DSA Sisteminin Çalıştırılması ve Test Edilmesi

Oğuz YAYLA

***Özet*—Açık anahtar altyapısı (AAA) sistemleri içinde kullanılan algoritmalardan bir tanesi DSA(Digital Signature Algorithm)'dır. Artık standartlaşmış olan DSA birçok ülke AAA sistemlerinde de tavsiye edilmektedir. Ülkemizde de yayınlanan e-imza uygulamalarına ilişkin tebliğde [6] DSA'nın kullanılabileceği belirtilmiştir. Çalıştırılacak DSA sisteminde FIPS 186-2'ün [7] güvenlik seviyesine göre uygun DSA parametreleri kullanılmalıdır. Ayrıca, kullanılacak özetleme ve rastsallık algoritmalarına ilişkin aynı standartta verilen kısıtlamalara uyulmalıdır. Bu çalışmada bir DSA uygulamasında gereken sistem gereksinimlerini en güncel haliyle belirteceğiz. Ayrıca, çalıştırılan bir DSA sisteminin güvenilir olması için gerek ve yeter koşulları belirteceğiz. Bunları geliştirilen bir DSA sistemi ile göstereceğiz.**

***Anahtar Sözcükler*— DSA, DSA Atakları, DLP.**

I. GİRİŞ

Açık Anahtar Altyapısında kullanıcı sertifikalarını oluştururken anahtar ikililerine ihtiyaç duyulmaktadır. Bu ikilileri üretmek için RSA, DSA ve EDSA algoritmalarını kullanılmaktadır. Bu çalışmada DSA sisteminin nasıl çalıştırılacağını ve test edileceğini anlatacağız.

DSA sistemi ayrık logaritma problemine dayanır ve bu problem ikilik(binary) cisimlerde Coppersimith'in [4] yöntemiyle diğer cisimlere göre daha hızlı çözülebilmektedir. Dolayısıyla ikilik cisimler DSA uygulamalarında kullanılmamalıdır. Diğer taraftan, sistemin rahat çalıştırılabilmesi daha fazla algoritmaya ihtiyaç duymaması için DSA yalnızca asal cisim üzerinde koşacaktır, diğer üst cisimler kullanılmayacaktır. Zaten DSA standardı [7] asal cisim üzerinde çalıştırılmış sistemleri tavsiye etmektedirler. Bu cisim aritmetiği için GMP [8] asal cisim kütüphanesi yeterli olacaktır.

II. DSA SİSTEMİ

A. Tanım Kümesi

DSA sistemi (p, q, g) tanım kümesinden oluşur. Bu tanım kümesi halka açıklanan elemanlardır. p ve q sistemin güvenlik seviyesini belirleyen asal sayılar ve g ise p elemanlı cisim içinde boyutu q olan alt grubun üretecidir. p ve q'nun büyüklüğü sistemin güvenlik gücünü belirtmektedir. Dolayısıyla, asal sayıların üretilmesi önem ihtiva etmektedir. Doğru ve hızlı yapılabilmesi için iki yöntem öne çıkmaktadır: Olasılıksal Miller-Rabin ve Kararlı Shawe-Taylor.

O.ODTÜ – Uygulamalı Matematik Enstitüsü Kriptografi Bölümü Ankara, Türkiye e118296@metu.edu.tr

Birinci yöntem olasılıksal yöntemle p ve q'nun üretimini yapmaktadır. Üretilen asal sayıların asal olmama olasılığı yeterince düşük yapabilmek için 2^{-100} olasılığı yeterli görülmüştür. Ayrıca, asal sayılar açık elemanlar oldukları için bu elemanların rasgele üretilmesi e-imza kullanıcılarına güven verecektir. Dolayısıyla, asal sayıların üretimi rasgele yapılmalıdır ve belli bir güven parametresi ile rasgele üretildikleri onaylanabilmelidir. Bu yöntemlerin algoritmik kodları Ek-1 de bulunmaktadır. Ayrıca, bu çalışmada anlatılan DSA sisteminde belirtilen bütün algoritmaların kodları yine Ek-1 de bulunmaktadır.

Tablo 1'de üretilecek asal sayıların bit uzunluklarına göre kaç defa olasılıksal Miller-Rabin testinden geçtiklerinde 2^{-100} olasılığında asal olmayacaklarını garanti etmektedir.[5]

İkinci yöntem ise p ve q asal sayılarını, güçlü asal sayı üretme yönteminin geliştirilmiş bir hali ile üretmektedir. Bu yöntem ilk yönteme göre biraz yavaştır, ancak üretilen sayılar kesinlikle asaldır. Bu yöntem DSA uygulama yazılımı içinde uygulanmayacaktır. Ancak, uygulanmasında bir sakınca yoktur.

Üreteç 2 ile q-1 arasında bir sayıdır. Üretecin üretilmesi yine e-imza kullanıcılarına güven verebilmek için rasgele üretilmeli ve üretme işlemi doğrulanabilir olmalıdır.

B. Anahtar İkilisi

DSA anahtar ikilisi (x,y)'den oluşur. x sayısı 1 ile q-1 arasında seçilen gizli anahtarı, y sayısı ise

Tablo 1: Miller-Rabin testi(t) - sayı bit uzunluğu(k)

k	t	k	t
0-160	34	265-278	16
161-163	33	279-294	15
164-166	32	295-313	14
167-169	31	314-334	13
170-173	30	253-264	17
174-177	29	265-278	16
178-181	28	335-360	12
182-185	27	361-392	11
186-190	26	393-430	10
191-195	25	431-479	9
196-201	24	480-542	8
202-208	23	543-626	7
209-215	22	627-746	6
216-222	21	747-926	5
223-231	20	927-1232	4
232-241	19	1233-1853	3
242-252	18	1853-…	2
253-264	17		

açık anahtarı belirtmektedir ve g^x mod p hesabının sonucudur. y sayısı 1 ile p-1 arasındadır. Anahtar ikilisi tüm kullanıcılarda farklı olması gerektiğinden üretilmesi tam rasgele olmalıdır; bunun için özel kriptografik rasgele sayı üreteçlerinden yararlanılmalıdır.

Verilen bir m mesajının DSA e-imzası 2q uzunluğundaki (r, s) ikilisinden oluşmaktadır. Dolayısıyla, DSA e-imza uzunluğu RSA e-imza uzunluğundan daha kısadır. Aynı güvenlik seviyelerinde DSA ve RSA sistemleri arasında 3/10 oranı vardır. Güvenlik seviyesi artırıldıkça bu oran azalmaktadır.

E-imza üretme algoritmasında geçen k sayısı rastsal olmalı ve yine gizli anahtarlara benzer yöntemlerle üretilmelidir. Dolayısıyla kullanılan rastsal sayı üreteci güvenlik seviyesini sağlamalıdır. Ayrıca rastsal üretilen k sayısında **sıfır byte (00000000 bit serisi)** olmamalıdır[3].

Ayrıca, kullanılan özetleme algoritması SHA-224 veya SHA-256 olmalıdır. Bu özetleme algoritması FIPS 180-2 standardında tanımlanmıştır. Tablo 2'de kullanılacak güvenlik seviyesine denk gelen özetleme fonksiyonunun minimum uzunluklarını göstermektedir.

Tablo 2: Güvenlik Seviyesi - Özetleme - DSA eşdeğer bit uzunlukları

Güvenlik Seviyesi(=AES)	HASH=Özetleme	DSA(=log p)
80	**224/256**	**1024**
128	256	3072
192	384	7680
256	512	15360

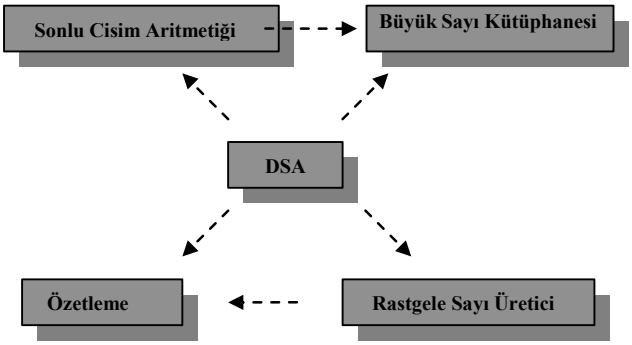
III. DSA YAZILIM MODÜLÜ

Bu modülde e-imza üretme ve doğrulama, olası asal sayı üretme ve testi için ilgili algoritmaları Ek-1 de belirtildiği gibi gerçekleştirilmiştir.

DSA sistemini çalıştırabilmek için öncelikle büyük sayı kütüphanesine ihtiyaç duyulur. Büyük sayı kütüphanesi asal cisim üzerindeki ve bazı sistem işlemlerini yapabilmek için gerekli olacaktır. DSA sisteminde zaman tutan işlemler (g^x gibi) asal cisim işlemleridir. Bu yüzden asal cisim işlemlerinin en etkili şekilde hazırlanması gerekmektedir [1]. Bu büyük sayı ve asal cisim kütüphanesi yazılım kütüphaneleri kullanılarak geliştirilebilir veya açık kaynak kod halinde, kullanımı tamamen serbest (ticari kullanımlar dahil) yazılım kütüphaneleri kullanılabilir. Internet ortamında bu tarz birçok kaynak vardır. GMP [8] de bunlara örnek bir kütüphanedir. Akademik makalelerde de geliştirme kütüphanesi olarak genellikle tercih edilmektedir. Oldukça hızlı büyük sayı ve asal cisim kütüphanelerini içermektedir.

Rastgele sayı üreteci ve özetleme algoritmaları da DSA sistemi içinde kullanılmaktadır. RIPEMD, SHA, WHIRLPOOL özetleme algoritmaları olarak kullanılabilir. Bunların içinden SHA sınıfını modül içine eklendi. Diğerleri daha sonra eklenecektir. SHA sınıfının yazılım kodları [9]'dan temin edinilebilir. Rastgele sayı üreteci de yine Internet ortamından [10] temin edinilebilir veya 8'nin içindeki rastgele sayı üreteci ve Ek -1 de belirtilen algoritmalar kullanılırsa yeterli güvenliğe erişilecektir. Bu modül içinde de ikinci yöntem tercih edilmiştir.

Şekil 1: DSA sisteminin genel yapısı



IV. TEST SİSTEMİ

DSA sistemi Ayrık Logaritma Problemine (ALP) dayandığı için bu sistemin testi öncelikle ALP üzerinden test ve ataklara dayanmaktadır. ALP probleminin çözümü günümüz bilgi düzeyiyle altüstsel sürede çözümü vardır.

$L_p[a,c] = O(\exp(c(\ln p)^a (\ln\ln p)^{(1-a)})$
a sıfıra yakınsarken **polinom** zamanlı
a bire yakınsarken **üstel** zamanlı
a arada bir değerken **alt üstsel** zamanlı denir.

Index Calculus Yöntemi: İlk aşamada koşulacak olan, uygulaması kolay, ALP çözen algoritmadır. $L_p[½,c]$ sürede ALP çözümü gerçekleştirebilmektedir. Bu yöntem Ek-2'deki yöntemle koşulmuştur. Çok büyük sayılarda sonuç vermesi beklenmemesine karşın; iyi tanımlanmamış tanım kümesi veya anahtar ikilisi için koşulan bir yöntemdir.

Pollard Rho Yöntemi: Yavaş olmasına karşın paralel hesaplamaya uygun olduğundan koşulması gerek görülen diğer algoritmadır. $L_q^{½}[1,c] = O(q^{½})$ sürede ALP çözümü gerçekleştirebilmektedir. Bu yöntem Ek-2'deki yöntemle koşulmuştur. Bu yöntem q parametresinin büyüklüğüyle orantılıdır; dolayısıyla uygun seçilmeyen q değerleri için çalışan bir yöntemdir.

Number Field Sieve Yöntemi: Bilinen en hızlı ALP çözen algoritmadır. Uygulanması zor da olsa gerekli görüldüğünde anahtarın güvenliğini belirttiğinden koşulması önemlidir. L[1/3, 1,923] sürede ALP çözümü gerçekleştirebilmektedir. Bu yöntem sistem içerisinde koşulmamıştır. Ama açık kaynak kodları üzerinden metot geliştirilmiştir.

Ayrık logaritma problemini çözebilmek için ortaya atılan Number Field Sieve yönteminin açık kaynak olarak bulunan en iyi yazılım Chris Studholme [2] tarafından geliştirilmiştir. Açık kaynak kodlar arasında en iyi performansı vermesine rağmen üzerinde iyileştirme yapılabilecek bir çok alan bulunmaktadır. Factor base ve polinom seçimi Index Calculus yöntemi ve Number Field Sieve için de oldukça önemlidir. [2] kütüphanesinin çalıştırıldığında verdiği bir sonuç şu şekilde olmaktadır.

$y = g^x \bmod q$ eşitliğindeki q, g, y sayıları verildiğinde x sayısını ve ne kadar sürede bu işlemi yaptığını belirtmektedir:

q:701927150347543584322583
g: 181168661459155964764704
y:274434000736645416233805
T:3.35049saniye
x: 179480586960441733839158

DSA sisteminin ikinci bir test yöntemi ise parametre ve algoritmaların uygunluğudur:

i. p ve q'nun uzunluğu Tablo 3'de belirtildiği gibi seçilmelidir:

Tablo 3: p'nin bit uzunluğu ve q'nun bit uzunluğu

L	N
1024	160
2048	224
2048	256
3072	256

ii. p ve q'nun asal olmama olasılığı 2^{-100} den az olmalıdır.

iii. g sayısı doğrulanabilir yöntemle üretilmeli ve test edilmelidir.

iv. Her e-imza üretme algoritmasında üretilen k sayısı tahmin edilemez olmalıdır.

v. Özetleme algoritmaları SHA-224 veya SHA-256 olmalıdır.

V. İLERİKİ ÇALIŞMALAR

DSA sistemi üzerine belirtilen ataklar birebir uygulanabilirse bu ataklara katkılar ve iyileştirilmeler yapılabilir. Özellikle [2] ve [3]'nin çalışmaları üzerine katkılar yapılabilir.

EK-1 DSA YAZILIM SİSTEMİ ALGORİTMALARI

E-imza Üretme Algoritması
INPUT: **m** = mesaj
OUTPUT: signature **(r,s)**
ALGORITHM:
1. $r = (g^k \bmod p) \bmod q$ where k is a random number generated by algorithms defined below
2. z = HASH(m) where hash algorithm is SHA224/256
3. $s = (k^{-1}(z + xr)) \bmod q$
4. Return "signature = (r, s)"

E-imza doğrulama algoritması
INPUT:
1. $\mathbf{m_0}$ gelen mesaj
2. $\mathbf{(r_0, s_0)}$ gelen e-imza ikilisi

OUTPUT: **status**: SUCCESS or FAILURE
ALGORITHM:
1. $z_0 = HASH(m_0)$ where hash is SHA256
2. if r_0 or $s_0 > q-1$ than FAILURE
3. $(g^{s0^{(-1)}z0 \bmod q} \; y^{s0^{(-1)}r0 \bmod q} \bmod p) \bmod q == r_0$ then SUCCESS else FAILURE.

Onaylı Özetleme Fonksiyonlarıyla Olası Asal Sayılar p ve q'nun Üretilmesi:
INPUT:
1. **L**: bit length of p
2. **N**: bit length of pq
3. **seedlen**: bit length of seed

OUTPUT:
1. **state** : SUCCESS or FAILURE.
2. **p, q**: primes
3. **domain parameter seed**: The seed used during the generation of p and q.
4. **counter** (optional)

ALGORITHM:
1. if (L,N) is not accepted pair then return FAILURE
2. If (seedlen < N), then return FAILURE.
3. n = ceil[L/outlen] − 1 where outlen is to be chosen as N.
4. b = L − 1 − (n * outlen).
5. Get an arbitrary sequence of seedlen bits as the domain parameter seed.
eg. by compiler's rand function.
6. $U = Hash(\text{domain parameter seed}) \bmod 2^N$.
7. $q = U \vee 2^{N-1} \vee 1$. (v means or)
8. Use Miller Rabin Algorithm(9) to test whether q is a prime.
9. If q is not a prime, then go to step 5.
10. offset = 1.
11. For counter = 0 to 4095 do
11.1 For j = 0 to n do
Vj = Hash((domain parameter seed + offset + j) mod $2^{seedlen}$).
11.2 $W = V_0+(V_1*2^{outlen})+...+(V_{n-1}*2^{(n-1)*outlen})+((V_n \bmod 2^b)*2^{n*outlen})$.
11.3 $X = W + 2^{L-1}$.
11.4 c = X mod 2q.
11.5 p = X − (c − 1).
11.6 If $(p < 2^{L-1})$, then go to step 11.9.
11.7 Use Miller Rabin Algorithm to test whether p is prime.
11.8 If p is determined to be prime, then return SUCCESS and the values of
p, q and
(optionally) the values of domain parameter seed and counter.
11.9 offset = offset + n + 1.
12. Go to step 5.

Onaylı Özetleme Fonksiyonlarıyla Üretilen Olası Asal Sayılar
p ve q'nun İlgili "seed" İle Geçerliliğinin Test Edilmesi
INPUT:
1. **p, q** generated prime numbers
2. **domain parameter seed**: The seed used during the generation of p and q.
3. **counter**.

OUTPUT:

1. **state** SUCCESS or FAILURE.

ALGORITHM:

1. L = len(p).
2. N = len(q).
3. Check that the (L,N) pair is in the list of acceptable (L,N) pairs.

If the pair is not in the list, return FAILURE.

4. If (counter > 4095), then return FAILURE.
5. seedlen = length of domain parameter seed.
6. If (seedlen < (N)), then return FAILURE.
7. U = Hash(domain parameter seed) mod 2^N.
8. computed q = U v 2^{N-1} v 1.
9. Use Miller Rabin Algorithm(9) to test whether computed q is prime. If

(computed q ≠ q) or (computed q is not prime), then return FAILURE.

10. n =ceil[L/outlen] − 1.
11. b = L − 1 − (n * outlen). Where outlen is to be chosen as N.
12. offset = 1.
13. For i = 0 to counter do

13.1 For j = 0 to n do

Vj = Hash((domain parameter seed + offset + j) mod $2^{seedlen}$).

13.2 W = V_0+(V_1*2^{outlen})+...+(V_{n-1}*$2^{(n-1)*outlen}$)+((V_n mod 2^b)*$2^{n*outlen}$).

13.3 X = W + 2^{L-1}.

13.4 c = X mod 2q.

13.5 computed p = X − (c − 1).

13.6 If (computed p < 2^{L-1}), then go to step 13.9

13.7 Use Miller Rabin Algorithm(9) to test whether computed p is a prime.

13.8 If computed p is determined to be a prime, then go to step 15.

13.9 offset = offset + n + 1.

14. If ((i ≠ counter) or (computed p ≠ p) or (computed p is not a prime)),

then return FAILURE.

15. Return SUCCESS.

Miller-Rabin Olasılıksal Asal Sayı Testi

INPUT:

w: odd number to be tested

OUTPUT:

status: PROBABLY PRIME or COMPOSITE.

ALGORITHM:

1. Let a be the largest integer such that 2^a divides w − 1.
2. m = (w − 1)/2^a.
3. wlen = len(w).
4. For i = 1 to iterations = 50 do

4.1 Obtain a string b of wlen bits from an random number generator.

4.2 If ((b <= 1) or (b >= w − 1)), then go to step 4.1.

4.3 z = b^m mod w.

4.4 If ((z = 1) or (z = w − 1)), then go to step 4.7.

4.5 For j = 1 to a − 1 do.

4.5.1 z = z^2 mod w.

4.5.2 If (z = w − 1), then go to step 4.7.

4.5.3 If (z = 1), then go to step 4.6.

4.6 Return COMPOSITE.

4.7 Continue.

5. Return PROBABLY PRIME

Üreteç g'nin üretilmesi

INPUT:

1. **p, q**: The primes.
2. **domain parameter seed**: The seed used during the generation of p and q.
3. **index**: The index to be used for generating g. Index is represented as an

unsigned 8-bit integer.

OUTPUT:

1. **status**: The status returned from the generation routine, where status is

either SUCCESS or FAILURE.

2. **g**: The value of g that was generated.

ALGORITHM: Note: count is an unsigned 16-bit integer.

1. If (index is incorrect), then return FAILURE.
2. N = len(q).
3. e = (p − 1)/q.
4. count = 0.
5. count = count + 1.
6. If (count = 0), then return FAILURE.
7. U = domain parameter seed||"ggen"||index||count. where "ggen" is the hex number "0x6767656e" as the value of "ggen" and || means concatenation.
8. W = Hash(U).
9. g = We mod p.
10. If (g < 2), then go to step 5.
11. Return SUCCESS and the value of g.

Üretecin Doğrulanması:

INPUT:

1. **p**, **q:** The primes.
2. **domain parameter seed:** The seed used to generate p and q.
3. **index:** The index was used in generation of x. index is represented as an

unsigned 8-bit integer.

5. **g** The value of g to be validated.

OUTPUT:

1. **status:** The status returned from the generation routine, where status is

either **SUCCESS** or **FAILURE**.

ALGORITHM:

Note: index is an unsigned 16-bit integer.

1. If (index is incorrect ie larger than 255), then return FAILURE.
2. Verify that 1 < g < p. If not true, return FAILURE.
3. If (g^q ≠ 1 mod p), then return FAILURE.
4. N = len(q).
5. e = (p − 1)/q.
6. count = 0.
7. count = count + 1.
8. If (count = 0), then return FAILURE.
9. U = domain parameter seed||"ggen"||index||count.

10. W = Hash(U).
11. computed g = We mod p.
12. If (computed g < 2), then go to step 7.
13. If (computed g = g), then return SUCCESS, else return FAILURE.

Anahtar İkililerinin Üretilmesi

Anahtar ikilisinin üretilmesi iki yolla yapılabilir.

1. Extra Rastgele Bitler Kullanarak
2. Adayları test ederek

Ekstra Rastgele Bitler Kullanarak
INPUT: **p, q, g**: Domain parameters
OUTPUT: **status** and
key pair **(x, y)**.
ALGORITHM:
1. N = len(q);L = len(p).
2. If the (L,N) pair is invalid, then return an ERROR,
Invalid x, and Invalid y.
3. requested security strength = the security strength associated with the
(L,N) pair; see (4).
4. Obtain a string of N + 64 returned bits from a random number generator(RBG) with a security strength of requested security strength or more. If an ERROR status is returned, then return an ERROR, Invalid x, and Invalid y.
5. if random number is not an integer, convert returned bits to the (non-negative) integer c.
6. x = (c mod (q − 1)) + 1.
7. $y = g^x \bmod p$.
8. Return SUCCESS, x, and y.

Adayları test ederek
INPUT:
p, q, g: Domain parameters
OUTPUT:
1. **status:** The status returned from the key pair generation process. The
status will indicate SUCCESS or an ERROR.
2. **(x, y):** The generated private and public keys. If an error is encountered
during the generation process, invalid values for x and y should be returned,
as represented by Invalid x and Invalid y in the following specification. x
and y are returned as integers. The generated private key x is in the range
[1, q − 1], and the public key is in the range [1, p − 1].
ALGORITHM:
1. N = len(q);L = len(p).
2. If the (L,N) pair is invalid, then return an ERROR, Invalid x,
and Invalid y.
3. requested security strength = the security strength associated with the
(L,N) pair;
4. Obtain a string of N returned bits from an RBG with a security strength of requested security strength or more. If an ERROR status is returned,
then return an ERROR, Invalid x, and Invalid y.
5. . if random number is not an integer, convert returned bits to the (non-negative) integer c.
6. If (c > q − 2), then go to step 4.
7. x = c + 1.
8. $y = g^x \bmod p$.
9. Return SUCCESS, x, and y.

k'nın Üretilmesi
E-imza üretme algoritmasında kullanılan **k** nın üretilmesi iki yolla yapılacaktır.

1. Extra Rastgele Bitler Kullanarak
2. Adayları test ederek

Extra Rastgele Bitler Kullanarak
INPUT: **p, q, g**: Domain parameters
OUTPUT: **status** and **(k, k^{-1})**: If an error is encountered during the generation
process, invalid values for k and k^{-1} should be returned, as represented by
Invalid k and Invalid k^{-1} in the following specification. k and k^{-1} are returned
as integers. The generated k is in the range [1, q − 1].
ALGORITHM:
1. N = len(q);L = len(p).
2. If the (L,N) pair is invalid, then return an ERROR,
Invalid k, and Invalid k^{-1}.
3. requested security strength = the security strength associated with the
(L,N) pair; see (4).
4. Obtain a string of N + 64 returned bits from a random number generator with a security
strength of requested security strength or more. If an ERROR status is
returned, then return an ERROR, Invalid k, and Invalid k^{-1}.
5. Convert returned bits to the (non-negative) integer c.
6. k = (c mod (q − 1)) + 1.
7. k^{-1} = inverse of k modulo q and if inverse is not exist than return INVALID.
8. Return SUCCESS, x, and y.

Adayları test ederek
INPUT:
p, **q**, **g**: Domain parameters
OUTPUT:
1. **status:** The status returned from the key pair generation process. The
status will indicate SUCCESS or an ERROR.
2. **(k, k^{-1})**: If an error is encountered during the generation process, invalid
values for k and k^{-1} should be returned, as represented by Invalid k and
Invalid k^{-1} in the following specification. k and k^{-1} are returned as integers.
The generated k is in the range [1, q − 1].

ALGORITHM:

1. N = len(q);L = len(p).
2. If the (L,N) pair is invalid, then return an ERROR, Invalid k,
and Invalid k^{-1}.
3. requested security strength = the security strength associated with the
(L,N) pair;
4. Obtain a string of N returned bits from a random number generator with a security strength
of requested security strength or more. If an ERROR status is returned,
then return an ERROR, Invalid k, and Invalid k^{-1}.
5. Convert returned bits to the (non-negative) integer c
6. If (c > q − 2), then go to step 4.
7. k = c + 1.
8. k^{-1} = inverse of k modulo q and if inverse is not exist than return INVALID.
9. Return SUCCESS, k, and k^{-1}.

EK-2 DSA TEST SİSTEMİ ALGORİTMALARI

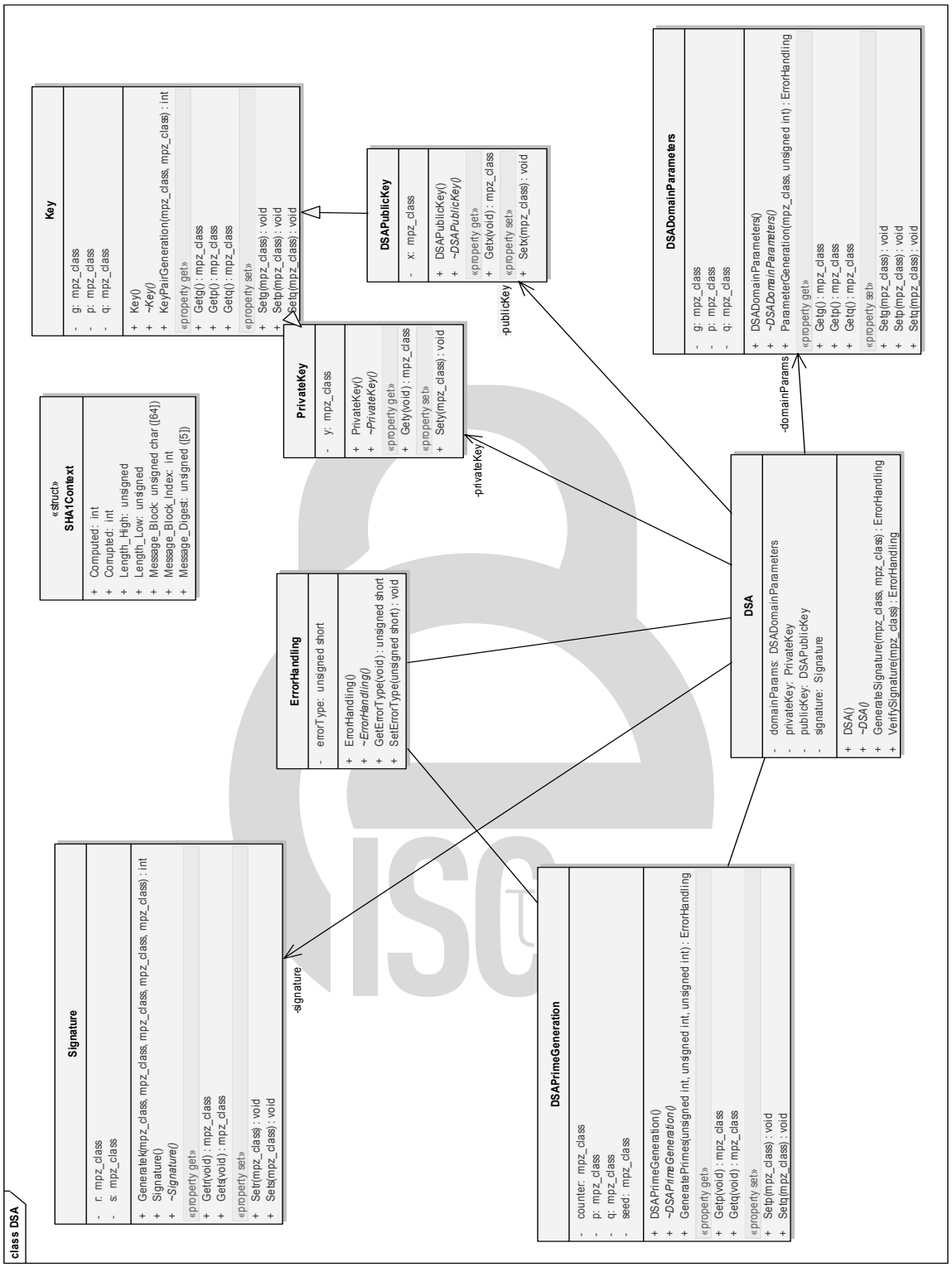
Index Calculus Yöntemi

INPUT : p, q, g, y
p an q prime numbers
q is a divisor of p-1.
g generator of F_q .
y a power of g which is between 1 and p − 1
b generator of F_p
OUTPUT : x a number between 1 and q − 1 such that $y = g^x \bmod p$.
ALGORITHM:
find e and z such that g = be mod p, y = bz mod p
1. F = {p_1, p_2, ..., p_t}, p_i = i th prime number, t = chosen exponent
2. form linear equations
   a. choose a random number k between 1 and p − 1
   b. $b^k \bmod p$
   c. b^k = Product(i=1 to t) p^{ci} c_i >= 0
   d. k = c_i*(Sum(i=1 to t)$\log_b p_i$
   e. apply 4 stps above until t + c lineq equations are obtained c is a parameter measures the probability of the system
   f. solve above linear system whose unknowns are $\log_b p$ by Gaussian elimination
3. find e:
   a. choose a random number between 1 and p − 1
   b. g * $b^k \bmod p$
   c. g * b^k = product(i=1 to t)p_i^{di} d_i >=0 if not go to 3.a back
   d. e = $\log_b g$ = Sum(i=1 to t)d_i*$\log_b p_i$ − k mod p
4. Step 2 and 3 is applied to find z
5. l = (p − 1)/(q − 1)
6. x = (z/l) *$(y/l)^{-1} \bmod q$
7. Return x

Pollard's rho algorithm for computing discrete logarithm

INPUT: a generator w of a cyclic group G of prime order n, and an element
s in G.
OUTPUT: the discrete logarithm x = $\log_w s$
ALGORITHM:
1. Set $x_0 = 1$, $a_0 = 0$, $b_0 = 0$.
2. For i = 1,2..., do the following:
   2.1 Using the quantities x_{i-1}; a_{i-1}; b_{i-1}, and x_{2i-2}; a_{2i-2}; b_{2i-2} computed previously,
   compute x_i; a_i; b_i and x_{2i}; a_{2i}; b_{2i} by modulo 3 casting of the whole set.
   2.2 If $x_i = x_{2i}$, then do the following: Set r = $b_i - b_{2i}$ mod n.
   If r = 0 then return failure;
   otherwise, compute x = $r^{-1}(a_{2i} - a_i) \bmod n$ and return x.

EK-3 DSA SİSTEMİNİN AYRINTILI YAZILIM YAPISI



TEŞEKKÜR

DSA sisteminin yazılım yapısındaki katkılarından dolayı Zaliha Yüce ve Çağdaş Çalık'a teşekkür ederiz.

KAYNAKLAR

[1] Knuth, Art of Computer Programming, 3rd edition, 1997.
[2] Chris Studholme ve Dr. Ian Blake'in alt-üstel zamanlı DLP çalışmaları http://www.cs.toronto.edu/~cvs/dlog/
[3] P.Q. Nguyen and I.E. Shparlinski, "The insecurity of the digital signature algorithm with partially known nonces," J. Cryptology, vol.15, pp.151–176, 2002.
[4] Coppersmith, Fast evaluation of logarithms in fields of characteristic two, *IEEE Trans. Inform.Theory* **IT-30** 1984, 587-594.
[5] IEEE 1363, Standard Specifications for Public-Key Cryptography, 2000
[6] Elektronik İmza ile İlgili Süreçlere ve Teknik Kriterlere İlişkin Tebliğ, 2005
[7] FIPS PUB 186-2 Dıgıtal Sıgnature Standard (DSS), U.S. Department Of Commerce / National Institute Of Standards And Technology
[8] http://gmplib.org/
[9] http://xyssl.org/code/source/sha2/
[10] http://xyssl.org/code/source/havege/