(12) 특허협력조약에 의하여 공개된 국제출원

(19) 세계지식재산권기구
국제사무국

(43) 국제공개일
2018 년 5 월 24 일 (24.05.2018)

WIPO | PCT

(10) 국제공개번호
WO 2018/092981 A2

(51) 국제특허분류:
H03M 7/04 (2006.01) *G06F 21/46* (2013.01)
G06F 7/533 (2006.01)

(21) 국제출원번호: PCT/KR2017/001113

(22) 국제출원일: 2017 년 2 월 2 일 (02.02.2017)

(25) 출원언어: 한국어

(26) 공개언어: 한국어

(30) 우선권정보:
10-2016-0154192 2016 년 11 월 18 일 (18.11.2016) KR

(71) 출원인: 부산대학교 산학협력단 (PUSAN NATIONAL UNIVERSITY INDUSTRY-UNIVERSITY COOPERATION FOUNDATION) [KR/KR]; 46241 부산시 금정구 부산대학로63번길 2 (장전동, 부산대학교), Busan (KR).

(72) 발명자: 황두희 (HWANG, Doo hee); 46241 부산시 금정구 부산대학로63번길 2, 313동 417호 (장전동, 부산대학교), Busan (KR). 최윤호 (CHOI, Yoon Ho); 46242 부산시 금정구 금샘로 262, 210동 1002호 (구서동, 쌍용예가 2단지), Busan (KR).

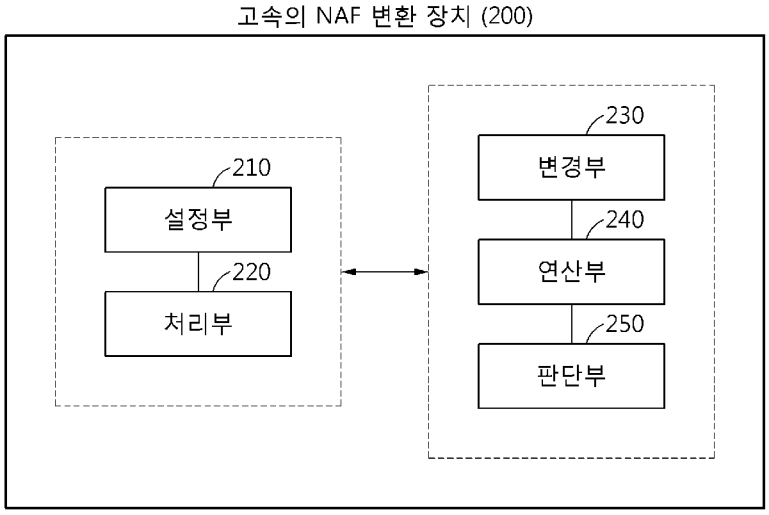
(74) 대리인: 특허법인 무한 (MUHANN PATENT & LAW FIRM); 06044 서울시 강남구 학동로 3길 9, 5층 (논현동, 명림빌딩), Seoul (KR).

(81) 지정국 (별도의 표시가 없는 한, 가능한 모든 종류의 국내 권리의 보호를 위하여): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KH, KN, KP, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

WO 2018/092981 A2

(54) **Title:** HIGH-SPEED NAF CONVERSION DEVICE AND HIGH-SPEED NAF CONVERSION METHOD

(54) **발명의 명칭:** 고속의 NAF 변환 장치 및 고속의 NAF 변환 방법



200 ... High-speed NAF conversion device
210 ... Setting unit
220 ... Processing unit
230 ... Changing unit
240 ... Calculating unit
250 ... Determining unit

(57) **Abstract:** The present invention relates to a high-speed NAF conversion device and a high-speed NAF conversion method. The high-speed NAF conversion device according to the present invention may comprise: a setting unit for setting a value of a window; and a processing unit which, on the basis of least significant bit (LSB) of a public key, primarily sets a window having the set value as the length to at least a part of the public key, and if a target bit to be encrypted in the public key is outside the window, secondarily sets the window by shifting right by ω-bits (where ω is a natural number) so that the target bit is included.

[다음 쪽 계속]

(84) 지정국 (별도의 표시가 없는 한, 가능한 모든 종류의 역내 권리의 보호를 위하여): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), 유라시아 (AM, AZ, BY, KG, KZ, RU, TJ, TM), 유럽 (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

공개:

— 국제조사보고서 없이 공개하며 보고서 접수 후 이를 별도 공개함 (규칙 48.2(g))

(57) 요약서: 본 발명은 고속의 NAF 변환 장치 및 고속의 NAF 변환 방법에 관한 것이다. 본 발명에 따른 고속의 NAF 변환 장치는, 윈도우의 값을 설정하는 설정부 및 공개키의 LSB(Least Significant Bit)를 기준으로, 상기 설정된 값을, 길이로 갖는 윈도우를, 상기 공개키의 적어도 일부에 제1 세팅하고, 상기 공개키 내 암호화 대상이 되는 타깃 비트가, 상기 윈도우의 밖에 있는 경우, 상기 타깃 비트가 포함되도록 상기 윈도우를, ω-비트 오른쪽 이동(ω-bit right shift)(상기 ω은 자연수)시켜, 상기 윈도우를 제2 세팅하는 처리부를 포함할 수 있다.

명세서

발명의 명칭: 고속의 NAF 변환 장치 및 고속의 NAF 변환 방법

기술분야

[1] 본 발명은 고속의 NAF 변환 장치 및 고속의 NAF 변환 방법에 관한 것이다.

배경기술

[2] 종래 기술에서의 비밀키 암호 방식은 하나의 암호화키를 사용하여 암호화 및 복호화 함으로써, 암호화키에 대한 유출 가능성이 매우 높다. 반면 공개키 암호 방식은 공개키와 비밀키를 사용하여 암호화함으로써, 키 교환에 대한 안정성을 보장하고 높은 보안 강도를 제공한다. 종래 기술에서의 대표적인 공개키 암호 방식은 RSA(Rivest, Shamir, Adleman)와 ECC(Elliptic Curve Cryptography)가 있다.

[3] 먼저, RSA는 소인수분해의 어려움에 기초한 알고리즘으로, 두 개의 소수들의 곱과 추가 연산을 통해 공개키 및 비밀키를 구할 수 있다. 즉, RSA는 핵심 기능으로 암호화, 복호화 과정에서 모듈러 멱승 연산을 사용한다. 예를 들면, RSA의 경우에서, p,q는 소수이고, e는 $\gcd(\varnothing(n),e)=1, 1<e<\varnothing(n)$이며, $d=e^{-1}(\mathrm{mod}\,\varnothing(n))$이다(비밀키=$\{d,p,q\}$, 공개키=$\{e,n\}$). 만일, 공개키가 {7, 187}이고, 비밀키가 {23, 17, 11}일 때 메시지 $M=88$ 이라면, 암호화된 메시지 C는 $C=88^7 \mathrm{mod}\,187=11$가 되고, C를 복호화하면 $11^{23}\mathrm{mod}\,187=88$이므로 처음 메시지 M가 구해질 수 있다. 이 때, 88^7과 11^{23}을 구하고 $\mathrm{mod}\,187$을 계산하는 것은 아주 많은 연산을 요구할 수 있다.

[4] 해당 연산은 RSA의 속도에 큰 영향을 미치기 때문에 모듈러 곱셈 최적화, 모듈러 곱셈 횟수 최적화에 대한 다양한 연구가 이루어지고 있다.

[5] 다음으로, ECC는 타원곡선 이론에 기초한 공개키 암호 방식으로, 유한체 F_q 상에 정의된 타원곡선 E에서의 이산로그문제에 기초한다. ECC는 핵심 기능으로 유한체 F_q상에 정의된 타원곡선 E위의 점 P를 양의 정수 k만큼 더한 kP를 계산하는 연산인 스칼라 곱셈을 사용한다. 스칼라 곱셈은 타원곡선 상의 점에 대한 연산인 포인트 더블링(point doubling) 연산과 포인트 에디션(point addition) 연산으로 구성되어 있다.

[6] 이때, ECC의 속도는 스칼라 곱셈 연산 속도에 큰 영향을 받을 수 있다. 그래서, ECC의 속도를 향상시키기 위해서, 스칼라 곱셈 연산 속도를 향상시키는 연구가 요구된다.

발명의 상세한 설명

기술적 과제

[7] 본 발명은 상기와 같은 문제점을 해결하기 위하여 안출된 것으로서, RSA의 모듈러 멱승 연산에서의 모듈러 곱셈 횟수와 ECC의 스칼라 곱셈 연산에서의

포인트 에디션 횟수를 감소시키도록 NAF(Non-Adgacetnt Form) 변환 과정의 속도를 향상시킬 수 있는 것을 목적으로 한다.

과제 해결 수단

[8] 상기의 목적을 이루기 위한 고속의 NAF 변환 장치는, 윈도우의 값을 설정하는 설정부 및 공개키의 LSB(Least Significant Bit)를 기준으로, 상기 설정된 값을, 길이로 갖는 윈도우를, 상기 공개키의 적어도 일부에 제1 세팅하고, 상기 공개키 내 암호화 대상이 되는 타깃 비트가, 상기 윈도우의 밖에 있는 경우, 상기 타깃 비트가 포함되도록 상기 윈도우를, ω-비트 오른쪽 이동(ω-bit right shift)(상기 ω은 자연수)시켜, 상기 윈도우를 제2 세팅하는 처리부를 포함할 수 있다.

[9] 또한, 상기 목적을 달성하기 위한 기술적 방법으로서, 고속의 NAF 변환 방법은, 윈도우의 값을 설정하는 단계, 공개키의 LSB(Least Significant Bit)를 기준으로, 상기 설정된 값을, 길이로 갖는 윈도우를, 상기 공개키의 적어도 일부에 제1 세팅하는 단계, 및 상기 공개키 내 암호화 대상이 되는 타깃 비트가, 상기 윈도우의 밖에 있는 경우, 상기 타깃 비트가 포함되도록 상기 윈도우를, ω-비트 오른쪽 이동(ω-bit right shift)(상기 ω은 자연수)시켜, 상기 윈도우를 제2 세팅하는 단계를 포함하여 구성할 수 있다.

발명의 효과

[10] 본 발명의 일실시예에 따르면, RSA의 모듈러 멱승 연산에서의 모듈러 곱셈 횟수와 ECC의 스칼라 곱셈 연산에서의 포인트 에디션 횟수를 감소시키도록 NAF 변환 과정의 속도를 향상시킬 수 있다.

[11] 또한, 본 발명의 일실시예에 따르면, $\lceil l/w \rceil$ 번의 비교 연산 과정을 수행함으로써, l번의 비트 비교 연산 과정을 수행하는 NAF_w 알고리즘의 연산 처리 속도를 최적화 할 수 있다(l은 양의 정수 k의 이진수 변환에 따른 비트 수, w 는 윈도우 크기).

[12] 또한, 본 발명의 일실시예에 따르면, NAF 변환 시, l(l은 양의 정수 k의 이진수 변환에 따른 비트 수)이 증가하거나 홀수인 경우를 발생시키는 패턴이 많이 존재하는 경우에 NAF 변환 시간을 단축할 수 있다.

도면의 간단한 설명

[13] 도 1a 내지 도 1i는 본 발명의 일실시예에 따른 RSA 또는 ECC에서 사용하는 알고리즘을 설명하기 위한 도면이다.

[14] 도 2는 본 발명의 일실시예에 따른 고속의 NAF 변환 장치를 나타내는 블록도이다.

[15] 도 3a 및 도 3b는 본 발명의 일실시예에 따른 고속의 NAF 변환 장치를 이용한 구현예를 설명하기 위한 도면이다.

[16] 도 4a 및 도 4b는 본 발명의 일실시예에 따른 고속의 NAF 변환 장치를 이용한 구현예를 설명하기 위한 도면이다.

[17] 도 5는 본 발명의 일실시예에 따른 고속의 NAF 변환 장치를 구현하기 위한

알고리즘을 도시한 도면이다.

[18] 도 6 내지 도 11은 본 발명의 일실시예에 따른 고속의 NAF 변환 장치를 구현하기 위한 알고리즘의 성능 평가를 설명하기 위한 도면이다.

[19] 도 12는 본 발명의 일실시예에 따른 고속의 NAF 변환 방법을 구체적으로 도시한 작업 흐름도이다.

발명의 실시를 위한 최선의 형태

[20] 이하에서, 본 발명에 따른 실시예들을 첨부된 도면을 참조하여 상세하게 설명한다. 그러나, 본 발명이 실시예들에 의해 제한되거나 한정되는 것은 아니다. 각 도면에 제시된 동일한 참조 부호는 동일한 부재를 나타낸다.

[21] 본 명세서에서 설명되는 고속의 NAF 변환 장치 및 고속의 NAF 변환 방법은 RSA의 모듈러 멱승 연산과 ECC의 스칼라 곱셈 연산 등에서 필수적으로 사용하는 양의 정수 k에 대한 NAF(k) 변환 과정의 속도를 향상시킬 수 있다.

[22] 명세서 상에서 설명의 이해를 돕기 위하여, 도 1a 내지 도 1i를 참조하여 RSA 또는 ECC에서 사용하는 알고리즘에 대하여 설명하고자 한다.

[23] 도 1a 내지 도 1i는 본 발명의 일실시예에 따른 RSA 또는 ECC에서 사용하는 알고리즘을 설명하기 위한 도면이다.

[24] 도 1a 및 도 1b는 스퀘어 앤드 멀티플라이(Square and Multiply) 기법을 설명하기 위한 도면이다. 도 1a는 스퀘어 앤드 멀티플라이 기법을 위한 제1 알고리즘으로서, RSA의 모듈러 멱승 연산에서 연산량을 줄이기 위해서 사용되는 알고리즘일 수 있다. $a^k \bmod n$ 형태의 계산에 있어서, 종래에는 k를 십진수로 표현하였지만, 도 1b에 도시된 바와 같은, 제1 알고리즘은 $k_l k_{l-1} \cdots k_0$ 형태의 이진수로 표현할 수 있다. 제1 알고리즘의 항목(1)에서, C는 0으로 초기화되어 알고리즘이 종료될 때는 k값이 될 수 있으며, f는 $a^k \bmod n$ 값을 저장하는 변수일 수 있다. 항목(2)에서 확인할 수 있듯이, 스퀘어 앤드 멀티플라이 기법은 i를 $l-1$부터 0까지 감소시키며 $k_{(2)}$의 각 비트(bit)들을 왼쪽-오른쪽(Left-to-Right) 방식으로 1-비트씩 이동(Shift)하며 확인할 수 있다. 이 때, 스퀘어 앤드 멀티플라이 기법은 $c \leftarrow 2 \times c$, $f \leftarrow (f \times f) \bmod n$ 연산을 수행하면서(항목(2.1)), $k_{(2)}$의 i번째 비트인 k_i가 1인 경우에는 추가적으로 $c \leftarrow c+1$과 $f \leftarrow (f \times a) \bmod n$ 연산을 수행할 수 있다(항목(2.2)). 따라서, 스퀘어 앤드 멀티플라이 기법은 $k_{(2)}$에서 1의 개수를 줄인다면 모듈러 곱셈 연산 횟수를 줄일 수 있다. 도 1b는 스퀘어 앤드 멀티플라이 기법을 이용한 연산 예를 나타내는 도면이다.

[25] 도 1c는 바이너리(Binary)기법을 설명하기 위한 도면이다. 바이너리 기법은 ECC의 $Q = kP$를 계산하는 스칼라 곱셈에서 가장 일반적으로 사용될 수 있다.

도 1c에 도시된, 바이너리 기법을 위한 제2 알고리즘의 항목(1)에 나타낸 바와 같이, 바이너리 기법은 $k_{(2)}$의 각 비트들을 왼쪽-오른쪽(Left-to-Right) 또는 오른쪽-왼쪽(Right-to-Left) 방식으로 1-비트씩 이동하며 확인할 수 있다. 이 때, 바이너리 기법은 항목(2.1)과 같이 각 비트마다 한 번의 포인트 더블링(point doubling) 연산을 수행할 수 있다. 비트가 0이 아닌 경우, 바이너리 기법은 항목(2.2)와 같이 추가적인 포인트 에디션(point addition) 연산을 수행하도록 0이 아닌 비트의 수만큼 포인트 에디션 연산을 수행할 수 있다. 따라서, 바이너리 기법은 $k_{(2)}$ 에서 1의 개수를 줄인다면 스칼라 곱셈 연산 속도를 향상시킬 수 있다.

[26] 바이너리 기법의 예시를 들어 설명하면, $k=7=(111)_2$일 때, $7P = 2(2(P)+P)+P$일 수 있고, $k=6=(110)_2$ 일 때, $6P = 2(2(P)+P)$ 일 수 있다.

[27] 도 1d 및 도 1e는 NAF 기법을 설명하기 위한 도면이다. RSA의 모듈러 멱승 연산을 최적화하기 위한 스퀘어 앤드 멀티플라이 기법과, ECC의 스칼라 곱셈에 사용되는 바이너리 기법은 $k_{(2)}$에서 0이 아닌 비트의 수가 많을수록 모듈러 곱셈 연산, 포인트 에디션 연산 횟수가 증가할 수 있다. NAF_w 알고리즘은 $k_{(2)}$의 각 비트에 부호가 있는 숫자를 사용할 수 있다. NAF_w 알고리즘은 0이 아닌 숫자가 연속된 부분을 제거하여 $k_{(2)}$ 보다 0이 아닌 숫자의 평균 밀도를 감소시킴으로써, RSA의 모듈러 멱승 연산에서는 모듈러 곱셈의 횟수를 줄이기 위해서 사용될 수 있다. 또한, NAF_w 알고리즘은 ECC의 스칼라 곱셈에서는 포인트 에디션 연산의 수를 줄이기 위해서 사용될 수 있다. 일반적으로 길이가 l인 $NAF_w(k)$를 인홀수, $k_{l-1}\neq 0$와 같이 표현할 수 있다. 이 때, $k=\sum_{i=0}^{l-1}k_i2^i,\ |k_i|<2^{w-1}$ $(NAF_2(k)=NAF(k))$ 일 수 있다.

[28] 도 1d에 도시된 바와 같이, 제3 알고리즘은 $k_{(2)}$을 $NAF_w(k)$로 변환하는 NAF_w 알고리즘일 수 있다. 제3 알고리즘은 양의 정수 k 와 윈도우 크기(window size)인 w를 입력 받아 k의 이진 표현인 $k_{(2)}$에서 0이 아닌 정수인 비트 수가 줄어들도록 변환한 $NAF_w(k)$를 반환할 수 있다. NAF 기법은 제3 알고리즘의 항목(2)와 같이, $k\geq 1$조건을 만족하면 와일(While)문을 반복하여 수행할 수 있고, k가 홀수일 때(즉, LSB(Least Significant Bit)가 1일 때) k_i의 값은 $k \bmod 2^w$가 되도록 할 수 있다(항목(2.1). 이 때, NAF 기법은 앞서 구한 k_i가 2^{w-1}보다 크거나 같으면 k_i 값은 k_i-2^w 가 되도록 하고, 그렇지 않을 경우에는 앞에서 구한 값을 사용할 수

있다(항목(2.1.1). 또한, NAF 기법은 k에 k_i를 빼줄 수 있다(항목(2.1.2)). 항목(2.2)에 나타낸 바와 같이, k 가 짝수일 경우(즉, LSB가 0일 경우) k_i값은 0이 될 수 있다. 항목(2.3)에서, NAF 기법은 $k \leftarrow k/2, i \leftarrow i+1$에서 $k \leftarrow k/2$ 는 k 값을 2로 나누는 수행을 할 수 있고, 이 과정이 오른쪽으로 1-비트 이동하는 동작일 수 있다. 예를 들어, 십진수 7을 이진수로 나타내면 111_2이고, 7을 2로 나누면 실제로는 3.5이지만, 제3 알고리즘에서 변수 k가 정수(Integer)로 선언되어있기 때문에 k값은 소수점을 제외한 3이 되고, 이를 이진수로 나타내면 11_2이 될 수 있다. 따라서, NAF 기법은 가장 오른쪽 비트를 없애고 모든 비트가 오른쪽으로 한 칸씩 이동하도록 할 수 있다. NAF 기법은 $k_{(2)}$의 길이가 l이므로, l번의 비트 비교 연산 과정을 수행할 수 있다. 도 1e는 NAF 기법을 이용한 연산 예를 나타내는 도면이다.

[29] 도 1f 및 도 1g는 레코딩 바이너리(Recording Binary) 기법을 설명하기 위한 도면이다. 레코딩 바이너리 기법은 NAF_w 알고리즘을 사용하여 모듈러 멱승 연산을 수행하는 알고리즘일 수 있다. 항목(2.3)과 같이, k_i가 -1인 경우를 처리하는 과정을 추가한다는 점과 $NAF_w(k)$를 사용한다는 점을 제외하면 스퀘어 앤드 멀티플라이 기법과 동일할 수 있다. 제4 알고리즘은 $\bmod n$에서 a의 곱셈에 대한 역원인 a^{-1} 가 미리 계산될 수 있다. 레코딩 바이너리 기법의 상세 동작 과정은 도 1f에서의 제4 알고리즘에 나타낸 바와 같고, 도 1g는 레코딩 바이너리 기법을 이용한 연산 예를 나타내는 도면이다.

[30] 도 1h는 윈도우 NAF(Window NAF) 기법을 설명하기 위한 도면이다. 윈도우 NAF 기법은 NAF_w 알고리즘을 사용하여 스칼라 곱셈을 수행하는 알고리즘일 수 있다. 또한, 윈도우 NAF 기법은 바이너리 기법에 NAF_w 알고리즘을 활용하여 포인트 에디션 연산을 줄일 수 있다. 윈도우 NAF 기법은, $P_i = iP, \{i \in 1, 3, \ldots, 2^{w-1}-1\}$ 에 해당하는 타원곡선 상의 점들을 미리 계산하여 저장할 수 있다. 따라서 윈도우 NAF 기법은 w의 크기에 따라 미리 계산한(pre-computation) 값들을 저장할 메모리가 요구될 수 있다. 윈도우 NAF 기법의 상세 동작 과정은 도 1h의 제5 알고리즘에 나타낸 바와 같다. 윈도우 NAF 기법을 이용한 연산 예로서, $k = 7 = (111)_2$일 때, $NAF_w(k) = 100-1$이고, $7P = 2(2(2(P)))-P$ 일 수 있다. 또한, 윈도우 NAF 기법을 이용한 연산 예로서, $k = 6 = (110)_2$ 일 때, $NAF_w(k) = 10-10$이고, $6P = 2(2(2(P))-P)$ 일 수 있다.

[31] 도 1i는 슬라이딩 윈도우(Sliding Window) 기법을 설명하기 위한 도면이다. 슬라이딩 윈도우 기법은 k에 대한 $NAF_w(k)$ 와 타원곡선 상의 점 $P_i = iP, \{i \in 1, 3, \ldots, 2(2^w-(-1)^w)/3-1\}$ 을 미리 계산하여 결과를 저장하기 때문에 w에 따라 미리 계산한(pre- computation) 값들을 저장할 메모리가 요구될

수 있다. 그리고 슬라이딩 윈도우 기법은 $w \geq 3$ 부터 윈도우 NAF 기법에 비해 미리 계산한 값(pre-computation)이 많은 단점이 있지만, 일반적으로 사용하는 w 크기인 $2 \leq w \leq 4$ 인 경우에 포인트 에디션 연산 횟수가 줄어든다는 장점이 있을 수 있다. 또한, 슬라이딩 윈도우 기법은 0,1로만 표현되는 일반적인 이진 표현을 사용할 수 있지만 이 경우는 타원곡선 상의 점 $P_i = iP, \{i \in 1, 3, \ldots, 2^w - 1\}$ 을 미리 계산하기 때문에 메모리 요구가 훨씬 증가할 수 있다. 따라서, 슬라이딩 윈도우 기법은 k를 $NAF_w(k)$로 변환하여 사용함으로써, 효율을 증가시킬 수 있다. 슬라이딩 윈도우 기법의 상세 동작 과정은 도 1i의 제6 알고리즘에 나타낸 바와 같다.

[32] 도 2는 본 발명의 일실시예에 따른 고속의 NAF 변환 장치를 나타내는 블록도이다.

[33] 본 발명의 고속의 NAF 변환 장치(200, 이하, NAF 변환 장치)는 설정부(210) 및 처리부(220)를 포함할 수 있다. 또한, 실시예에 따라, NAF 변환 장치(200)는 변경부(230), 연산부(240) 및 판단부(250)를 추가하여 구성할 수 있다.

[34] 설정부(210)는 윈도우의 값을 설정한다. 즉, 설정부(210)는 양의 정수인 윈도우의 값을 설정할 수 있다. 예를 들면, 설정부(210)는 2 이상의 정수 중 적어도 하나의 값을 윈도우의 값으로 설정할 수 있다.

[35] 처리부(220)는 공개키의 LSB(Least Significant Bit)를 기준으로, 상기 설정된 값을, 길이로 갖는 윈도우를, 상기 공개키의 적어도 일부에 제1 세팅한다. 즉, 처리부(220)는 제1 세팅으로서, 설정부(210)에 의해 설정된 값만큼 공개키의 일부분에 윈도우를 세팅할 수 있다. 예를 들면, 윈도우의 값이 3으로 설정된 경우, 처리부(220)는 공개키 중 LSB로부터 3 비트만큼을 제1 세팅할 수 있다.

[36] 또한, 처리부(220)는 상기 공개키 내 암호화 대상이 되는 타깃 비트가, 상기 윈도우의 밖에 있는 경우, 상기 타깃 비트가 포함되도록 상기 윈도우를, ω-비트 오른쪽 이동(ω-bit right shift)(상기 ω은 자연수)시켜, 상기 윈도우를 제2 세팅한다. 즉, 처리부(220)는 타깃 비트를 윈도우에 포함시키기 위하여, 윈도우의 값인 w-비트만큼 이동시켜 윈도우를 제2 세팅할 수 있다. 예를 들면, 윈도우의 값이 3으로 설정된 경우, 처리부(220)는 윈도우를 3-비트 이동시켜 타깃 비트가 포함되도록 세팅할 수 있다.

[37] 설명의 이해를 돕기 위하여, 이하 도 3a, 도 3b, 도 4a, 및 도 4b를 참조하면서 도 2에서의 NAF 변환 장치(200)를 설명하고자 한다.

[38] 도 3a 및 도 3b는 본 발명의 일실시예에 따른 고속의 NAF 변환 장치를 이용한 구현예를 설명하기 위한 도면이다.

[39] 도 3a 및 도 3b에 도시된 바와 같이, 처리부(220)는 NAF_w 에 대해 $w = 3$으로 설정되면, NAF_3 알고리즘에서 k 가 홀수인 경우(즉, LSB가 1인 경우)의 윈도우를, 제1 세팅할 수 있다(회색 음영 부분, 311).

[40] 변경부(230)는 상기 제1 세팅된 윈도우에 포함되는, 상기 공개키 내 비트를

모두 '0'으로 변경할 수 있다. 즉, 변경부(230)는 제1 세팅된 비트에 대하여 모두 '0'으로 변경할 수 있다. 도 3a의 일례에서, 변경부(230)는 윈도우에 포함된(즉, 회색 음영 부분) 비트를 모두 '0'으로 변경할 수 있다(312).

[41] 연산부(240)는 상기 제1 세팅된 윈도우에 포함되는, 상기 공개키 내 비트에 대한 환산값을 산출할 수 있다. 도 3a의 일례에서, 연산부(240)는 가장 오른쪽의 3-비트가 011이라면 제3 알고리즘에서의 항목(2.1)인 $k_i \leftarrow k \bmod 2^3$에 대하여 $k_i \leftarrow 3$ 을 산출할 수 있다.

[42] 이때, 변경부(230)는 산출된 상기 환산값이 선정된 기준값을 넘는지를 확인하여, 정해진 수를 가산 또는 감산하여, 상기 환산값을 '0'을 만들 수 있다. 예를 들어, 기준값이 2^2로 선정된다면, 변경부(230)는 환산값 k_i가 $k_i \geq 2^2$ 인지를 확인할 수 있다. 도 3a의 일례에서, 변경부(230)는 k_i 값이 제3 알고리즘에서의 항목(2.1.1)의 $k_i \geq 2^2$를 만족시키지 않으므로, $k \leftarrow k - k_i$ 를 수행할 수 있다. 이때, 변경부(230)는 k_i가 3이므로 $k \leftarrow k - 3$ 을 수행하여 가장 오른쪽의 3-비트가 000이 되도록 할 수 있다(312).

[43] 이때, 처리부(220)는 상기 ω을 '1'로 결정하고, 상기 윈도우를, 1-비트 오른쪽 이동(1-bit right shift)시키되, 상기 타깃 비트가 상기 윈도우에 포함될 때까지 상기 1-비트 오른쪽 이동을 반복하여 상기 제2 세팅할 수 있다. 도 3a의 일례에서, 처리부(220)는 제3 알고리즘의 항목(2.3)을 수행하여 k값을 2로 나누어 오른쪽으로 1-비트 이동시킬 수 있다(313). 즉, 도 3a에 도시된 바와 같이, 가장 오른쪽 3-비트를 표시하는 회색 음영은 이동될 수 있다.

[44] 여기서, NAF 변환 장치(200)는 NAF_3 알고리즘을 통해, 도 3a 및 도 3b와 같은 동작을 수행한 후 오른쪽으로 1-비트 이동시키는 과정에서, 현재 n 번째 비트를 본다고 가정할 때 n 번째 비트부터 $n+w-1$ 번째 비트는 모두 0으로 바뀌는 것을 알 수 있다.

[45] 또한, 처리부(220)는 상기 윈도우의 1-비트 오른쪽 이동에 연동하여, 상기 공개키의 LSB를 이동할 수 있다. 도 3a에 도시된 바와 같이, 처리부(220)는 i 값을 1 증가시킴으로써, LSB를 가리키는 화살표를 이동시킬 수 있다.

[46] 또한, 처리부(220)는 상기 1-비트 오른쪽 이동 후, 상기 윈도우에서 이탈되는 비트값 '0'의 비트를 제거할 수 있다. 즉, 처리부(220)는 이전에(단계311, 312) LSB로 지정되었던 비트를 실제로 사라지도록 제거할 수 있다. 예를 들면, 도 3a에서, 단계(313)에 도시된 바와 같이, 처리부(220)는 윈도우에서 이탈된 부분(회색 음영이 아닌 부분)의 비트값을 제거할 수 있다.

[47] 도 3b의 일례에 대해 설명하면, 먼저, 단계(311)에서 설명한 바와 같이 처리부(220)는 $w=3$ 으로 설정되면, 윈도우를 제1 세팅할 수 있다(회색 음영 부분, 321). 다음으로, 변경부(230)는 가장 오른쪽의 3-비트가 101이라면 제3

알고리즘에서의 항목(2.1)인 $k_i \leftarrow k \bmod 2^3$ 에 대하여 $k_i \leftarrow 5$ 으로 할 수 있다. 이때, 변경부(230)는 k_i 값이 항목(2.1.1)의 $k_i \geq 2^2$ 를 만족시키므로, k_i 값이 $k_i - 2^3$ 인 -3이 되고, 항목(2.1.2)의 $k \leftarrow k - k_i$ 를 수행할 수 있다. 변경부(230)는 k_i 가 -3이므로 $k \leftarrow k + 3$ 을 수행하면 가장 오른쪽의 3-비트가 000이 되도록 할 수 있는데(322), 이때 전환(carry)이 발생할 수 있다. 본 명세서에서, 전환이 발생하는 것은 0 또는 1(0 or 1)이 1 또는 0(1 or 0)으로 바뀌는 것으로 표현하지만, 이에 한정되지는 않는다.

[48] 그 다음으로, 처리부(220)는 제3 알고리즘의 항목(2.3)을 수행하여 k 값을 2로 나누어 오른쪽으로 1-비트 이동시킬 수 있다(323). 또한, 처리부(220)는 i 값을 1증가시킴으로써, LSB를 가리키는 화살표를 이동시킬 수 있다.

[49] 실시예에 따라, NAF 변환 장치(200)는 NAF_w 알고리즘에 슬라이딩 윈도우 개념을 적용하고, NAF_w 알고리즘에서 k_{i-1} 의 값이 반드시 $2^{w-1}-1$ 보다 작은 양의 홀수가 된다는 특성을 활용할 수 있다. 예를 들어, $w=3$ 인 경우, NAF 변환 장치(200)는 001 및 011인 1 및 3만이 k_{i-1} 의 값이 될 수 있다는 특성을 활용할 수 있다. 또한, NAF 변환 장치(200)는 010, 100 및 110과 같은 짝수의 경우, LSB가 1이 될 때까지 오른쪽으로 이동하여 001 및 011의 형태가 되고, 101 및 111의 경우에는 전환(carry)가 발생하여 1000이 된 다음 LSB가 1이 될 때까지 오른쪽으로 이동하여 001의 형태가 될 수 있다는 특성을 활용할 수 있다. 따라서, NAF 변환 장치(200)는 k_{i-1} 의 값이 반드시 $2^{w-1}-1$ 보다 작은 양의 홀수가 되는 특성을 활용할 수 있다.

[50] 도 4a 및 도 4b는 본 발명의 일실시예에 따른 고속의 NAF 변환 장치를 이용한 구현예를 설명하기 위한 도면이다.

[51] 도 4a 및 도 4b에 도시된 바와 같이, 처리부(220)는 NAF_w 에 대해 $w=3$ 으로 설정되면, NAF_3 알고리즘에서 k 가 홀수인 경우(즉, LSB가 1인 경우)의 윈도우를, 제1 세팅할 수 있다(회색 음영 부분, 411, 421).

[52] 판단부(250)는 산출된 상기 환산값이 선정된 기준값을 넘는지를 판단할 수 있다. 즉, 연산부(240)에 의해 산출된 환산값에 대하여, 판단부(250)는 기준값을 기준으로 판단할 수 있다. 예를 들면, 판단부(250)는 기준값 2^{w-1} 을 기준으로, 환산값이 기준값을 넘는지를 판단할 수 있다.

[53] 이때, 처리부(220)는 상기 판단 결과, 넘지 않는 경우, 상기 윈도우를 상기 ω-비트 오른쪽 이동시켜, 상기 윈도우의 세팅 위치를 상기 타깃 비트로 조정할 수 있다. 즉, 처리부(220)는 기준값 2^{w-1}보다 환산값이 작은 경우, 제1 세팅된 윈도우 크기(ω)만큼을, ω 비트만큼 오른쪽 이동시킬 수 있다. 도 4a에 도시된

바와 같이, 처리부(220)는 $k_i < 2^{w-1}$ 인 경우 오른쪽으로 ω-비트 이동시킬 수 있다(413). 단계(413)에서, ω-비트 이동한 후, NAF 변환 장치(200)는 윈도우가 이동한 비트의 값을 체크할 수 있다. 즉, 처리부(220)는 윈도우 크기가 3으로 제1 세팅된 경우, 윈도우 크기가 3인 윈도우를 3-비트만큼 오른쪽 이동시킬 수 있다. 3-비트 이동한 후, NAF 변환 장치(200)는 윈도우에 포함된 비트에 대하여, 0 또는 1 중 하나를 비트값으로 체크할 수 있다. 이 과정에서, 처리부(220)는 NAF_3 알고리즘의 연산 과정을 생략할 수 있다.

[54] 또한, 처리부(220)는 상기 판단 결과, 넘는 경우, 상기 타깃 비트의 비트값을 전환(carry)하고, 상기 윈도우를 상기 ω-비트 오른쪽 이동시켜, 상기 윈도우의 세팅 위치를 상기 타깃 비트로 조정할 수 있다. 즉, 처리부(220)는 전환(carry)을 발생시킴에 따라, $k = k - k_i$ 대신 $k = k + 2^w$ 를 수행할 수 있다(422). 또한, 처리부(220)는 비트값을 전환한 후, 도 4b에 도시된 바와 같이, 윈도우 크기가 ω인 윈도우를 ω 비트만큼 오른쪽 이동시킬 수 있다(423). 단계(423)에서, ω-비트 이동한 후, NAF 변환 장치(200)는 윈도우가 이동한 비트의 값을 체크할 수 있다. 예를 들면, 처리부(220)는 제1 세팅된 윈도우 크기가 3인 윈도우를 3-비트만큼 오른쪽 이동시킬 수 있다. 3-비트 이동한 후, NAF 변환 장치(200)는 윈도우에 포함된 비트에 대하여, 0 또는 1 중 하나를 비트값으로 체크할 수 있다.

[55] 또한, 처리부(220)는 상기 윈도우의 ω-비트 오른쪽 이동에 연동하여, 상기 공개키의 LSB를 상기 타깃 비트로 조정할 수 있다. 즉, 도 4a 및 도 4b에 도시된 바와 같이, 처리부(220)는 오른쪽으로 ω-비트 이동을 하여 $n+w$ 번째 비트로 이동함으로써, LSB를 가리키는 화살표를 이동시킬 수 있다.

[56] 이러한, NAF 변환 장치(200)는 $\lceil l/w \rceil$ 번의 비교 연산 과정을 수행함으로써, l 번의 비트 비교 연산 과정을 수행하는 NAF_w 알고리즘의 연산 처리 속도를 최적화 할 수 있다.

[57] 또한, NAF 변환 장치(200)는 NAF 변환 시, l (l 은 양의 정수 k 의 이진수 변환에 따른 비트 수)이 증가하거나 홀수인 경우를 발생시키는 패턴이 많이 존재하는 경우에 NAF 변환 시간을 단축할 수 있다(l 은 양의 정수 k 의 이진수 변환에 따른 비트 수, w는 윈도우 크기).

[58] 도 5는 본 발명의 일실시예에 따른 고속의 NAF 변환 장치를 구현하기 위한 알고리즘을 도시한 도면이다.

[59] 도 5에 도시된 바와 같이, NAF 변환 장치(200)는 제7 알고리즘($SNAF_w$ 알고리즘)으로서 수도 코드(pseudo code)로 표현될 수 있다. NAF 변환 장치(200)는 제7 알고리즘의 항목(2)에서의 While문의 수행 조건을 $k \geq 1$으로 설정하지 않을 수 있다. 그 이유는, k_{i-1} 의 값이 반드시 $2^{w-1}-1$ 보다 작은 양의 홀수가 되므로 NAF 변환 장치(200)는 와일(While)문의 중단 여부를

항목(2.3.1)에서만 판단하도록 할 수 있다. 항목(2.1)의 와일(While)문은 연속된 0을 빠르게 처리하기 위한 코드일 수 있다. 와일(While)문을 빠져 나와 항목(2.2)에 왔을 때는 k 는 반드시 홀수인 정수가 될 수 있다.

[60] 제7 알고리즘에서의 항목(2.3.2)는 도 4a의 동작 과정을 코드로 표현한 것이다. 이때, NAF 변환 장치(200)는 제3 알고리즘의 항목(2.1.2)과 '0임을 확신할 수 있는 w-비트에 대한 1-비트씩 이동(shift) 하는 과정'을 생략할 수 있다. 즉, NAF 변환 장치(200)는 i 를 w만큼 증가시키고 d를 w만큼 오른쪽으로 이동할 수 있다.

[61] 제7 알고리즘에서의 항목(2.4)는 도 4b의 동작 과정을 코드로 표현한 것이다. 이 경우, NAF 변환 장치(200)는 알고리즘에서 전환(carry)이 발생하기 때문에 $k \leftarrow k+2^w$ 를 수행할 수 있다. NAF 변환 장치(200)는 나머지 부분에 대하여 항목(2.3.2)와 같이 수행할 수 있다.

[62] 도 6 내지 도 11은 본 발명의 일실시예에 따른 고속의 NAF 변환 장치를 구현하기 위한 알고리즘의 성능 평가를 설명하기 위한 도면이다.

[63] $SNAF_w$ 알고리즘(제7 알고리즘)의 성능 평가를 도시한 도면이다. $SNAF_w$ 알고리즘에 대한 성능 평가는 입력 값 k 와 w에 대해 $NAF_w(k)$를 계산하는 데에 걸리는 사이클 카운터(Cycle Counter)를 NAF_w 알고리즘과 비교하는 방식으로 평가할 수 있다. NAF_w 알고리즘과 $SNAF_w$ 알고리즘은 $k_{(2)}$의 길이 l 이 늘어나면 전체 사이클 카운터가 증가할 수 있다. 두 알고리즘은 오른쪽 이동(right shift)하는 과정에서 홀수인 경우가 증가할수록 사이클 카운터가 증가할 수 있다.

[64] 예를 들면, $k_{(2)}$의 길이가 8이고 w가 2일 때, 10000000_2은 홀수인 경우가 1번 발생하지만 10010101_2와 10110011_2 은 홀수인 경우가 4번 발생하기 때문에 사이클 카운터가 더 클 수 있다. 10110011_2 이 홀수인 경우가 4번이 되는 이유는 도 6에 도시된 바와 같이, $k_i \geq 2^{w-1}$ 일 때 발생하는 전환(Carry)에 의해 w-비트 앞에 1이 발생하기 때문일 수 있다.

[65] 따라서, 본원의 성능 평가에서는 $k_{(2)}$의 길이 l 에 대한 사이클 카운터 비교를 할 수 있다. 그뿐만 아니라, 본원의 성능 평가에서는 홀수의 경우가 발생하는 횟수에 대한 비교도 포함할 수 있다. 즉, 본원의 성능 평가에서는 0과 1로 조합되는 0,1,01,001,011,101 등과 같은 패턴들의 반복 횟수에 대해서 w 크기가 2인 경우와 3인 경우를 저성능 8-비트 마이크로프로세서 아트메가128(ATmega128)에서 구현하고 사이클 카운터를 측정하여 성능 평가를 수행할 수 있다.

[66] $k_{(2)}$의 이진수 비트수는 RSA에서 1024-비트 이상, ECC에서 160-비트 이상이 사용되며, 이러한 비트들은 0과 1에 의한 다양한 패턴들의 반복이라고 볼 수

있다. 따라서, 도 7에 도시된 바와 같이, 3-비트 이하에서 발생 가능한 다양한 반복 패턴에 따른 NAF_2 알고리즘 대비 $SNAF_2$ 알고리즘의 평균 및 편차 이득을 요약하여 도시한다. NAF_2 알고리즘에 비해 $SNAF_2$ 알고리즘의 평균 사이클 카운터가 약 20% 정도 감소하고 편차 또한 30% 이상 감소한 것을 확인할 수 있다. 특히, $SNAF_2$ 알고리즘은 NAF_2 알고리즘에 비해 편차에서 우수한 성능을 나타낼 수 있다. 이는, $SNAF_w$ 알고리즘이 $k_{(2)}$의 길이 l 이 늘어나더라도 사이클 카운터 증가량이 NAF_w 알고리즘 보다 작아서 다양한 입력 패턴에 대해 일정한 성능을 나타낼 수 있음을 의미할 수 있다.

[67] 또한, 본원에서는 두 알고리즘의 성능에 영향을 미치는 주요한 반복 패턴인 1과 01 비트 패턴에서의 반복 횟수에 따른 두 알고리즘의 사이클 카운터를 상세히 분석한 결과를 도 8 및 도 9에 도시한다. 도 8 및 도 9에 도시된 바와 같이, $SNAF_2$ 알고리즘은 NAF_2 알고리즘에 비해 각각 평균 이득이 18.6%와 19.4%, 편차 이득이 31.3%와 31.8% 성능 향상을 나타낼 수 있다. 또한, 도 10 및 도 11에 도시된 바와 같이, $SNAF_3$ 알고리즘은 NAF_3 알고리즘에 비해 각각 평균 이득이 11.8%와 14.3%, 편차 이득이 26.7%와 29.9% 성능 향상을 나타낼 수 있다.

[68] 도 11에 도시된 바와 같이, 반복 회수가 2에서 3으로(또는, 5에서 6으로) 증가함에 따라 클락 카운터(Clock Counter)가 증가하지 않는 이유는 w-비트 이동(shift) 횟수가 동일하기 때문일 수 있다. 예를 들어, $w=3$ 일 때, 0101은 전환(carry)이 발생해 1000로 변환된 후 3 오른쪽 이동(right shift)을 한 다음 0001을 처리하기 때문에 한 번의 w-비트 이동(shift)이 발생할 수 있다. 010101도 전환(carry)이 발생해 011000으로 변환된 후 오른쪽으로 3-비트 이동(shift)을 한 다음 011을 처리하기 때문에 이동(shift)이 한 번 발생할 수 있다. 반면에, 01010101은 전환(carry)이 발생하여 01011000으로 변환된 후 오른쪽으로 3-비트 이동(shift)을 한 다음 01011이 되고, 다시 01000으로 변환된 후 오른쪽으로 3-비트 이동(shift)을 하여 01을 처리하기 때문에 이동(shift)이 두 번 발생할 수 있다. 따라서, 01의 연속 횟수가 2번일 때와 3번일 때의 사이클 카운터는 동일하지만 4번일 때는 사이클 카운터가 증가하게 될 수 있다.

[69] 도 12는 본 발명의 일실시예에 따른 고속의 NAF 변환 방법을 구체적으로 도시한 작업 흐름도이다.

[70] 우선 본 실시예에 따른 고속의 NAF 변환 방법은 상술한 NAF 변환 장치(200)에 의해 수행될 수 있다.

[71] 먼저, NAF 변환 장치(200)는 윈도우의 값을 설정한다(1210). 즉, 단계(1210)는 양의 정수인 윈도우의 값을 설정하는 과정일 수 있다. 예를 들면, NAF 변환 장치(200)는 2 이상의 정수 중 적어도 하나의 값을 윈도우의 값으로 설정할 수

있다.

[72] 다음으로, NAF 변환 장치(200)는 공개키의 LSB(Least Significant Bit)를 기준으로, 상기 설정된 값을, 길이로 갖는 윈도우를, 상기 공개키의 적어도 일부에 제1 세팅한다(1220). 즉, 단계(1220)는 제1 세팅으로서, 단계(1210)에서 설정된 값만큼 공개키의 일부분에 윈도우를 세팅할 수 있다. 예를 들면, 윈도우의 값이 3으로 설정된 경우, NAF 변환 장치(200)는 공개키 중 LSB로부터 3 비트만큼을 제1 세팅할 수 있다.

[73] 다음으로, NAF 변환 장치(200)는 상기 공개키 내 암호화 대상이 되는 타깃 비트가, 상기 윈도우의 밖에 있는 경우, 상기 타깃 비트가 포함되도록 상기 윈도우를, ω-비트 오른쪽 이동(ω-bit right shift)(상기 ω은 자연수)시켜, 상기 윈도우를 제2 세팅한다(1230). 즉, NAF 변환 장치(200)는 타깃 비트를 윈도우에 포함시키기 위하여, 윈도우의 값인 ω-비트만큼 이동시켜 윈도우를 제2 세팅할 수 있다. 예를 들면, 윈도우의 값이 3으로 설정된 경우, NAF 변환 장치(200)는 윈도우를 3-비트 이동시켜 타깃 비트가 포함되도록 세팅할 수 있다.

[74] 실시예에 따라, NAF 변환 장치(200)는 상기 제1 세팅된 윈도우에 포함되는, 상기 공개키 내 비트를 모두 '0'으로 변경할 수 있다. 즉, NAF 변환 장치(200)는 제1 세팅된 비트에 대하여 모두 '0'으로 변경할 수 있다.

[75] 실시예에 따라, NAF 변환 장치(200)는 상기 제1 세팅된 윈도우에 포함되는, 상기 공개키 내 비트에 대한 환산값을 산출할 수 있다. 예를 들면, NAF 변환 장치(200)는 가장 오른쪽의 3-비트가 011이라면 제3 알고리즘에서의 항목(2.1)인 $k_i \leftarrow k \bmod 2^3$ 에 대하여 $k_i \leftarrow 3$을 산출할 수 있다.

[76] 상기 '0'으로 변경하는 단계는, 산출된 상기 환산값이 선정된 기준값을 넘는지를 확인하여, 정해진 수를 가산 또는 감산하여, 상기 환산값을 '0'을 만드는 과정일 수 있다. 예를 들어, 기준값이 2^2로 선정된다면, NAF 변환 장치(200)는 환산값 k_i 가 $k_i \geq 2^2$ 인지를 확인할 수 있다. NAF 변환 장치(200)는 k_i 값이 제3 알고리즘에서의 항목(2.1.1)의 $k_i \geq 2^2$를 만족시키지 않으므로, $k \leftarrow k - k_i$ 를 수행할 수 있다. 이때, NAF 변환 장치(200)는 k_i 가 3이므로 $k \leftarrow k-3$을 수행하여 가장 오른쪽의 3-비트가 000이 되도록 할 수 있다.

[77] 이때, 단계(1230)는 상기 ω을 '1'로 결정하고, 상기 윈도우를, 1-비트 오른쪽 이동(1-bit right shift)시키되, 상기 타깃 비트가 상기 윈도우에 포함될 때까지 상기 1-비트 오른쪽 이동을 반복하여 상기 제2 세팅하는 과정일 수 있다. 예를 들면, NAF 변환 장치(200)는 제3 알고리즘의 항목(2.3)을 수행하여 k 값을 2로 나누어 오른쪽으로 1-비트 이동시킬 수 있다.

[78] 또한, 단계(1230)는 상기 1-비트 오른쪽 이동 후, 상기 윈도우에서 이탈되는 비트값 '0'의 비트를 제거하는 과정일 수 있다. 즉, NAF 변환 장치(200)는 이전에 LSB로 지정되었던 비트를 실제로 사라지도록 제거할 수 있다. 예를 들면, NAF

변환 장치(200)는 윈도우에서 이탈된 부분의 비트값을 제거할 수 있다.

[79] 또한, 단계(1230)는 상기 윈도우의 1-비트 오른쪽 이동에 연동하여, 상기 공개키의 LSB를 이동하는 과정일 수 있다. 예를 들면, NAF 변환 장치(200)는 i 값을 1 증가시킴으로써, LSB를 이동할 수 있다.

[80] 실시예에 따라, NAF 변환 장치(200)는 산출된 상기 환산값이 선정된 기준값을 넘는지를 판단할 수 있다. 즉, 상기 환산값을 산출하는 단계에서 산출된 환산값에 대하여, NAF 변환 장치(200)는 기준값을 기준으로 판단할 수 있다. 예를 들면, NAF 변환 장치(200)는 기준값 2^{w-1}을 기준으로, 환산값이 기준값을 넘는지를 판단할 수 있다.

[81] 실시예에 따라, NAF 변환 장치(200)는 상기 판단 결과, 넘지 않는 경우, 상기 윈도우를 상기 ω-비트 오른쪽 이동시켜, 상기 윈도우의 세팅 위치를 상기 타깃 비트로 조정할 수 있다. 즉, NAF 변환 장치(200)는 기준값 2^{w-1} 보다 환산값이 작은 경우, 제1 세팅된 윈도우 크기(ω)만큼을 ω 비트만큼 오른쪽 이동시킬 수 있다. NAF 변환 장치(200)는 $k_i < 2^{w-1}$ 인 경우 오른쪽으로 ω-비트 이동시킬 수 있다. ω-비트 이동한 후, NAF 변환 장치(200)는 윈도우가 이동한 비트의 값을 체크할 수 있게 된다. 예를 들면, NAF 변환 장치(200)는 윈도우 크기가 3으로 제1 세팅된 경우, 윈도우 크기가 3인 윈도우를 3-비트만큼 오른쪽 이동시킬 수 있다. 3-비트 이동한 후, NAF 변환 장치(200)는 윈도우에 포함된 비트에 대하여, 0 또는 1 중 하나를 비트값으로 체크할 수 있다. 이 과정에서, NAF 변환 장치(200)는 NAF_3 알고리즘의 연산 과정을 생략할 수 있다.

[82] 실시예에 따라, NAF 변환 장치(200)는 상기 판단 결과, 넘는 경우, 상기 타깃 비트의 비트값을 전환(carry)하고, 상기 윈도우를 상기 ω-비트 오른쪽 이동시켜, 상기 윈도우의 세팅 위치를 상기 타깃 비트로 조정할 수 있다. 즉, NAF 변환 장치(200)는 전환(carry)을 발생시킴에 따라, $k = k - k_i$ 대신 $k = k + 2^w$ 를 수행할 수 있다. 또한, NAF 변환 장치(200)는 비트값을 전환한 후, 윈도우 크기가 ω인 윈도우를 ω 비트만큼 오른쪽 이동시킬 수 있다. ω-비트 이동한 후, NAF 변환 장치(200)는 윈도우가 이동한 비트의 값을 체크할 수 있다. 예를 들면, NAF 변환 장치(200)는 제1 세팅된 윈도우 크기가 3인 윈도우를 3-비트만큼 오른쪽 이동시킬 수 있다. 3-비트 이동한 후, NAF 변환 장치(200)는 윈도우에 포함된 비트에 대하여, 0 또는 1 중 하나를 비트값으로 체크할 수 있다.

[83] 실시예에 따라, NAF 변환 장치(200)는 상기 윈도우의 ω-비트 오른쪽 이동에 연동하여, 상기 공개키의 LSB를 상기 타깃 비트로 조정할 수 있다. 예를 들면, NAF 변환 장치(200)는 오른쪽으로 ω-비트 이동을 하여 $n+w$번째 비트로 이동함으로써, LSB를 가리키는 화살표를 이동시킬 수 있다.

[84] 이러한, NAF 변환 방법은 $\lceil l/w \rceil$ 번의 비교 연산 과정을 수행함으로써, l 번의

비트 비교 연산 과정을 수행하는 NAF_w 알고리즘의 연산 처리 속도를 최적화 할 수 있다(i 은 양의 정수 k 의 이진수 변환에 따른 비트 수, w는 윈도우 크기).

[85] 또한, NAF 변환 방법은 NAF 변환 시, l (l 은 양의 정수 k 의 이진수 변환에 따른 비트 수)이 증가하거나 홀수인 경우를 발생시키는 패턴이 많이 존재하는 경우에 NAF 변환 시간을 단축할 수 있다.

[86] 본 발명의 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(magnetic media), CD-ROM, DVD와 같은 광기록 매체(optical media), 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical media), 및 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 상기된 하드웨어 장치는 실시예의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

[87] 이상과 같이 실시예들이 비록 한정된 실시예와 도면에 의해 설명되었으나, 해당 기술분야에서 통상의 지식을 가진 자라면 상기의 기재로부터 다양한 수정 및 변형이 가능하다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다.

[88] 그러므로, 다른 구현들, 다른 실시예들 및 특허청구범위와 균등한 것들도 후술하는 특허청구범위의 범위에 속한다.

청구범위

[청구항 1] 윈도우의 값을 설정하는 설정부;
공개키의 LSB(Least Significant Bit)를 기준으로, 상기 설정된 값을, 길이로 갖는 윈도우를, 상기 공개키의 적어도 일부에 제1 세팅하고, 상기 공개키 내 암호화 대상이 되는 타깃 비트가, 상기 윈도우의 밖에 있는 경우, 상기 타깃 비트가 포함되도록 상기 윈도우를, ω-비트 오른쪽 이동(ω-bit right shift)(상기 ω은 자연수)시켜, 상기 윈도우를 제2 세팅하는 처리부
를 포함하는 고속의 NAF 변환 장치.

[청구항 2] 제1항에 있어서,
상기 제1 세팅된 윈도우에 포함되는, 상기 공개키 내 비트를 모두 '0'으로 변경하는 변경부
를 더 포함하고,
상기 처리부는,
상기 ω을 '1'로 결정하고, 상기 윈도우를, 1-비트 오른쪽 이동(1-bit right shift)시키되, 상기 타깃 비트가 상기 윈도우에 포함될 때까지 상기 1-비트 오른쪽 이동을 반복하여 상기 제2 세팅하는
고속의 NAF 변환 장치.

[청구항 3] 제2항에 있어서,
상기 처리부는,
상기 1-비트 오른쪽 이동 후, 상기 윈도우에서 이탈되는 비트값 '0'의 비트를 제거하는
고속의 NAF 변환 장치.

[청구항 4] 제2항에 있어서,
상기 처리부는,
상기 윈도우의 1-비트 오른쪽 이동에 연동하여, 상기 공개키의 LSB를 이동하는
고속의 NAF 변환 장치.

[청구항 5] 제2항에 있어서,
상기 제1 세팅된 윈도우에 포함되는, 상기 공개키 내 비트에 대한 환산값을 산출하는 연산부
를 더 포함하고,
상기 변경부는,
산출된 상기 환산값이 선정된 기준값을 넘는지를 확인하여, 정해진 수를 가산 또는 감산하여, 상기 환산값을 '0'을 만드는
고속의 NAF 변환 장치.

[청구항 6] 제1항에 있어서,
상기 제1 세팅된 윈도우에 포함되는, 상기 공개키 내 비트에 대한

환산값을 산출하는 연산부; 및
산출된 상기 환산값이 선정된 기준값을 넘는지를 판단하는 판단부
를 더 포함하고,
상기 처리부는,
상기 판단 결과, 넘지 않는 경우,
상기 윈도우를 상기 ω-비트 오른쪽 이동시켜, 상기 윈도우의 세팅 위치를 상기 타깃 비트로 조정하는
고속의 NAF 변환 장치.

[청구항 7] 제6항에 있어서,
상기 판단 결과, 넘는 경우,
상기 처리부는,
상기 타깃 비트의 비트값을 전환(carry)하고, 상기 윈도우를 상기 ω-비트 오른쪽 이동시켜, 상기 윈도우의 세팅 위치를 상기 타깃 비트로 조정하는
고속의 NAF 변환 장치.

[청구항 8] 제6항 또는 제7항에 있어서,
상기 처리부는,
상기 윈도우의 ω-비트 오른쪽 이동에 연동하여, 상기 공개키의 LSB를 상기 타깃 비트로 조정하는
고속의 NAF 변환 장치.

[청구항 9] 윈도우의 값을 설정하는 단계;
공개키의 LSB(Least Significant Bit)를 기준으로, 상기 설정된 값을, 길이로 갖는 윈도우를, 상기 공개키의 적어도 일부에 제1 세팅하는 단계; 및
상기 공개키 내 암호화 대상이 되는 타깃 비트가, 상기 윈도우의 밖에 있는 경우,
상기 타깃 비트가 포함되도록 상기 윈도우를, ω-비트 오른쪽 이동(ω-bit right shift)(상기 ω은 자연수)시켜, 상기 윈도우를 제2 세팅하는 단계
를 포함하는 고속의 NAF 변환 방법.

[청구항 10] 제9항에 있어서,
상기 제1 세팅된 윈도우에 포함되는, 상기 공개키 내 비트를 모두 '0'으로 변경하는 단계
를 더 포함하고,
상기 제2 세팅하는 단계는,
상기 ω을 '1'로 결정하는 단계; 및
상기 윈도우를, 1-비트 오른쪽 이동(1-bit right shift)시키되, 상기 타깃 비트가 상기 윈도우에 포함될 때까지 상기 1-비트 오른쪽 이동을 반복하여 상기 제2 세팅하는 단계
를 포함하는 고속의 NAF 변환 방법.

[청구항 11] 제10항에 있어서,

상기 제2 세팅하는 단계는,
상기 1-비트 오른쪽 이동 후, 상기 윈도우에서 이탈되는 비트값 '0'의 비트를 제거하는 단계
를 더 포함하는 고속의 NAF 변환 방법.

[청구항 12] 제10항에 있어서,
상기 제2 세팅하는 단계는,
상기 윈도우의 1-비트 오른쪽 이동에 연동하여, 상기 공개키의 LSB를 이동하는 단계
를 더 포함하는 고속의 NAF 변환 방법.

[청구항 13] 제10항에 있어서,
상기 NAF 변환 방법은,
상기 제1 세팅된 윈도우에 포함되는, 상기 공개키 내 비트에 대한 환산값을 산출하는 단계
를 더 포함하고,
상기 '0'으로 변경하는 단계는,
산출된 상기 환산값이 선정된 기준값을 넘는지를 확인하여, 정해진 수를 가산 또는 감산하여, 상기 환산값을 '0'을 만드는 단계
를 포함하는 고속의 NAF 변환 방법.

[청구항 14] 제9항에 있어서,
상기 제1 세팅된 윈도우에 포함되는, 상기 공개키 내 비트에 대한 환산값을 산출하는 단계;
산출된 상기 환산값이 선정된 기준값을 넘는지를 판단하는 단계; 및
상기 판단 결과, 넘지 않는 경우,
상기 윈도우를 상기 ω-비트 오른쪽 이동시켜, 상기 윈도우의 세팅 위치를 상기 타깃 비트로 조정하는 단계
를 더 포함하는 고속의 NAF 변환 방법.

[청구항 15] 제14항에 있어서,
상기 판단 결과, 넘는 경우,
상기 타깃 비트의 비트값을 전환(carry)하는 단계; 및
상기 윈도우를 상기 ω-비트 오른쪽 이동시켜, 상기 윈도우의 세팅 위치를 상기 타깃 비트로 조정하는 단계
를 더 포함하는 고속의 NAF 변환 방법.

[청구항 16] 제14항 또는 제15항에 있어서,
상기 윈도우의 ω-비트 오른쪽 이동에 연동하여, 상기 공개키의 LSB를 상기 타깃 비트로 조정하는 단계
를 더 포함하는 고속의 NAF 변환 방법.

[도1a]

입 력 : $k=(k_{l-1},\ldots,k_1,k_0)_2, a, n$

출 력 : $f=a^k \bmod n$

1. $c \leftarrow 0, f \leftarrow 1$
2. For i from $l-1$ downto 0 do
   2.1 $c \leftarrow 2\times c$, $f \leftarrow (f\times f) \bmod n$
   2.2 If $k_i = 1$ then
   $c \leftarrow c+1, f \leftarrow (f\times a) \bmod n$
3. Return(f)

[도1b]

$a^k \bmod n$, *where* $a=7,\ k=560=1000110000, n=561$

i	9	8	7	6	5	4	3	2	1	0
k_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
f	7	49	157	526	160	241	298	166	67	1

[도1c]

입 력 : $k=(k_{l-1},\ldots,k_1,k_0)_2, P \subset E(F_q)$

출 력 : kP

1. $Q \leftarrow \infty$
2. For i from $l-1$ downto 0 do
   2.1 $Q \leftarrow 2Q$
   2.2 If $k_i = 1$ then $Q \leftarrow Q+P$
3. Return(Q)

[도1d]

입 력 : 양의 정수 k , w $(w \geq 2)$

출 력 : $NAF_w(k) = k_{l-1}, k_{l-2}, \ldots, k_1, k_0$

1. $i \leftarrow 0$
2. While $k \geq 1$ do
   2.1 If k is odd then $k_i \leftarrow k \bmod 2^w$
   2.1.1 If $k_i \geq 2^{w-1}$ then $k_i \leftarrow k_i - 2^w$
   2.1.2 $k \leftarrow k-k_i$
   2.2 Else $k_i \leftarrow 0$
   2.3 $k \leftarrow k/2, i \leftarrow i+1$
3. Return($k_{l-1}, k_{l-2}, \ldots, k_1, k_0$)

[도1e]

$k=103, w=2$ $k_{(2)}=1100111$								
i	0	1	2	3	4	5	6	7
k	103	52	26	13	6	3	2	1
$k_{(2)}$	1100111	110100	11010	1101	110	11	10	1
k_i	-1	0	0	1	0	-1	0	1

[도1f]

입 력 : $k=(k_{l-1},...,k_1,k_0)_2, a, a^{-1}, n$

출 력 : $f=a^k \bmod n$

1. Algorithm 3로부터 얻은
   $NAF_w(k) = k_{l-1}, k_{l-2}, \ldots, k_1, k_0$
2. $c \leftarrow 0, f \leftarrow 1$
3. For i from $l-1$ downto 0 do
   2.1 $c \leftarrow 2 \times c$, $f \leftarrow (f \times f) \bmod n$
   2.2 If $k_i = 1$ then
   $c \leftarrow c+1, f \leftarrow (f \times a) \bmod n$
   2.3 Else if $k_i = -1$ then
   $c \leftarrow c-1, f \leftarrow (f \times a^{-1}) \bmod n$
4. Return(f)

[도1g]

$a^k \bmod n$, $where\ a=7, k=560=1000110000, n=561$ $NAF_w(k) = 10010-10000$ $a^{-1}=481$										
i	9	8	7	6	5	4	3	2	1	0
k_i	1	0	0	1	0	-1	0	0	0	0
c	1	2	4	9	18	35	70	140	280	560
f	7	49	157	316	559	241	298	166	67	1

[도1h]

입 력 : 양의 정수 $k, P \in E(F_q), w$ $(w \geq 2)$

출 력 : $Q=kP$

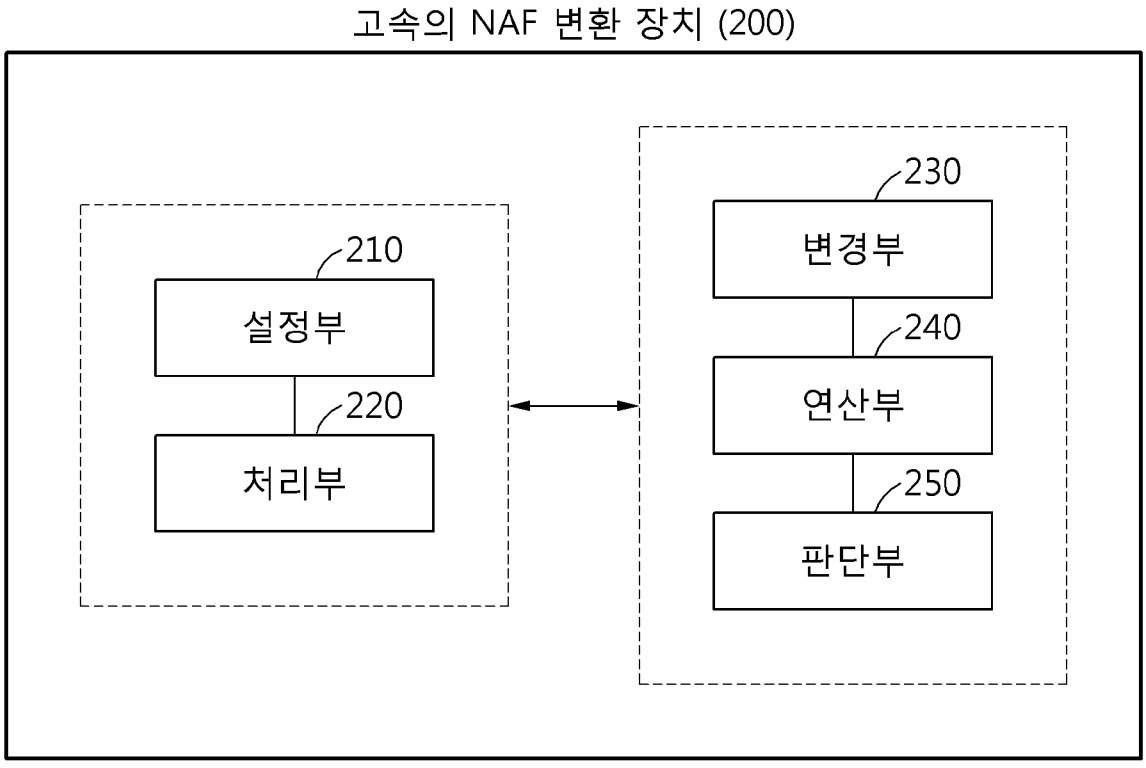
1. Algorithm 3로부터 얻은
   $NAF_w(k) = k_{l-1}, k_{l-2}, \ldots, k_1, k_0$
2. $P_i = iP, \{i \in 1, 3, \ldots, 2^{w-1}-1\}$
3. $Q \leftarrow \infty$
4. For i from $l-1$ downto 0 do
   2.1 $Q \leftarrow 2Q$
   2.2 If $k_i \neq 0$ then
   If $k_i > 0$ then $Q \leftarrow Q + P_{k_i}$
   Else $Q \leftarrow Q - P_{k_i}$
5. Return(Q)

[도1i]

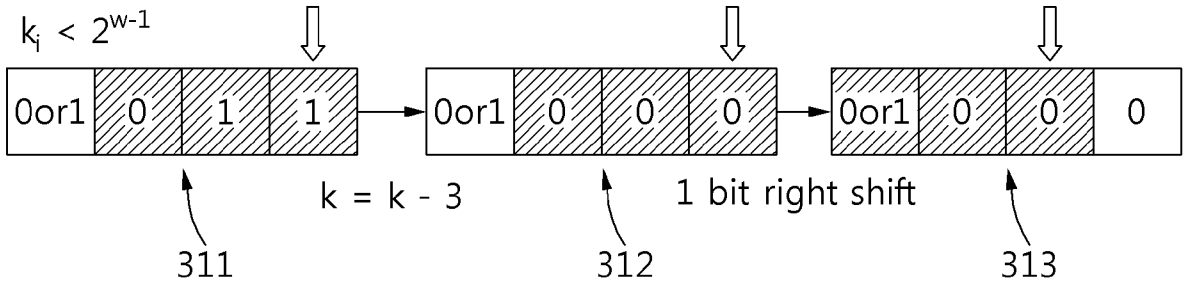
입 력 : 양의 정수 $k, P \in E(F_q), w \ (w \geq 2)$
출 력 : $Q = kP$

1. Algorithm 3로부터 얻은
   $NAF_w(k) = k_{l-1}, k_{l-2}, \ldots, k_1, k_0$
2. $P_i = iP, \{i \in 1, 3, \ldots, 2(2^w - (-1)^w)/3 - 1\}$
3. $Q \leftarrow \infty$, $i \leftarrow l-1$
4. While $i \geq 0$ do
   4.1 If $k_i = 0$ then $t \leftarrow 1, u \leftarrow 0$
   4.2 Else find the largest $t < w$
   such that $u \leftarrow (k_i, \ldots, k_{i-t+1})$ is odd
   4.3 $Q \leftarrow 2^t Q$
   4.4 If $u > 0$ then $Q \leftarrow Q + P_u$
   else if $u < 0$ then $Q \leftarrow Q - P_{-u}$
   4.5 $i \leftarrow i - t$
5. Return(Q)

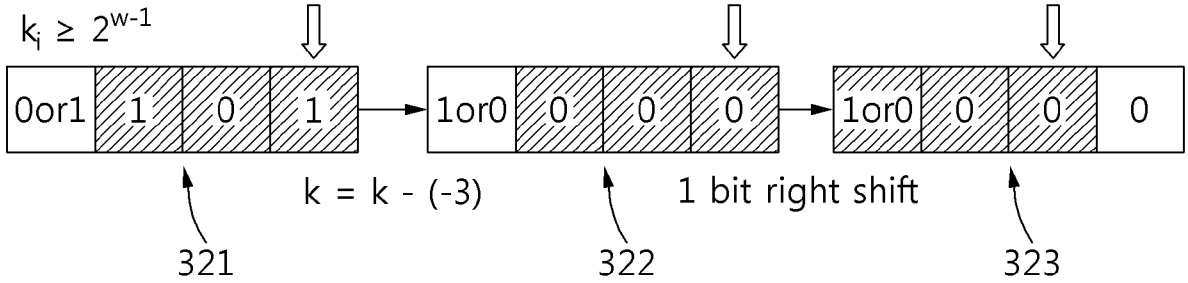
[도2]



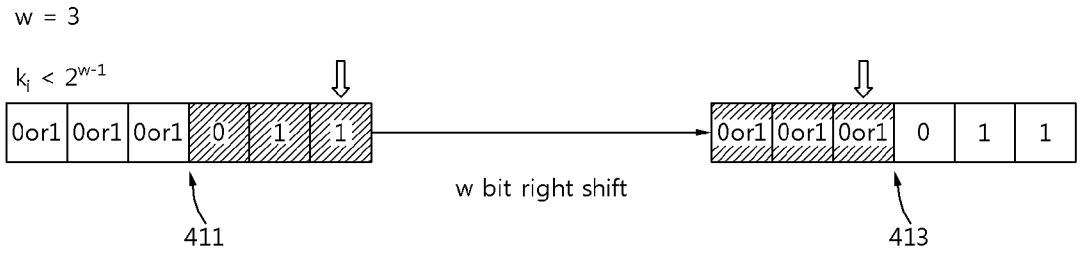
[도3a]



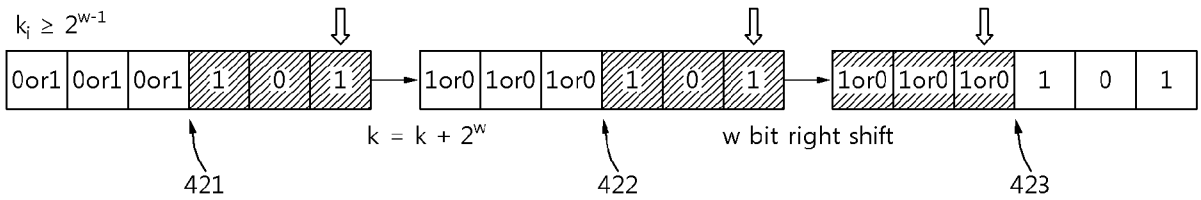
[도3b]



[도4a]



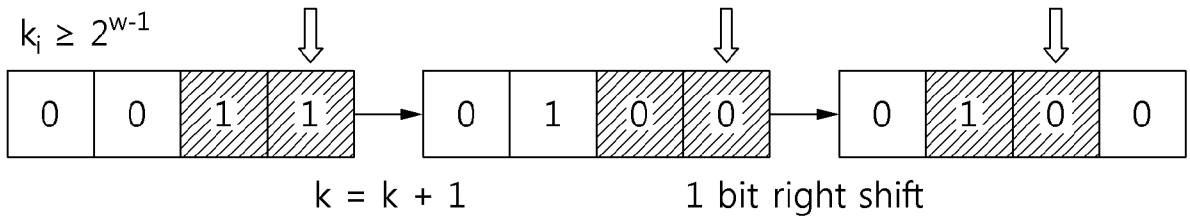
[도4b]



[도5]

입 력 : 양의 정수 k , w $(w \geq 2)$
출 력 : $NAF_w(k) = k_{l-1}, k_{l-2}, \ldots, k_1, k_0$
1. $i \leftarrow 0$, x
2. While *true* do
  2.1 While k is even then
    $k_i \leftarrow 0, k \leftarrow k \gg 1, i \leftarrow i+1$
  2.2 $x \leftarrow k\&(2^w - 1)$
  2.3 If $x < 2^{w-1}$ then
    2.3.1 If $x = k$ then
      $k_i \leftarrow x, i \leftarrow i+1, break$
    2.3.2 Else $k_{i+w-1} \leftarrow 0, \ldots, k_{i+1} \leftarrow 0$
      $k_i \leftarrow x, k \leftarrow k \gg w, i \leftarrow i+w$
  2.4 Else $k_{i+w-1} \leftarrow 0, \ldots, k_{i+1} \leftarrow 0$
    $, k_i \leftarrow x - 2^w, k \leftarrow k + 2^w$
    $, k \leftarrow k \gg w, i \leftarrow i+w$
3. Return$(k_{l-1}, k_{l-2}, \ldots, k_1, k_0)$

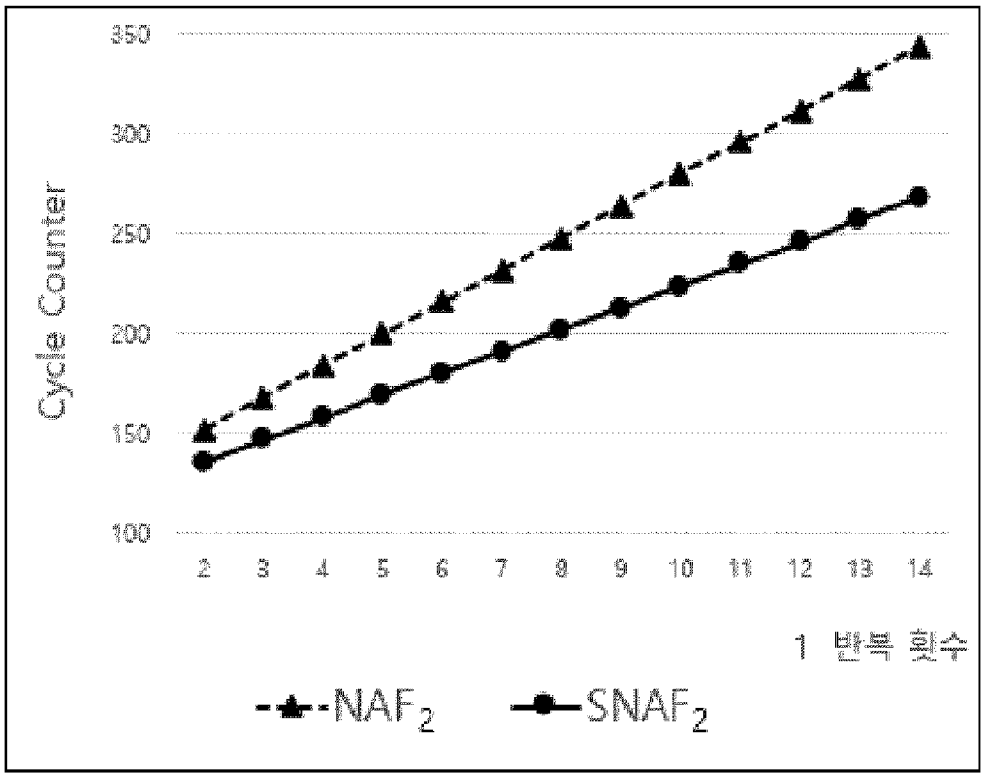
[도6]



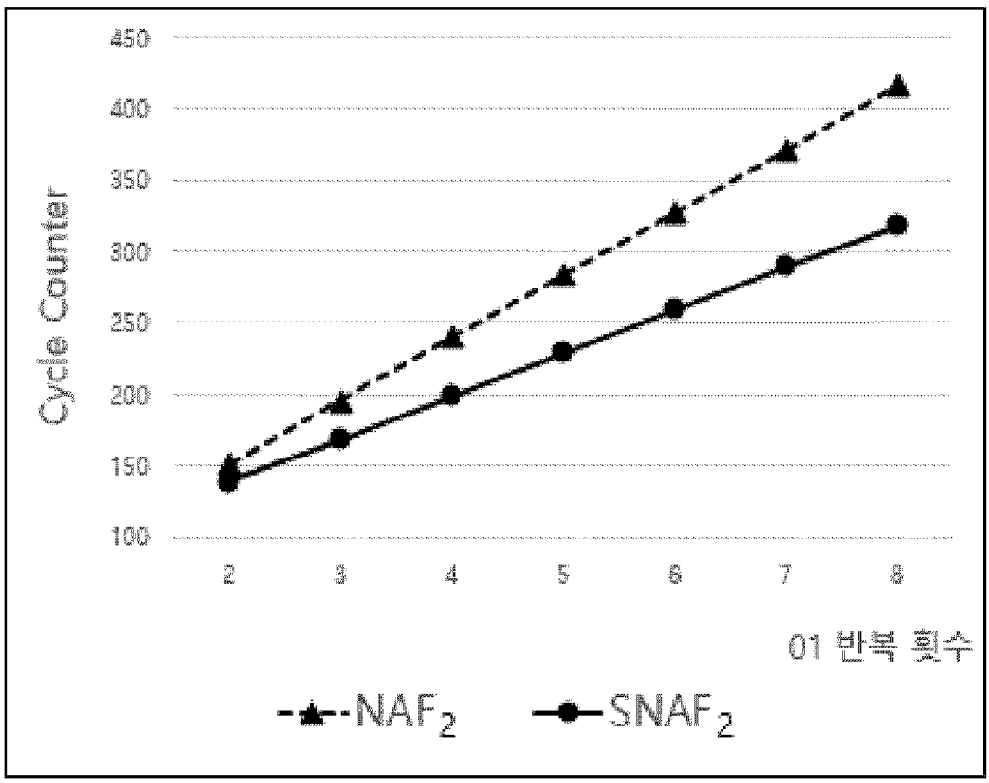
[도7]

	Repeated Patterns							
	1	01	10	001	011	100	101	110
Average gain	18.6	19.4	16.0	18.0	22.1	18.1	22.6	22.6
Deviation gain	31.3	31.8	31.3	31.7	36.7	31.7	37.2	36.7

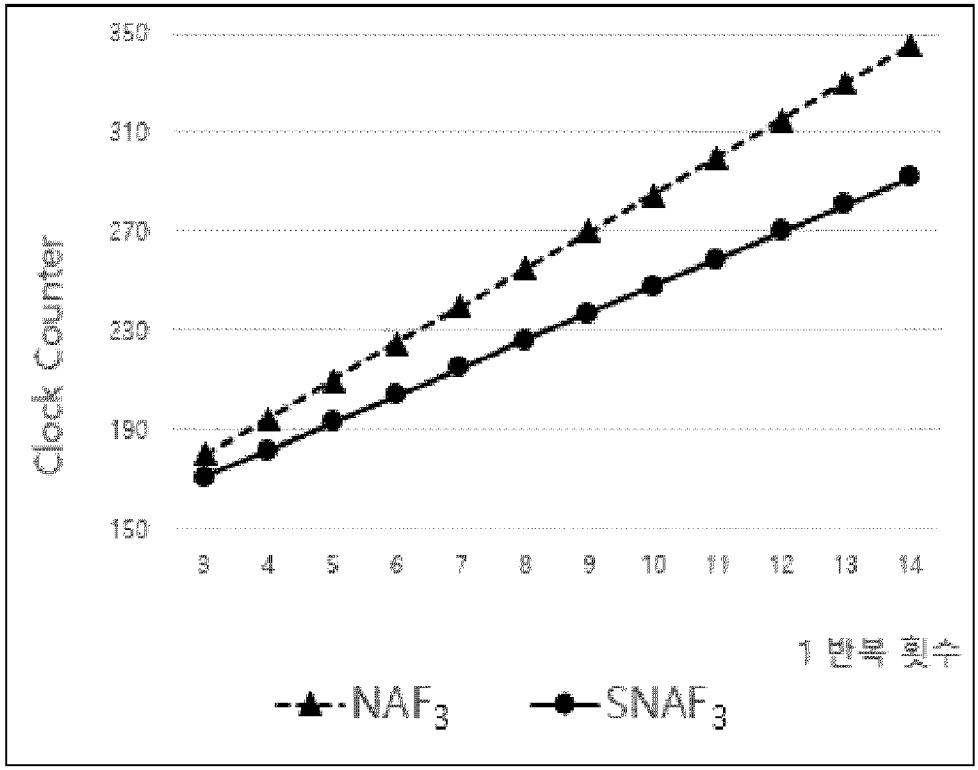
[도8]


350
300
250
200
150
100
Cycle Counter
2 3 4 5 6 7 8 9 10 11 12 13 14
1 반복 횟수
NAF2
SNAF2

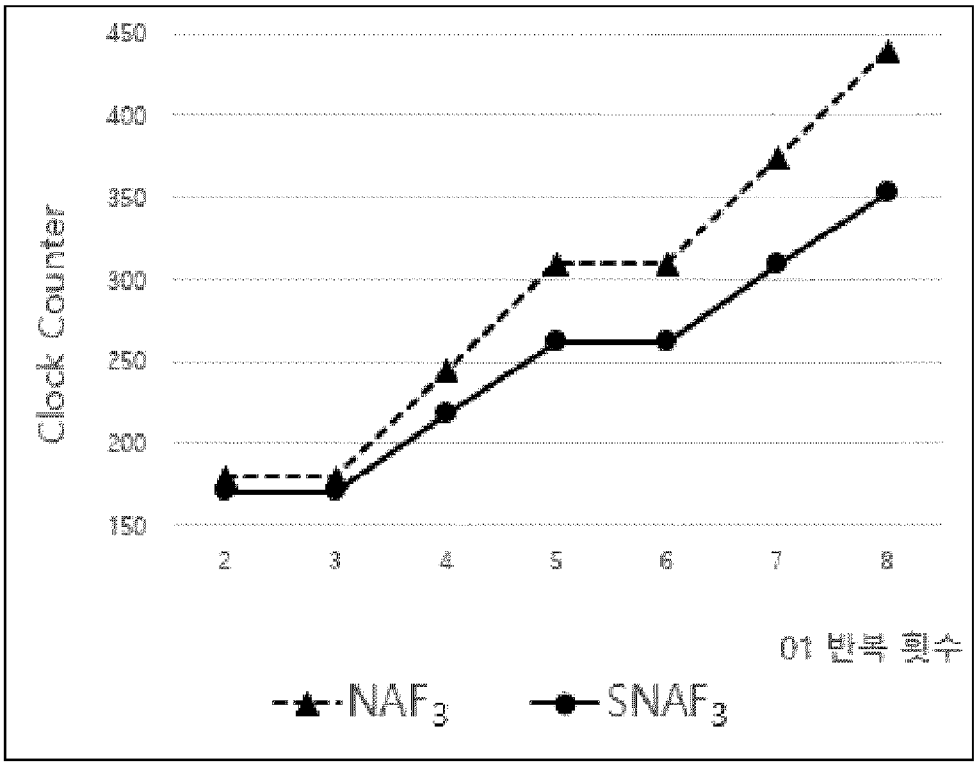
[도9]


450
400
350
300
250
200
150
100
Cycle Counter
2 3 4 5 6 7 8
01 반복 횟수
NAF2
SNAF2

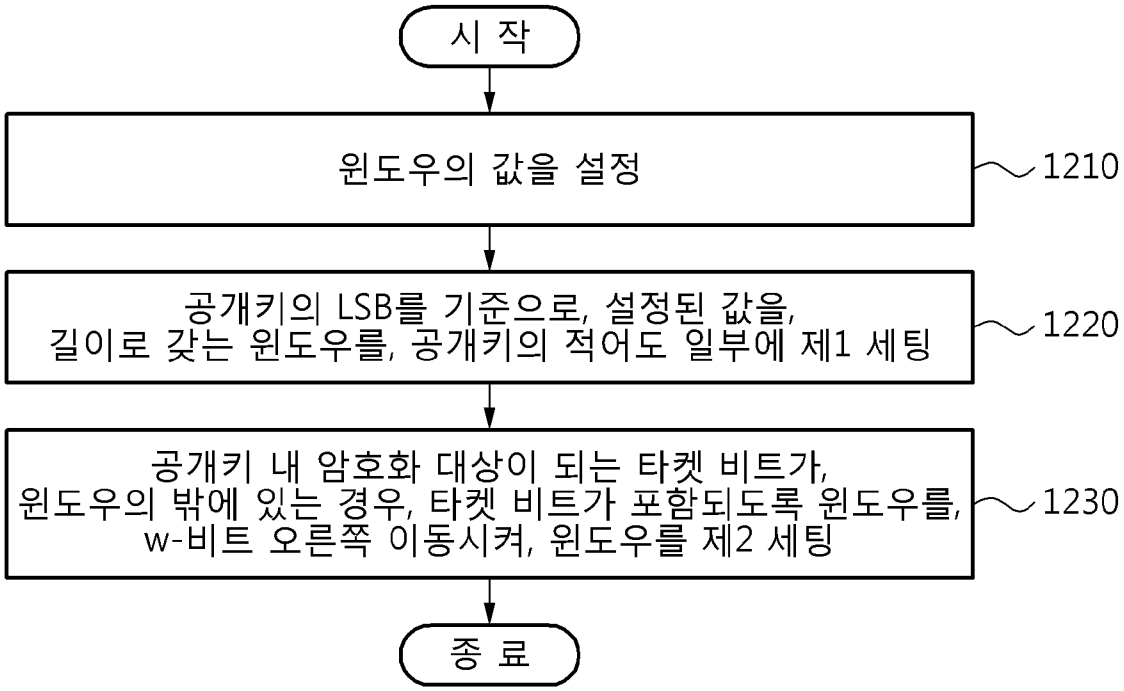
[도10]


350
310
270
230
190
150
Clock Counter
3
4
5
6
7
8
9
10
11
12
13
14
1 반복 횟수
NAF3
SNAF3

[도11]


450
400
350
300
250
200
150
Clock Counter
2
3
4
5
6
7
8
01 반복 횟수
NAF3
SNAF3

[도12]


시 작
윈도우의 값을 설정
1210
공개키의 LSB를 기준으로, 설정된 값을,
길이로 갖는 윈도우를, 공개키의 적어도 일부에 제1 세팅
1220
공개키 내 암호화 대상이 되는 타켓 비트가,
윈도우의 밖에 있는 경우, 타켓 비트가 포함되도록 윈도우를,
w-비트 오른쪽 이동시켜, 윈도우를 제2 세팅
1230
종 료