제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

스마트 카드에서의 Multiplicative Inverse 연산을 위한 효율적인 하드웨어의 구현

엄준형*, 이상우**, 박영수**, 전성익**,
*한국과학기술원 전산학과
**한국전자통신연구원
e-mail : jhum@vlsisyn.kaist.ac.kr

Implementation of Efficient Inverse Multiplier for Smart Card

Junhyung Um*, Sangwoo Lee**, Youngsoo Park**, Sungik Jeon**,
*Dept. of Computer Science, Korea Advance Institute of Science and Technology
**Electronics and Telecommunications Research Institute

요 약

여러 내장형 시스템에 탑재되는 암호모듈의 구현에 있어, 공개키 알고리즘을 위한 ECC 연산의 지연시간을 단축시키기 위해 유한체 연산은 하드웨어로 구현되는 경우가 많다. 그 중에서도 역원 연산은 지연시간 및 전력 소모, 또한 회로 면적에 있어 가장 주요한 부분을 차지하기 때문에 보다 효율적으로 구현하는 것이 필요하다. 본 논문에서 우리는 효율적인 역원 연산, 즉 작은 회로의 역원기를 위한 하드웨어의 구조를 제안한다. 실험에서, 우리가 구현한 구조는 기존에 주로 쓰이는 Modified Inverse Algorithm 의 구현에 비해 비슷한 지연시간을 가지면서 회로 면적에 있어 큰 감소를 보이며 이는 스마트 카드 뿐 아니라 여러 mobile 내장형 시스템에 광범위하게 쓰일 수 있다.

1. 서론

통신 및 반도체의 발전, 그리고 인터넷의 발전에 따라 정보 보호는 현재에 있어 상당히 중요한 문제로 대두되었다. 이러한 정보 보호를 위해 여러 가지 알고리즘들이 하드웨어로 개발 되어 사용되고 있으며, 이 적용 분야는 내장 시스템에까지 확장되고 있다. 대표적 공개키 알고리즘인 ECC(Elliptic Curve Cryptography)는 긴 지연시간을 요구하기 때문에 소프트웨어 보다는 하드웨어로 이를 구현하는 추세이다. 더욱이, portable 기기에 탑재되는 암호 모듈의 요구가 절실히 요구되는 상황에서, 내장형 시스템에 ECC 를 구현할 경우 가장 문제가 되는 것은 무엇보다 회로를 작게 만드는 것이다. 실제로 우리는 스마트 카드에 탑재를 목표로 연구를 진행해 나갔으며, 칩의 사이즈가 상당히 작기 때문에 암호 모듈의 크기를 크게 하며 지연시간을 빠르게 하는 것 보다 암호 모듈의 크기를 작게 하며 그 상위단계 부분을 좀더 빠르게 설계하는 것이 효율적이었다. 그래서 본 연구에서 우리는 최종 목표를 회로 면적을 줄이는 것으로 잡았다. 실제로 실험에서 우리의 결과는 기존의 것에 비해 비슷한 지연시간 또한 갖는다.

ECC 의 모든 연산은 유한체 $GF(2^n)$ 위에서 이루어진다. 이때 n 은 0 이 아닌 양수이고, GF(2)를 기반으로 하여 2^n 개의 원소를 갖는다. 또한 이 유한체 위의 모든 원소는 차수가 n 이하인 연산식으로 유일하게 표현된다. ECC 에서는 이러한 연산식을 더하고, 빼고, 곱하고, 나누는 등의 연산을 행함으로써 암호 연산을 행하게 된다. 이때, 일반적인 연산을 하는 것이 아니라, ECC 의 구현을 위해서는 n 차 다항식 연산식 F(x)에 대한 모듈로 연산을 행하게 된다.

이러한 모든 연산 중에, 나눗셈 연산은 가장 시간을 많이 소요하는 연산이다. $GF(2^n)$ 위에서의 연산식 A(x)를 B(x)로 나눈 후 결과를 n 차 다항식 F(x)의 모

듈로 값으로 구하는 것은, 주로 다음과 같은 두 단계로 이루어진다:

(1) 역원 $B(x)^{-1}$ 구함.

(2) $A(x)/B(x) = A(x) \cdot B(X)^{-1}$.

다시 말해서, 나눗셈 연산은 역원 연산을 수행한 후에 다시 곱셈 연산을 수행함으로써 이루어 진다. $GF(2^n)$ 위의 역원연산의 구현을 위해 많은 알고리즘이 제시되어 있으며[1,2,3,4,5,6], 그 중 가장 광범위하게 쓰이는 것으로는 Extended Eucliean Algorithm(EEA)를 들 수 있다. EEA 는 연산식을 기반으로 한 연산 뿐 아니라 다른 기반을 이용한 연산에도 광범위하게 사용될 수 있으나, 우리는 본 논문에서 연산식 기반을 목표로 한다.

2. 역원 계산을 위한 알고리즘

우리는 먼저 Euclid 의 GCD 알고리즘을 수정한 Modified GCD 알고리즘을 소개한다. 그리고, 이를 효율적인 하드웨어 구현을 위해 수정한 알고리즘[6]을 소개한다. 다음의 알고리즘에서 A(x)를 역원 연산이 행해질 연산식으로 가정한다. 또한 F(x)는 $GF(2^n)$ 위의 나누어 떨어지지 않는 연산식이다. 이때, modified GCD 알고리즘은 그림 1 과 같이 나타내진다.

알고리즘(Modified GCD)

$R_{-1}(x) := A(x); R_0(x) := A(x); U_{-1}(x) := 0, U_0(x) := 1$
repeat
$i = i + 1$
$Q_i(x) := \lfloor R_{i-2}(x) / R_{i-1}(x) \rfloor$
$R_i(x) := R_{i-2}(x) + Q_i(x) R_{i-1}(x)$
$U_i(x) := U_{i-2}(x) + Q_i(x) U_{i-1}(x)$
until $R_i(x) := 0$
$A^{-1}(X) := U_{-1}(x)$

그림 1. Modifed GCD 알고리즘

이를 그대로 하드웨어로 구현하려면, $Q_i(x)$ 를 계산하기 위하여 많은 시간이 요구된다. Hasan[6]은 이를 하드웨어로 효율적으로 구현할 수 있도록 수정을 가하였다. 수정된 알고리즘은 그림 2 에 주어진다.

수정된 알고리즘은, $Q_i(x)$ 를 직접 구하는 것이 아니라 여러 번에 나누어 구하는 것이다. 즉, $Q_i(x) = x^{d(i,0)} + x^{d(i,1)} + \ldots + x^{d(i,m_i-1)}$ 로 나타내면, 두개의 중첩된 반복문 중 내부 반복문에서 $x^{d(i,j)}$ 를 하나씩 구해주게 된다. 우리는 여기서 두 가지 관찰을 할 수 있다 :

관찰 1) 그림 2 의 수식 (1),(2)에서 보면, U 에 관한 연산, 그리고 R 에 관한 연산이 모두 동일한 크기의 shift 연산을 수행한 후 XOR 연산을 행한 것을 볼 수 있다;

관찰 2) 수정된 알고리즘은 다음과 같은 두 수식을 만족시킨다 (자세한 증명은 [6] 참조)

$$\deg U_{(i)}(x) + \deg R_{(i-1)}(x) = n$$
$$\deg U_{(i)}(x) + \deg R_{(i-1)}(x) < n$$

관찰 1)에서 우리는 그림 4 의 수식 (1),(2)가 동시에 shift 연산으로 수행될 수 있음을 알 수 있다. 또한, 관찰 (2)에서 우리는 U 와 V 를 위한 XOR 게이트의 개수를 n 개에 가깝게 사용할 수 있다는 점을 발견할 수 있다(기존의 구현[7,8]의 경우에는 2n 개의 XOR 게이트 이용). 이제 우리는 위와 같은 성질을 이용해서 우리는 기존의 역원 하드웨어보다 전력 손실이 적은 하드웨어를 구현한다. Hasan 의 알고리즘에서 무엇보다 지연시간이 많이 걸리는 부분은 $\delta^{(i,j)}$ 를 찾는 부분이다. 각 내부 반복문에서 $\delta^{(i,j)}$ 을 구한 후, 이 값을 이용해서 다음 번 내부 반복문 수행에 있어 shift 의 값을 결정하게 된다. 이를 하드웨어로 구현하기 위해서는 비트를 스캔해야 하며, 이는 다소 많은 지연시간을 요구한다. Hasan 은 이를 위해 LUT(Look Up Table) 방법을 제안하였으나, 우리의 경우는 스마트 카드나 내장형 시스템에 탑재를 목표로 하고 있기 때문에, 큰 회로 면적 및 전력 소모를 필요로 하는 LUT 를 사용할 수 없었다. 다음 장에서는 그림 2 에 제시된 알고리즘을 위한 우리의 설계를 자세한 예와 함께 설명한다.

단계 1.
$R_{-1}(x) := A(x); R_0(x) := A(x); U_{-1}(x) := 0, U_0(x) := 1$
$\deg R_{(-1,0)}(x) := \deg R_{-1}(x) = n,$
$\deg R_{(0)}(x)$계산
$d_{(1,0)} = n - \deg R_{(0)}(x), i = 0$
단계 2
do {
$i = i + 1$
$j = 0$
$R_{(i-2,0)}(x) := R_{(i-2)}(x)$
$U_{(i-2,0)}(x) := U_{(i-2)}(x)$
while $(d_{(i,j)} \geq 0)$ do{
$R^{*}_{(i-2,j+1)}(x) := R_{(i-2,j)}(x) - x^{d(i,j)} R_{(i-1)}(x)$ (1)
$U_{(i-2,j+1)}(x) := U_{(i-2,j)}(x) - x^{d(i,j)} U_{(i-1)}(x)$ (2)
$j = j + 1;$
Find $\delta_{(i,j)}$ from LUT
$\deg R_{(i-2,j)}(x) := \deg R_{(i-2,j-1)}(x) - \delta_{(i,j)}$
$d_{(i,j)} := \deg R_{(i-2,j)}(x) - \deg R_{(i-1)}(x)$
}
$R_{(i)}(x) := R_{(i-2,j)}(x)$
$U_{(i)}(x) := U_{(i-2,j)}(x)$
$\deg R_{(i)}(x) := \deg R_{(i-2,j)}(x)$
}while $(R_{(i)}(x) \neq 0)$
$A^{-1}(X) := U_{-1}(x)$

그림 2. Hasan 의 수정 알고리즘

3. 역원 알고리즘의 하드웨어 구현

$GF(2^n)$ 상에서 이루어지는 역원 연산을 위해 우리는 (n+2)개의 XOR 게이트, 그리고 3(n+2)개의 레지스터를 이용한다. 우리는 각 레지스터 열을 tmpReg, reg1, reg2 라 명명한다. (Hasan 의 논문에서는 2n 개의 레지스터의 가능성에 대해 언급하였으나, 내부 반복 문에서 생성되는 중간 값의 저장이 필요하므로 2n 개의 레지스터로는 구현이 불가능하다.) 그 중 reg1, reg2 는 shift 레지스터를 사용한다. 우리는 작은 예를 들어 구

현된 하드웨어의 동작 방법을 설명한다. n=10 으로 가정하고, $F(x)=x^{10}+x^5+x^2+1, A(x)=x^4+x^3+x$ 로 놓았다. 알고리즘에서, $R_{(-1)}(x)=F(x), R_{(0)}(x)=A(x)$ 가 된다. 그림 3 은 제시된 예에 대한 레지스터와 XOR 게이트 열의 간단한 예이다. 레지스터는 각각 12 의 크기를 가지며, U 부분을 위한 부분, 그리고 R 를 위한 부분으로 각각 나뉘어 있다. R 을 위한 레지스터열에서, 가장 상단 레지스터로부터 MSB 를 나타내는 방식으로 값을 할당하였다. 또한 U 를 위한 레지스터도 동일한 방식을 취하였으며, R 을 위한 레지스터와 U 를 위한 레지스터의 경계는 margin 로 표기하였다. 우리의 예에서, 초기 R 의 자리수는 10 으로 주어지고, U 들의 자리수는 0 으로 주어지기 때문에 R 은 11 칸을 차지하고, U 는 1 칸을 차지한다.

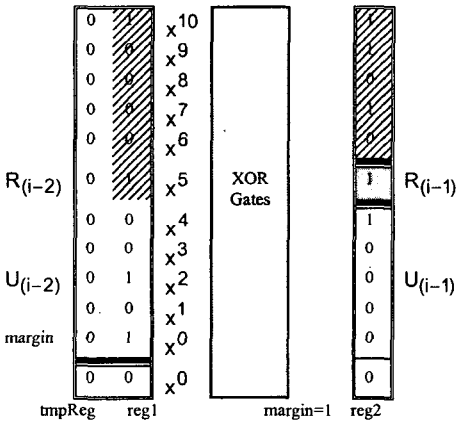


그림 3. 간단한 전체 구조

알고리즘의 $x^{d(i,j)}$는 얼마만큼의 shift 를 하고 XOR 연산을 해야 가장 상위비트의 1 을 없앨 수 있는가를 나타내기 때문에, 우리의 예에서는 5 로 계산된다. 그러므로, R_{i-1} 에서 빗금 아래부분, 즉 A(x)에 해당되는 부분은 5 만큼 shift 되어 F(x)와 XOR 연산이 행하여 진다. 그리고, margin 아래부분 역시 5 만큼 shift 되어 tmpReg 와 XOR 연산을 행한다.(그림 4 에서 하향 가로줄로 표시되었다.)

우리의 구조는 항상 reg1 과 reg2 의 MSB 가 1 이 되도록 한다. 쉬프트연산의 규칙은 다음과 같다.

(1) reg1 의 MSB 가 1 인 경우 : reg1 의 MSB 가 1 이 될때까지 reg1 의 R 을 위한 부분을 위로 shift 를 수행한다. Shift 를 수행할 때마다 reg2 의 U 를 위한 부분을 아래로 shift 해준다.

(2) Reg2 의 MSB 가 1 인 경우 :reg2 의 MSB 가 1 이 될때까지 reg2 전체를 위로 shift 수행한다.

또한, 연산기의 XOR 연산의 규칙은 다음과 같다:

(1) reg1 의 R 을 위한 부분은 reg2 의 R 을 위한 부분과 연산한다.

(2) TmpReg 와 reg2 의 U 를 위한 부분을 연산한다.

연산을 수행한 결과는, 그림 4(b)에 주어진다. (간략한 표기를 위해 XOR 게이트 열을 생략하고 도표로 표현하였다.) 결과를 보면, 새로이 갱신된 $R_{(-1)}$ 에 해당되는 부분의 자리수가 9 임을 알 수 있다. 그러므로 다음번 내부반복을 위한 $\delta^{(i,j)}$의 값은 1 가 된다. 그러므로 reg1 은 1 만큼 상향으로 shift 되고, 이에 맞추어 reg2 의 U 를 위한 부분은 1 만큼 아래로 shift 되게 된다. 초기 U_0 은 1 로 주어졌고 이 값 또한 5 만큼 shift 되어 tmpReg 와 XOR 연산이 행해진 결과가 tmpReg 에 저장되어 있다. 그림 4 의 (c), (d), (e)는 각 레지스터의 변화를 설명하고 있다. 다섯번의 내부 반복문 수행후, $\delta^{(i,j)}$ 값이 음수값을 가지게 되었다. Hasan 의 알고리즘에 따르면, 이 경우 내부 반복문에서 외부 반복문으로 빠져 나오게 된다. 또한, $R_{(i-2)}$ 는 $R_{(i)}$ 로 변하며, $U_{(i-2)}$는 $U_{(i)}$로 변하게 된다.

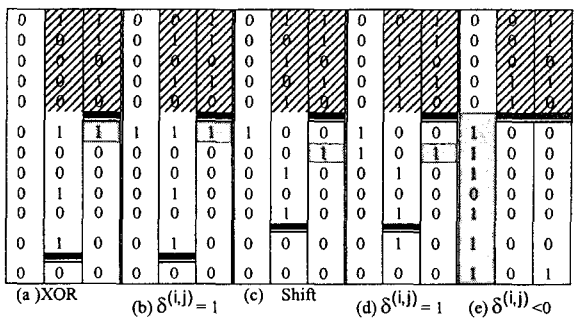
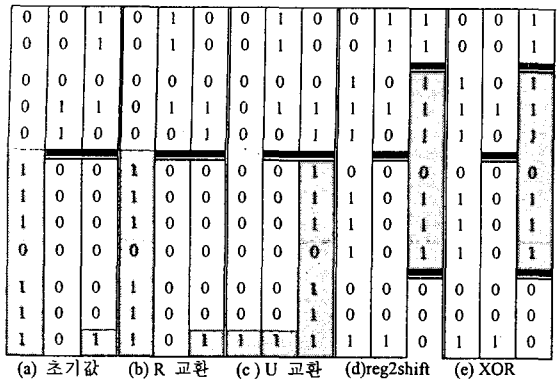


그림 4. 내부반복문 수행시의 레지스터의 변화의 예

그림 5(b)에 보이는 바와 같이 $R_{(i-2)}$ 에 해당되는 레지스터와 $R_{(i-1)}$에 해당되는 내부값을 교환한다우리는 이제 그림 5(c)에 보이는 바와 같이 tmpReg 의 값을 $U_{(i-1)}$에 넣고, $U_{(i-1)}$에 있던 값을 tmpReg 에 넣는다. 이제 reg2 의 R 을 위한 부분의 가장 윗 자리수, 즉 MSB 가 0 값을 갖는 것을 알수 있다. 이 값이 1 이 될때까지 reg 전체를 shift 한다. 그 결과가 그림 5(c)에 나타나 있다. $\delta^{(i,j)}$의 값이 이제 양수로 변화하게 되며, 이제 첫번째 내부반복문 실행에서 행했던 바와 같이 두 번째 반복을 같은 방법으로 $\delta^{(i,j)}$의 값이 음수가 될 때까지 수행해 준다. 첫번째 XOR 연산을 행한 결과를 보면 $\delta^{(i,j)}=1$ 이므로 reg2 의 U 에 대한 레지스터를 아래로 shift 를 한번 수행한다. 이 결과가 그림 5(e)에 나타나 있다. 이제 다시 $\delta^{(i,j)}=3$ 이 되었으므로, reg2 의 U 에 해당하는 부분을 아래로 3 만큼 shift 한 후 XOR 연산을 수행하며, 이러한 과정은 그림 6(a)-,(d),(e)에 나타나 있다. 우리가 설계한 하드웨어는 이러한 반복을 $R_{(i)}$의 값이 0 이 될 때 까지 계속 수행하여 역원 연산을 행하게 된다. 그림 6 에서 우리는 은 계속된 수행에 따른 레지스터 변화의 예를 좀더 추가하였다.



(a) 초기값 (b) R 교환 (c) U 교환 (d)reg2shift (e) XOR

그림 5. 외부반복문 수행시의 레지스터의 예

하드웨어 수행 시 가장 많은 지연시간을 차지하는 부분은 $x^{d(i,j)}$를 결정짓기 위해 $\delta^{(i,j)}$를 알고리즘에서 구하는 부분이다 이 경우, 가장 상위비트부터 $R_{(i)}$가 1을 갖는 레지스터까지 스캔해 나간다. 이 값의 상한선은 이전 반복문에서 값을 알고 있기 때문에 스캔에 소요되는 시간이 많이 줄어드는 것을 알 수 있다. Hasa 은 이 경우 LUT 를 사용하여 지연시간을 감소시키고 있지만, 우리의 경우, 즉 스마트 카드를 위한 경우에 있어서는 회로 면적을 작게 하는 것이 지연시간만큼 아니면 보다 중요하기 때문에, 우리는 비트를 스캔하는 방법을 취한다.

4. 실험 결과

우리는 제안된 구조의 효율성을 보이기 위해 널리 사용되는 163 자리수의 ECC 연산의 역원 연산을 수행하였고, 이를 비교하기 위해 [7]을 구현하였다. ([7]은 Modified Almost Inverse Algorithm 을 구현하였다. 이 구조는 적은 회로 면적을 목표로 하는 특징을 가지고 있기 때문에 주어진 알고리즘을 최소 면적으로 수행할 수 있도록 구현하였다.) 모든 실험은 1-GB Memory 를 가지고 있는 Pentium-3 500MHx Linux 에서 수행되었다. 하드웨어의 크기에 있어서 XOR 게이트는 50% 가량의 감소를 가지고 왔으며 레지스터의 개수에 있어서는 거의 비슷한 숫자를 보였다. 전체적 구조나 컨트롤에 있어서도 우리가 제안한 구조가 훨씬 간단함이 보여졌다.

$F(x) = x^{163} + x^7 + x^6 + x^3 + 1$

입력(핵사표현)	[7,8] (clock)	Ours (clock)
00000005 08ab0f0e 043a6718 03c8653c 09812de1 03b7e15e	413	351
00000005 04057b12 08e60ef5 05ae446b 05e3bd0c 123c2118	369	354
00000004 07d6e50c 1e826555 1fa7cd95 012e9e50 1017db38	398	365
00000005 01dca96c 3a327e74 13e5a89a 35d857cd 09ecdb03	410	366
00000006 12a2b07c 25463550 00660d50 02d2ef64 07bb71b6	399	387
00000007 0c194840 1700dd38 28457587 02c7fd79 12cc4947	409	377

도표 1. 163 비트의 입력에 대한 실험 결과

도표 1 은 여러 개의 163 비트의 임의의 수식에 대한 실험시간의 결과이다. 우리는 비교를 위해 임의의 수식에 대해 Modified Almost Inverse Algorithm[7]의 구현이 몇 번의 clock 을 소모하여 결과값을 내는지, 그리고 우리의 알고리즘이 몇 번의 clock 을 소모하여 결과값을 내는지를 simulation 하였고, 그 결과를 도표 1 에 정리하였다. 도표 1 에서, 우리의 구조의 지연시간은 Modified Almost Inverse Algorithm 과 지연시간에 있어서도 향상을 가지고 오는 것을 볼 수 있다. 이는 우리의 구조가 저전력과 적은 회로 공간을 필요로 하는 스마트 카드에 적합하다는 것을 보여준다.

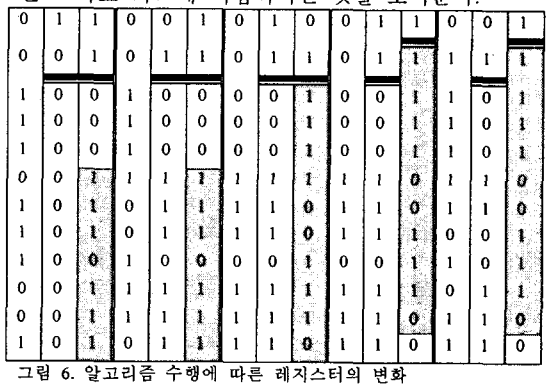


그림 6. 알고리즘 수행에 따른 레지스터의 변화

5. 결론 및 향후 과제

본 논문에서 우리는 스마트 카드의 암호 모듈 구현을 위한 역원 연산 수행을 위한 회로를 제안하였다. 이는 기존의 회로에 비해 비슷한 지연시간을 가지면서 회로 면적에 있어 큰 향상을 가지고 오는 것을 관찰할 수 있었다. 우리는 이 결과를 바탕으로 하여 역원 계산뿐 아니라 나눗셈을 효율적으로 하는 회로 구조 및 지연시간을 향상하는 구조를 연구하고자 한다.

참고문헌

[1] T. Itoh, "A fast algorithm for computing multiplicative inverses in $GF(2^n)$", Infor. and Comp. Vol. 78, pp. 171-177, 1988

[2] M.A. Hasan and V. K. Bhargava, "Bit-Serial Systolic Divider and Mulitplier for $GF(2^n)$", IEEE Trans. Compu., vol. 41, pp. 972-980, Aug. 1992.

[3] H.Brunner, A. Curiger, and M. Hofstetter, "On Computing Multiplicative Inverses in $GF(2^n)$", IEEE Trans. Compu. Vol. 42, pp. 1010-1015, Aug. 1993.

[4] C.C. Wang, T.K.Truong, H.M.Shao, L.J. Deutsch, J.K. Omura, and I.S. Reed, "VLSI Architecture for Computing Multiplications and Inverses in $GF(2^n)$", IEEE Trans. Compu. Vol: C-34, pp.709-717, Aug. 1985.

[5] M. Morii and M. Kasahara, "Efficient construction of gate circuit for computing multiplicative inverses over $GF(2^n)$", Trans. IEICE, vol. E72, pp. 37-42, 1989.

[6] M.A. Hasan, "Efficient Computation of Multiplicative Inverses for Cryptographic Applications", 15th IEEE Symposium on Computer Arithmetic, pp.66-72, 2001.

[7] D. Hankerson, J. L. Hernandez, and A.Menezes, "Software Implementation of Elliptic Curve Crypography over Binary Fields", CHES 2000.