부가형 디지탈서명 방식 표준(안) 제 2부 : 확인서 이용 디지탈서명 알고리즘

문상재[1], 박성준[2], 원동호[2], °이필중[3], 임채훈[3], 장청룡[4]
경북대학교[1], 성균관 대학교[2], 포항공과대학교[3], 한국통신[4]

(Proposed) Standard for Digital Signature Mechanisms with Appendix - Part 2 : KCDSA(Korean Certificate-based Digital Signature Algorithm)

Chung-Ryong JANG[1], Sang Jae MOON[2], °Pil Joong LEE[3], Chae Hoon LIM[3], Dong Ho WON[4]
Korea Telecom Research Lab.[1], Kyungpook Nat'l Univ.[2],
Pohang Univ. of Sci. & Tech.[3], Sung Kyun Kwan Univ.[4]

요약문

디지탈서명 알고리즘은 정보보호를 위한 필수적 조건이며 금년에 발표된 전산망 보급확장과 이용 촉진에 관한 법률의 개정안에도 전자문서의 법적효력을 인정하고 있고, 전자서명이 들어가야 함을 원칙으로 하고 있다. 따라서, 디지탈서명의 국가 표준화가 시급해졌다. 이러한 때에 '94년부터 한국전자통신연구소 정보통신 표준 연구센터의 지원을 받아 디지탈서명 표준화가 진행되고 있는 것은 다행한 일이라하겠다. 작년의 결과인 "부가형 디지탈서명 방식 - 제 1부 : 기본구조 및 모델"이 한국통신기술협회(TTA)를 통하여 한국전기통신 표준(KCS)으로 올려져 금년안에 국가 표준화를 목적으로 작업이 진행되고 있다. 올해에는 지난 9월 표준안 발표회를 통하여 "부가형 디지탈서명 방식 표준(안) - 제 2부 확인서 이용 디지탈서명 알고리즘"이 발표되었다. 본 원고는 발표되었던 내용에 논의된 사항들을 반영하여 수정한 후 정리한 것이다. 본 표준안은 보완되어 11월 말에 한국전기통신 표준(KCS)으로 올려질 예정이다. 본문에서는 표준안으로 제안된 디지탈서명 알고리즘을 기술하였고, 부기에서는 알고리즘 수행시 유의점과 요구되는 부분함수들의 예를 기술하였다.

1. 개요

본 표준은 정보처리 시스템 또는 정보통신망에서 임의의 길이를 갖는 메시지에 대한 확인서 이용 부가형 디지탈서명의 생성과 검증을 위한 알고리즘 KCDSA(Korean Certificate-based Digital Signature Algorithm)를 기술한 것이다. 부기에 있는 참조사항들은 본 표준을 사용하거나 이해함에 도움을 주기위해 기술된 것으로 준수사항은 아니다.

2. 적용범위

본 표준에서 기술되는 확인서 이용 디지탈서명 알고리즘 KCDSA는 정보처리시스템 또는정보통신망 환경하에서 인증, 무결성 및 부인봉쇄 등의 정보보호 서비스를 제공하기 위하여 서명자가 디지탈서명을 생성할 때와 검증자가 서명된 메시지의 서명을 검증할 때 적용될 수 있다.

3. 준용표준

본 표준에서 별도로 규정하지 아니한 사항은 다음의 표준을 준용한다.

1) KCS 표준 XXX : 부가형 디지탈서명 방식 표준(안) - 제 1부 : 기본 구조 및 모델

4. 정의와 표기

4.1 준용한 정의

다음은 준용표준에서 준용한 정의들이다.

가. 검증과정
나. 공개 검증키
다. 메시지
라. 비공개 서명키
마. 서명과정
바. 서명된 메시지
사. 해쉬코드
아. 해쉬함수

4.2 기호와 표기

h, ‖를 제외한 기호는 음이 아닌 정수 값이다.

g	$a^{(p-1)/q} \bmod p$. $1 < a < p-1$이고 $a^{(p-1)/q} \bmod p > 1$을 만족함
h	해쉬함수
H	해쉬코드. $H < 2^{L_q}$
K	난수값. $0 < K < q$
L_p	p의 비트길이. $512 + 64i$, $0 \le i \le 8$
L_q	q의 비트길이. $128 + 16j$, $0 \le j \le 8$
O_I	선택 사항인 사용자 입력. 서명자와 검증자가 알고있는 데이터 (예) 서명자의 식별자, 검증자의 식별자, 공개 시스템 파라미터 등
p	소수. $2^{L_p-1} < p < 2^{L_p}$
q	p-1을 나누는 소수. $2^{L_q-1} < q < 2^{L_q}$
R	서명의 앞부분. $R < q$
S	서명의 뒷부분. $S < q$
Σ	서명. $\Sigma = R \| S$
X	비공개 서명키. $0 < X < q$
Y	공개 검증키. $Y = g^X \bmod p$
Z	수신되었거나 다시 만든 Z의 값
‖	연접

p, q, g는 모든 사용자들이 공용으로 사용하는 변수이고, X는 사용자의 비공개 서명키, Y는 사용자의 공개 검증키이다. 이상의 5개 변수는 고정 변수이고, 난수값 K는 서명 생성시마다 새롭게 선택된다. X와 K는 서명과정에서 사용되고 남에게 알려지지 않

아야한다. h는 충돌저항성을 갖는 해쉬함수이다.

L_p, L_q의 선택은 부기 3에, p와 q 를 만드는 방법은 부기 4에, X 와 K를 만드는 방법은 부기 5에, g를 만드는 방법은 부기 6에 설명되어 있는 것을 사용할 수도 있다.

5. 서명과정

메시지 M에 대한 서명은 다음 단계를 거쳐 계산된 $\Sigma = R \| S$이다.

단계 1. $H = h(M \| O_1)$을 계산한다.

단계 2. 난수값 K를 {1, ..., q-1}에서 선택한다.

단계 3. $R = (g^K \bmod p) \bmod q$를 계산하고 R = 0이면 단계 2로 간다.

단계 4. $E = ((RH) \bmod q) \bmod 2^{L_q/2}$을 계산한다.

단계 5. 서명자의 비공개 서명키 X를 이용하여 $S = (K - EX) \bmod q$를 계산하고S = 0 이면 단계2로 간다.

단계 6. $\Sigma = R \| S$를 출력한다.

6. 검증과정

$\{\underline{M}, \underline{\Sigma}\}$를 검증자가 받은 서명된 메시지라 하자. Y는 서명자의 공개 검증키이며 서명자의 것임을 확인하였다고 가정한다. 검증자가 서명을 검증하기위하여 먼저 $\underline{\Sigma} = \underline{R} \| \underline{S}$에 대하여 $0 < \underline{R} < q$이고 $0 < \underline{S} < q$ 임을 확인한다. 두 개의 조건을 모두 만족하는 경우에 다음의 계산을 한다.

단계 1. $\underline{H} = h(\underline{M} \| O_1)$을 계산한다.

단계 2. $\underline{E} = ((\underline{R}\,\underline{H}) \bmod q) \bmod 2^{L_q/2}$ 을 계산한다.

단계 3. 서명자의 공개 검증키 Y를 이용하여 $V = g^{\underline{S}} Y^{\underline{E}} \bmod p$ 를 계산한다.

단계 4. $V \bmod q = \underline{R}$이 성립하는지 확인한다.

단계 4에서 결과가 참이면 서명 $\underline{\Sigma} = \underline{R} \| \underline{S}$는 받은 메시지 $\underline{M}$에 대하여 공개 검증키 Y를 가진 사용자가 비밀 서명키 X로 서명하였음이 확인된 것이다. $\underline{M} = M$, $\underline{\Sigma} = \Sigma$ 일때, $V \bmod q = \underline{R}$이 됨을 부기 2에서 증명하였다.

결과가 거짓이면 메시지 $\underline{M}$에 불법적인 방법으로 서명이 되었거나 공격자에 의하여 메시지가 변경된 것이다. 따라서, $\underline{M}$은 가치없는 데이터가 된다.

부기 1. KCDSA의 사용시 유의사항

1.1. 확인서

KCDSA는 확인서를 이용한 디지탈서명 방식이다. 디지탈서명의 안전성을 위하여 검증과정에서 사용되는 공개키가 누구의 것인지를 확인할 필요가 있다. 이것은 사용자들 상호간에 신뢰할 수 있는 제 3자가 사용자의 공개 검증키와 이름 등이 포함된 신용장에 서명을 하여 발행해 주는 확인서를 사용함으로서 가능하다. 확인서를 만드는 방법이나 분배는 이 표준의 범위 밖이다.

1.2. 특허

관련된 외국 특허는 한국에 등록되어 있지 않은 상태이며, 본 KCDSA의 제안자들은 KCDSA가 한국내에서 사용되는 한 본 표준에 수용된 지적권리를 행사하지 않을 것이므로 누구나 국내 제조-사용시에는 사용료 없이 사용할 수 있다.

부기 2. V mod q = $\underline{R}$ 증명

검증자가 받은 서명된 메시지 {$\underline{M}$, $\underline{\Sigma}$}가 서명자가 바르게 서명하여 전송한 데이터 {M, Σ}이면 검증과정 단계 4에서 V mod q = $\underline{R}$가 성립함을 증명하고자 한다.

정리. $\underline{M}$=M, $\underline{R}$= R, $\underline{S}$= S 이면 V mod q = $\underline{R}$이다.

증명 : $\underline{M}$=M 이므로 $\underline{H}$= h($\underline{M}$||O) = H 이고, $\underline{R}$= R 이므로

$$\underline{E} = ((\underline{R}\,H) \bmod q) \bmod 2^{L_q/2}$$
$$= ((R\,H) \bmod q) \bmod 2^{L_q/2}$$
$$= E \text{ 이다.}$$

$\underline{S}$= S = (K - EX) mod q 이고 $\underline{E}$= E 이므로

$$V = (g^{\underline{S}} Y^{\underline{E}}) \bmod p$$
$$= (g^{K-EX} Y^{E}) \bmod p$$
$$= (g^{K-EX} (g^{X} \bmod p)^{\underline{E}}) \bmod p$$
$$= g^{K-EX+XE} \bmod p$$
$$= g^{K} \bmod p \text{ 이다.}$$

따라서,

$$V \bmod q = (g^{K} \bmod p) \bmod q$$
$$= R \text{ 이다.}$$

$\underline{R}$= R 이므로 V mod q = $\underline{R}$이다. ■

부기 3. 안전성 분석 - L_p, L_q의 선택

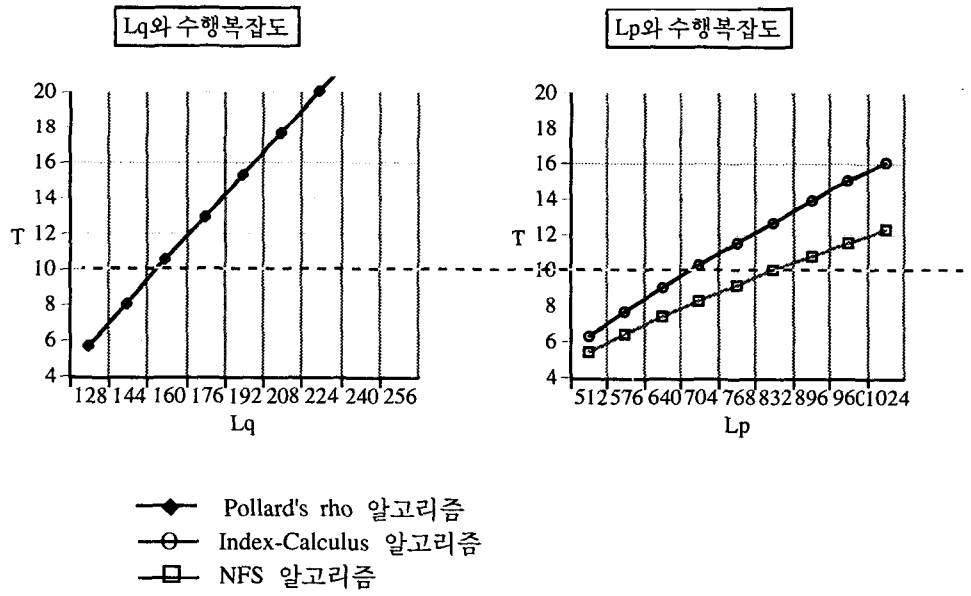
KCDSA는 유한체 GF(p)위에서 정의된 알고리즘이며 KCDSA의 안전도는 유한체에서 이산대수 문제를 풀기 어렵다는 사실에 기초하고 있다. KCDSA에서 이용된 이산대수 문제는 GF(p) 의 원소 g ($g^q \bmod p = 1$)로 생성되는 부분 곱셈군 <g>의 원소 Y에 대하여 $g^X \bmod p = Y$를 만족하는 X를 찾는 문제이다.

이산대수를 푸는 빠른 알고리즘으로 Index-Calculus 알고리즘과 [Adle] Number Field Sieve(NFS) 알고리즘이 있다 [Lens]. q가 작을때는 Pollard's rho 알고리즘을 사용하는 것이 효과적이다 [Kobl].

그림1은 p와 q의 비트 길이 L_p, L_q에 따라 각 알고리즘을 수행할 때의 예상 수행 시간을 나타낸 것이다. 가로축은 L_p, L_q를 나타내고 세로축은 q가 L_q비트 소수(p가 L_p비트 소수)일때 <g>위에서(GF(p)위에서)의 이산대수 문제를 풀기위하여 필요한 연산수이다. MIPS-year 는 1 MIPS(million instructions per second)의 프로세서가 1년동안 행할 수 있는 연산수를 나타낸다.

그림1에 나타난 수행복잡도를 바탕으로 안전도에 따른 p와 q의 비트 길이 L_p, L_q를 결정할 수 있다. 예를 들어 10^{10} MIPS-years 의 안전도가 요구되면 L_q=160, L_p=832로 선택해야한다. 10^{10} MIPS-years 의 안전도란 이산대수 문제를 풀기위하여 10^{10} MIPS-years 연산(예를 들면, 100 MIPS 의 컴퓨터 백만개가 100년 동안 수행해야 하는 정도의 연산)을 해야한다는 것이다. 표1에 안전도에 따른 L_q와 L_p를 정리하였다.

$T = \log_{10}(\text{MIPS-years})$



(그림 1) L_p, L_q에 따른 이산대수 문제의 수행복잡도

안전도(MIPS-years)	10^6	10^8	10^{10}	10^{12}
q의 비트길이 (L_q)	144	144	160	176
p의 비트길이 (L_p)	576	704	832	1024

(표 1) 안전도에 따른 L_p, L_q

L_q비트의 해쉬코드를 출력하는 해쉬함수는 birthday attack을 이용하여 $2^{L_q/2}$ 의 연산으로 충돌쌍을 찾을 수 있다[Stal]. 충돌쌍의 생성이 쉽다면 서명자에 의한 서명의 위조가 쉬워지므로 해쉬함수가 birthday attack에 견디도록 L_q가 선택되어야 한다. Birthday attack에 요구되는 연산수 $2^{L_q/2}$는 대략적으로 Pollard's rho 알고리듬에 의한 이산대수 계산에 요구되는 연산수와 거의 같으므로 해쉬코드의 길이를 q의 비트길이 L_q와 같게 잡았다.

부기 4. 소수 p, q 의 생성

이 부기는 참고문헌 [Dss]의 부록 2에 소개된 알고리즘을 KCDSA 에 맞게 수정하여 KCDSA에 사용될 수 있는 소수 p, q를 만드는 알고리즘을 설명하고 있다. 이들 알고리즘에는 난수 발생 함수(부기 5 참조)와 효율적인 모듈라 멱승 알고리즘이 필요하다.

4.1 확률적인 소수 판정 알고리즘

소수 p와 q를 만들기 위해서 소수 판정 알고리즘이 필요하다. 사용가능한 몇 가지의 빠른 확률적인 알고리즘이 있다. 다음의 알고리즘은 부분적으로 Gary L. Miller의 생각에 기초를 두고 M.O. Rabin이 만든 방법을 간단히 한 것이다[Knuth]. 이 알고리즘이 n번 반복되면 알고리즘을 통과한 정수가 소수가 아닐 확률은 $1/4^n$보다 크지 않다. 따라서 n이 50이상이면 받아들일 수 있는 오류 확률을 가진다. 정수가 소수인지 알아보기 위해서 다음 알고리즘을 수행한다.

단계 1. $i = 1$, $n \geq 50$ 으로 한다.

단계 2. w는 테스트할 정수이고, $w = 1 + 2^a m$라 둔다. 여기서 m은 홀수이고 2^a은 w-1을 나누는 2의 가장 큰 멱승이다.

단계 3. $1 < b < w$인 난수 b를 만든다.

단계 4. $j = 0$, $z = b^m \bmod w$을 계산한다.

단계 5. $j = 0$, $z = 1$이거나 $z = w - 1$이면 단계 9로 간다.

단계 6. $j > 0$이고 $z = 1$이면 단계 8로 간다.

단계 7. $j = j+1$, $j < a$이면 $z = z^2 \bmod w$로 하고 단계 5로 간다.

단계 8. w는 소수가 아니다. 그만둔다.

단계 9. $i < n$이면 $i = i+1$로 하고 단계 3으로 간다. 그렇지 않다면 아마도 w는 소수이다.

4.2 소수 p, q 만들기

KCDSA는 다음의 세 가지 조건을 만족하는 두 소수 p, q를 필요로한다.

a. $2^{L_p-1} < p < 2^{L_p}$, $L_p = 512 + 64i$ $(0 \leq i \leq 8)$
b. $2^{L_q-1} < q < 2^{L_q}$, $L_q = 128 + 16j$ $(0 \leq j \leq 8)$

c. q는 p-1을 나눈다.

이러한 소수의 생성은 먼저 해쉬함수와 사용자가 제공한 SEED를 이용하여 $2^{L_q-1} < q < 2^{L_q}$인 q를 만드는 것부터 시작한다. 다음에는 같은 SEED로 $2^{L_p-1} < x < 2^{L_p}$인 x를 만들고, 소수 p는 아래에 명시한 것처럼 x를 잘라서 mod 2q로 1과 합동인 수로 만든다.

$0 \le x < 2^d$인 정수 x는 다음과 같이 길이 d인 비트열로 변환될 수 있다.

$$x = x_1*2^{d-1} + x_2*2^{d-2} + ... + x_{d-1}*2 + x_d \quad -> \quad \{x_1, ... , x_d\}$$

반대로 길이 d인 비트열 $\{x_1, ... , x_d\}$는 다음과 같이 정수로 변환될 수 있다.

$$\{x_1, ... , x_d\} \quad -> \quad x = x_1*2^{d-1} + x_2*2^{d-2} + ... + x_{d-1}*2 + x_d$$

수열의 첫번째 비트는 정수 x의 최상위 비트에 해당하고 마지막 비트는 최하위 비트에 해당한다.

다음은 Miller - Rabin 의 확률적인 소수 판정 알고리즘이다.
$L_p-1 = nL_q + b$라 두자. 여기서 b와 n은 모두 $0 \le b$, $n < L_q$인 정수이다.

단계 1. 최소 L_q비트인 정수를 임의로 선택해서 이것을 SEED라 부른다.
d를 SEED의 비트길이라 하자.

단계 2. $U = h[SEED] \oplus h[(SEED+1) \bmod 2^d]$를 계산한다.

단계 3. U에서 최상위 비트와 최하위 비트를 1로 하여 q를 만든다. 부울 연산의 용어로는 $q = U \vee 2^{L_q-1} \vee 1$이다. $2^{L_q-1} < q < 2^{L_q}$임을 주의하라.

단계 4. 강한 소수 판정 알고리즘을[1] 이용하여 q가 소수인지를 검사한다.

단계 5. q가 소수가 아니면 단계 1로 간다.

단계 6. counter = 0, offset = 2로 둔다.

단계 7. $k = 0, ... , n$에 대해 $v_k = H[(SEED+offset+k) \bmod 2^d]$로 둔다.

단계 8. w와 x를 다음과 같이 놓는다.

$$w = v_0 + v_1*2^{L_q} + ... + v_{n-1}*2^{(n-1)*L_q} + (v_n \bmod 2^b)*2^{n*L_q}$$
$$x = w + 2^{L_p-1}$$

$L_p-1 = nL_q + b$ 이므로 $0 \le w < 2^{L_p-1}$이고, $2^{L_p-1} \le x < 2^{L_p}$임을 주의하라.

단계 9. $c = x \bmod 2q$, $p = x - (c - 1)$를 계산한다.

---

[1] 강한 소수 판정 알고리즘은 소수가 아닌 정수가 알고리즘을 통과할 확률이 2^{-80}이하이어야 한다.

단계 10. $p < 2^{L_p-1}$이면 단계 13으로 간다.

단계 11. p에 대해 강한 소수 판정을 한다.

단계 12. p가 단계 11을 통과하면 단계 15로 간다.

단계 13. counter= counter + 1, offset = offset + n + 1로 한다.

단계 14. counter $\geq 2^{12} = 4096$ 이면 단계 1로 가고 그렇지 않으면(counter < 4096)이면 단계 7로 간다.

단계 15. 만들어진 p와 q가 적절한 것인지 확인할 때 사용하기 위해 SEED와 counter를 저장한다.

부기 5. KCDSA를 위한 난수값 X, K 만들기

이 부기는 참고문헌[Dss]의 부록 3을 기본으로 하여 KCDSA에 맞게 수정한 것이다.

디지탈서명 알고리즘을 구현하기 위해서는 난수 혹은 의사난수(pseudorandom)를 생성할 수 있어야 한다. 사용자의 비밀 서명키 X와 각 메시지별 난수값 K는 0과 소수 q 사이에서 난수로 선택되어야 하고, 이 부기에서 제공하는 방법을 사용할 수도 있다.

5.1절에 있는 알고리즘을 비밀 서명키 X를 생성하는 데에 쓸 수 있다. 5.2절의 알고리즘은 난수값 K를 만들어 $R = g^K \bmod p$를 미리 계산하고, 미리 계산해 둔 R을 이용하여 메시지에 서명하는 과정을 기술한 것이다. $g^K \bmod p$는 메시지와 독립적인 계산이므로 미리 계산해 둘 수 있고, 이 알고리즘은 서명할 메시지에 대한 지식 없이 미리 계산할 수 있는 대부분의 서명 계산에 쓸 수 있다.

각 알고리즘들은 일방향 함수 G(u, c)를 사용하는데, 여기서 u는 L_q비트 정수이고 c는 b비트($L_q \leq b \leq 512$, b는 32의 배수) 정수이며, G(u,c)는 L_q비트 정수를 출력한다. 5.3절에서 Data Encryption Standard (DES)를 사용하여 함수 G를 만드는 방법을 설명하고 있다.

5.1. 난수값을 찾는 알고리즘

다음과 같이 m개의 난수값 X를 생성할 수 있다.

단계 1. seed-key인 XKEY에 쓸 새 비밀값을 고른다.

단계 2. 16진 표기로 초기값 u를 다음과 같이 둔다.

u = (2C80246B B2BC4C4A B078C7A3 15641642 FDD1FFF1
7C373875 1D1C357F 2A651CAB)의 하위 L_q 비트

단계 3. j = 0, ... , m-1에 대하여 다음을 계산한다.
a. $XSEED_j$ = 선택 사항인 사용자 입력
b. $XVAL = (XKEY + XSEED_j) \bmod 2^b$

c. $X_j = G(u, XVAL) \bmod q$
d. $XKEY = (1 + XKEY + X_j) \bmod 2^b$

5.2. K와 $g^K \bmod p$ 값들을 미리 계산하는 알고리즘

이 알고리즘은 m개의 메시지에 서명을 하기 위하여 난수값 K를 만들어 $g^K \bmod p$를 미리 계산하는 데에 쓸 수 있다.

단계 1. seed-key인 KKEY에 쓸 비밀 초기값을 고른다.

단계 2. 16진 표기로 u를 다음과 같이 초기화한다.

u = (B2BC4C4A B078C7A3 15641642 FDD1FFF1 7C373875
1D1C357F 2A651CAB 2C80246B) 의 하위 L_q 비트

단계 3. j = 0, ... , m-1에 대하여 다음을 계산한다.
a. $K_j = G(u, KKEY) \bmod q$
b. $R_j = (g^{K_j} \bmod p) \bmod q$
c. $KKEY = (1 + KKEY + K_j) \bmod 2^b$

단계 4. $M_0, \ldots, M_{m-1}$ 을 서명할 m개 메시지라고 하자.
j = 0, ... , m-1 에 대하여 다음을 계산한다.
a. $H = h(M_j \| O_{1,j})$라 하자
b. $E = (R_j H \bmod q) \bmod 2^{L_q/2}$
c. $S_j = (K_j - EX) \bmod q$
c. M_j에 대한 서명은 (R_j, S_j)이다.

단계 5. u = H 라 둔다.

단계 6. 단계 3으로 간다.

단계 3에서 다음의 m개의 메시지에 서명하는 데 필요한 값들을 미리 계산하며, 단계 4는 이 m개 메시지의 첫 메시지가 준비되었다면 언제든지 시작할 수 있다. 다음의 m개 메시지가 준비되지 않았다면 언제든지 단계 4의 실행을 중단할 수 있다. 단계 4와 단계 5가 완료되자마자 단계 3이 실행될 수 있고 m개 메시지의 다음 그룹의 처음이 준비될 때까지 결과를 저장할 수 있다.

KKEY에 쓸 공간에 대해 부언하면, 단계 3에서 계산한 $K_0, \ldots, K_{m-1}$과 $R_0, \ldots, R_{m-1}$을 저장하는 데에 길이 m의 배열 2개가 필요하다.

5.3. DES로부터 G함수 만들기

비트열 a와 b의 비트연산 exclusive-or를 $a \oplus b$로 표기하자. a1, a2, b1, b2를 32비트 비트열이라 하고, b1'은 b1의 하위 24비트라 하자. 또 $K = b1' \| b2$, $A = a1 \| a2$라 하고 다음을 정의한다.

$DES_{b1,b2}(a1, a2) = DES_K(A)$

이때 $DES_K(A)$는 56비트 키 K를 이용하여 64-비트 블락 A를 상용 DES로 암호화한 것을 나타낸다. u는 L_q비트, c는 b비트 비트열 일때, 다음의 알고리즘으로 L_q비트 비트열을 출력하는 G(u, c)를 계산할 수 있다. $d = [L_q/32]$, $e = b/32$로 정의한다. 이때 L_q를 32로 나눈 나머지가 0이 아니고 16이 될 수도 있는데(L_q가 16의 배수이므로), 이 경우 남는 16비트를 한번 더 반복하여 32비트를 만들고 $d = d + 1$로 둔다.

단계 1. u, c를 다음처럼 쓴다.

$u = u_1 \| u_2 \| u_3 \| \ldots \| u_d$

$c = c_1 \| c_2 \| c_3 \| \ldots \| c_e$

(u_i와 c_i는 32 비트)

단계 2. $i = 1, \ldots, e$에 대하여 다음을 계산한다.

$x_i = u_i \oplus c_i,\ i = 1, 2, \ldots, d$

$x_i = u_{i-d} \oplus c_i,\ i = d+1, d+2, \ldots, 2d$

$x_i = u_{i-2d} \oplus c_i,\ i = 2d+1, 2d+2, \ldots, 3d$

...

$x_i = u_{i-kd} \oplus c_i,\ i = kd+1, kd+2, \ldots, b-1, b$

단, $k = [b/d]$.

단계 3. $i = 1, \ldots, d$에 대하여 다음을 계산한다.

$b1 = c_{((i-2) \bmod e)+1}$

$b2 = c_{((i-3) \bmod e)+1}$

$a1 = x_i$

$a2 = x_{((i-5) \bmod e)+1} \oplus x_{((i-2) \bmod e)+1}$

$y_{i,1} \| y_{i,2} = DES_{b1,b2}(a1, a2)$

($y_{i,1}$와 $y_{i,2}$는 32비트)

단계 4. $i = 1, \ldots, d$에 대하여 다음을 계산한다.

$z_i = y_{i,1} \oplus y_{((i-4) \bmod e)+1,2} \oplus y_{((i-3) \bmod e)+1,1}$

단계 5. $G(u, c) = z_1 \| z_2 \| z_3 \| \ldots \| z_d$의 하위 L_q비트

부기 6. g 만들기

이 부기는 참고문헌 [Dss]의 부록 4에 소개된 g를 만드는 알고리듬이다. KCDSA에서 사용되는 g를 생성할 때에도 사용할 수 있다.

단계 1. 부기 D에서 설명한 방법으로 p, q를 생성한다.

단계 2. $e = (p - 1)/q$.

단계 3. a를 $1 < a < p-1$ 이고 이전에 시도되지 않은 임의의 정수값으로 둔다.

단계 4. $g = a^e \bmod p$를 계산한다.

단계 5. 만일 $g = 1$이면 단계 3으로 간다.

부기 7. 참고문헌

[Adle] L.Adleman, "A subsexponenential algorithm for the discrete logarithm problems with applications to cryptography", in *Proc, IEEE 20th Annual Symposium on Foundations of Compuer Science*, 1979, pp.55-60.

[Dss] Digital Signature Standard (DSS), FIPS PUB 186, 1994.

[Knuth] Knuth, *The Art of Computer Programming*, Vol. 2, Addison-Wesley, 1981, Algorithm P, page 379.

[Kobl] N.Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, 1987.

[Lens] A.K.Lenstra, H.W.Lenstra.Jr.(Eds.), *The development of the number field sieve*, Springer-Verlag, 1993.

[Stal] W.Stallings, *Network and Internetwork Security*, Prentice Hall, 1995.

(본 표준안을 작성하는 데 많은 도움을 준 포항공과대학교 정보통신연구소 이은정 연구원에게 감사를 표한다.)