CRYMPIX: KRİPTOGRAFİK ÇOK-BASAMAKLI KÜTÜPHANE

Hüseyin HIŞIL[1]

[1]Bilgisayar Mühendisliği Bölümü
Mühendislik Fakültesi
İzmir Yüksek Teknoloji Enstitüsü, 35430, Urla, İzmir
[2] e-posta: huseyinhisil@iyte.edu.tr

Anahtar sözcükler: Sayı teorisi, kriptografi, cebir, modüler aritmetik, asimetrik kriptosistem.

ABSTRACT

This paper summarizes the experiences gained throughout the development of a cryptographic multi-precision library, CRYMPIX, coded in IS^3 *Labs. of Izmir Institute of Technology. CRYMPIX is mainly designed to supply code readability and portability. The whole work is achieved by detailed investigation of current algorithms and multi-precision libraries. The selected algorithms are discussed by means of efficiency and implementation techniques. The time measurements and speedup values are also included.*

1 GİRİŞ

Kriptografik ürünlerin genel anlamda başarısı alt yapı olarak kullandıkları kriptografik kütüphanelerde yatmaktadır. Kriptografik bir kütüphane gerek kodlamada gösterilmesi gereken duyarlılık ve gerekse seçilen algoritmaların matematiksel anlamda yetkinliği gibi birçok parametre ile geliştirilebilmektedir. Anılan parametrelerin çokluğu ve karmaşası kriptografik kütüphane yazımını güçleştirmektedir. Bu çalışma yukarıda tanımlandığı gibi, kriptografik ürün ortaya koymanın ilk basamaklarından biri olan kriptografik kütüphane yazımı üzerinedir. Halen birçok kriptografik kütüphane mevcut olup, bu çalışmada ortaya konan CRYMPIX kütüphanesi, söz konusu benzerlerinden bazı tasarım ilkeleri doğrultusunda ayrılmakta ve performans açısından geride kalmamaktadır. Farklı tasarım ilkelerinin uygulanması sonucunda ortaya çıkan, kod okunabilirliği, taşınabilirliği gibi kullanım kolaylıkları ve diğer yararlılılar, CRYMPIX ve diğer kütüphanelerin karşılaştırılması biçiminde aşağıda ortaya konmuştur.

Bu bildiride bir işlemci mimarisinde tek bir değişken içinde saklanamayacak kadar büyük olan sayılar *çok-basamaklı sayı* (multi-precision number) olarak anılmıştır. Asimetrik kriptografiye ilişkin tanımlanmış algoritmaların, uygulama ortamlarında çok-basamaklı sayılar ile çalışması gerektiği bilinmektedir. Örneğin, günümüz koşullarında RSA kriptosistemi 1024-4096 bit anahtar uzunluklarında kullanılmaktadır. 32 bitlik bir mimaride 4096 bitlik bir anahtar 128 kelimede (basamakta) saklanmaktadır.

Kriptografik uygulamaların bu gereksinimini karşılamak üzere çok-basamaklı sayı kütüphaneleri geliştirilmektedir. Bu kütüphanelerden en popüler olanları GNU GMP, Shamus Software MIRACL, PARI/GP, BigNum, Java BigInteger, Bouncy Castle, Magma, Maple, Mathematica, MuPAD gibi yazılımlardır. Anılan kütüphanelerin başarısı, temel olarak sundukları hizmetin hızı ile değerlendirilmekte; dolayısıyla, geliştirilen yazılımlarda mimariye dayalı kodlamanın vazgeçilemez olduğu görülmektedir. Bu yaklaşım, yüksek performanslı uygulamalara imkan vermesine karşın, farklı mimariler için kod geliştirme sürecinin tekrarlanması anlamına da gelmektedir.

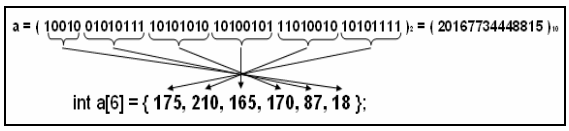
Bu çalışmada, kod geliştirme sürecinde harcanan eforun, hızdan ödün verilmeden en az seviyeye indirilmesi için uygulanabilecek yaklaşımlar incelenmiş, çok-basamaklı sayı kütüphanelerinin tasarımında karşılaşılan sorunlar ve çözümleri tartışılmıştır. Tercih edilen yaklaşımlar deneysel sonuçlarla desteklenmiş ve nihayet geliştirilmekte olan CRYMPIX kütüphanesi, diğer kütüphanelerle karşılaştırmalı olarak sunulmuştur.

2 TEMEL TASARIM İLKELERİ

Yazılım tasarım ve geliştirme süreçlerine yönelik tanımlanmış birçok kriter mevcuttur [10]. Çok-basamaklı sayı kütüphanelerinde öne çıkan temel tasarım kriterleri; sayıların temsil ediliş biçimi, programlama dili seçimi, bellek yönetimi, kod okunabilirliği ve taşınabilirliği, algoritma seçimi ve kullanım kolaylığıdır [1,9]. Başarılı bir çok-basamaklı sayı kütüphanesinden, anılan özelliklere ilişkin optimum yaklaşımları içermesi ve mevcut donanımı en verimli şekilde kullanarak, alternatif yazılımların sağladığı hızı sunması beklenir. İzleyen bölümlerde, anılan tasarım kriterlerine göre var olan kütüphanelerle, CRYMPIX kütüphanesi karşılaştırılmaktadır.

2.1 Sayıların temsil edilişi

Bu alanda geliştirilmiş tüm kütüphanelerde sayılar radiks gösterimine uygun olarak pozitif tamsayı dizilerinde tutulur [7]. Şekil 1'de görüldüğü gibi, sayının en anlamlı basamağı dizinin ilk elemanı olacak şekilde sıralanır. CRYMPIX de bu temsili kullanmaktadır.



Şekil 1. Sayıların temsil edilişi.

2.2 Programlama Dili seçimi

Çok-basamaklı sayı kütüphanelerinde genel olarak tercih edilen diller: Assembly, C, C++, Fortran ve Java'dır. Assembly değerlendirme dışında tutulursa, yazılımın hızında dil seçiminden çok, kullanılan derleyici ve programcının dili kullanma yetisi ön plana çıkmaktadır. Diğer dillerin assembly kadar hızlı olmayacağı açıktır.

CRYMPIX için programlama dili olarak ANSI C seçilmiştir. Bu kararda, C dilinin sunduğu işaretçi aritmetiği ve ileriye yönelik olarak dağıtık mimariye geçişte Message Passing Interface (MPI) teknolojisi ile entegrasyon kolaylığı hedefi etkili olmuştur. Genel performansı belirleyen iç döngülerin assembly dilindeki alternatiflerininse derleme zamanında kullanıcıya sunulması öngörülmüş; henüz uygulamaya alınmamıştır.

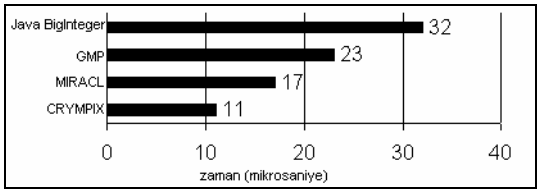
Tablo 1'de MIRACL 4.8, GMP 4.1.4, Java BigInteger ve CRYMPIX kütüphanelerinin çarpma işlemindeki performans değerleri karşılaştırılmıştır. Verilen değerler 1K, 2K, 4K ve 8K büyüklüğündeki rasgele seçilen 1000'er adet sayının çarpımı için farklı koşullarda geçen sürelerin ortalaması alınarak hesaplanmıştır. Deneyde seçimli assembly kullanımı, işletim sistemi, derleyici optimizasyonları ve donanım, değişken parametreler olarak kullanılmıştır. İşletim sistemleri olarak RedHat Linux 9.0 ve Microsoft Windows XP SP2 baz alınmıştır. Derleyici olarak GNU GCC_v3.2.2 seçilmiş ve optimizasyon 0, 1 ve 2 derecelerinde ölçüm yapılmıştır. Donanım seçimleri işlemci bazında yapılarak Intel Centrino 1400Mhz, IBM RISC RS/6000 133Mhz ve Intel P4 1700Mhz işlemcileri kullanılmıştır. IBM RS/6000 133Mhz mimarisinde sadece Linux İşletim sistemi (Yellow Dog 2.3) üzerinde ölçüm yapılmıştır. Java BigInteger kütüphanesinin performans testi ise Sun Microsystems, Inc. tarafından geliştirilen Java2 Standard Development Kit (J2SDK) v1.4.2 üzerinde gerçekleştirilmiştir. Belirlenen farklı koşullarının sonuçlar arasındaki oranı etkilemediği görülmüştür. Olası tüm kombinasyonları içeren tablo geniş yer tuttuğundan, bu bildiride Intel Centrino 1400Mhz işlemci, RedHat Linux 9.0 işletim sistemi ve GNU GCC_v3.2.2 derleyicisiyle elde edilen ölçümler verilmiştir.

Tablo 1. Kütüphanelerin çarpma işlemindeki performans karşılaştırması. (mikrosaniye).

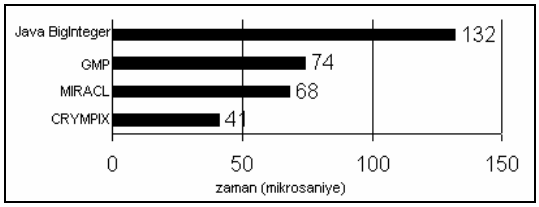
	CRYMPIX	**MIRACL**		**GMP**		**Java**
	C	**C**	**C+Asm**	**C**	**C+Asm**	**BigI.**
1K	11	17	6	23	4	32
2K	41	68	26	74	15	132
4K	133	277	104	235	47	512
8K	410	1097	411	731	154	2630

Assembly dili hariç tutulduğunda ve yalnız C dili kullanıldığında CRYMPIX diğer kütüphanelerden geri kalmamaktadır. Performanstaki farkın nedeni MIRACL kütüphanesinin çarpma işleminde sadece kağıt-kalem algoritmasının kullanılmasından ileri gelmektedir. Anılan kütüphane Karatsuba/Comb metodlarını içerse de bu hızlı yaklaşımları modüler üs alma gibi daha pahalı işlemlerde kullanmaktadır. GMP kütüphanesi ise salt C versiyonunda güncel derleyiciler tarafından desteklenen çift basamaklı (double precision) çarpma işlemini kullanmamakta ve bu nedenle yavaş kalmaktadır.

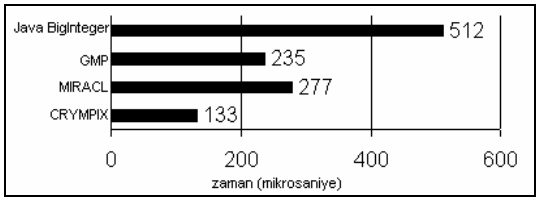
Java BigInteger kütüphanesinin performansı, üzerinde çalıştırıldığı Java Sanal Makinesine bağlı olsa da, Java BigInteger kütüphanesi her durumda alternatiflerinden yavaş kalmaktadır. Şekil 2, 3, 4 ve 5'te sırası ile 1K, 2K, 4K ve 8K'lık deney uzayı üzerinde ölçülen performans değerleri karşılaştırmalı olarak sunulmuştur. Bu değerler salt C dili kullanıldığında elde edilen verilerdir.



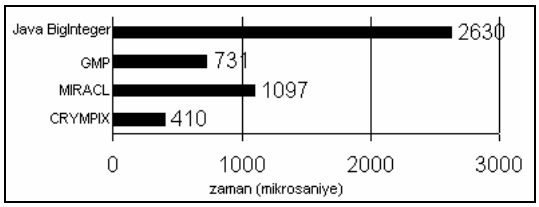
Şekil 2. 1K büyüklüğündeki iki sayının çarpımı için gereken süre.



Şekil 3. 2K büyüklüğündeki iki sayının çarpımı için gereken süre.



Şekil 4. 4K büyüklüğündeki iki sayının çarpımı için gereken süre.



Şekil 5. 8K büyüklüğündeki iki sayının çarpımı için gereken süre.

2.3 Bellek Yönetimi

Kriptografik uygulamaların tamamının modüler aritmetik ilkelerine göre tasarlandığı göz önüne alındığında, aritmetik işleme tabi tutulan sayıların ne kadar büyüklüğe ulaşacağı kesin olarak bilinmektedir. Bu durumda sayılar için ayrılan bellek alanlarının büyüklüklerini sabitlemek, bellek parçalanmasını önleyecektir. Uygulamanın içinde çekirdek olarak görev yapacak bir katman oluşturulması ve anılan çekirdekte özel olarak bellek yönetimi yapılması, uygulamanın hızını arttıracaktır. Yapılan deneylerde statik bellek yönetiminin, genel performansı ihmal edilemeyecek derecede arttırdığı görülmüştür. Bu tip bir yaklaşım gerçek zamanlı sistemlerde tercih edilmektedir.

MIRACL kütüphanesinin çalışması bir sistem olarak tasarlanmış ve sayıların büyüklüğünün sistem başlangıcında belirtilmesi şart koşulmuştur. Bellek ayırımları malloc fonksiyonuyla her sayı için ayrı ayrı yapılmaktadır. Kullanıcı fonksiyonlarındaki parametrelere atanan sayılar için bellek alanının önceden ayrıldığı varsayılmıştır. MIRACL, kullanıcıdan soyutlanan iç kısımlarında, çeşitli hesaplamalarda kullanmak üzere sistem başlangıcında bir çalışma alanı da oluşturmaktadır. Bu yaklaşımla lokal değişkenler için sürekli bellek ayırımı ve boşaltımı önlenmiştir. Şekil 6, MIRACL kütüphanesinde toplamayı göstermektedir.

```
mirsys(100, 2); //Max bits per num. and num. base
big a = mirvar(0);
big b = mirvar(0);
big c = mirvar(0);
..................
add(a, b, c); // c = a + b
..................
mirkill(a);
mirkill(b);
mirkill(c);
```

Şekil 6. MIRACL ile toplama.

GMP kütüphanesinde bellek alanı malloc fonksiyonu ile ayrılmaktadır. Sayıların büyüklüğüne ilişkin herhangi bir sınır konmamıştır. Sayı büyüdükçe bellek alanı genişlemektedir. Bu yaklaşım bellek parçalanmasına açık olduğu için GMP kütüphanesini yavaşlatmaktadır fakat kullanılmayan sayıların bellek alanı sisteme hemen geri dönmemekte, ihtiyaç halinde yeniden kullanılmaktadır. Dolayısıyla yazılımın alternatiflerinden yavaş kalması önlenmiştir. Bu yaklaşımın kabul edilemeyeceği durumlar içinse tasarımda özel bellek ayırımı adında bir açık nokta bırakılmıştır. Şekil 7 GMP toplama örneğini vermektedir.

```
mpz_t a, b, c;
mpz_init(a);
mpz_init(b);
mpz_init(c);
..................
mpz_add(c,a, b); // c = a + b
..................
mpz_clear(a);
mpz_ clear (b);
mpz_ clear (c);
```

Şekil 7. GMP ile toplama.

Java BigInteger API nesne yaklaşımına uygun olarak tasarlanmıştır. Sayıların bellekte kapladıkları alan için herhangi bir sınırlama getirilmemiştir. Bu kütüphanenin hızını Java Sanal Makinasının performansı ve Garbage Collector belirlemektedir. Alternatiflerine kıyasla kullanımı basit olmasına karşın, performans bakımından rakiplerinin gerisinde kalmaktadır. Şekil 8, Java ile toplamayı vermektedir.

```
import java.math.BigInteger;
BigInteger a, b, c;
a = new BigInteger();
b = new BigInteger();
c = a.add(b); // c = a + b
..................
```

Şekil 8. Java BigInteger ile toplama.

CRYMPIX kütüphanesi yukarıda tartışılan ilkeler doğrultusunda kendi bellek yönetimini yapmaktadır. Sayılar için bellek ayırımı ve bu sayıların sisteme iade edilmesi hızlı yapıldığı için MIRACL'da benimsenen çalışma alanı kullanılmamıştır. Kriptografik perspektife uygun olarak sayı uzunlukları sabit tutulmuştur. Herhangi bir çöp toplama (garbage collector) mekanizması olmadığından, sayıların yaşam döngüsünden tamamen programcı sorumlu kılınmıştır. CRYMPIX toplama yöntemi şekil 9'da verilmiştir.

```
CZ a, b, c;
crympix_init(100, 20); //Max columns per number.
...
a = cz_init();
b = cz_init();
c = cz_init();
...
cz_add(c, a, b); // c = a + b
...
cz_kill(a);
cz_kill(b);
cz_kill(c);
...
```

Şekil 9. CRYMPIX ile toplama.

2.4 Kod Okunabilirliği ve Taşınabilirliği

Kod okunabilirliği birçok kütüphanede ikinci planda bırakılırken, CRYMPIX kütüphanesinde okunabilirlik ve taşınabilirlik ön planda tutulmuştur. Kod taşınabilirliğine iki farklı biçimde yaklaşılabilinir: İlkinde kodda farklılaştırma gerektirecek her türlü unsur için tüm kod baştan yazılır; bu GMP kütüphanesinde uygulanmıştır. Bu yaklaşımda kod okunurluğu ve taşınabilirliği korunsa da koda destek sağlanması zorlaşır. Anılan probleme ilişkin diğer çözüm ise kodda farklılaşma gerektiren kısımlar için jenerik ifadeler tanımlanması ve bu tip durumların makrolar halinde ele alınmasıdır. CRYMPIX kütüphanesinin tüm tasarım ilkesi bu esas üzerine oturtulmuş olup, buna ilişkin örnek şekil 10'da görülmektedir.

```
DPUP carry;
POS i, j;
BIG z;

z = big_init();
z->len = a->len + b->len;
carry.spu[HIGH] = 0;
for(j = 0; j < b->len; j++){
        DPU_MULTIPLY_WITH_CARRY(
                carry.dpu,
                a->num[0],
                b->num[j],
                carry.spu[HIGH]
        );
        z->num[j] = carry.spu[LOW];
}
z->num[j] = carry.spu[HIGH];
for(i = 1; i < a->len; i++){
        carry.spu[HIGH] = 0;
        for(j = 0; j < b->len; j++){
                DPU_MULTIPLY_AND_ADD_WITH_CARRY(
                        carry.dpu,
                        a->num[i],
                        b->num[j],
                        z->num[i + j],
                        carry.spu[HIGH]
                );
                z->num[i + j] = carry.spu[LOW];
        }
        z->num[i + j] = carry.spu[HIGH];
}
if((z->num[z->len - 1] == 0) && (z->len > 1)){
        z->len--;
}
z->sign = a->sign * b->sign;
return z;
```

Şekil 10. CRYMPIX kütüphanesinde kağıt-kalem algoritması ile çarpma kod örneği.

Kod içindeki `DPU_MULTIPLY_AND_ADD_WITH_CARRY (r, a, b, c, d)` ifadesi bir makrodur ve r = a * b + c + d ifadesine karşılık gelmektedir. Bu makro çarpma fonksiyonun hızını belirlemektedir. Farklı mimariler için sadece bu ifadeyi baştan yazmak kod okunabilirliği ve taşınarbilirliğini etkilemeyeceği gibi koda destek sağlamada da programcıya kolaylık sağlamaktadır. Bu yaklaşım, performasın önem taşıdığı noktalar göz önüne alınarak uygulanmıştır. Projenin ileryen aşamalarında iç döngüleri içerek bir makro katmanın daha tasarıma eklenmesi öngörülmüştür.

2.5 Algoritma seçimi

Tüm kütüphanelerin benzer algoritmaları tercih ettiği görülmüştür. Anılan kütüphanelerin birçoğu henüz geliştirilme aşamasında olduğundan, algoritma tercih koşulları tartışılmıştır.

2.5.1 Toplama, Çıkarma ve Kaydırma İşlemleri

Toplama ve çıkarma işlemi kağıt-kalem metodu ile yapılır. Sayılar en az anlamlı basamaklarından başlanarak işleme alınır. Oluşan elde/borç değerleri tespit edilerek diğer basamaklara aktarılır. Bazı mimarilerde elde/borç işlemleri otomatik olarak yapılır, fakat bu özelliğin kullanılabilmesi için assembly dilinin kullanması gerekmektedir.

Kaydırma işlemi için programlama dilinin sunduğu bit düzeyinde operatörler kullanılır. Kaydırma işlemi sayı tabanının katı ise işlem, basamakların taşınması şeklinde yapılır. Genel olarak, anılan iki çözüm birarada kullanılmaktadır.

2.5.2 Çarpma İşlemi

Bilinen tüm asimetrik kriptosistem uygulamalarının performansı, çarpma işleminin hızı ile orantılıdır [7]. Çarpma işleminde kullanılan algoritmalar tablo 3'de kompleksite değerleri ve kullanım aralıkları ile verilmiştir.

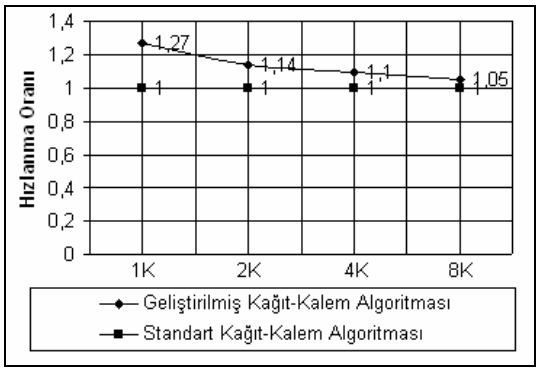
Tablo 2. Çok-basamaklı çarpma işleminde kullanılan algoritmalar.

Algoritma	Kompleksite	Kullanımı
Kağıt-Kalem	$O(n^2)$	0-1 Kb
Karatsuba	$O(n^{1.585})$	1-6 Kb
Tom-Cook	$O(n^{1.465})$	6-24 Kb
FFT	$O(n^{1.4})$	24Kb ve üstü

Kriptografik uygulamalarda Kağıt-Kalem algoritması [7,5] ve Karatsuba algoritması [5,6] yoğun olarak kullanılmaktadır. Fast Fourier Transform (FFT) tekniği asimptotik olarak daha hızlı olsa da kriptografik gereksinimler göz önüne alındığında tercih edilmektedir. CRYMPIX'te de kullanılan Kağıt-Kalem algoritmasında ilk işlem, sonucun yazılacağı bellek alanına sıfır atamaları ile başlar. CRYMPIX'te bu alanı sıfırlamak yerine devirsiz çarpma ile işleme başlanmaktadır. Bu yaklaşım ile 1-10 basamaklı sayılarda daha iyi sonuç alınmıştır. Ancak, sayılar büyüdükçe anılan yaklaşımın önemi azalmaktadır. Tablo 2'de 64-512 bit uzunluğundaki sayılar için ölçülen değerler sunulmuştur. Hızlanma oranları (speed-up) ise şekil 11'de sunulmuştur. Deney ortamında bölüm 2.2'de değinilen kriterler baz alınmıştır.

Tablo 3. CRYMPIX kütüphanesinde uygulanan Geliştirilmiş Kağıt-Kalem Çarpması (G.K.K.Ç.) ile Standart Kağıt-Kalem Çarpması (S.K.K.Ç.) uygulamasının performans karşılaştırması. (mikrosaniye).

Sayı büyüklüğü	64	128	256	512
G.K.K.Ç.	181	392	1153	4175
S.K.K.Ç.	213	447	1277	4425



Şekil 11. Crympix kütüphanesinde uygulanan Geliştirilmiş Kağıt-Kalem Çarpması'nın, Standart Kağıt-Kalem Çarpması üzerine hızlanma oranı.

2.5.3 Bölme İşlemi

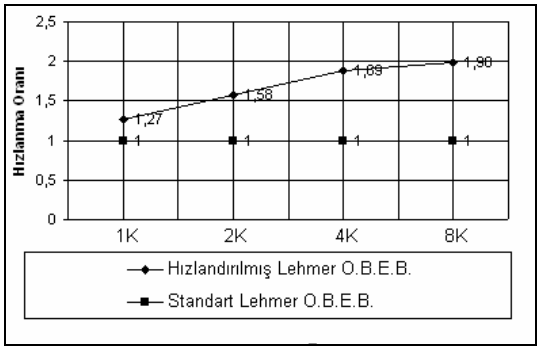
Bölme işleminde kağıt-kalem algoritması Knuth tarafından tanımlanmıştır [5]. Güncelde bölme işleminde kullanılan sayılar 50 basamağı aştığında Moenck, Borodin, Jebelean, Burnikel and Ziegler tarafından geliştirilen böl-ve-yönet algoritması tercih edilmektedir [2,8]. CRYMPIX'te kağıt-kalem algoritması uygulanmıştır.

2.5.4 Ortak Bölenlerin En Büyüğü İşlemi (O.B.E.B.)

Ortak bölenlerin en büyüğünü hesaplamak için bilinen temel algoritma, $O(n^2)$ kompleksitesine sahip Euclid'in klasik algoritmasıdır, fakat anılan algoritma ardarda bölme işlemini kullandığı ve bit düzeyinde çalıştığı için yavaş kalmaktadır. Aynı kompleksiteye sahip diğer algoritmalar, basamak düzeyinde işlem yaparak daha hızlı çalışan hatta paralelleştirilebilen uygulamalara imkan vermektedir. Anılan algoritmalardan, kongruans çözümlerinde kullanılan Genişletilmiş Euclid algoritmasına en uygun olanı, Lehmer'in yaklaşımıdır [7]. Bu algoritma sayılar, en anlamlı basamaklarından başlanıp, döngü başına yarım basamak kadar azaltılır. Anılan algoritmayı hızlandırmak için izlenebilecek üç yöntem Jebelean tarafından önerilmiştir [3]. CRYMPIX'te uygulanan Genişletilmiş Euclid algoritmasında anılan iyileştirmelerin ilk ikisi dikkate alınmış ve Jebelean tarafından öngörülen değerlere ulaşılmıştır. Bu uygulamaya ilişkin ölçülen değerler tablo 4'de, elde edilen hızlanma değerleri ise şekil 12'de verilmiştir.

Tablo 4. Standart Lehmer O.B.E.B. algoritması ve geliştirilmiş halinin performans karşılaştırması. (mikrosaniye).

Bit Uzunluğu	1024	2048	4096	8192
Standart Lehmer	201	557	1746	6228
Geliştirilmiş Lehmer	158	351	921	3131

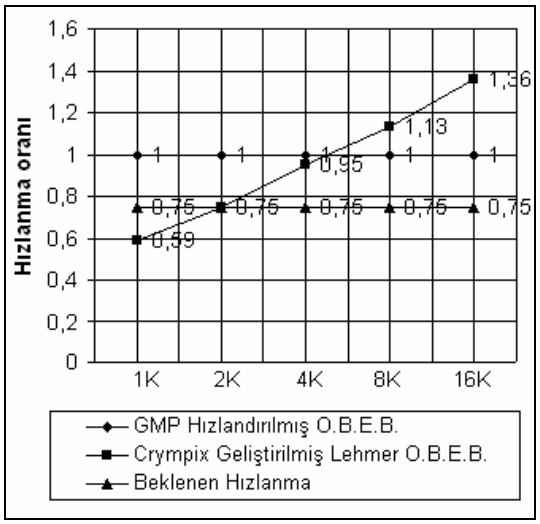


Şekil 12. Crympix kütüphanesinde uygulanan Geliştirilmiş Lehmer O.B.E.B.'in, Standart Lehmer O.B.E.B. uygulaması üzerine hızlanma oranı.

O.B.E.B hesaplamasında bilinen en hızlı algoritma Jebelean ve Weber'in eşzamanlı özgün çalışmalarında ortaya konmuştur [4]. Anılan algoritma, Hızlandırılmış/Genişletilmiş O.B.E.B. olarak bilinmektedir. Bu algoritma sistolik parallelleştirmeye uygundur ve ayrıca SIMD yapılarında da Lehmer'in algoritmasından daha hızlı çalışmaktadır. Anılan algoritmada sayılar Lehmer algoritmasının tersine, en az anlamlı basamaklarından başlanarak işleme alınmakta olup, bu algoritmanın CRYMPIX'te uygulanması halen devam etmektedir. Anılan iki yaklaşımın performans karşılaştırması tablo 5'de sunulmuştur. Bu karşılaştırmada Hızlandırılmış/Genişletilmiş O.B.E.B. algoritması için GMP kütüphanesinden, Lehmer O.B.E.B. algoritması içinse CRYMPIX kütüphanesinden yararlanılmıştır. Teorik olarak, Hızlandırılmış/Genişletilmiş O.B.E.B.'in Lehmer O.B.E.B'ten 1,33 kat hızlı çalışması beklenir. [3, 4]. CRYMPIX kütüphanesinin beklenenden daha hızlı çalışması, tasarım ve/veya uygulama platformları açısından, GMP kütüphanesinden ayrıldığı noktayı göstermektedir. Hızlanma değerleri şekil 13'de verilmiştir.

Tablo 5. O.B.E.B. hesaplamada kullanılan iki yaklaşımın performans karşılaştırması. (mikrosaniye).

Sayı büyüklüğü	1024	2048	4096	8192	16384
CRYMPIX O.B.E.B.	149	351	922	2806	8535
GMP O.B.E.B.	89	265	880	3190	11672



Şekil 13. Crympix kütüphanesinde uygulanan Geliştirilmiş Lehmer O.B.E.B.'in, GMP kütüphanesindeki Hızlandırılmış O.B.E.B. uygulaması üzerine hızlanma oranı.

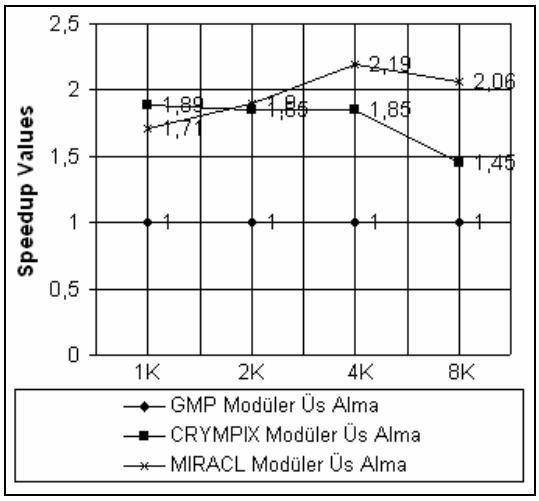
2.5.5 Modüler Üs Alma

Modüler Üs Alma işlemi anılan işlemler içinde en pahalı olanıdır. Bu işlem için kullanılan ve ardışık kare alma (successive squaring) olarak bilinen temel algoritmaya birçok farklı yaklaşım geliştirilmiştir. İyi bir modüler üs alma uygulaması, bilinen yaklaşımların bir arada kullanılması ile oluşur [6]. CRYMPIX, anılan algoritma ile birlikte soldan sağa üs tarama (left-to-right exponent scanning), değişken uzunluklu pencere kaydırması (variable-length-window-sliding) ve Montgomery çarpmasını kullanmaktadır. Çarpma işlemine kullanılan algoritmalar ise 2.5.2'de ele alınmıştır. Tablo 6'da 1K, 2K, 4K ve 8K büyüklüğündeki sayılarla gerçekleştirilen modüler üs alma işleminin GMP, CRYMPIX, MIRACL kütüphaneleri için performans değerleri sunulmuştur. Hızlanma oranları ise şekil 14'de verimiştir.

MIRACL kütüphanesinin hızı Montgomery sadeleştirmesinde (Montgomery REDC) kullanılan özyinelemeli (recursive) yarım çarpma tekniğinden ileri gelmektedir. 8K değerinde MIRACL kütüphanesindeki hızlanma oranındaki azalma ise GMP kütüphanesinin bu değerde ToomCook algoritmasını kullanması ile oluşmaktadır. Modüler üs alma işleminin CRMYPIX kütüphanesindeki implementasyonu halen sürmektedir.

Tablo 6. Kütüphanelerin modüler üs alma işlemindeki performans karşılaştırması. (milisaniye).

Sayı büyüklüğü	1K	2K	4K	8K
GMP Mod. Exp.	54	389	2841	16734
CRYMPIX Mod. Exp.	28	210	1532	11525
MIRACL Mod. Exp.	31	204	1298	8132



Şekil 14. CRYMPIX ve MIRACL kütüphaneleri ile gerçekleştirilen modüler üs alma işleminin GMP kütüphanesi ile gerçekleştirilen modüler üs alma işlemi üstüne hızlanma oranı.

3 SONUÇ

Bu çalışmada, CRYMPIX olarak isimlendirilen yeni bir kriptografik çok-basamaklı sayı kütüphanesi geliştirilmiştir. Tasarım ilkelerinin belirlenmesinde kod okunabilirliği ve yönetilebilirliği önemsenmiştir. Ortaya çıkarılan ürün halen geliştirilme aşamasındadır. CRYPMIX'in gelinen noktada; yukarıda tanımlanmış olan kod okunabilirliği-yönetilebilirliği-taşınabilirliği gibi temel tasarım parametrelerinden ödün vermeksizin, en performanslı kütüphanelere yakın performansta çalıştığı ve hatta bazı koşullarda daha da iyi olabildiği gözlenmiştir. Projenin, ilerleyen aşamalarda dağıtık mimarilere adaptasyonu planlanmaktadır.

KAYNAKLAR

[1] Bosselaers A., A Fast and Flexible Software Library for Large Integer Arithmetic, Werkgemeenschap voor Informatie- en Communicatietheorie, pp. 82-89, 1994.

[2] Burnikel C., Fast Recursive Division, Max-Planck-Institut fuer Informatik Research Report MPI-I-98-1-022.

[3] Jebelean T., Improving the Multiprecision Euclidean Algorithm, Proceedings of DISCO'93, Springer-Verlag LNCS 772, (1993) 45-58, 1993.

[4] Jebelean T., A Generalization of the Binary GCD Algorithm, ISSAC 93, pp. 111-116, June 1993.

[5] Knuth D.E., The Art of Computer Programming, Volume 2, Seminumerical

Algorithms, 3rd edition, Addison-Wesley, 1998.

[6] Koc C.K., High Speed RSA Implementation, RSA Laboratories, TR201, 1994.

[7] Menezes A., Handbook of Applied Cryptography, CRC Press, 1996, ISBN: 0-8493-8523-7.

[8] Paul Zimmermann, A Proof of GMP Fast Division and Square Root Implementations, September 2000.

[9] Tiplea F.L., MpNT: A Multi-Precision Number Theory Package, Number-Theoretic Algorithms (I), May 2003, TR 03-02, ISSN 1224-9327.

[10] Whitten L.J., Systems Analysis and DesignMethods 5th Edition, McGraw Hill., ISBN-0071180702.