논문에서는 다항식 기저 방식을 사용하였다. F_2^m상의 타원곡선 E는 위의 방정식을 만족하는 F_2^m상의 모든 점 (x, y)와 무한원점 0으로 구성되고, 아래와 같은 덧셈법칙을 만족한다[3-6].

(1) $0 + 0 = 0$.

(2) 모든 점 $P \in E$ 에 대해,
$0 + P = P + 0 = P$ 이다.

(3) $P = (x, y)$일 때, $-P = (x_1, x_1 + y_1)$이고
$P + (-P) = 0$이다.

(4) $P = (x_1, y_1), Q = (x_2, y_2)$라고 할 때,
$P + Q = (x_3, y_3)$라고 하면 다음과 같다.

① $P \neq Q$일 때, $L = (x_1 + x_2)^{-1} \cdot (y_1 + y_2)$
$P = Q$일 때, $L = x_1^{-1} \cdot y_1 + x_1$

② $x_3 = L^2 + L + (x_1 + x_2) + a$

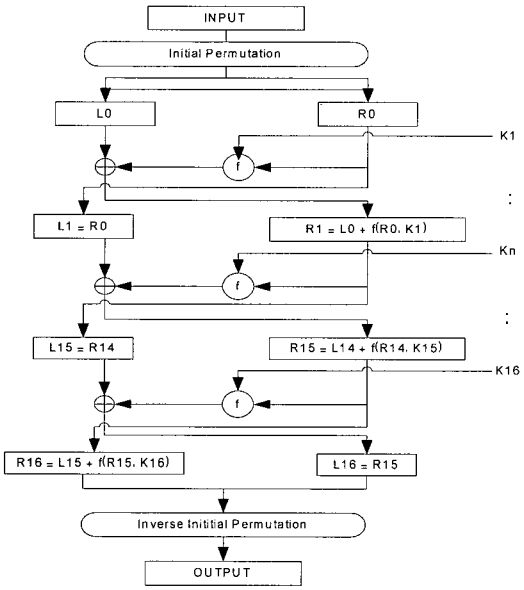
③ $P \neq Q$일 때, $y_3 = L \cdot (x_1 + x_3) + x_3 + y_1$
$P = Q$일 때, $y_3 = x_{+(L+1) \cdot x_3 1}^2$

2.2 DES

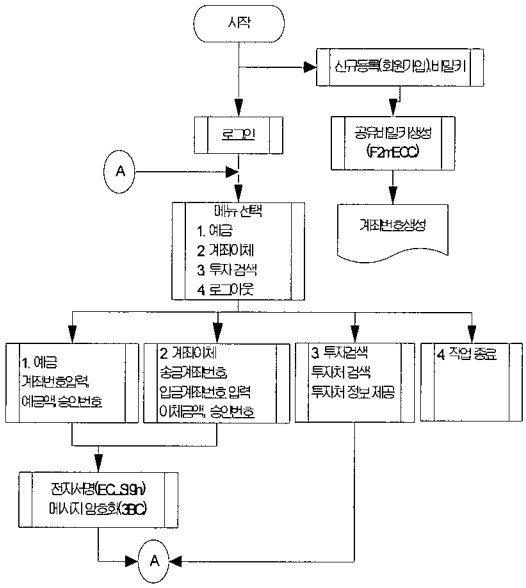
ANSI에 의해 DEA(Data Encryption Algorithm)와 ISO에 의해 DEA-1로 알려진 DES[7,8]는 세계적인 표준으로 사용된 64비트 블록 암호 알고리즘으로 평문 64비트 블록이 입력되어 16라운드를 거치면서 암호문 64비트 블록으로 출력되는데, 키의 길이 64비트 중에서 56비트만 사용되고 나머지 8비트는 패리티 체크에 사용된다.

DES의 구조는 크게 데이터 암호화부와 키 생성부로 구성되어 있으며, 먼저 키 생성부에서 생성된 48비트의 16개 라운드 키는 데이터 암호화부의 각 라운드로 들어가 평문 블록과 함께 치환, 대치 그리고 키 스케줄 등을 통하여 암호문을 만들어내게 되고, 복호화는 암호화의 역순이다.

그림 1은 DES의 기본 흐름도를 설명한 것으로, 64비트인 메시지 블록을 IP(Initial Permutation)를 이용해 치환한 후, 각각 왼쪽의 32비트를 L_{n-1}, 오른쪽 32비트를 R_{n-1}로 나눈다. 오른



〈그림 1〉 DES의 기본 흐름도



〈그림 2〉 MDIT 에이전트 설계도

쪽 R_{n-1}이 왼쪽의 다음 단계인 L_n이 되고, 오른쪽 R_{n-1}와 키 값 K_{n-1}을 f함수를 이용해 계산한 후, 그 결과와 왼쪽의 L_{n-1}을 Exclusive-OR 연산을 한 것이 다음 단계 R_n이 된다. 즉, DES는 아래의 식 (1), (2)를 이용해 16라운드를 반복하면, 메시지 암호화가 된다[9-11].

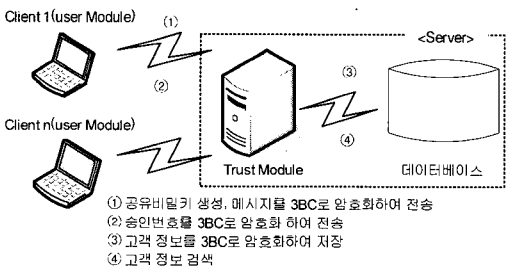
$$L_n = R_{n-1} \quad \cdots\cdots (1)$$

$$R_n = L_{n-1} \oplus f(R_{n-1}, K_{n-1}) \quad \cdots\cdots (2)$$

3. MDIT 에이전트 설계

MDIT 에이전트는 모바일 디지털 투자 신탁 시스템으로써, 시스템 내에서 계좌 생성 및 거래를 익명화함으로써 고객의 정보를 내부 직원으로부터 보호할 뿐 아니라, 보안문제를 해결하려고 한다. 특히, 모바일 디지털 투자 신탁 시스템의 고객으로 회원 가입할 시, 별도의 개인 정보는 필요하지 않으며, 사용자 모듈은 타원곡선의 공유 비밀키 방식을 이용하여 고객의 계좌번호를 생성한다. 생성된 고객의 계좌번호는 MDIT 서버와 사용자만이 알 수 있는 정보로써, 공유 비밀키가 노출되더라도 사용자의 개인키는 절대 풀 수 없는 이산대수 기법을 이용하고, 이 공개키는 본 논문에서 제안하는 대칭키 알고리즘인 3BC의 대칭키 생성에 필요한 원형키가 된다. 그림 2는 MDIT 에이전트의 설계도이다.

MDIT 에이전트는 크게 사용자 모듈, Trust 모듈, 모바일 디지털 투자 신탁 데이터베이스로 구성되며, MDIT 에이전트의 구성도를 대략 살펴보면 그림 3과 같다[12].



〈그림 3〉 MDIT 에이전트 구성도

3.1 사용자 모듈

사용자 모듈은 Trust 서버에 접근하려는 일반적인 웹 브라우저와 관련된 작업을 하며, 계좌번호를 생성하고 저장한다.

3.1.1 모바일 디지털 투자 신탁 계좌 생성

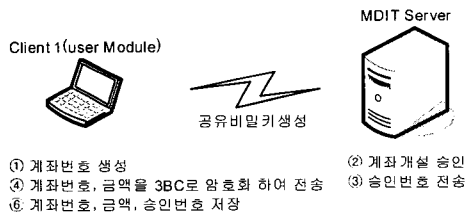
모바일 디지털 투자 신탁 계좌를 개설하는 것은 계좌에 현금을 입금시키는 것만큼 간단하다. 고객은 익명이므로 모바일 디지털 투자 신탁은 고객 정보가 필요하지 않으며, 계좌 번호는 고객의 사용자 모듈에 의해 생성되어, Trust 모듈에 의해 예금 기록과 함께 모바일 디지털 투자 신탁 데이터베이스에 저장된다.

그림 5는 사용자 모듈이 Trust 모듈에 계좌 생성을 요청하는 알고리즘이다. 계좌 번호는 사용자 모듈에 의해 랜덤하게 생성된 후, EC-sign으로 메시지 인증 코드를 생성한다. 생성된 계좌번호와 금액에 대한 정보는 3BC 알고리즘으로 암호화되어 MDIT 서버의 Trust 모듈에 전송된다. 사용자 모듈은 계좌 개설 승인 번호를 Trust 모듈에 의해 전송 받은 후, 계좌번호, 계좌정보, 승인번호를 3BC 알고리즘으로 암호화하여 저장한다.

3.1.2 paying 고객 알고리즘

모바일 디지털 투자 신탁 시스템의 고객들끼리 계좌 이체를 하는 경우로, 누구든지 paying 고객 혹은 receiving 고객이 될 수 있다.

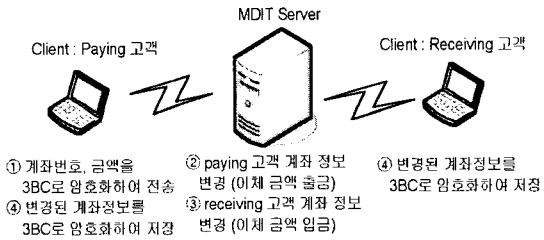
그림 7은 paying 고객에 대한 알고리즘으로 Trust 모듈에 계좌이체에 대한 정보를 전송하여 계좌이체를 한 후, Trust 모듈로부터 계좌 잔액에 대한 정보를 전송 받아 paying 고객의 컴퓨터에 계좌번호와 금액, 승인번호를 3BC 알고리즘으로 암호화하여 저장한다.



〈그림 4〉 신탁 계좌 생성 흐름도

```
User_module(x, C){
 x : user's account number
 C : e-money
   x = generate_number();
   user_pk = generate_private_key();
   sk=generate_shared_key(user_pk);
   sign = EC_sign(x);
   data = en_3bc(sk, sign, x, C);
   send_MDIT(data);
   data2=de_3bc(sk, receive(data1));
   if (check(ok_sign, no)){
     data2 = en_3bc(sk, x, C, no);
     save_user(data2);
   }
}
```

〈그림 5〉 User Module 알고리즘



〈그림 6〉 고객간 계좌이체 흐름도

```
pay_module(sk, x1, x2, T){
 sk : shared secret key
 x1 : payer's account number
 x2 : receiver's account number
 T : transfer e-money
    sign = EC_sign(x1);
    data = en_3bc(sk, sign, x1, x2, T, no);
    send_MDIT(data);
    data2=de_3bc(sk, receive(data1));
    if(check(ok_sign, no)){
       data2 = en_3bc(sk, x1, C, no);
       save_user(data2);
  }
}
```

〈그림 7〉 paying 고객 알고리즘

```
Trust_module(data){
 data : received cipher text
    trust_pk = generate_private_key();
    sk = generate_shared_key(trust_pk);
    data1 = de_3bc(sk, data) //
    복호화된 내용(x, sign, C)
    if(check(sign, x)){
       no = create_trust_number();
       ok_sign = EC_sign(no);
       data2 = en_3bc(sk, x, C, no, ok_sign);
       send_user(data2);
       data3 = en_3bc(sk, x, C, no);
       save_MDIT_db(data3);
    }
}
```

〈그림 8〉 Trust Module 알고리즘-계좌 승인 알고리즘

```
Trust_pay_module(data){
data : received data
 data1 = de_3bc(sk, data);
 if(search_ok(x1, x2)==TRUE){
   C = C - T;
   call_trust_rec_module(data);
   ok_sign = EC_sign(no);
   data2 = ec_3bc(sk, x1, C, no, ok_sign);
   send_user(data2);
   data3 = en_3bc(sk, x1, C, no);
   save_MDIT_db(data3);
 }
}
```

〈그림 9〉 Trust Module 알고리즘-paying 측면

3.2 Trust 모듈

Trust 모듈은 모바일 디지털 투자 신탁 시스템을 제어하기 위해 웹서버와 관련된 작업을 하며, Trust 번호를 생성하고 기록을 작성한다. 또한 익명 계좌에 대해 예금 혹은 인출을 관리하며, 모바일 디지털 투자 신탁 계좌간의 계좌 이체를 관리한다.

3.2.1 계좌 승인 알고리즘

그림 8은 고객이 요청한 계좌 개설을 승인하는 알고리즘으로 고객의 메시지 인증 코드를 확인한 후, 고객의 정보가 일치한다면, Trust 모듈은 Trust의 고유번호 즉, 계좌 승인 번호를 생성한다. 생성된 계좌 승인 번호를 EC-sign으로 메시지 인증코드를 생성한 후, 3BC 알고리즘으로 암호화하여 고객에게 전송한다.

3.2.2 paying/receiving 고객 계좌 관리 알고리즘

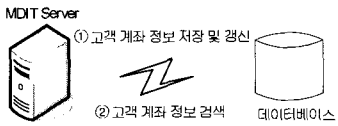
paying 고객일 경우 Trust 모듈에 paying 고객의 계좌 정보를 전송하여, paying 고객의 계좌에서 이체 금액을 뺀 남은 금액으로 모바일 디지털 투자 신탁 데이터베이스 정보를 갱신하고, receiving 고객일 경우 Trust 모듈이 receiving 고객의 정보를 전송 받아, receiving 고객의 계좌에 이체 금액을 더하여 모바일 디지털 투자 신탁 데이터베이스의 receiving 고객의 정보를 갱신한다.

그림 9는 paying 측면의 Trust 모듈 알고리즘으로써, paying 고객의 계좌에서 이체 금액 뺀 후, receiving 측면의 알고리즘을 호출하여 receiving 고객의 정보를 변경시키고, paying 고객의 변경된 정보를 3BC 알고리즘으로 암호화 하여 고객과 모바일 디지털 투자 신탁 데이터베이스에 각각 전송한다.

3.3 데이터베이스

F2mECC 알고리즘을 이용해 공유 비밀키를 생성한 후, 공유 비밀키를 이용하는 암호화 알고리즘인 3BC로 Trust 계좌 정보를 데이터베이스에 저장한다.

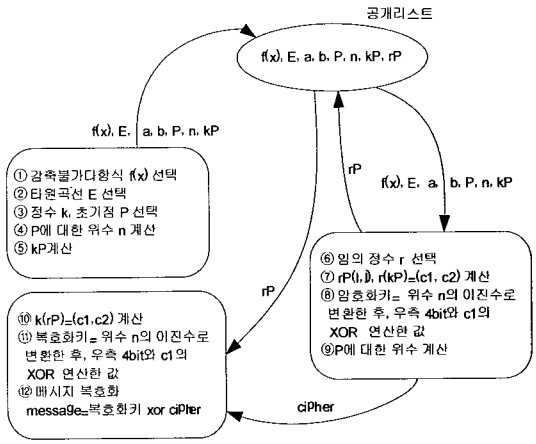
데이터베이스 정보의 흐름은 그림 10과 같다.



〈그림 10〉 데이터베이스 흐름도

```
MDIT_db(data){
data : received data
data1 = de_3bc(sk, data)
while(!EOF){
  if(search_ok(no)){
    data = en_3bc(sk, x, C, no);
    update_db(data);
  }
  else{
    data = en_3bc(sk, x, C, no);
    save_db(data);
  }
}
}
```

〈그림 11〉 모바일 디지털 투자 신탁 데이터베이스 알고리즘



〈그림 12〉 F2mECC 암·복호화 과정

그림 11은 계좌 개설이 승인된 정보를 모바일 디지털 투자 신탁 데이터베이스에 저장하는 알고리즘으로 고객 정보인 금액, 계좌번호, 승인번호에 대한 보안을 위해 3BC 암호화 기법을 사용하였다.

4. MDIT 에이전트 보안 설계

4.1 F2mECC 알고리즘 설계

본 논문에서는 공개키 알고리즘으로 F2mECC 암호 알고리즘을 이용해 공유 비밀키를 생성한 후, 공유 비밀키로 사용자 A, B가 메시지를 암호화하여 보내는 과정을 아래의 그림 12와 같다.

사용자 A는 우선 감축불가 다항식 f(x)와 타원곡선 E를 선택하고, 타원곡선 E의 벡터 a, b를 결정한다. 또한, 타원곡선 위의 한 점을 초기점 P로 선택하고 초기점 P에 대한 위수(order) n을 구한다. 비밀키 k만큼 덧셈 연산한 kP를 계산한 후, 공개 리스트에 타원곡선 E, 감축 불가 다항식 f(x), 벡터 a, b, 초기점 P, 위수 n, 공개키 kP를 등록한다. 사용자 B는 공개 리스트에 있는 감축불가 다항식 f(x)와 타원곡선 E를 이용해 사용자 B의 비밀키 r만큼 덧셈 연산한 사용자 B의 공개키 rP를 공개 리스트에 등록시키고, 사용자 A의 공개키 kP를 비밀키 r만큼 덧셈 연산한 공유 비밀키 r(kP)로 메시지를 암호화시킨 후 암호화된 메시지를 사용자 A에게 전송한다. 사용자 A는 공개리스트에서 사용자 B의 공개키인 rP를 자신의 비밀키인 k만큼 덧셈 연산한 공유 비밀키 k(rP)를 이용해 암호화된 메시지를 복호화 시킨다.

타원곡선 알고리즘의 암호화·복호화 과정을 단계별로 살펴보면 다음과 같다.

[단계 1] 사용자 A : m=4 인 경우, 감축불가 다항식 $f(x)=x^4+x+1$을 선택한 후, F_2^4의 생성자 g=0010이며, F_2^4의 모든 원소들의 벡터 값 16개를 계산한다.

[단계 2] 사용자 A : 타원곡선 상에서 $y^2+xy=x^3+ax^2+b,\ a, b\in F_2^m$인 수식을 선택하고 벡터 값 a, b를 선택한다. 사용자 A는 타원곡선 상의 초기점 P를 찾는다. 감축 불가 다항식이 $f(x)=x^4+x+1$이고, a=g, b=1인 타원곡선 $y^2+xy=x^3+gx^2+1$인 좌표를 계산한다.

```
ecc_key_generation(f(x), a, b, E, P)
{
 f(x) : irreducible polynomial
 a, b : vector value
 E : elliptic curve
 P : initial point
   input_private_key(k);
   for (i=k ; i>=1 ; i--){
     if (x1 == x2 && y1 == y2){
       Equal_addition(x1, y1);
       x2 = x3; y2 = y3;
     }
     else if (x1 != x2 && y1 != y2){
       Not_equal_addition(x1, y1, x2, y2);
       x2 = x3; y2 = y3;
     }
   }
   return(kP);
}
```

〈그림 13〉 공개키 생성 알고리즘

```
Ecc_encrypt(f(x), a, b, P, kP, n, E, message)
{
 f(x) : irreducible polynomial
 a, b : vector value
 E : elliptic curve
 P : initial point
 kP : user A's public key
 n : point P's order
 message : plaintext
   input_private_key(r);
   ecc_key_generation(f(x), a, b, E, P, kP);
   e_key = concatenate(bin(n, 4), x2);
   cipher = e_key ⊕ message;
   return(cipher)
}
```

〈그림 14〉 F2mECC를 이용한 암호화 단계 알고리즘

[단계 3] 사용자 A : 정수 k를 선택한 후, kP를 계산한다.

[단계 4] 사용자 A : $f(x)$, E, a, b, P, kP의 값을 공개 리스트에 등록시킨다.

F2mECC에서 사용하는 공개키 생성 알고리즘은 그림 13과 같다.

[단계 5] 사용자 B: 공개 리스트에 등록되어 있는 사용자 A의 공개키 kP를 이용해 $r(kP)=(c_1, c_2)$ 를 계산한다.

[단계 6] 사용자 B : 사용자 A에게 전송할 메시지 m을 $r(kP)=(c_1, c_2)$ 로 암호화하여 사용자 A에게 보낸다.

F2m ECC를 이용한 암호화 단계를 알고리즘으로 나타내면 그림 14와 같다.

[단계 7] 사용자 A : 공개 리스트에서 사용자 B의 공개키 rP를 이용해 $k(rP)=(c_1, c_2)$) 을 계산한 후, 사용자 B가 암호화하여 전송한 메시지 m을 복호화 한다.

F2mECC를 이용한 복호화 단계를 알고리즘으로 나타내면 그림 15와 같다.

```
Ecc_decrypt(f(x), a, b, rP, E, n, cipher)
{
 f(x) : irreducible polynomial
 a, b : vector value
 E : elliptic curve
 P : initial point
 kP : user A's public key
 n : point P's order
 cipher : cipher text
   input_private_key(k);
   ecc_key_generation(f(x), a, b, E, P, rP);
   d_key = concatenate(bin(n, 4), x2);
   message = cipher ⊕ d_key;
   return(message);
}
```

〈그림 15〉 F2mECC를 이용한 복호화 단계 알고리즘

4.2 3BC 설계

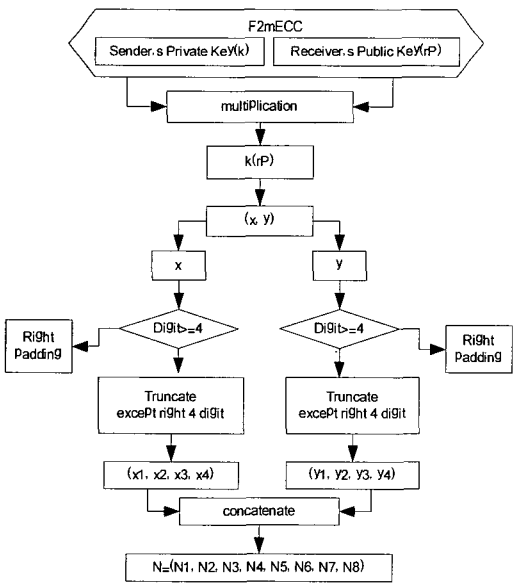
제안된 대칭키 알고리즘인 3BC는 블록 암·복호화 알고리즘으로 블록 단위의 바이트 교환과 비트 연산(xor)을 통해 암호화되는 대칭키 알고리즘이면서도 대칭키의 전송을 필요로 하지 않는다. 3BC 알고리즘은 바이트 교환키와 비트 연산키를 생성하는 대칭키 생성 단계와 암호처리를 위해 평문 데이터를 처리하는 과정인 블록 형성과 데이터 패딩 단계, 바이트 교환 후 비트 연산으로 이어지는 암호화 단계로 구분할 수 있다. 복호화는 블록 계산과 블록 생성 과정이 불필요하며 패딩 과정이

없이 암호화의 역순으로 처리된다. 본 논문에서는 데이터 길이 표시 부분을 제외한 실제 입력된 데이터만을 가지고 3BC의 암·복호화 처리 과정을 설명한다[13,14].

4.2.1 암호화

1) 대칭키 생성

데이터 암호화를 위해 F2mECC의 상수배 연산을 이용해 공유 비밀키를 계산한다. 공유 비밀키가 생성되면 키 값의 좌표 x, y값을 각각 4자리씩 자른다. 다시 말해서 좌표길이가 4자리보다 클 때에는 오른쪽에서부터 4자리를 선택해서, 나머지는 잘라버리고(truncate), 좌표길이가 4자리보다 작을 때에는 오른쪽에 0을 패딩 한다. 이 결과 x, y를 연접시키고(concatenate) 각각의 숫자를 mod 8로 연산한 결과를 원형 키(prototype key) N으로 한다. 예를 들어, 공유 비밀키의 계산 결과가 (456789, 567)일 경우 N은 67015670이 된다.



〈그림 16〉 대칭키 생성

(1) 바이트 교환키의 생성

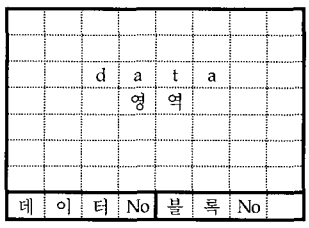
바이트 교환 키(byte-exchange key)는 데이터 블록 A와 블록 B의 위치를 지정하는 행열로 구성되며 원형키 N을 기반으로 키를 생성한다.

바이트 교환키는 바이트 교환 암호화를 위한 2개 단위 블록의 바이트 교환을 처리한 횟수인 n과 계산되어 생성되며 64개의 행열 값으로 나타나고 이 키에 따라 블록 A, B간에 바이트를 교환한다. 블록 A의 위치와 B의 교환 위치는 블록 A, B간에 동일 위치가 될 수도 있고 반복 교환이 발생할 수도 있다. 또한, 블록이 호출되어 처리되는 횟수에 따라 행열 값이 변화하므로 교환되는 위치가 변화된다. N이 67015670일 경우 생성되는 1회 바이트 교환키는 표1과 같고, 2회 바이트 교환키는 표2와 같다.

(2) 비트 연산(bit-wise xor)키 생성

비트 연산키는 원형키 N을 아스키코드로 변환시킨 값과 변환된 값을 역전시킨 값을 or 연산을 하여 기본키를 만들고 비트 연산(xor) 수행 횟수에 따라 기본키를 1비트씩 오른쪽 shift를 하여 키의 모양을 변환시켜 사용한다.

예를 들어, N=97015670일 경우 기본키는 00111110 11111111 01111100 10111101 10111101 00111110 11111111 01111100이 된다. 이 기본키를 암호화 과정에서 비트 연산 처리 횟수에 따라 1비트씩 오른쪽 쉬프트 시켜 비트 연산(xor)에 사용한다.



〈그림 17〉 블록 구조

2) 블록 형성과 데이터 패딩

데이터 암·복호화는 8 바이트 × 8 바이트의 단위 블록으로 수행되며, 한 개 블록의 크기는 64 바이트로 그림 17과 같이 데이터 영역 56 바이트와 데이터 블록 순서 번호 4 바이트, 블록 번호

4 바이트로 구성된다. 데이터 블록 순서 번호는 입력되는 원 메시지를 56 바이트 단위로 나누어 순차적으로 붙여지는 번호이며, 블록 번호는 블록 간에 바이트 교환을 하기 위해 무작위로 붙여지는 번호이나 두 개 블록만 동일한 번호를 갖는다.

평문 데이터가 입력되면 생성시킬 블록의 개수를 계산하고 필요한 경우 블록을 생성한다. 블록의 개수 계산식은 다음과 같다.

① data_block = Input_data_length/56
② Input_data_length % 56 = 0인 경우
- data_block % 2 = 0 일때,
  Block = data_block
- data_block % 2 ≠ 0 일때,
  Block = data_block + 1

③ Input_data_length % 56 ≠ 0인 경우
- data_block % 2 = 0 일때,
  Block = data_block + 2
- data_block % 2 ≠ 0 일때,
  Block = data_block + 1

블록의 계산이 끝나면 padding_data = Block * 56 - Input_data_length 식을 이용하여 패딩될 바이트 수를 계산하여 필요한 경우에 임의 문자로 패딩을 하여 2의 배수가 되는 블록을 생성하게 된다. 입력 데이터가 "This project propose to design an ISEP(Improved Secure Electronic Payment protocol), which consist of the ECC(Elliptic Curve Cryptosystem)."라고 할 때 처리 과정은 다음과 같다. 입력 테이터는 블랭크를 포함해 139개 문자가 되고, 이는 그림 18과 같이 3개의 블록이 만들어지지만, 암호화를 위해서는 2의 배수로 블록이 만들어져야 하기 때문에 암호를 위한 블록은 4개이다.

따라서 한 개의 블록을 더 생성하여 전체 블록을 살펴보고 블록 내 비어있는 모든 데이터 영역은 그림 19와 같이 임의 문자로 무작위로 패딩된다.

블록과 데이터 패딩이 끝난 후 각각의 블록에 데이터 블록 번호를 데이터 순서대로 부여시키고 블록 번호를 임의수로 부착을 한다. 단, 임의로 발생되는 블록의 번호는 동일한 번호가 2회까지만

〈표 1〉 n=1일 때의 바이트 교환 위치(행,열)

processing order	block A		exchange	block B	
	A_Matbj			B_Matdj	
	row	col		row	col
1	6	7	↔	2	2
2	7	1	↔	5	2
≀	≀	≀	≀	≀	≀
33	2	3	↔	6	6
34	3	5	↔	1	6
≀	≀	≀	≀	≀	≀
63	0	5	↔	6	6
64	1	0	↔	1	6

〈표 2〉 n=2일 때의 바이트 교환 위치(행,열)

processing order	block A		exchange	block B	
	A_Matbj			B_Matdj	
	row	col		row	col
1	4	4	↔	4	4
2	6	7	↔	2	3
≀	≀	≀	≀	≀	≀
33	0	0	↔	0	0
34	2	7	↔	6	7
≀	≀	≀	≀	≀	≀
63	7	3	↔	3	7
64	1	6	↔	1	6

T	h	i	s		p	r	o
j	e	c	t		p	r	o
p	o	s	e		t	o	
d	e	s	i	g	n		a
n		I	S	E	P	(	I
m	p	r	o	v	e	d	
S	e	c	u	r	e		E

l	e	c	t	r	o	n	i
	P	a	y	m	e	n	t
	p	r	o	t	o	c	o
l	)	,		w	h	i	c
h		c	o	n	s	i	t
	o	f		t	h	e	
E	C	C	(	E	l	l	i

p	t	i	c		C	u	r
v	e		C	r	y	p	t
o	s	y	s	t	e	m	)
.							

〈그림 18〉 데이터의 블록 형성

T	h	i	s		p	r	o
j	e	c	t		p	r	o
p	o	s	e		t	o	
d	e	s	i	g	n		a
n		I	S	E	P	(	I
m	p	r	o	v	e	d	
S	e	c	u	r	e		E
0	0	0	1	0	0	0	1

l	e	c	t	r	o	n	i
	P	a	y	m	e	n	t
	p	r	o	t	o	c	o
l	)	,		w	h	i	c
h		c	o	n	s	i	t
	o	f		t	h	e	
E	C	C	(	E	l	l	i
0	0	0	2	0	0	0	2

p	t	i	c		C	u	r
v	e		C	r	y	p	t
o	s	y	s	t	e	m	)
.	g	h	j		k	s	s
f	w	p	b	p	/	,	.
/	D	f	h	g	j	r	f
d	f	g	'	[		t	'
0	0	0	3	0	0	0	1

1	2	r	e	g	}	f	m
f	t	g		r	<	;	h
f	g	k	o	l	k	,	l
]	@	4	$		f	k	z
z	n	u	+	\	!	G	B
D	R	H	l		.	p	d
e	f	b	[	s	l	v	e
0	0	0	4	0	0	0	2

〈그림 19〉 데이터 번호와 블록 번호의 부착

발생되어야 하며, 블록 번호가 같은 블록은 단 2개만 존재하게 된다. 발생되는 블록 번호의 상한은 (전체 블록의 수/2)가 된다.

위의 그림 19는 그림 18의 블록에 데이터 순서 번호와 블록 번호를 부착한 완성된 블록의 상태를 보여준다.

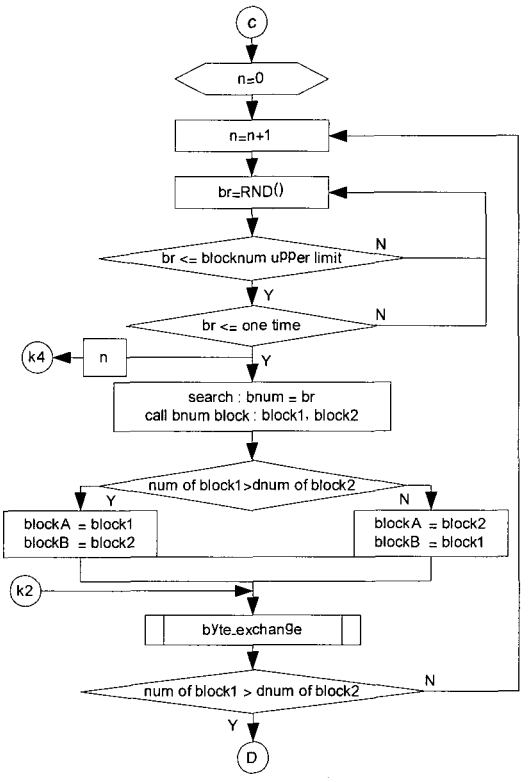
3) 데이터 암호화

데이터 암호화는 (1)에서 생성시킨 바이트 교환키를 사용하여 그림 20과 같이 먼저 바이트 교환 후 이어서 (2)에서 생성시킨 비트 연산(xor)키를 사용하여 그림 21과 같이 비트 연산을 하게 된다.

바이트 교환은 그림 22와 같이 임의 난수를 발생시켜 블록 번호가 동일한 블록 두 개를 호출한 후 데이터 블록 번호가 큰 것을 블록 A로, 데이터 블록 번호가 작은 것을 블록 B로 하여 바이트 교환키의 지정 행열 위치에 따라 값을 교환한다.

바이트 교환이 반복 실행됨에 따라 바이트 교환 처리 횟수인 n의 값이 증가하여 바이트 교환키의 값이 바뀌게 되고, 바이트 교환키가 가리키는 블록 내 바이트의 위치인 행열의 좌표도 바뀌게 되어 바이트 교환 처리마다 바이트 교환이 되는 위치가 일정치 않다. 바이트 교환 암호화가 끝나면 비트 연산을 하게 된다.

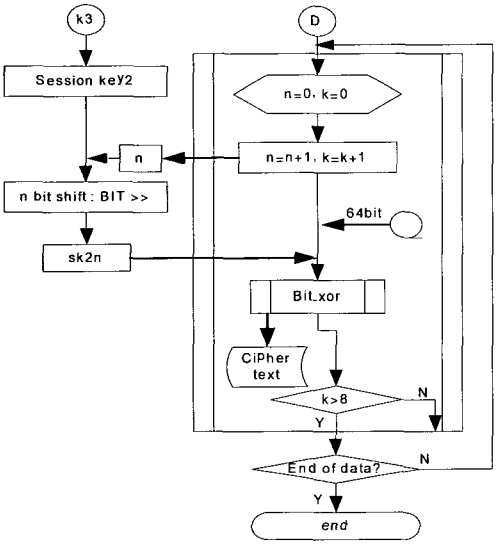
비트 연산은 그림 21과 같이 비트 연산키와 데이터를 64 비트 단위로 절단하여 xor 연산을 한다. 비트 연산도 반복 실행됨에 따라 실행 횟수인 n의 값이 증가하여 비트 교환키의 값이 바뀌게 된다. 비트 연산까지 끝나면 암호화된 순서대로



〈그림 20〉 바이트 교환 암호화

전송한다.

그림 19를 기준으로 난수 2가 발생되었을 경우의 암호화 과정은 다음과 같다. 블록 번호가 0002인 블록 두 개가 호출되고, 호출된 두 개의 블록은 데이터 번호가 0002인 것과 0004인 두 개이다. 그림 22와 같이 데이터 번호 0004인 블록이 블록 A, 0002인 것이 블록 B가 된다.



〈그림 21〉 비트 연산(Xor) 암호화

그림 22의 오른쪽 두 개 블록에 표시된 1~64의 숫자는 바이트 교환 처리 순서를 나타내고 있으며 블록 A와 B의 동일 숫자가 표시된 위치의 바이트를 교환하게 된다.

표 1의 키에 따라 바이트 교환된 결과는 그림 23에 나타나고, 바이트 교환이 끝난 후 비트 연산을 한다. 블록 A에 대한 xor비트 연산 결과는 표 3과 같다.

4) 암호문 완성

암호화가 끝나면 아래와 같은 문자의 열을 획득하게 되고, 이 암호문을 2진 비트열로 전송한다.

[iscgootftP4,0(knc2\c@ Donl0zeiEobei)

Block A Block B

[	i	s	c	g	o	o	t
f	t	P		4	,		0
(		k	n		c	2	\
c	@	D		o	n	l	0
z	e	i	E	o	b	e	i
)	0	2	w	0		h	d
o	n	o	C	o	E	v	h
e	f	0	i	2	0	y	m

l	G	e	t	l	e	f	,
.	g	a	0	2	0	f	t
g	p	r	f	m	l	c	s
l		〈	;	k	h	4	)
}	l	k	!	n	r	B	t
$				t	p	e	r
+	1	C	f	l	l	r	u
h	0	z	0	0	0	R	H

〈그림 23〉 바이트 교환 후의 모양

02w0hdonoCoEvhef0i20ymlGetlef,.ga020ftgp rfmlcsl<;kh4]}lk!nrBt$tper+1Cfllruh0z000R H·····

4.2.2 복호화

복호화는 그림 24와 같이 수신측에서도 ECC의 상수배 연산을 통해 공유 비밀키를 계산하여 원형키를 생성시키고, 비트 연산키 기본키와 바이트 교환키를 만든 후, 비트연산 처리 횟수에 따른 비트 연산키와 수신된 데이터를 64 비트 단위로 절단하여 xor연산한다. 비트 연산이 끝나면 64 바이트 단위로 절단하여 순서대로 두 개 블록씩 바이트 교환키를 이용하여 블록 간 바이트 교환을 한다. 바이트 교환키를 적용할 때는 생성된 바이트 교환키의 역순으로 적용하여 바이트 교환을 하여야 한다. 이 과정이 완료되면 블록 내에 데이터 번호가 원래의 위치에 나타나게 되고 이후 데이터 번호 순서대로 블록의 데이터 영역인 56 바이트만을 정렬시키면 평문이 나타나게 된다.

Block A

1	2	r	e	g	}	f	m
f	t	g		r	<	;	h
f	g	k	o	l	k	,	l
]	@	4	$		f	k	z
z	n	u	+	\	!	G	B
D	R	H	l		.	p	d
e	f	b	[	s	l	v	e
0	0	0	4	0	0	0	2

Block B

l	e	c	t	r	o	n	i
	P	a	y	m	e	n	t
	p	r	o	t	o	c	o
l	)	,		w	h	i	c
h		c	o	n	s	i	t
	o	f		t	h	e	
E	C	C	(	E	l	l	i
0	0	0	2	0	0	0	2

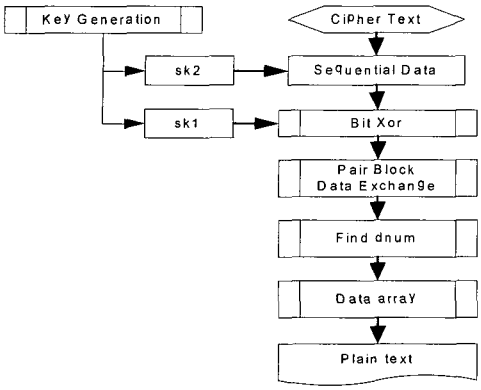
Block A

58	8	54	3		49,63	12	45
64	46	59		55,41	4	37	50
38	51		33,47	60	29	42	56
43		39,25	52	21	34	48	30
	17,31	44	13	26	40	22	35
9,23	36	5	18	32	14	27	
28	61	10	24	6	19		1,15
53	2	16	62	11		7	20

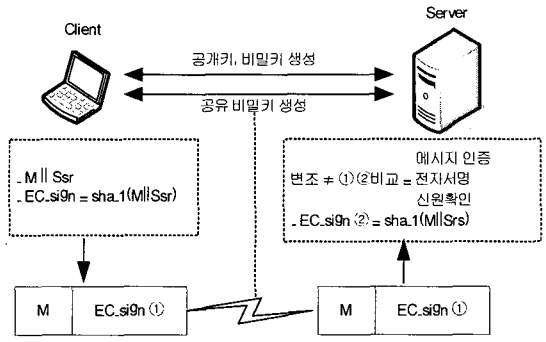
Block B

	22	3		17	28	61	8,4
14	59		9,39	20	53	34,6	
51		1,31	12	45	26,5		6
	23,5	4	37	18,4		62	43
15,4	60	29	10,4		54	35	
52	21	2,32		46	27		7,41
13	24,5		38	19		33,6	42
16,5		30	11		25,5	36	5

〈그림 22〉 블록간 교환 처리 순서 및 교환 위치



〈그림 24〉 복호화 과정



〈그림 25〉 EC-Sign 검증 개략도

4.3 전자 봉투의 생략

ECC의 전자 서명을 통해 메시지 무결성과 인증은 가능하고, 공개된 상수배 값을 선택하여 필요에 따라 공유 비밀키를 생성하여 3BC 알고리즘에 사용함으로 송·수신자간에 대칭키를 전송할 필요성이 없다. 그러므로 대칭키 전송에 사용되는 전자 봉투를 생략할 수 있다. 이러한 생략은 대칭키 해킹을 방지하여 암호 데이터의 보안성을 증대시킬 수 있고, 전송 데이터 량과 전송시간을 감소시킨다. 또한, 전자 봉투를 생성하는 암호화 과정과 대칭키를 복구하기 위한 복호화 과정이 생략되어 처리 시간이 단축된다.

4.4 EC-Sign를 이용한 메시지 인증 알고리즘

본 논문에서 제안된 EC-Sign은 F2m상의 타원곡선 알고리즘으로 공유 비밀키를 생성하는 Keyed HMAC로서 송신자와 수신자만이 알고 있는 공유 비밀키를 사용해 메시지 인증 코드를 생성한다. 따라서 해쉬 함수가 단지 무결성 만을 확인하는데 비해 EC-Sign 알고리즘은 공유 비밀키를 모르면 메시지를 인증할 수 없으므로 메시지의 인증은 물론 상대방의 신원 확인 및 전자서명에도 사용할 수 있다. 송신자는 본인의 공유 비밀키와 메시지를 결합하여 메시지 인증 코드를 생성하여 상대방에게 전송한다. 수신자는 전송된 메시지에 송신자와 같은 인증 알고리즘을 이용해 메시지 인증 코드(MAC*)를 생성한 후 전송된 MAC와 비교한다. 만약 두 값이 같다면 전송 중 메시지에 변조가 없다는 것을 검증할 수 있다.

EC-Sign를 이용한 메시지 인증 코드 생성과정은 다음과 같다.

[단계 1] F2m상의 타원곡선으로 생성된 공유 비밀키 Sx, Sy를 각각 바이트 변환, 정수 변환하여 그 값을 연결(concatenation)하여 S_K변수에 저장한다.

[단계 2] 공유 비밀키 S_K의 길이를 검사한다.

[단계 3] 키의 길이가 64바이트이면, 공유 비밀키를 S_K_0에 기억시키고, 단계 5로 이동한다.

[단계 4] 공유 비밀키 S_K의 길이가 64바이트보다 작으면, 공유 비밀키의 길이가 64바이트가 되도록 S_K의 끝에 0을 채운 후에 S_K_0에 기억시키고 단계 5로 이동한다.

[단계 5] 공유 비밀키 S_K의 길이가 64바이트보다 큰 경우에는 공유 비밀키 S_K를 Sha-1 함수로 다이제스트 한다. 공유 비밀키 S_K를 다이제스트 하면, 항상 20바이트가 출력되므로 나머지 44바이트는 0으로 채운 후에 S_K_0에 기억시킨다.

[단계 6] S_K_0와 ipad 값인 0x36을 Exclusive -OR 연산을 한다. 즉, S_K_0 ⊕ 0x36을 한다.

[단계 7] 다섯째의 결과 끝에 메시지를 연결시킨 후, SHA-1 함수로 다시 한번 다이제스트를 한다. 즉, Sha-1((S_K_0 ⊕ ipad) || Message)를 한다.

[단계 8] S_K_0와 opad 값인 0x5C와 Exclusive -OR 연산을 한다. 즉, S_K_0 ⊕ 0x5C 을 한다. 끝으로, 단계 7의 결과와 단계 8의 결과를 연결한 후 Sha-1함수로 다이제스트 한 결과 20바이트 값이 된다. 즉, EC-Sign = Sha-1((S_K_0 ⊕ opad) || Sha-1(S_K_0 ⊕ ipad || Message)) 이다.

5. 성능 분석

성능분석은 MDIT 에이전트 보안설계에 중점을 두었으며, MDIT 에이전트 보안설계의 성능 평가를 위한 구현환경은 다음과 같다.

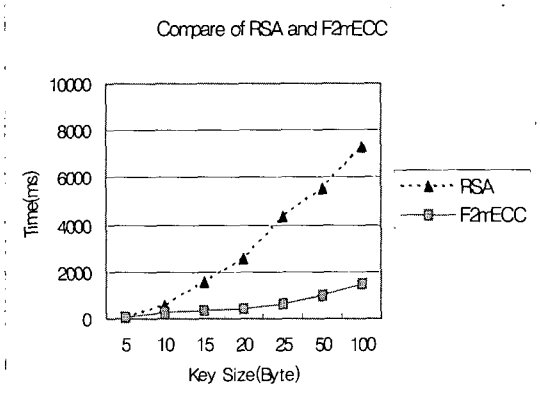
- 운영체제 : Windows XP professional
- 하드웨어 : Intel Pentium IV, 256MB
- 개발도구 : Visual C++, BREW

성능평가는 RSA와 F2mECC에 대해서는 키의 크기에 따라 20회 반복하여 암·복호화 하는데 걸리는 시간을 비교하였고, DES와 3BC에 대하여는 1블록의 크기를 128 바이트로 하여 블록 수에 따라 20회 반복하여 암·복호화를 처리하는데 걸리는 시간을 비교하였다.

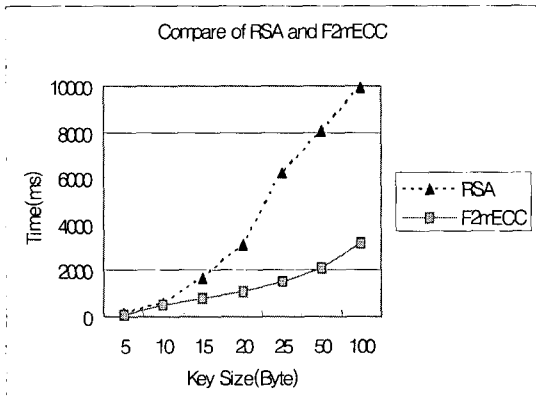
5.1 RSA와 F2mECC

RSA 알고리즘은 개인키와 공개키를 이용하여 암·복호화를 하는데 보안성을 위해 키 길이가 커지고 있다. 이렇게 키의 길이가 커지게 되면 메모리 사용량이 증가될 뿐만 아니라 송수신 양의 증가로 처리 속도가 떨어진다. 반면에 F2mECC는 키의 길이가 크지 않으면서도 키의 보안 능력은 RSA보다 우수하다.

본 논문에서 제안된 3BC 대칭키 알고리즘에 사용되고 전송할 필요성이 없는 대칭키를 생성하기 위해 F2mECC를 사용하였다. 그림 26은 RSA와 F2mECC의 암호화 시간을 비교한 것으로 F2mECC의 암호화 시간이 RSA보다 빠르게 나타났으며 키 크기에 따라 암호화 시간 격차는 증가되는 것으로 나타났다.



〈그림 26〉 RSA와 F2mECC의 암호화 시간 비교



〈그림 27〉 RSA와 F2mECC의 복호화 시간 비교

그림 27은 RSA와 ECC의 키 크기에 따른 복호화 시간의 비교를 그래프로 나타낸 것이다. ECC의 복호화 시간이 RSA보다 빠르게 나타났으며 키 길이 증가에 따라 복호화 시간 격차는 크게 나타났다.

5.2 3BC와 DES

DES는 데이터 암·복화를 위해 정형화된 IP (Initial Permutation) 테이블과 PE(Permutation Extend) 테이블, s-박스, IP^{-1} 테이블을 사용하고 있으며 1블록의 데이터를 16회에 걸쳐서 암·복호화를 하고 있다. 이러한 DES에 비해 본 논문에서 제안된 3BC 알고리즘은 64바이트 2개 블록에 대한 바이트 교환과 64비트 블록에 대한 비트 연산의 2단계 암호화로 처리되고 있어 속도면에서 DES보다 빠르다.

그림 28은 3BC와 DES의 메시지 암호화 시간을 블록 수에 따라 20회 반복하여 비교한 평균값을 그래프로 표현한 것이다. 3BC의 암호화 시간이 DES의 암호화 시간보다 빠르다는 것을 알 수 있다.

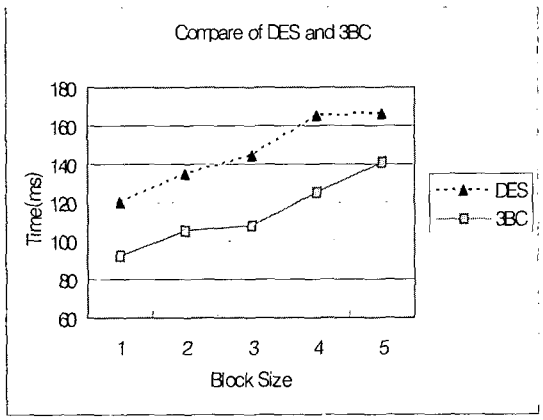
그림 29는 1 블록의 크기를 128 바이트로 했을 때 각 블록 수마다 20회 반복 비교된 3BC와 DES의 암호화 평균 시간을 나타내고 있는 것으로 3BC의 복호화 시간은 기존의 DES의 복호화 시간보다 빠르다는 것을 알 수 있다. 결과적으로 3BC를 사용함으로서 기존의 DES에 비해 암호화·복호화 시간을 줄일 수 있다.

6. 결 론

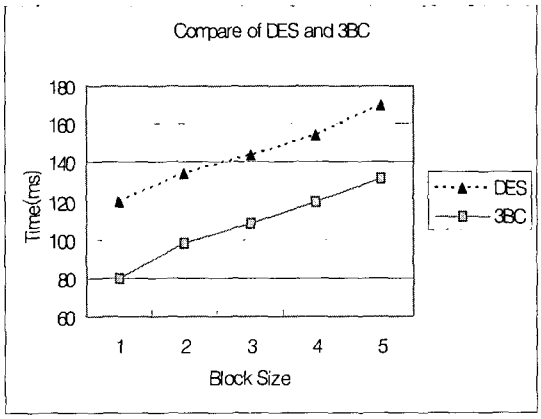
본 논문에서 제안한 MDIT 에이전트에선 현재 프로토콜로 널리 이용되는 기존의 SSL(Secure Socket Layer)의 보안 기법에 공개키 보안을 강화한 F2m ECC와 제안된 블록 암호화 기법인 3BC를 사용하였다. 특히, 사용자 모듈과 Trust 모듈 사이에서 데이터를 전송할 때, 데이터베이스에 계좌정보를 저장할 때, 제안된 블록 암호화 기법인 3BC 암호화 방식을 사용하였으며, 블록 암호화 기법인 3BC의 대칭키는 F2mECC 알고리즘을 이용해 공유 비밀키를 생성한 후, 공유 비밀키를 이용해 바이트 교환키와 비트 연산키를 생성하여 사용하였다.

MDIT 에이전트에선 MDIT 서버 데이터베이스에 데이터를 3BC로 암호화하여 저장함으로써 제 3자로부터 고객 정보를 보호할 수 있다. 또한, MDIT 에이전트는 계좌 생성 및 계좌 이체를 익명으로 처리하므로, 고객의 정보와 자산을 보호할 수 있는 차세대 금융 관리 시스템이라고 볼 수 있다.

향후 3BC에서 사용되는 대칭키로 F2mECC의



〈그림 28〉 3BC와 DES의 암호화 시간 비교



〈그림 29〉 3BC와 DES의 복호화 시간 비교

키 길이 160 비트에 대응하는 확장된 대칭키를 사용하고 블록의 크기를 확대한 후 바이트 교환과 비트 연산 암호화를 하게 되면 보다 강화된 암호화 강도를 얻을 것으로 기대된다.

참 고 문 헌

[1] 이은철, "뉴밀레니엄 시대의 전자화폐(상)", 지식재산21, 통권 제59호, 2000년 3월.

[2] 이은철, "뉴밀레니엄 시대의 전자화폐(하)", 지식재산21, 통권 제60호, 2000년 5월.

[3] N.Koblitz, "Elliptic Curve Cryptosystems", Mathematics of Computation, 48, pp.203-209, 1987.

[4] V.S. Miller, "Use of elliptic curve in cryptography", Advances in Cryptology-Proceedings of Crypto'85, Lecture Notes in Computer Science, 218, pp.417-426, Springer-Verlag, 1986.

[5] ANSI X9.63, "Public key cryptography for the financial services industry : Key agreement and key transport using elliptic curve cryptography", Key Management Workshop, Feb. 2000.

[6] C. P. Schnorr, "Efficient Signature Generation for Smart Cards", Advances in Cryptology CRYPTO'89 Proceeding, Springer- Verlag, pp.239-252, 1990.

[7] ANSI X3.92, "American National Standard Data Encryption Algorithm", American National Standards Institute, Approved Dec. 30. 1980.

[8] ANSI X.3.106, "American National Standard for Information Systems - Data Encryption Algorithm - Modes of Operation", American National Standards Institute, Approved May. 16. 1983.

[9] Federal Information Processing Standards Publication(FIPS PUB) 46, "Data Encryption Standard", Jan. 15. 1977.

[10] Federal Information Processing Standards Publication(FIPS PUB) 46-1, "DES Modes of Operation", Reaffirmed Jan. 22. 1978.

[11] Federal Information Processing Standards Publication(FIPS PUB) 81, "DES Modes of Operation", Dec. 2. 1980

[12] 정은희, 이병관 "DIT 시스템 설계를 위한 계좌관리" 정보처리학회 춘계학술발표논문집, 제9권 1호, pp951-954, 2002.

[13] In-seock, Cho and Byung-Kwan, Lee, "ASEP (Advanced Secure Electronic Payment) Protocol Design", ICIS, 2nd International Conferance on Computer and Information Science, pp.366-372, Aug. 8. 2002.

[14] 이병관, 조인석, 정은희, 양승해, "ThreeB 대칭키 알고리즘을 이용한 SET의 성능 개선", Telecommunication Review, 제13권 제6호, pp1009~1026, 2003.

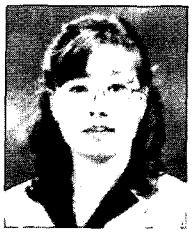
[15] 한국정보통신기술협회, "타원곡선을 이용한 인증서 기반 전자서명 알고리즘",pp.8~11, pp.14, 2001.

[16] 정은희, "타원곡선 보안 소켓층 프로토콜 설계 및 평가", pp.47~49, 관동대학교, 2003.

[17] SSL 3.0 Specification, http://www.netscape.com /eng/ssl3/draft302.txt

[18] SSL 3.0 Implementation Assistance, http: //www. netscape.com/eng/ssl3/traces/index.html

◐ 저 자 소 개 ◑



정 은 희(Eun Hee Jeong)
1991년 2월 강릉대학교 통계학과 이학사
1998년 2월 관동대학교 전자계산공학과 공학석사
2003년 2월 관동대학교 전자계산공학과 공학박사
2003년 9월~현재 삼척대학교 경제통상학과 전임강사
관심분야 : 네트워크 보안, 전자상거래, 웹 프로그래밍
E-mail : jeh@samcheok.ac.kr

이 병 관 (Byung Kwan Lee)
1975년 2월 부산대학교 기계설계학과 학사
1986년 2월 중앙대학교 전자계산공학과 석사
1990년 2월 중앙대학교 전자계산공학과 박사
1988년 3월~현재 관동대학교 멀티미디어공학전공 교수
관심분야 : 네트워크 보안, 전자상거래, 컴퓨터 네트워크
E-mail : bklee@kwandong.ac.kr