İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

RSA ALGORİTMASININ ÜÇ ÇEKİRDEKLİ LEON3 İŞLEMCİSİ TABANLI SİSTEM ÜZERİNDE HATA ENJEKTE ETME ATAĞINA DAYANIKLI GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

İsmail DEMİR

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Mart 2018

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

RSA ALGORİTMASININ ÜÇ ÇEKİRDEKLİ LEON3 İŞLEMCİSİ TABANLI SİSTEM ÜZERİNDE HATA ENJEKTE ETME ATAĞINA DAYANIKLI GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

İsmail DEMİR
(504121387)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Sıddıka Berna Örs YALÇIN

Mart 2018

İTÜ, Fen Bilimleri Enstitüsü'nün **504121387** numaralı Yüksek Lisans Öğrencisi **İsmail DEMİR**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"RSA ALGORİTMASININ ÜÇ ÇEKİRDEKLİ LEON3 İŞLEMCİSİ TABANLI SİSTEM ÜZERİNDE HATA ENJEKTE ETME ATAĞINA DAYANIKLI GERÇEKLENMESİ"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Doç. Dr. Sıddıka Berna Örs YALÇIN** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Yrd. Doç. Dr. Şerif Bahtiyar** İstanbul Teknik Üniversitesi	.....
	Doç.Dr. Burak Kelleci Okan Üniversitesi	.....

Teslim Tarihi : 19 Mart 2018
Savunma Tarihi : 28 Mart 2018

Eşime,

ÖNSÖZ

Tez çalışmam boyunca benden yardımını esirgemeyen değerli hocam Doç. Dr. Sıddıka Berna Örs YALÇIN'a ve manevi desteğini her zaman hissettiğim eşime teşekkürü borç bilirim.

Mart 2018

İsmail DEMİR
Elektronik Mühendisi

İÇİNDEKİLER

Sayfa

KISALTMALAR

DES	**:** Data Encryption Standard
AES	**:** Advanced Encryption Standard
ECC	**:** Elliptic Curve Cryptography
RSA	**:** Rivest Shamir Adleman
FPGA	**:** Field Programmable Gate Array
VHDL	**:** Very High Speed Integrated Circuits Hardware Description Language
TMR	**:** Triple Modular Redundancy
SMP	**:** Symmetric Multi Processing
AMP	**:** Asymmetric Multi Processing
SPARC	**:** Scalable Processor Architecture
SPARC OPENPROM	**:** SPARC Programmable Read Only Memory
FPU	**:** Floating Point Unit
GRLIB	**:** Gaisler Library
GRMON	**:** Gaisler Monitor
IP	**:** Intellectual Property
CRT	**:** Chinese Remainder Theorem
CRC	**:** Cyclic Redundancy Check

ŞEKİL LİSTESİ

Sayfa

RSA ALGORİTMASININ ÜÇ ÇEKİRDEKLİ LEON3 İŞLEMCİSİ TABANLI SİSTEM ÜZERİNDE HATA ENJEKTE ETME ATAĞINA DAYANIKLI GERÇEKLENMESİ

ÖZET

Günümüzde kriptografik sistemler verinin gizliliğini, bütünlüğünü ve kaynağını doğrulamak için kullanılmaktadır. Kriptografik sistemler simetrik ve asimetrik kripto sistemleri olmak üzere iki başlık altında toplanabilir. Simektrik kripto sistemlerinde şifreleme ve şifre çözme işleminde aynı anahtarın kullanılması gerekmektedir. Bu nedenle haberleşmeden önce anahtar güvenli bir kanal üzerinden paylaşılmalıdır. Bu durum uygulamada zorluklara neden olmaktadır. İlk olarak Diffie-Hellman'ın önerdiği asimetrik kripto sistemleri bu zorluğun üstesinden gelmektedir. Asimetrik kripto sistemlerinde şifreleme ve şifre çözme işlemi için açık ve kapalı anahtar olmak üzere iki farklı anahtar kullanılmaktadır ve sadece açık anahtarın paylaşılması haberleşme için yeterlidir. Bu durumda güvenli kanal kullanılması gerekmemektedir. Asimetrik kripto sistemlerinde en yaygın kullanılan algoritma Rivest Shamir Adleman (RSA) algoritmasıdır. Hem simetrik hem de asimetrik kripto sistemlerine matematiksel altyapılarını hedef alan ve gerçeklemeyi hedef alan yan kanal analizi yöntemleri ile atak yapılmaktadır. Başlıca yan kanal analizi atakları zamanlama analizi, güç analizi ve hata enjekte etme ataklarıdır.

Bu çalışmanın amacı RSA algoritmasının geçici bit hataları oluşturmak şeklinde gerçeklenen hata enjekte etme ataklarına dayanıklı olacak şekilde gerçeklenmesidir. Bu amaçla gerçeklenebilir bir işlemci olan Leon3 kullanılarak 3 çekirdekli bir kırmık üstü sistem tasarlanmış ve sentezlenerek Digilent firmasının Nexys4DDR isimli Sahada Programlanabilir Kapı Dizisi (Field Programmable Gate Array - FPGA) geliştirme kartı üzerinde çalıştırılmıştır. Daha sonra Linux işletim sistemi Leon3 için derlenerek tasarlanan sistem üzerinde koşması sağlanmıştır. Son olarak C++ programlama diliyle RSA algoritması yazılmıştır. Her bir işlemci çekirdeği üzerinde aynı şifre çözme işlemi yapılarak yazılım tabanlı Üçlü Modül Yedekleme (Triple Modular Redundancy – TMR) gerçeklemesi yapılmıştır. Bu gerçeklemede her bir işlemci üzerinde çalışan RSA algoritması belli bir zaman farkı ile çalışmaya başlamaktadır. Bu yöntemle geçici bit bozulmalarından etkilenerek yanlış sonuç üreten bir işlemci olması durumunda diğer iki işlemcinin ürettiği sonuç doğru kabul edilecektir.

Gerçeklenen sistem üzerinde RSA algoritmasına kapalı anahtarın rastgele bir bitinin yanlış okunmasına neden olan geçici hata oluşturarak atak yapılmıştır. Tek işlemci kullanılması durumunda yanlış şifre çözme sonucu üretilmiş ve kapalı anahtarın hata yaptırılan biti elde edilmiştir. 3 işlemci kullanılarak atak gerçeklendiğinde ise işlemcilerin RSA algoritmasını başlatmaları arasındaki zaman farkı enjekte edilen hatanın süresine göre makul bir şekilde büyük olunca sistemin hatalı olan şifre çözme işlemi sonucunu elediği görülmüştür. Ayrıca önerilen yöntem tek bir iş parçacığının şifre çözme yaptığı durum ve yine benzer şekilde TMR kullanan ancak

iş parçacıklarının çalışmaya başlama zamanları arasında süre farkı olmadığı ve aynı kapalı anahtarın birer kopyasının iş parçacıkları tarafından kullanıldığı durum ile karşılaştırılmıştır. Önerilen yöntemin avantajları ve dezavantajları değerlendirilmiştir. Sonuçta Döngüsel Artıklık Denetimi (Cyclic Redundancy Check – CRC) ile kapalı anahtarın periyodik olarak kontrol edilmesi yöntemiyle birlikte kullanıldığında önerilen yöntemin RSA'i hata enjekte etme atağına karşı koruduğu aynı zamanda hata düzeltme de yapabildiği gösterilmiştir.

IMPLEMENTATION OF RSA ALGORITHM RESISTANT TO FAULT INJECTION ATTACK ON SYSTEM BASED ON TRIPLE CORE LEON3 PROCESSOR

SUMMARY

Nowadays, cryptographic systems are used to verify the confidentiality, integrity and source of the data. Cryptographic systems can be grouped under two categories as symmetric and asymmetric crypto systems. In symmetric crypto systems,the screet key needs to be used for both encryption and decryption. For this reason, the screet key must be shared over a secure channel before communication. This situation creates difficulties in practice. Asymmetric crypto systems proposed by Diffie-Hellman overcome from this difficulty. Asymmetric crypto systems use two different keys for encryption and decryption named public and private and sharing only the public key is sufficient for communication. In this case it is not necessary to use a secure channel. The most widely used algorithm for asymmetric crypto systems is the Rivest Shamir Adleman (RSA) algorithm. Both symmetric and asymmetric cryptosystems are attacked by attacks which target mathematical infrastructures and side channel analysis attacks which target implementation. Major side channel analysis attacks are timing analysis, power analysis and fault injection attacks.

Multi-core processors offer higher processing power and lower operation frequency than single-core processors. Multi-core processors also reduces system cost due to lower power consumption and less space occupancy. Multiprocessing consists of Asymmetric Multi Processing (AMP) and Symmetric Multi Processing (SMP) modes. In SMP mode, one operating system manages all processor cores and applications can use any of the processor cores. In AMP mode, each core runs a different operating system or a copy of the same operating system. In this thesis, Leon3 processor, a synthesizable processor designed by Gaisler Aeroflex, is used in SMP mode with 3 cores. The operating system running on Leon3 cores is chosen as Linux operating system compiled with SMP support. The Pthreads library is used to take advantage of multithreading support of Linux. Pthreads library provides facilities such as writing code that multiple threads work in the same program and inter-thread synchronization mechanisms. It also allows configuration of the desired thread so that it runs on the desired processor core.

The purpose of this work is to implement the RSA algorithm to be resistant to fault injection attacks, which are implemented in the form of temporary bit errors on the secret key. For this purpose, a system on chip was designed and synthesized using 3 Leon3 core and it was run on Digilent's Nexys4DDR Field Programmable Gate Array (FPGA) development card. Then Linux operating system was compiled for the Leon3 microprocessor and it run on the Leon3 system on chip. Finally, the RSA algorithm is written in C ++ programming language. Software based Triple Modular Redundancy (TMR) is implemented by performing the same decryption process on each processor core. In this realization, the RSA algorithm running on each processor

is starting to work with a certain time difference. If one of the processor cores is affected by temporary bit errors and produces a wrong result, the result produced by the other two processors will be accepted correctly.

For analysing the proposed method three case are compared with each other. First case is the one that only one thread is running for RSA decryption, that is, TMR method is not used. Second case is the one that three thread is running, that is TMR method is used, for RSA decryption of the same ciphertext with one copy of the same secret key. Third case which is the proposed method is the one that three thread is running, that is TMR method is used, for RSA decryption of the same ciphertext with the same secret key. In the proposed method, there is only one copy of the secret key and all of three threads are using this copy. After analysis of these three cases, effect of Cylic Redundancy Check(CRC) for checking integrity of the secret key is analyzed when used with the proposed method. Firstly, the proposed method is tested on Linux operating system running on triple core Leon3 processor. It is shown that the method is succesfully running on the system. After that, for detailed analysis of the proposed method, Linux operating system installed on PC is used because of performance constraint of Leon3 processor system I synthesized.

In the analysis of the first case, that is only one thread is running for RSA decryption, the program written for analysis does different number of decryption at each step (from 50 to 2000 inreasing with 50 at each step). For each step percentage of succeeded fault attacks are calculated. Analysis results for first case shows that at each step approximately 13,33% of fault attacks are succeeded. The same test is done when CRC is performed on the secret key with 1 ms period. In this case, 1,348% of fault attacks are succeeded.

For the second and third cases that TMR method is used, the most important situation is the situtation that the result of RSA decryption is wrong, while TMR method is succeeded, that is, at least two of three thread are generated the same result for RSA decryption. In this situation fault attack is succeeded, because the system generates faulty output. Another important situation is the situation that TMR method is failed, that is, all threads are generated different decryption results with each other. In this situation neither fault attack is succeeded, nor true result for decryption is generated.

In the analysis of the second case, that is TMR method is used and all threads use one copy of the same secret key, the program written for analysis does different number of decryption at each step (from 500 to 32000 inreasing with 500 at each step). For each step percentage of succeeded fault attacks are calculated. At the same time the percentage of the situation that TMR is failed is calculated. Analysis results for second case shows that at each step approximately 0,1% of fault attacks are succeeded. Also 8,282% of fault attacks causes TMR method to fail. The same test is done when CRC is performed on each secret key with 1 ms period. In this case, 0,002578% of fault attacks are succeeded.

In the analysis of the third case (the proposed method), that is TMR method is used and all threads use the same copy of the secret key, for different start delays between threads, analyzes are done. It is seen that when start delay is increasing, the success of fault attack is decreased, but not as much as in situation 2. The same test is done, when CRC period is performed on the secret key with different CRC period values. According to results, we can say that when CRC period and start delay of threads increasing, success rate of fault attack is decreasing. For every CRC period, if start

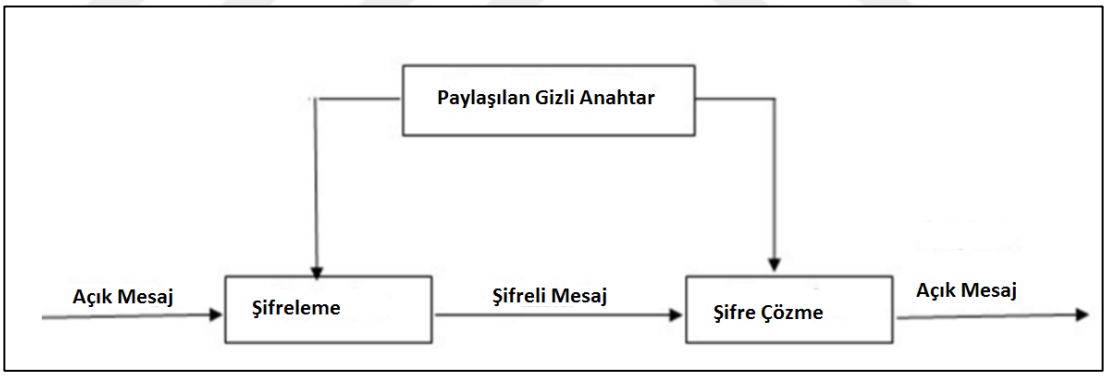
delay between thread 1-2 and 2-3 is greater than a specific value, success rate of fault attack goes to 0%.

When we compare the second and the third case, we see that third case which is the proposed method is more secure because of usage of one copy for the secret key. On the other hand, the second method has advantage over the proposed method in terms of time consumed for one RSA decryption operation. In terms of the situation that TMR is failed, the proposed method is more effective than the second case.

1. GİRİŞ

Kriptografi güvenli haberleşme için antik çağlardan beri kullanılmaktadır [1]. Özellikle günümüzün gelişen haberleşme teknolojisiyle birlikte kriptografi alanındaki çalışmalar da hız kazanmıştır. Kriptografi verinin gizliliği ve bütünlüğünü sağlamak ve kaynağını doğrulamak için kullanılabilir.

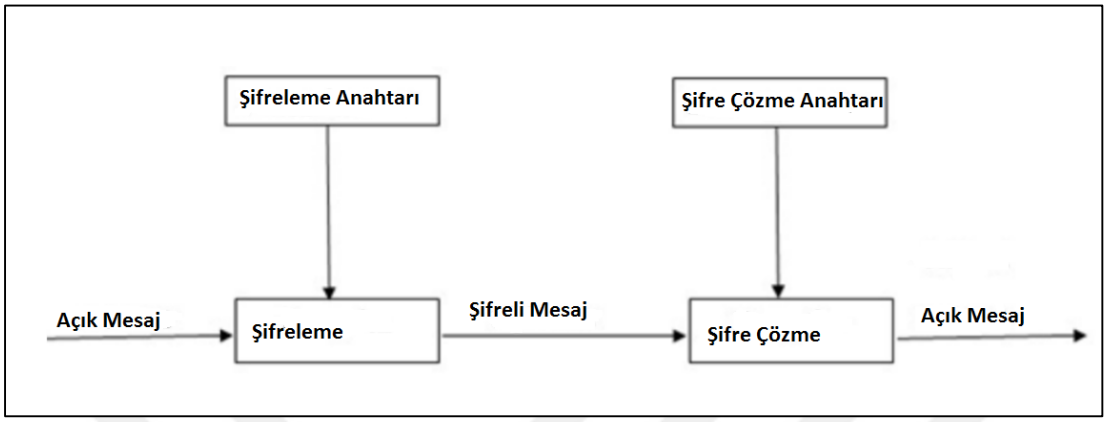
Kripto sistemleri kullanılan anahtar sayısına göre simetrik ve asimetrik kripto sistemleri olmak üzere ikiye ayrılabilir [1]. Simetrik kripto sistemlerinde Şekil 1.1' de görüldüğü üzere hem şifreleme hem de şifre çözme için aynı anahtar kullanılır. Bu nedenle anahtarın güvenli bir kanal üzerinden kullanıcılara dağıtılması gerekliliği vardır. Simetrik kripto sistemlerinde kullanılan algoritmalara Veri Şifreleme Standardı (Data Encryption Standard - DES) [2] ve İleri Şifreleme Standardı (Advanced Encryption Standard - AES) [3] örnek verilebilir.



Şekil 1.1 : Simetrik kripto sistemi [1].

Simetrik kripto sistemlerinde haberleşecek olan taraflar dışında kimsenin bilmemesi gereken anahtarın güvenli kanal üzerinden dağıtılması gerekmektedir [4]. Diffie ve Hellman yayınladıkları makale ile ilk kez bir asimetrik kripto sistemi önermesinde bulunmuş ve güvenli olmayan kanal üzerinden anahtar dağıtımı ile güvenli haberleşme sağlanabileceğini göstermiştir. Asimetrik kripto sistemlerinde Şekil 1.2' de görüldüğü üzere açık ve kapalı anahtar olmak üzere iki anahtar kullanılır. Açık anahtar ile isteyen herkes kapalı anahtarın sahibine şifreli veri yollayabilir. Ancak

şifreli veriyi sadece kapalı anahtarın sahibi çözebilir. Asimetrik kripto sistemlerine Rivest Shamir Adleman (RSA) [5] ve Eliptik Eğri Kriptografisi (Elliptic Curve Cryptography - ECC) [6] örnek verilebilir.



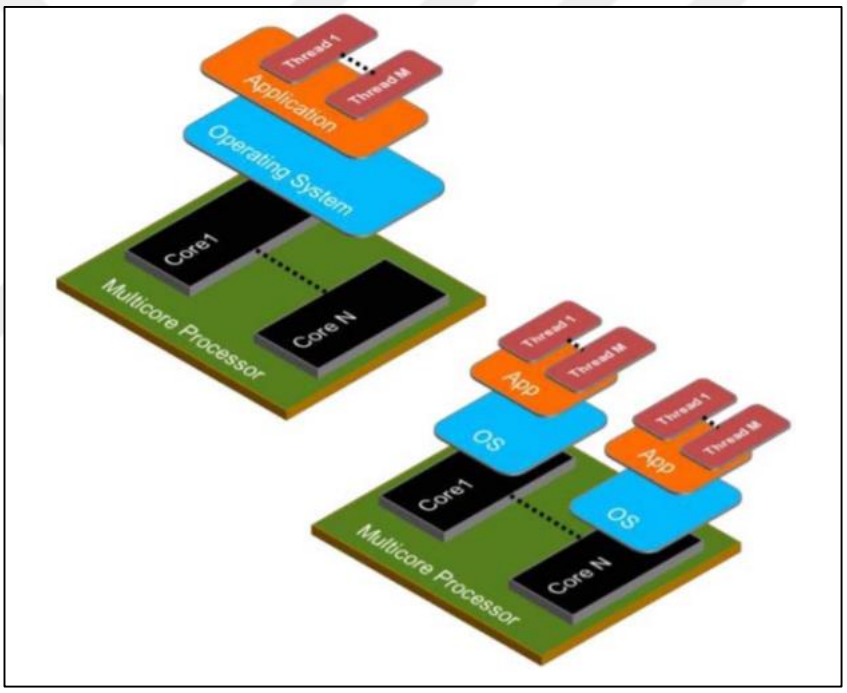
Şekil 1.2 : Asimetrik kripto sistemi [1].

Asimetrik kripto sistemleri kaynak doğrulama için de kullanılabilir [7]. Kaynak doğrulama uygulamalarının başında elektronik imza gelmektedir. Elektronik imza atmak için imza atılacak verinin sonuna bir özet fonksiyonuyla elde edilmiş ve kapalı anahtar ile şifrelenmiş imza eklenir. Şifreli imza açık anahtar ile çözülüp imza atılmış olan verinin özeti olduğu görülünce kapalı anahtarın sahibi tarafından imzalandığı doğrulanmış olur.

Hem simetrik hem de asimetrik kripto sistemlerinin gelişmesine paralel olarak bu sistemlere kapalı anahtarı ele geçirmek için yapılan atak yöntemleri de gelişmektedir [8]. Bu tez kapsamında yaygın olarak kullanılan asimetrik kripto sistemlerinden RSA, Sahada Programlanabilir Kapı Dizisi (Field Programmable Gate Array - FPGA) üzerinde koşan gömülü Linux işletim sistemi üzerinde hata enjekte etme ataklarına dayanıklı olacak şekilde gerçeklenmiştir. Öncelikle açık kaynak kodlu, sentezlenebilir bir mikroişlemci olan Leon3 işlemcisi üç çekirdekli olacak şekilde gerçeklenmiştir. Daha sonra Linux işletim sistemi Leon3 üzerinde çok çekirdekli yapıyı destekleyecek şekilde derlenip Digilent firmasının Nexys4ddr isimli FPGA kartı üzerinde çalıştırılmıştır. Son olarak RSA C++ programlama dili ile yazılmış ve üç işlemci çekirdeği üzerinde Üçlü Modül Yedekleme (Triple Modular Redundancy - TMR) yapısı kullanılarak [41]'de Bölüm 2'de anlatılan hata enjekte etme atağına dayanıklı olması sağlanmıştır.

1.1 Simetrik ve Asimetrik Çoklu İşlem

Çok çekirdekli işlemciler tek çekirdekli işlemcilere göre daha yüksek işlem gücü ve daha düşük frekanslarda çalışma olanağı sunar. Aynı zamanda daha düşük güç tüketimi ve daha az yer kaplaması dolayısıyla sistem maliyetini düşürür. Çoklu işlem Asimetrik Çoklu İşlem (Asymmetric Multi Processing - AMP) ve Simetrik Çoklu İşlem (Symmetric Multi Processing - SMP) modlarından oluşur [9]. Şekil 1.3'de SMP ve AMP çoklu işlem modları gösterilmektedir. SMP modunda tüm işlemci çekirdeklerini bir işletim sistemi yönetir ve uygulamalar herhangi bir çekirdeği kullanabilir. AMP modunda ise her bir çekirdek farklı işletim sistemi ya da aynı işletim sisteminin bir kopyasını çalıştırır [9].

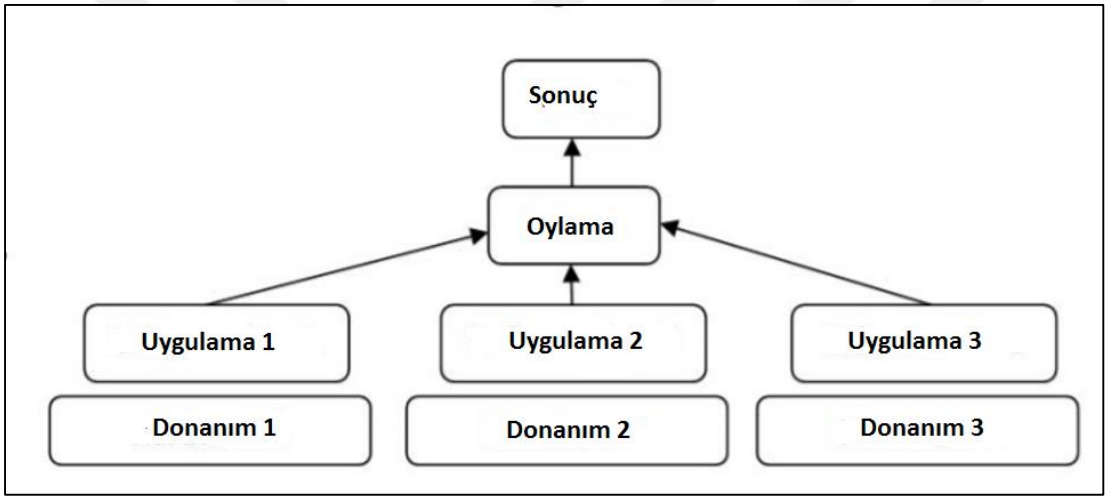


Şekil 1.3 : SMP(üstte) ve AMP(altta) [10].

Tez kapsamında gerçeklenen Leon3 çok çekirdekli sistemi SMP modunda çalışacak şekilde sentezlenmiştir. Daha sonra Linux [11] işletim sistemi SMP desteğiyle derlenmiş ve Leon3 sistemi üzerinde çalıştırılmıştır. Linux üzerinde çoklu işlemi destekleyen uygulamalar yazabilmek için OpenMP [12] ve Pthreads [13] gibi kütüphaneler kullanılmaktadır. Bu tez kapsamında Pthreads kütüphanesi kullanılmıştır.

1.2 Üçlü Modül Yedekleme

Hataya açık sistemlerin normal davranışını sürdürebilmesi için birincil yaklaşım hata maskelemedir. N Versiyon Programlama (N-Version Programming - NVP) ve N Modül Yedekleme (N-Modular Redundancy - NMR) en bilinen hata maskeleme yöntemleridir. Her iki yöntem de yedek modüller ve bir karar birimi kullanarak oluşan hataların sistem çıkışında görülmesini engeller. NMR'nin en basit hali TMR'dır. Üç modül aynı giriş üzerinde birbirinin aynı işi yapar ve sonuçları karar birimine iletir. Karar birimi çoğunlukta olan sonuca göre çıkışı belirler [14]. Örnek bir TMR sistemi Şekil 1.4'de görüldüğü gibidir. TMR yöntemindeki modüller lojik kapılar olabileceği gibi, mikroişlemci, FPGA hatta yazılım bile olabilir [15].



Şekil 1.4 : TMR yapısı [15].

Tez kapsamında RSA algoritmasının şifre çözme işlemi esnasında hata enjekte etme atağı nedeniyle kapalı anahtarın rastgele bir biti üzerinde oluşabilecek geçiçi okuma hatalarının şifre çözme işlemi sonucunu bozmasını engellemek için yazılım tabanlı TMR gerçeklenmiştir. Leon3 sistemindeki her bir işlemci bir TMR modülü olarak kullanılmış ve Pthreads kütüphanesi kullanılarak paralel çalışmaları sağlanmıştır. Kapalı anahtar her 3 TMR modülü tarafından ortak kullanılacaktır. Bu nedenle kapalı anahtarda hata enjekte etme atağı nedeniyle geçiçi bir bit okuma hatası oluşursa ve her üç TMR modülü hatanın oluştuğu anda kapalı anahtarın aynı biti ile çalışırlarsa üçü de yanlış sonuç verecek ve karar birimi yanlış olan bu sonucu geçerli kabul edecektir. Bunun önüne geçebilmek için [16]'da önerilen zaman çoğullamalı TMR yöntemi yazılım tabanlı olarak kullanılmıştır. Bu yöntemde her bir TMR modülü

aynı zaman diliminde kapalı anahtarın farklı bir bitini kullanacaktır. Önerilen yöntemin analizi için şu üç durum arasında karşılaştırma yapılmıştır:

I. Tek bir modülün çalıştığı, yani TMR yönteminin kullanıldığı durum.

II. Her üç TMR modülünün kapalı anahtarın bir kopyasını tuttuğu fakat modüllerin çalışmaya başlama zamanları arasında zaman farkı olmadığı durum.

III. Her üç TMR modülünün kapalı anahtarın tek bir kopyasını ortak kullandığı ve çalışmaya başlama zamanları arasında zaman farkı olduğu durum.

1.3 Literatür Araştırması

Bu bölümde sırasıyla teze konu olan LEON3 işlemcisinin çok çekirdekli gerçeklemeleri, RSA algoritmasının çok çekirdekli gerçeklemeleri ve TMR yönteminin çok çekirdekli işlemciler üzerinde kullanılması üzerine çalışmalardan bahsedilmiştir.

1.3.1 Çok çekirdekli LEON3 işlemcisi gerçeklemeleri

[58]'de yapılan çalışmada Gaisler firmasının çift çekirdekli LEON3 işlemcisi GR712RC üzerinde RTEMS işletim sistemi için SMP desteği sağlanmış ve paralel programlama için gerekli kütüphaneler geliştirilmiştir. [59]'de Gaisler firmasının çift çekirdekli LEON3 işlemcisi barındıran GR-CPCI-XC4V isimli geliştirme kartı üzerinde bir trafik işareti tanıma sistemi geliştirilmiştir. Bu çalışmada LEON3 üzerinde gerçek zamanlı bir işletim sistemi olan eCos işletim sistemi koşturulmuştur. Bir diğer çalışma olan [60]'de ise çift çekirdekli LEON3 işlemcisi Xilinx'in ML-509 geliştirme kartı üzerinde her bir çekirdeğe bağlı bir yeniden konfigüre edilebilir yardımcı işlemci bulunacak şekilde gerçeklenmiştir. Bu sistem üzerinde Linux işletim sistemi SMP desteğiyle çalıştırılmıştır. Yeniden konfigüre edilebilir olan yardımcı işlemciler Xilinx'in parçalı konfigürasyon yükleme imkanı veren ICAP modülü üzerinden istenildiğinde bağli bulunduğu LEON3 çekirdeği tarafından konfigüre edilebilmektedir. Bu çalışmada ICAP modülüne erişebilmek için sürücü yazılımı da yazılmıştır. Son olarak [61]'deki çalışmada 4 çekirdekli LEON3 işlemcisi üzerinde gerçek zamanlı işletim sistemleri için bir zamanlama analizi yapılmıştır.

1.3.2 RSA algoritmasının çok çekirdekli gerçeklemeleri

RSA algoritmasının çok çekirdekleri gerçeklemeleri üzerine yapılan çalışmalar daha çok hız açısından performansı arttırmak üzerine yapılmıştır. [62]'deki çalışmada Montgomery modüler çarpma işlemi paralelleştirilerek 4 çekirdekli ARM A15 işlemcisi üzerinde çalıştırılmış ve RSA algoritmasının paralel olmayan versiyonuna kıyasla 7.3 kat daha hızlı çalıştığı gösterilmiştir. [63]'deki çalışmada ise RSA modüler üs alma yöntemi paralelleştirilmiştir. Ancak gerçekleme yapılmamıştır. [64]'daki çalışmada ise RSA modüler üs alma yöntemi GPU üzerinde paralelleştirilerek çalıştırılmıştır.

1.3.3 RSA algoritmasının hata enjekte etme atağına dayanıklı gerçeklemeleri

RSA algorirtmasının hata enjekte etme ataklarına dayanıklı gerçeklemelerine [65,66]'daki modüler üs alma yöntemini hata enjekte etme ataklarına dayanıklı hale getirmek için yapılan çalışmalar örnek verilebilir. Rivain [65]'deki yöntemde aynı anda iki üs alma işlemi yapılacak şekilde algoritma geliştirilmiştir. Sonuçta hem $m^d \ mod \ N$ hem de $m^{(\Phi(N)-d)} \ mod \ N$ hesaplanmış olacak ve $m^d \times m^{(\Phi(N)-d)} \ mod \ N = 1$ eşitliğinin sağlanması gerekliliğinden yararlanılarak hata enjekte edilip edilmediği anlaşılacaktır. [66]'deki çalışmada modüler üs alma işlemi hata enjekte etme ataklarının yanısıra basit güç analizi ataklarına da dayanıklı hale getirilmiştir. Bu çalışmada da benzer şekilde iki üs alma işlemi aynı anda yapılmakta ve sonuçların birbirine eşit olması esasına göre hata enjekte edilip edilmediği kontrol edilmektedir.

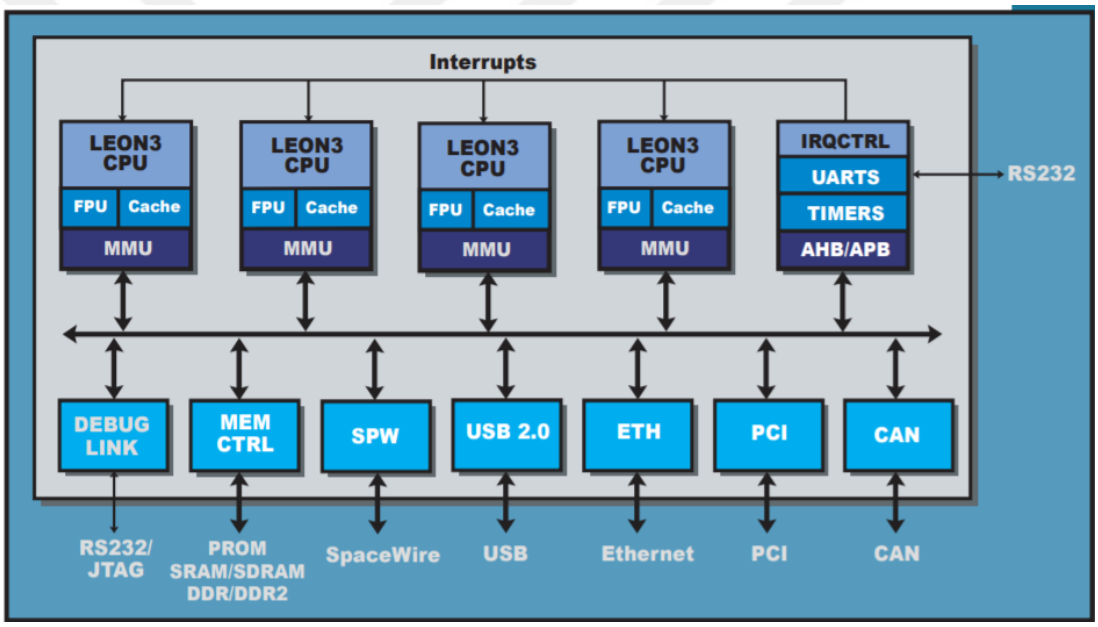
1.3.4 TMR yönteminin çok çekirdekli işlemcilerde kullanılması

[67]'daki çalışmada 4 çekirdekli P2041 işlemcisi üzerinde TMR yöntemiyle hata düzeltme yöntemi analiz edilmiştir. Hatanın doğrudan işlemci kütüklerine ve paylaşılan belleğe yapılması farklı sonuçlar vermektedir. paylaşılan belleğe hata enjekte edildiğinde %20 oranındaki hata sonucu değiştirmemektedir. Geriye kalan %80 oranındaki hatanın %99.99'u farkedilip düzeltilebilmektedir. Hata enjekte etme işleminin işlemci kütüklerine yapılması durumunda da yine çoğu hata düzeltilmektedir. Fakat bazı kritik kütüklere hata enjekte edildiğinde hata düzeltme performansı ciddi şekilde kötüleşmektedir. Ancak işlemci kütüklerinin program ve data belleğine kıyasla %0.6 yer kaplaması bu kütüklere hata ejekte edilmesi

olasılığını azaltmaktadır. Sonuç olarak TMR çok çekirdekli işlemcili ve çok işlemcili sistemlerde güvenilirliği arttırmak için iyi bir alternatif olabilir.

2. LEON3

Cobham Gaisler firması tarafından geliştirilmiş olan Leon3 SPARC V8 mimarisi tabanlı ve çok çekirdekli yapıyı destekleyen 32-bit bir işlemcidir. Çok Yüksek Hızlı Tümleşik Devre Donanım Tasarım Dili (Very High Speed Integrated Circuits Hardware Description Language - VHDL) ile yazılmış, açık kaynak kodlu ve tamamen sentezlenebilir yapıdadır [17]. Leon3 işlemci mimarisi Şekil 2.1'de görüldüğü üzere Ethernet, PCI, USB ve RS232 gibi arayüzleri desteklemektedir.



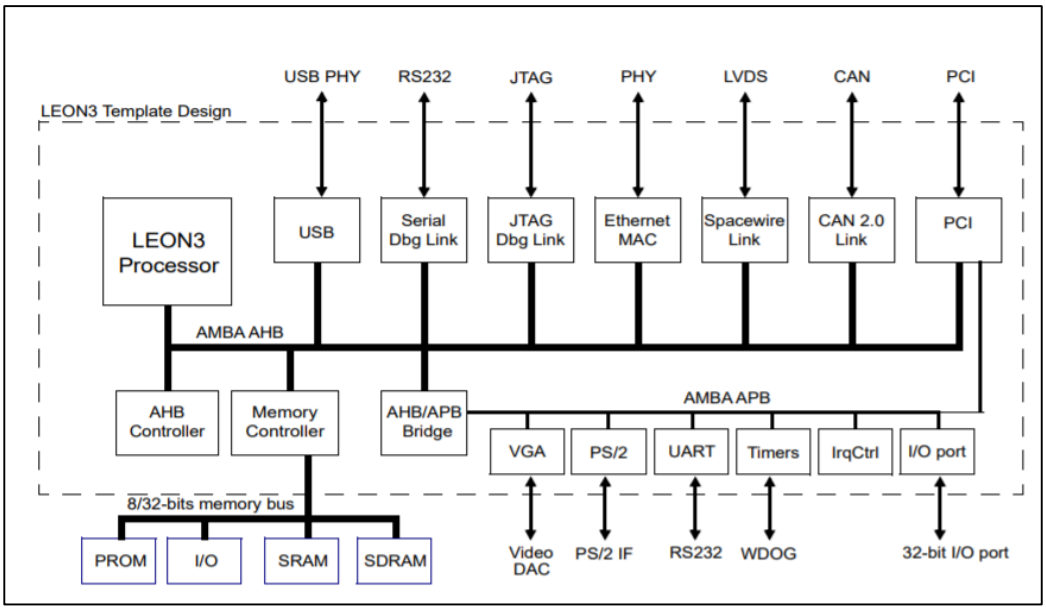
Şekil 2.1 : Leon3 işlemci mimarisi [17].

Leon3 işlemcisi hem SMP hem de AMP yapısını 16 çekirdeğe kadar desteklemektedir. Leon3 çok çekirdekli yapıda tek çekirdekli işlemcilere kıyasla daha düşük frekanslarda çalışıp daha yüksek performans sağlamaktadır. Bu sayede önemli ölçüde güç ve maliyet tasarrufu sağlamaktadır. Leon3 işlemcisinin genel özellikleri yüksek konfigüre edilebilirlik, yüksek performans, enerji verimliliği, tasarım ve entegrasyon kolaylığı, yazılım desteği ve kaynak kodlarına erişilebilirlik gösterilebilir. İşlemcinin Kayan Nokta Birimi (Floating Point Unit - FPU) ve önbellek gibi birçok özelliği sentez esnasında düzenlenebilmektedir. VxWorks [18],

eCos [19] ve Linux 2.6 işletim sistemlerini SMP ve RTEMS [20], VxWorks, uCLinux ve ThreadX işletim sistemlerini AMP yapısında desteklemektedir.

2.1 Gaisler Kütüphanesi

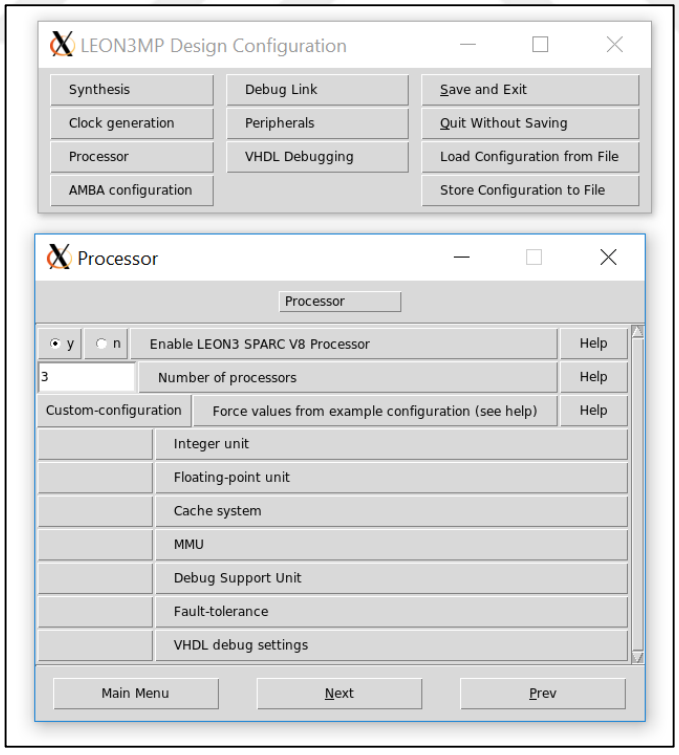
Cobham Gaisler firması Leon3 işlemcisi içeren kırmık üstü sistem tasarımı için Gaisler Kütüphanesi (Gaisler Library - GRLIB) [21] adında bir VHDL kütüphanesini açık kaynak kodlu olarak sunmaktadır. [21] kütüphane organizasyonu, tasarım ve simülasyon araçları kullanımı ve kırmık üstü veri yolu hakkında bilgi vermektedir. [22] Gaisler kütüphanesinde bulunan Intellectual Property (IP)'ler hakkında detaylı bilgi içermektedir. Son olarak [23] ise Leon3 sistem tasarımı hakkında geliştiricilere bilgi sağlamaktadır. Gaisler kütüphanesi Modelsim, Ncsim, Aldec, Sonata ve GHDL gibi simülasyon araçları ve Synopsys, Synplify, Cadence, Mentor, Actel, Altera, Lattice, eASIC ve Xilinx sentez araçlarını desteklemektedir. Diğer sentez ve simülasyon araçları için destek kolaylıkla eklenebilmektedir. Örnek bir Leon3 sistemi Şekil 2.2' de görülmektedir. Görüldüğü gibi tüm IP'ler Leon3'e Gelişmiş Mikrokontrolör Veri Yolu Mimarisi (Advanced Microcontroller Bus Architecture - AMBA) [24] üzerinden bağlanmıştır. Hızlı çalışması gereken IP'ler AMBA Gelişmiş Yüksek Hızlı Veri Yolu (Advanced High Speed Bus - AHB)'na diğer IP'ler daha az karmaşıklığı olan AMBA Gelişmiş Çevresel Birim Veri Yolu (Advanced Peripheral Bus - APB)'na bağlanabilir.



Şekil 2.2 : Leon3 kırmık üstü sistem örneği [21].

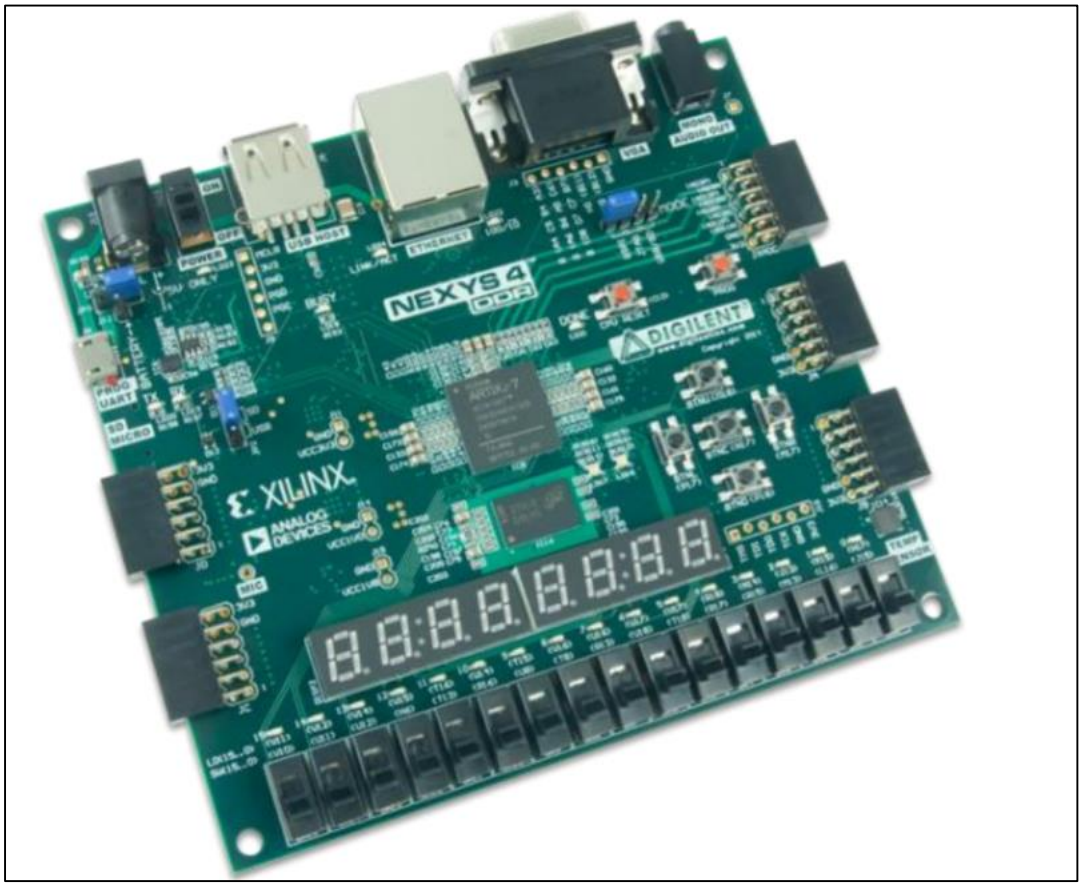
Gaisler kütüphanesindeki tüm IP'ler tak-oynat(plug and play) özelliğine sahiptir. Tak-oynat bilgisi IP'ye özgü bir ID numarası, AMBA AHB/APB adres haritası ve kullanılan kesme vektörü bilgisini içerir. Bu bilgi AMBA AHB/APB'ye bağlı IP'ler tarafından veri yolu yöneticisine gönderilir. Veri yolu yöneticisine üzerinde bulunan küçük bir salt okunur bellekte tutulan tak-oynat bilgileri herhangi bir AMBA AHB yöneticisi (bir işlemci veya başka bir IP olabilir) tarafından okunabilir. Tak-oynat özelliğini destekleyen işletim sistemleri desteklenebilmektedir.

Gaisler kütüphanesi kullanıcılar için birçok FPGA geliştirme kartı için hazır tasarımlar sunmaktadır. Make dosyası aracılığıyla sunulan xonfig menüsü kullanılarak işlemci konfigürasyonu yapılabilmekte ve istenen IP'ler sisteme eklenip çıkarılabilmektedir. Xconfig Şekil 2.3'de görüldüğü gibi menü yapısında bir arayüz sunarak kütüphanenin kullanımını kolaylaştırmaktadır. Leon3 sistem tasarımı istenen FPGA geliştirme kartı için hazırlanmış olan proje kullanılarak yapılır. Proje içerisinde make xconfig komutu çalıştırılarak menü üzerinden işlemcinin konfigürasyonu yapılır ve geliştirme kartının desteklediği IP'lerden istenenler sisteme eklenebilir.



Şekil 2.3 : Xconfig menüsü.

Proje kapsamında 3 Leon3 çekirdeği içeren, ethernet ve RS232 arayüzlerini destekleyen bir kırmık üstü sistem tasarımı Şekil 2.4'de görülen Digilent firmasının Nexys4DDR [25] isimli FPGA geliştirme kartı üzerinde gerçeklenmiştir.



Şekil 2.4 : Nexys4DDR FPGA kartı [25].

Nexys4DDR FPGA kartının sahip olduğu önemli bazı özellikler şunlardır:

- 128 MiB DDR2
- Micro SD kart girişi
- 10/100 Ethernet PHY
- İvme ve sıcaklık sensörü
- Xilinx Artix-7 100T FPGA

2.2 Geliştirme Ortamı

Üzerinde Linux işletim sistemi koşan Leon3 sistemi tasarımı için kullanılan geliştirme ortamı şu şekildedir:

- Xconfig menüsünün kullanılabilmesi için Linux işletim sistemi veya Windows işletim sistemi üzerinde Linux benzeri bir ortam sunan Cygwin [26] isimli açık kaynaklı ve GNU araçlarından oluşan platform kullanılabilir. Tez kapsamında Cygwin kullanımı tercih edilmiştir.
- Tasarlanan Leon3 sisteminin sentezlenmesi ve Nexys4DDR kartına yüklenmesi için ise Xilinx firmasının Vivado isimli geliştirme ortamı kullanılmıştır. Tez kapsamında Vivado'nun ücretsiz sürümü olan WebPack kullanılmıştır.
- Nexys4DDR kartına Leon3 üzerinde çalışması için yüklenecek Linux işletim sisteminin ve herhangi bir programın derlenmesi için kullanılacak GCC derleyicisinin çapraz derlenmesi için Windows üzerinde sanal makine olarak çalışan Linux tabanlı Ubuntu işletim sistemi kullanılmıştır.

  Derlenen Linux işletim sisteminin Nexys4DDR kartına yüklenmesi işlemi için Gaisler firmasının Gaisler Monitor (GRMON) [27] isimli programı kullanılmıştır.

3. MATEMATİKSEL ÖN BİLGİ

Bu bölümde RSA algoritmasının gerçeklenmesi için gerekli olan bazı matematiksel ön bilgiler verilecektir. Aşağıdaki bölümlerde sırasıyla rastgele asal sayıların bulunması, bir sayının mod alma işlemine göre tersinin bulunması, karatsuba çarpma işlemi, mod alma ve çarpma işlemlerinin bir arada yapıldığı Montgomery modüler çarpma işlemi ve Barrett mod alma işlemleri detaylı bir şekilde anlatılacaktır.

3.1 Rastgele Asal Sayıların Bulunması

İstenen büyüklükte rastgele belirlenen tek sayılardan asal olan biri bulunarak p ve q belirlenir. Bir sayının asal olup olmadığının test edilmesi için [5] tarafından [28]' de anlatılan asallık testini önermiştir. Bunun dışında [29,30,31]' de önerilen asallık testleri de kullanılabilir. [7] tarafından kullanılması önerilen Rabin-Miller testi Algoritma 3.1'de gösterilmektedir.

Algoritma 3.1 : Asallık testi [7].

GİRİŞ : w asal olup olmadığı test edilmek istenen sayı
iterations asallık testi için kaç deneme yapılacağını gösteren sayı
ÇIKIŞ : w'nin asal olup olmadığı bilgisi

1. 2^a, $w-1$'i bölen bir sayı olmak üzere en büyük a değeri seçilir.
2. $m = (w-1)/2a$.
3. $wlen = bitlen(w)$.
4. For $i = 1$ to *iterations* do
   4.1. $1 < b < w-1$ olmak üzere *wlen* bit uzunluklu bir b sayısı rastgele seçilir.
   4.2. Eğer $((b \leq 1) \,||\, (b \geq w-1))$ ise adım 4.1'den devam edilir.
   4.3. $z = b \times m \bmod w$.
   4.4. Eğer $((z = 1) \,||\, (z = w-1))$ ise adım 4.7'den devam edilir.
   4.5. For $j = 1$ to $a - 1$ do.
      4.5.1. $z = z^2 \bmod w$.
      4.5.2. Eğer $z = w-1$ ise adım 4.7'den devam edilir.
      4.5.3. Eğer $z = 1$ ise adım 4.6'den devam edilir.
   4.6. w sayısının asal bir sayı olmadığı bilgisi dönülür.
   4.7. Adım 4'den devam edilir.
5. w sayısının olasılıksal bir asal sayı olduğu bilgisi dönülür.

Algoritmadan görüldüğü üzere Rabin-Miller testi verilen sayının asal omadığını söylüyorsa verilen sayı kesinlikle asal değildir. Ancak asal olduğunu söylüyorsa k asallık testi için yapılacak deneme sayısı olmak üzere $1/2^{2k}$ olasılıkla asal olmama ihtimali vardır. $k = 50$ için bu değer 2^{-100} olmaktadır [31].

3.2 Mod İşlemine Göre Bir Sayının Çarpımsal Tersinin Bulunması

x, y, z tam sayılar olmak üzere $x \times y \bmod z = 1$ ise x ve y'ye $mod\ z$'ye göre birbirlerinin çarpımsal tersi denir. [7]'de Bölüm C.1'de anlatılan algoritmaya göre bir sayının mod işlemine göre çarpımsal tersinin bulunması Algoritma 3.2'de gösterilmiştir.

Algoritma 3.2 : Mod işlemine göre çarpımsal ters alma [7].

GİRİŞ : a mod alınacak olan sayı
z, $mod\ a$'ye göre çarpımsal tersi bulunmak istenen sayı
ÇIKIŞ : z^{-1}, z'nin $mod\ a$'ye göre tersi

1. a ve z'nin tamsayı ve $a < z$ olup olmadığı kontrol edilir; değilse hata dönülür.
2. $i = a, j = z, y2 = 0$, ve $y1 = 1$.
3. $quotient = \lfloor i/j \rfloor$.
4. $remainder = i - (j \times quotient)$.
5. $y = y2 - (y1 \times quotient)$.
6. $i = j, j = remainder, y2 = y1$, ve $y1 = y$.
7. $j > 0$ ise adım 3'den devam edilir.
8. $i \neq 1$ ise hata dönülür.
9. Başarılı dönüş ve $y2\ mod\ a$ dönülür.

Algoritmaya bakıldığında 8. adımda hata dönülebileceği görülmektedir. Bu durumun oluşmaması için z ile a'nın aralarında asal olması gerekmektedir [32].

3.3 Karatsuba Çarpma İşlemi

Karatsuba Çarpma işlemi büyük sayılarla çarpma işlemini etkili bir şekilde yapmayı sağlayan bir çarpma algoritmasıdır [33]. Algoritmanın uygulanması çarpılacak sayıların yüksek ve düşük anlamlı parçalar olarak ikiye ayrılmasıyla gerçekleşir. Her iterasyonda 3 çarpma 4 toplama işlemi gerektirir. Algoritma 3.3'de gösterilmiştir.

Algoritma 3.3 : Karatsuba çarpma işlemi [33].

GİRİŞ : $A = (a_n, a_{n-1}, \ldots., a_1, a_0)_r$
$B = (n, b_{n-1}, \ldots., b_1, b_0)_r$ r tabanında $n + 1$ basamaklı sayılar
ÇIKIŞ : $C = A \times B$

1. If $n = 1$ then return $A \times B$.
2. A ve B basamak sayısına göre iki parçaya ayrılır:
   $A = A_L \times r^{n/2} + A_R$
   $B = B_L \times r^{n/2} + B_R$
3. Aşağıdaki çarpma işlemleri yapılır:
   $d_1 = Karatsuba(A_L, B_L)$
   $d_0 = Karatsuba(A_R, B_R)$
   $d_{0,1} = Karatsuba(A_L + A_R, B_L + B_R)$
4. $C = d_1 \times r^n + (d_{0,1} - d_0 - d_1) \times r^{n/2} + d_0$

3.4 Montgomery Modüler Çarma İşlemi

$X \times Y \ mod \ M$ işleminin sonucunu çarpma ve mod işlemlerini ayrı ayrı yapmadan ve bölme işlemine ihtiyaç duymadan yapabilmek için Montgomery tarafından [34]'de önerilmiştir.

Algoritma 3.4: Montgomerty modüler çarma işlemi [35].

GİRİŞ : $M = (m_{n-1}, m_{n-2}, \ldots., m_1, m_0)_b$
$X = (x_{n-1}, x_{n-2}, \ldots., x_1, x_0)_b$
$Y = (y_{n-1}, y_{n-2}, \ldots., y_1, y_0)_b$ b tabanında sayılar
$0 \leq X, Y < M, R = b^n$ ve $M' = -M^{-1} \ mod \ b$
ÇIKIŞ : $X \times Y \times R^{-1} \ mod \ M$

1. $A = 0$
2. For i from 0 to $(n - 1)$ do
   2.1. $u_i = (a_0 + x_i \times y_0) \times M' \ mod \ b$
   2.2. $A = (A + x_i \times Y + u_i \times M)/b$
3. If $A \geq M$ then $A = A - M$
4. Return A

Algoritmanın tanımından da görüldüğü gibi Montgomery modüler çarpma işlemi ile "$X \times Y \ mod \ M$" değil "$X \times Y \times R^{-1} \ mod \ M$" işleminin sonucunu hesaplanmaktadır. "$X \times Y \ mod \ M$" işleminin sonucunun hesaplanabilmesi için X ve Y Montgomery modüler çarpma işlemine girmeden önce bir dönüşüme tabi tutulmalıdır. Bu dönüşümden Montgomery modüler çarpma işleminin kullanılacağı ilerleyen bölümlerde bahsedilecektir.

3.5 Barrett Mod Alma Yöntemi

Barrett mod alma yöntemi ile $r = x \, mod \, m$ işleminin sonucu hesaplanır [35]. Bölme işlemi içeren bir ön hesaplama gerektirir. Bu nedenle çok fazla mod alma işlemi yapıldığında Barrett mod alma yöntemi ile avantaj sağlanmış olur. Yöntem Algoritma 3.4'de gösterilmiştir.

Algoritma 3.5 : Barrett mod alma yöntemi [35].

GİRİŞ : $M = (m_{k-1}, m_{k-2}, \ldots., m_1, m_0)_b$ ve
$X = (x_{2k-1}, x_{2k-2}, \ldots., x_1, x_0)_b$ b tabanında sayılar
$u = \lfloor b^{2k}/m \rfloor$

ÇIKIŞ : $r = x \, mod \, m$

1. $q_1 = \lfloor x/b^{k-1} \rfloor$, $q_2 = q_1 \times u$ ve $q_3 = \lfloor q_2/b^{k+1} \rfloor$
2. $r_1 = x \, mod \, b^{k+1}$, $r_2 = q_3 \times m \, mod \, b^{k+1}$ ve $r = r_1 - r_2$
3. If $r < 0$ then $r = r + b^{k+1}$
4. While $r \geq m$ do: $r = r - m$
5. Return r

Barrett mod alma yönteminde 1 numaralı adımda görülen bölme işlemleri b tabanında sağa kaydırma işlemi ile yapılabileceğinden işlem karmaşıklığı açısından bir yük getirmediği görülmektedir [35].

4. RİVEST SHAMIR ADLEMAN ALGORİTMASI

RSA; Rivest, Shamir ve Adleman tarafından 1977 yılında önerilmiş bir asimetrik kripto algoritmasıdır [5]. Diffie ve Hellman tek yönlü ve arka kapılı fonksiyonlar ile asimetrik kripto sistemleri önermesinde bulunmuştur [4]. Tek yönlü ve arka kapılı fonksiyonlar bir yönde kolay hesaplanabilen, tersi yönde ise ek bir bilgi olmadan hesaplamanın çok zor olduğu fonksiyonlardır. RSA tek yönlü ve arka kapılı fonksiyonlar kullanarak asimetrik kripto algoritmasının ilk örneği olmuştur. RSA algoritmasının güvenliği çok büyük sayıların çarpanlarına ayrılmasının zorluğuna dayanmaktadır [5]. Şifreleme ve şifre çözme işlemleri şu şekilde yapılmaktadır:

M : şifrelenecek veri, $0 \leq M \leq n-1$

C : şifreli veri,

E : şifreleme algoritması

D : şifre çözme algoritması olmak üzere;

$$C = E(M) = M^e \, mod \, n \tag{4.1}$$

$$D(C) = C^d \, mod \, n \tag{4.2}$$

Görüldüğü üzere şifreleme sonucunda oluşan şifreli veri C de şifrelenen veri olan M ile aynı şekilde 0 ile $n-1$ aralığındadır.

Şifreleme ve şifre çözme işlemlerine baktığımızda açık anahtarın (e, n), kapalı anahtarın ise (d, n) olduğu görülmektedir. Bu noktada karşımıza n, e ve d'nin nasıl belirleneceği sorusu çıkmaktadır.

p ve q çok büyük rastgele asal sayılar olmak üzere;

n denklem 4.3'e göre belirlenir.

$$n = p \times q \tag{4.3}$$

$\Phi(N) = (p-1) \times (q-1)$ olmak üzere d denklem 3.4'ü sağlayan çok büyük bir değer olarak belirlenir.

$$\gcd(d,\ \Phi(N)) = 1 \tag{4.4}$$

e denklem 4.5 çözülerek bulunur.

$$d\ \times e\ mod\ \Phi(N) = 1 \tag{4.5}$$

Denklem 4.5'de gösterilen işlemle bulunan e değerine d'nin $mod\ \Phi(N)$ işlemine göre çarpımsal tersi denir.

n değeri açık olmasına rağmen çarpanlarına ayrılmasının çok zor bir problem olması nedeniyle p ve q'nun bulunması çok zordur. Bu zorluk aynı zamanda denklem 4.4'in çözülüp d'nin yani kapalı anahtarın bulunmasını da çok zor hale getirmektedir [5].

4.1 Anahtar Üretimi

Şifreleme anahtarı (e, n) ve şifre çözme anahtarı (d, n)'nin nasıl belirleneceği denklem 4.3, 4.4 ve 4.5 ile gösterilmiştir. Bu noktada çözülmesi gereken şu üç problem ortaya çıkmaktadır:

- p ve q'nun çok büyük asal sayılar olarak seçilmesi
- $\Phi(N)$ ile en büyük ortak böleni 1 olan bir d değerinin bulunması
- d'nin $mod\ \Phi(N)$'e göre çarpımsal tersi olan e değerinin bulunması

Bu problemlerin çözümü için .Bölüm 3.1 ve 3.2' de anlatılan yöntemler kullanılabilir. Eğer Bölüm 3.1'de anlatılan Rabin-Miller asallık testi p ya da q değeri için asal olmayan bir sayının asal olduğunu söylerse ve bu değer kullanılarak açık ve kapalı anahtarlar belirlenirse bu şifre çözmenin düzgün çalışmamasından dolayı kolaylıkla farkedilebilir [5]. d değeri de p ve q değerlerine benzer bir şekilde bulunabilir. $d > \max(p, q)$ olan bir asal sayı olarak seçilir.

4.2 Şifreleme ve Şifre Çözme

Denklem 4.1 ve 4.2'ye bakıldığında hem şifreleme hem de şifre çözme işlemlerinin parametreleri dışında aynı olduğu görülmektedir. d ve e parametreleri çok büyük sayılar olduğundan şifreleme ve şifre çözme işlemlerinin yapılmasının normal yollarla çok fazla işlem yükü gerektirdiği açıktır. Bu işlem yükünün üstesinden gelebilmek için aşağıda verilen kare al ve çarp algoritması (square and multiply algorithm) kullanılır.

Algoritma 4.1 : Kare al ve çarp algoritması [35].

GİRİŞ : a, k, n ve k'nın bit sayısı t

ÇIKIŞ : $a^k \ mod \ n$

1. $b = 1$. If $k = 0$ then return b.
2. $A = a$.
3. If $k[0] = 1$ then $b = a$.
4. For i from 1 to t do.
   4.1. $A = A^2 \ mod \ n$.
   4.2. If $k[i] = 1$ then $b = A \times b \ mod \ n$.
5. Return b.

Şifre çözme işleminin zaman açısından daha verimli bir şekilde yapılabilmesi için p ve q değerlerinin şifre çözen tarafından biliniyor olmasından yararlanan Çinli Kalan Teoremi (Chinese Remainder Theorem - CRT) yöntemi kullanılabilir. CRT yöntemi Algoritma 4.2'de verilmiştir.

Algoritma 4.2 : CRT yöntemi ile modüler üs alma [36].

GİRİŞ : P, Q ve $M = P \times Q$

ÇIKIŞ : $S = A^D \ mod \ M$

1. $D_p = D \ mod \ (P-1)$
   $D_q = D \ mod \ (Q-1)$
2. $C_p = Q^{(P-1)} \ mod \ M$
   $C_q = P^{(Q-1)} \ mod \ M$
3. $A_p = A \ mod \ P$
   $A_q = A \ mod \ Q$
4. $B_p = {A_p}^{D_p} \ mod \ P$
   $B_q = {A_q}^{D_q} \ mod \ Q$
5. $S_p = B_p \times C_p \ mod \ M$
   $S_q = B_q \times C_q \ mod \ M$
6. $S = S_p + S_q$
7. $S(A) = (if \ S \geq M \ then \ S - M \ else \ S)$

CRT yöntemine bakıldığında 1 ve 2 numaralı adımların sadece bir kere hesaplanıp şifre çözme işleminde sabit değişken olarak kullanılarak algoritmanın hızlandırılabileceği görülmektedir.

RSA şifreleme ve şifre çözmede kullanılan kare al ve çarp ve CRT işlemlerine bakıldığında yoğun olarak çok büyük sayılar üzerinde çarpma ve mod işlemi yapıldığı görülmektedir. Çok büyük sayılarla çarpma ve mod işlemleri standart çarpma ve bölme işlemine dayalı olarak gerçeklendiğinde RSA şifreleme ve şifre çözme işlem karmaşıklığı yüksek olmaktadır. Bu işlem zorluklarının üstesinden gelebilmek Bölüm 3.3, 3.4 ve 3.5' de incelenen yöntemler kullanılabilir.

4.3 RSA Algoritmasına Yapılan Yan Kanal Analizi Atakları

Kriptografi şifreleme yöntemlerinin geliştirilmesiyle ilgilenirken, kriptoanaliz kriptografik algoritmaların kırılmasıyla ilgilenir [8]. Kriptografi ve kriptoanaliz biri geliştikçe diğeri de gelişen bilim dallarıdır. Kriptoanaliz kriptografik algoritmaların zaafiyet taraması için kullanılan önemli bir araçtır.

RSA algoritmasına yönelik ataklar genel olarak matematiksel altyapısını hedef alan ve gerçeklemeyi hedef alan ataklar olmak üzere iki ana başlıkta toplanabilir [1]. RSA'in matematiksel altyapısını hedef alan ataklara kapalı anahtarın yeterince büyük olmamasından yararlanan [37]'deki atak ve kapalı anahtarın bir kısmının bilinmesine dayanan [38]' deki atak örnek verilebilir. Gerçeklemeye yönelik ataklara ise zamanlama [39], güç analizi [40] ve hata enjekte etme [41,42] atakları örnek verilebilir. Bu bölümde gerçeklemeye yönelik olan ataklar incelenecektir.

4.3.1 Zamanlama atağı

İlk olarak Kocher tarafından önerilen zamanlama atağı kripto sistemlerinin bazı giriş değerlerini işlerken normalden daha fazla zaman harcamasına dayanır [39]. Bunun nedenleri çeşitli perfomrans optimizasyonları, çarpma ve bölme gibi sabit zamanda işlenmeyen işlmeci komutları, gereksiz adımların atlanması ve dallanma olabilir. Zamanlama farklılıklarından yararlanılarak gizli anahtarın tamamı ele geçirilebilir.

Zamanlama ataklarına dirençli kripto sistemleri geliştirebilmek için çeşitli yöntemler kullanılabilir. Tüm işlemlerin aynı sürede yapılması sağlanabilir, rastgele gecikmeler kullanarak anlamsız zamanlama karakteristiklerine sahip bir kripto sistemi oluşturulabilir. Bir diğer yöntem ise giriş verisini rastgele bir v_i değeriyleçarptıktan sonra RSA algoritmasına sokup şifre çözme bitince $v_f = ({v_i}^d)^{-1} \, mod \, n$ değeriyle çarpmaktır.

4.3.2 Güç analizi atağı

Güç analizi atağı kriptografik cihazların kriptografik işlemler esnasında harcadığı güç analiz edilerek yapılan atak türüdür. Çarpma modülleri giriş değerlerine bağlı olarak farklı güç tüketir. Modüler üs alıcılar çarpma işlemleri ile kare alma işlemlerinde farkıl güç tüketim karakteristikleri gösterir. Bu gibi giriş değerlerine bağlı olarak farklı güç tüketim karakteristiklerinden yararlanılarak kripto sistemleri

kırılabilir [40]. Güç analizi ataklarını engellemek sadece bir kaç kriptografik işlem için kullanılan oturum anahtarları kullanılabilir. Böylece güç analizi için çok az veri elde edilmesi sağlanabilir. Güç tüketiminin her işlem için aynı olması veya rastgele olması sağlanabilir [43].

4.3.3 Hata enjekte etme atağı

Mikrodalgaya maruz bırakmak gibi bazı fiziksel etkilerle kurcalamaya dayanıklı cihazlarda kriptografik işlemler esnasında geçici hatalar meydana getirmek olasıdır [41]. [41]'de geçici hatalardan yararlanarak yapılan iki atak önerilmiştir. Önerilen ikinci atak şu şekildedir:

$d = (d_{n-1}d_{n-2} \dots d_i \dots d_1 d_0)$ kapalı anahtar

$c_i = c^{2^i} \bmod n$ olmak üzere;

1. Rastgele bir m açık verisi seçilir ve şifrelenerek c şifreli verisi elde edilir.
2. Şifreli veri c atak yapılan sisteme çözdürülmek üzere verilir.
3. Şifre çözme esnasında kapalı anahtar d'nin rastgele bir biti (i. bit) fiziksel bir müdahale ile geçici olarak bozulur.
4. Atak yapılmamış olsaydı şifre çözme sonucunda açık veri şu şekilde olacaktı:
   $m = ({c_{n-1}}^{d_{n-1}} \times {c_{n-2}}^{d_{n-2}} \times \dots \times {c_i}^{d_i} \times \dots \times {c_1}^{d_1} \times {c_0}^{d_0}) \bmod n$
   Atak yapıldığı durumda ise şu şekilde olacaktır:
   $m' = \left({c_{n-1}}^{d_{n-1}} \times {c_{n-2}}^{d_{n-2}} \times \dots \times {c_i}^{d_i'} \times \dots \times {c_1}^{d_1} \times {c_0}^{d_0}\right) \bmod n$
5. $\frac{m'}{m} = \frac{{c_i}^{d_i'}}{{c_i}^{d_i}} \bmod n$ hesaplanır.
6. Eğer adım 3'deki hesaplama sonucu $\frac{1}{c_i} \bmod n$ ise $d_i = 1$, $c_i \bmod n$ ise $d_i = 0$ olduğu anlaşılır.

Benzer şekilde geçici bit bozulması kapalı anahtar d yerine şifreli veri c'de oluşursa; $m' = \left({c_{n-1}}^{d_{n-1}} \times {c_{n-2}}^{d_{n-2}} \times \dots \times {c_i'}^{d_i} \times \dots \times {c_1}^{d_1} \times {c_0}^{d_0}\right) \bmod n$ ve $\frac{m'}{m} = \frac{{c_i'}^{d_i}}{{c_i}^{d_i}} \bmod n$ olur. Bu durumda hesaplama sonucu $\frac{c_i'}{c_i}$ ise $d_i = 1$, 1 ise $d_i = 0$ olduğu anlaşılır.

Bir diğer hata enjekte etme atağı ise [42]'deki CRT kullanan RSA şifre çözme gerçeklemelerine yönelik hata enjekte etme atağıdır. Bu atak literatürde Bellcore

atağı olarak geçmektedir. Tek bir hatalı şifre çözme işlemi yaptırılarak kapalı anahtar bulunmaktadır. Tez kapsamında CRT yöntemi kullanılmadığından bu hata enjekte etme yöntemi üzerinde durulmamıştır.

Tez kapsamında [41]'de Bölüm 2.2'de anlatılan atak ile RSA algoritmasına atak yapılmış ve bu atağa karşı önlem olarak [16]'de önerilen zaman çoğullamalı TMR yöntemini esas alan bir yöntem kullanılmıştır.

5. ÜZERİNDE LİNUX İŞLETİM SİSTEMİ KOŞAN 3 ÇEKİRDEKLİ LEON3 İŞLEMCİSİNİN GERÇEKLENMESİ

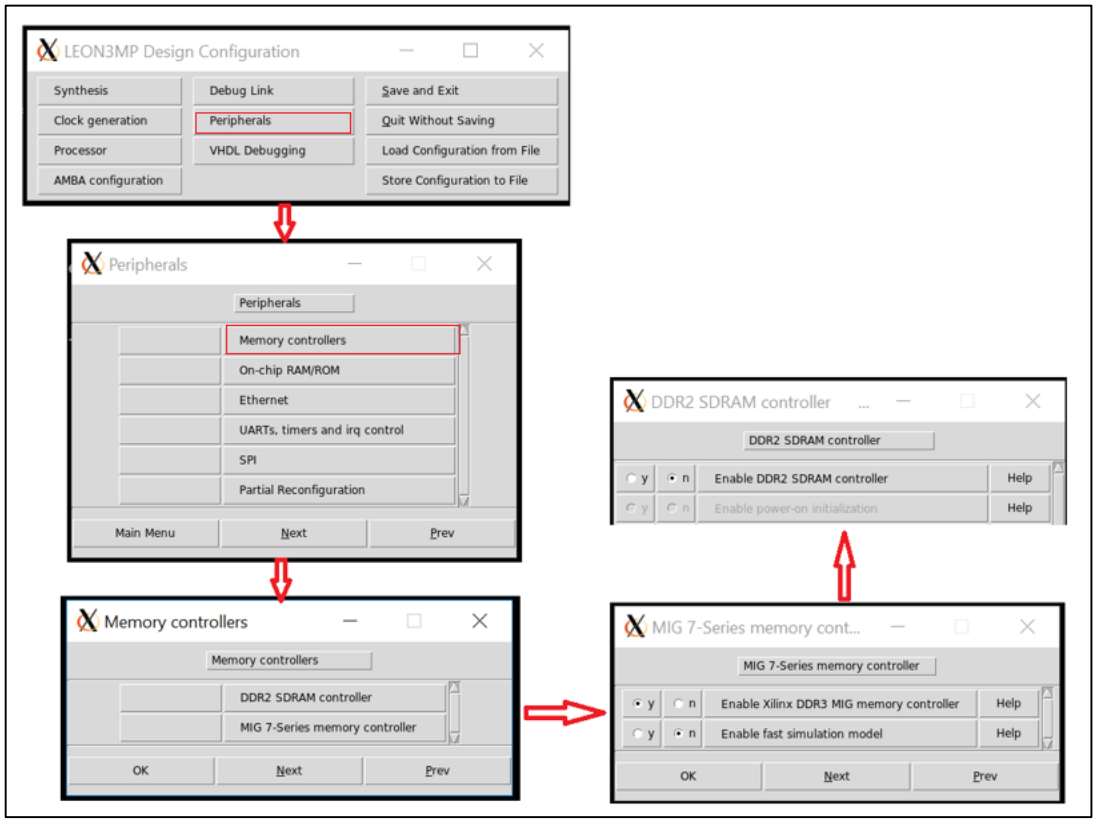
Bu bölümde sırasıyla 3 çekirdekli Leon3 işlemcisinin Nexys4DDR FPGA kartı üzerinde gerçeklenmesi, GCC derleyicisinin çapraz derlenmesi, kök dosya sisteminin oluşturulması ve Linux işletim sisteminin derlenme işlemleri anlatılmıştır.

5.1 3 Çekirdekli Leon3 Sisteminin Gerçeklenmesi

Leon3 işlemcisi tabanlı kırmık üstü sistem tasarımı tipik olarak GRLIB kütüphanesinde bulunan örnek tasarımlar kullanılır [21]. Bu örnek tasarımlara GRLIB kütüphanesi klasörünün altındaki designs klasörü altında bulunur. Bu örnek tasarımlar her bir FPGA geliştirme kartı için leon3-<üretici adı>-<kart> şeklinde isimlendirilmiştir. Tez kapsamında kullanılan Nexys4DDR kartı için GRLIB kütüphanesinde bulunan leon3-digilent-nexys4ddr projesi kullanılmıştır. Proje içerisinde bulunan leon3mp.vhd isimli VHDL kaynak kodu çok çekirdekli tasarımı destekleyecek şekilde yazılmıştır. Leon3 işlemcisinin veri ve komut önbelleği, FPU, MMU(Memory Management Unit) gibi desteklediği özellikler tamamiyle konfigüre edilebilir halde parametrik olarak yazılmıştır. Aynı zamanda leon3mp.vhd VHDL kaynak kodu içerisinde Nexys4DDR kartının desteklediği RS232, Ethernet, Video Grafik Dizisi (Video Graphics Array - VGA) gibi arayüzlere ve DDRAM'e erişim için kullanılan modüllerde parametrik olarak aktif veya pasif hale getirilebilmektedir. Bahsedilen işlemci özellikleri ile Nexys4DDR FPGA geliştirme kartı arayüzlerine erişim modüllerinin aktif veya pasif hale gelmesi için kullanılan parametreler leon3mp.vhd ile aynı dizinde bulunan config.vhd dosyasında tanımlanmıştır. Bu parametreler xconfig kullanıcı arayüzü üzerinden kontrol edilebilmektedir.

Tasarım esnasında bazı sorunlarla karşılaşılmış ve gerek internetten araştırma gerekse GRLIB kütüphanesi dökümanları incelenerek çözümler bulunulmuştur. Nexys4DDR örnek tasarımı Leon3 işlemcisinin ana belleği olarak kartın üzerinde bulunan DDRAM'i kullanmaktadır. Ancak tasarım sentezlenip karta yüklendiğinde DDRAM'a erişimde hatalar alınmış ve Leon3 üzerinde çalıştırılmak istenen örnek

bir kod yüklenip çalıştırılamamıştır. Bu sorunu çözebilmek için xconfig arayüzünde Şekil 5.1'de görüldüğü gibi GRLIB'in DDRAM'a erişim modülü olan DDR2 SDRAM kontrolcü isimli çevresel modülün pasif, Xilinx MIG-7 Serisi Bellek kontrolcüsü [44]'nün aktif hale getirilmesi gerekmektedir.

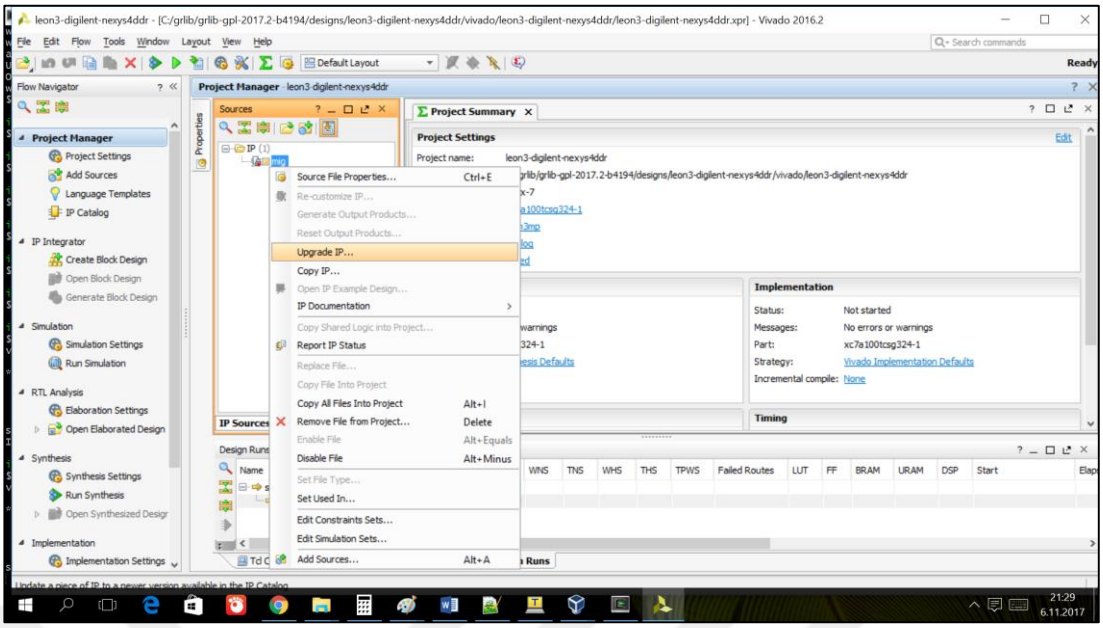


Şekil 5.1 : DDRAM'e erişim için gerekli ayarların yapılması.

Şekil 5.1'de gösterilen işlem yapıldıktan sonra Leon3 işlemcisinin Xilinx MIG-7 Serisi Bellek kontrolcüsü'nün senteze dahil olabilmesi için şu komut çalıştırılır:

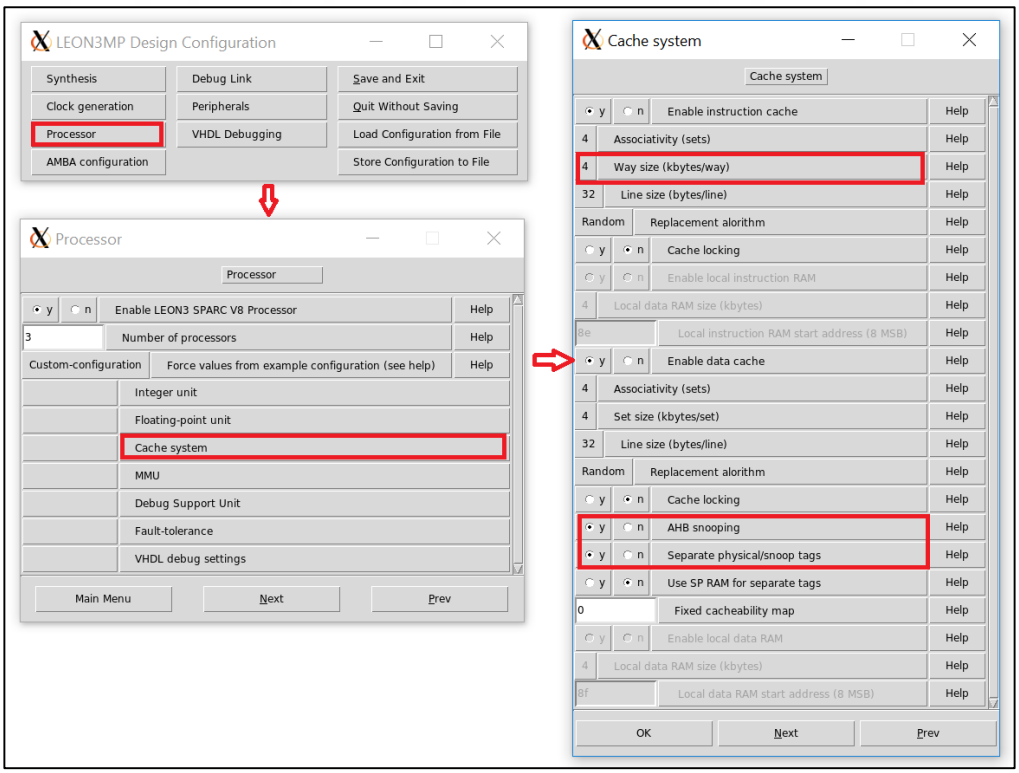
```
make mig_7series
```

Kullanılan Vivado versiyonu "make mig_7series" komutuyla oluşturulan Xilinx MIG-7 Serisi Bellek kontrolcüsü IP'sinin versiyonuyla uyumsuz ise sentez aşamasında hata oluşacaktır. Bu hata Şekil 5.2'de gösterilen şekilde Xilinx MIG-7 Serisi Bellek kontrolcüsü IP'sinin Vivado versiyonuyla uyumlu olması için güncellenmesi ile düzeltilir.



Şekil 5.2 : Xilinx MIG-7 Serisi Bellek kontrolcüsü'nün güncellenmesi.

Dikkat edilmesi gereken bir diğer nokta ise Linux işletim sisteminin SMP modunda çalışacak olmasıdır. Leon3 işlemcisinin SMP modunda çalışmayı desteklemesi için çekirdek sayısının 3 olarak ayarlanması yeterli değildir. Aynı zamanda veri ve komut ön belleği için Şekil 5.3'deki ayarların da yapılması gerekmektedir.



Şekil 5.3 : Leon3 için veri ve komut ön bellek ayarlarının yapılması.

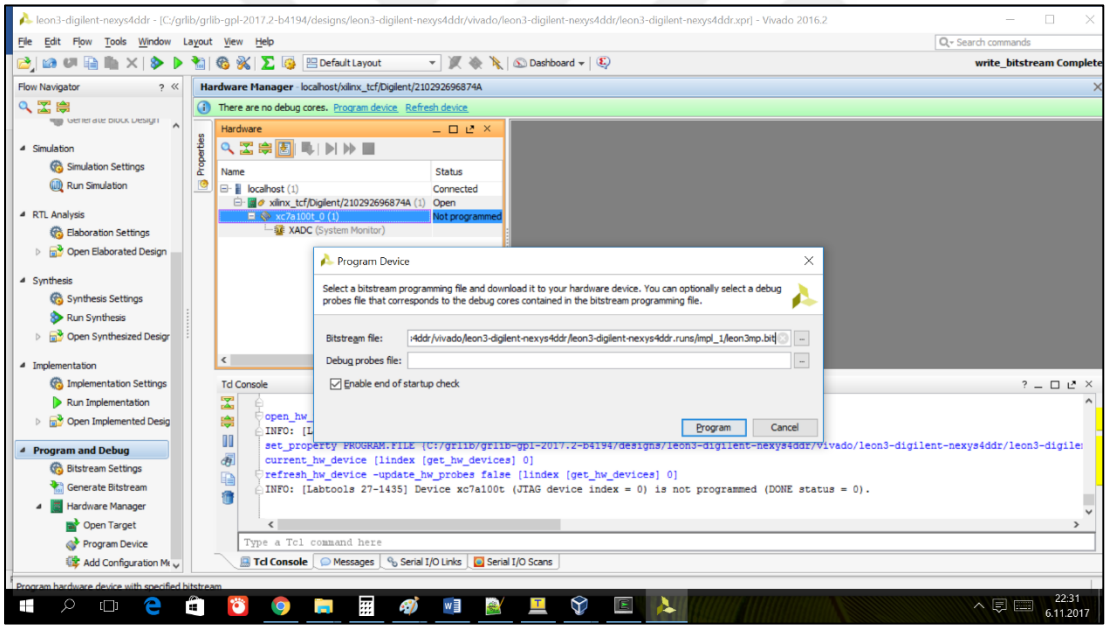
Bu aşamada artık tasarım son haline ulaşmıştır. Aşağıdaki komut çalıştırılarak sentezin yapılacağı Vivado projesi oluşturulur.

```
make vivado-launch
```

Vivado projesi oluşturulduktan sonra Vivado üzerinde aşağıdaki adım izlenerek FPGA'ya yüklenecek olan dosya (.bit uzantılı dosya – bit dosyası) üretilir.

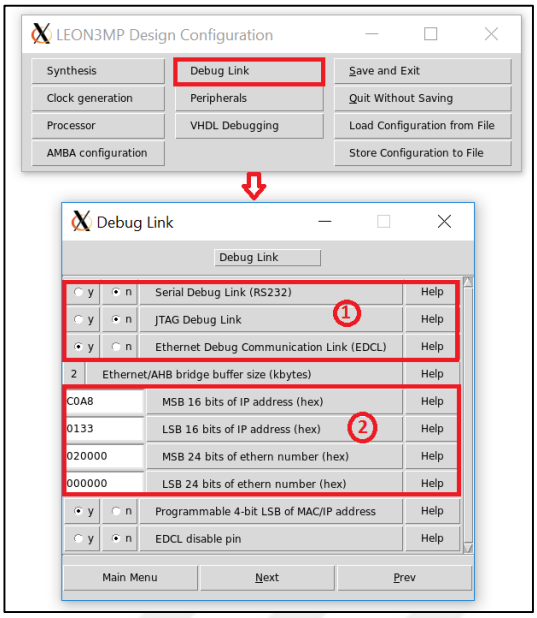
```
Flow Navigator→Program And Debug→Generate Bitstream
```

"Generate Bitstream" işlemi bittiğinde "designs\leon3-digilent-nexys4ddr\vivado\leon3-digilent-nexys4ddr\leon3-digilent-nexys4ddr.runs\impl_1" dizininin altında leon3mp.bit dosyası oluşturulmuş olur. Leon3mp.bit dosyası FPGA'ya yüklenecek olan konfigürasyon dosyasıdır. Bu dosya Vivado üzerinde Şekil 5.4'de gösterildiği gibi FPGA'ya yüklenir.



Şekil 5.4 : Leon3 sisteminin Nexys4DDR geliştirme kartına yüklenmesi.

Bu adımdan sonra Leon3 kırmık üstü sistemine GRMON ile erişip DDRAM'e kod yüklenerek çalıştırılabilir. Leon3 sistemine program yükleme ve hata ayıklama amacıyla RS232, Joint Test Action Group (JTAG) veya Ethernet arayüzleri kullanılabilir. Bu arayüzlerin biri veya birkaçı Şekil 5.5'deki gibi xconfig menüsü üzerinden aktif veya pasif hale getirilerek kullanılabilir.



Şekil 5.5 : Leon3 sisteminin program yükleme ve hata ayıklama arayüzünün seçilmesi.

Tez kapsamında program yükleme ve hata ayıklama arayüzü olarak ethernet arayüzü seçilmiştir. Ethernet arayüzü için Ortam Erişim Kontrolü (Media Access Control – MAC) ve İnternet Protokolü (Internet Protocol - IP) adresi Şekil 5.5'de görülen 2 numaralı alandan ayarlanabilmektedir. GRMON ile Leon3 sistemine ethernet arayüzünden bağlanmak için şu komut çalıştırılır:

```
grmon.exe -nb -eth ip-address-of-leon3
```

GRMON ile Leon3 sistemine bağlandığımızda Şekil 5.6'da görüldüğü gibi sistemde bulunan IP'ler ve işlemci çekirdekleri görülmektedir.

Leon3 sistemine bağlandıktan sonra Leon3 işlemcisi için derlenmiş bir program veya Linux işletim sistemi imajı Şekil 5.7'de görüldüğü gibi DDRAM'e "load program-adı" komutu ile yüklenip "run" komutu ile çalıştırılabilir.

5.2 GCC Derleyicisinin Leon3 İşlemcisi İçin Çapraz Derlenmesi

Leon3 üzerinde çalışacak Linux işletim sisteminin derlenebilmesi ve aynı zamanda C/C++ dilleri ile yazılmış herhangi bir programın derlenmiş olan Linux üzerinde çalışabilmesi için GCC derleyicisinin Leon3 işlemcisine kod üretecek şekilde derlenmiş olarak bahsi geçen derleme işlemlerinde kullanılması gerekmektedir.

```
/cygdrive/c/grlib/grlib-gpl-2017.2-b4194/designs/leon3-digilent-nexys4ddr

isdemir@isdemir /cygdrive/c/grlib/grlib-gpl-2017.2-b4194/designs/leon3-digilent-nexys4ddr
$ grmon.exe -nb -eth 192.168.1.48

  GRMON2 LEON debug monitor v2.0.85 64-bit eval version

  Copyright (C) 2017 Cobham Gaisler - All rights reserved.
  For latest updates, go to http://www.gaisler.com/
  Comments or bug-reports to support@gaisler.com

  This eval version will expire on 14/12/2017

 Ethernet startup...
  GRLIB build version: 4194
  Detected frequency:  70 MHz

  Component                            Vendor
  LEON3 SPARC V8 Processor             Cobham Gaisler
  LEON3 SPARC V8 Processor             Cobham Gaisler
  LEON3 SPARC V8 Processor             Cobham Gaisler
  GR Ethernet MAC                      Cobham Gaisler
  AHB/APB Bridge                       Cobham Gaisler
  LEON3 Debug Support Unit             Cobham Gaisler
  Xilinx MIG DDR2 Controller           Cobham Gaisler
  Generic AHB ROM                      Cobham Gaisler
  Generic UART                         Cobham Gaisler
  Multi-processor Interrupt Ctrl.      Cobham Gaisler
  Modular Timer Unit                   Cobham Gaisler

  Use command 'info sys' to print a detailed report of attached cores

grmon2>
```

Şekil 5.6 : GRMON ile Ethernet arayüzünden Leon3 sistemine bağlanma.

```
grmon2> load linux.dsu
load linux.dsu
  40000000 .text                    4.2kB /   4.2kB    [===============>] 100%
  400010B0 .data                     80B              [===============>] 100%
  40004000 .vmlinux                 6.6MB /   6.6MB    [===============>] 100%
  4069A720 .startup_prom           31.4kB /  31.4kB    [===============>] 100%
  Total size: 6.62MB (23.17Mbit/s)
  Entry point 0x40000000
  Image C:/grlib/grlib-gpl-2017.2-b4194/designs/leon3-digilent-nexys4ddr/linux.dsu loaded

grmon2> run
run
```

Şekil 5.7 : DDRAM'e program yükleme.

GCC'nin bir işletim sistemi üzerinde başka bir işletim sistemine kod üretecek şekilde derlenmesi işlemi çapraz derleme olarak adlandırılmaktadır. Çapraz derlenmiş GCC derlendiği işletim sistemi üzerinde çalışacak ancak derlediği kodlar sadece Leon3 üzerinde koşan Linux işletim sisteminde çalışabilecektir. GCC derleyicisini ve Linux işletim sistemini derlemek için öncelikle aşağıdaki komutla istenen bir dizine 'sparc-linux' isimli bir klasör oluşturulur:

```
mkdir /somewhere/sparc-linux
```

Daha sonra aşağıdaki komutlarla bazı ortam değişkenleri oluşturulur:

```
PATH="$PATH:/somewhere/sparc-linux/usr/bin"
```

```
TOOLS=/somewhere/sparc-linux
```

GCC'nin çapraz derlenmesi için [49]'da anlatılan 1. Bölümdeki adımlar izlenerek GCC Leon3 için çapraz derlenir. Derleme adımları şu şekildedir:

- GNU bağlayıcı(linker) ve assembler(çevirici) gibi araçları içeren binutils paketinin 2.23 versiyonu [45] adresinden indirilir. Aşağıdaki komutlar çalıştırılarak SPARC V8 mimarisi için binutils kurulumu yapılır:

```
tar xzvf binutils-2.23.tar.gz
mkdir b-binutils
cd b-binutils
../binutils-2.23/configure --prefix=${TOOLS}/usr \
--target=sparc-leon3-linux --with-sysroot=${TOOLS}\
--disable-nls   --disable-multilib --with-cpu=v8 \
--with-float=soft
make
make install
cd ..
rm -r b-binutils
```

  'configure' komutunun çalışması esnasında alınan ve uyarıların hata olarak algılanıp konfigürasyon işleminin durdurulmasına neden olan bazı hataları gidermek için çalıştırılan komuta "--disable-werror" parametresi eklenerek bu hatalar çözülmüştür.

- Linux 3.8 çekirdeği [46] adresinden indirilir. Aşağıdaki komutlar çalıştırılarak linux çekirdeği başlık dosyaları kurulumu yapılır:

```
tar xjvf linux-3.8.tar.bz2
cd linux-3.8
LINUXSRC=$PWD

make mrproper
make ARCH=sparc headers_check
make ARCH=sparc INSTALL_HDR_PATH=${TOOLS}/usr\
headers_install
```

- Bu adımda GCC derleyicisi kismi olarak derlenir. GCC derleyicisinin derlenebilmesi için libc küüphanesine ihtiyaç vardır. Ancak aynı şekilde libc kütüphanesinin derlenebilmesi için de GCC'ye ihtiaç vardır. Bu durumun üstesinden gelebilmek için önce GCC derleyicisi libc'yi derleyebilecek şekilde kısmi olarak derlenir. Daha sonra kısmi derlenmiş olan GCC ile libc derlenir. Son olarak GCC derleme işlemi tam olacak şekilde yeniden yapılır. GCC'nin 4.7.2 versiyonu [47] adresinden indirilir. Aşağıdaki komutlarla GCC kısmi olarak derlenmiş olur:

```
tar xjvf gcc-4.7.2.tar.bz2
mkdir b-gcc
cd b-gcc
../gcc-4.7.2/configure --prefix=${TOOLS}/usr \
--target=sparc-leon3-linux \
--disable-nls --disable-shared --disable-multilib \
--disable-libgomp --disable-libmudflap \
--disable-libssp --disable-threads \
--enable-languages=c --with-cpu=v8 --with-float=soft
make all-gcc all-target-libgcc
make install-gcc install-target-libgcc
cd ..
rm -r b-gcc
```

  Bu adımda alınan bölümleme hatası(segmentation fault)'nın çözümü için GCC paketi içerisindeki "ira-int.h" başlık dosyası [50]'den indirilen ira-int.h dosyası ile değiştirilmiştir.
- uClibc C kütüphanesinin 0.9.33.2 versiyonu [48] adresinden indirilir. Aşağıdaki komutlarla libc derlenir:

```
tar xjvf uClibc-0.9.33.2.tar.bz2
cd uClibc-0.9.33.2
make menuconfig
make
make PREFIX=${TOOLS} install
```

Bu adımında “make PREFIX=${TOOLS} install” komutunun sonuna “CROSS_COMPILE=sparc-linux-“ eklenmelidir. Aksi takdirde uClibc kütüphanesinin kurulumu için gerekli derleyici olan sparc-linux-gcc derleyicisi bulunamamaktadır.

Yukarıdaki komutlardan 'make menuconfig' çalıştırılınca libc'nin konfigürasyonunu yapabileceğimiz bir menüyle karşılaşılır. Bu menü üzerinden Şekil 5.8'deki ayarların yapılması gerekmektedir. Libc 0.9.33.2 versiyonu [48] adresinden indirilir. Aşağıdaki komutlarla libc derlenir:

```
Target Architecture          :   sparc
    Target Options
    Processor type           :   SPARC v8
    Target has MMU           :   yes
    Utilize MMU              :   yes
    Enable floating point    :   yes
    Target has FPU           :   no
    Enable C99 math          :   yes
    KERNEL_HEADERS           :   /somewhere/sparc-linux/usr/include
General Library Settings
    Thread support           :   native POSIX threading
    SUSv3 legacy functions   :   yes
    SUSv3 legacy macros      :   yes
    SUSv4 legacy functions   :   yes
Networking
    IPv6                     :   yes
    RPC support              :   yes
        Full RPC             :   no
    DNS resolver functions   :   yes
String support
    ctype argument checking  :   detect and handle
Installation options
    RUNTIME_PREFIX           :   /
    DEVEL_PREFIX             :   /usr
    HARDWIRED_ABSPATH        :   no
Development
    CROSS_COMPILER_PREFIX    :   sparc-leon3-linux-
```

Şekil 5.8 : GCC derleyicisinin ayarları.

- Bu adımda GCC derleyicisi aşağıdaki komutlar çalıştırılarak tamamen kullanılabilecek şekilde derlenir:

```
mkdir b-gcc
cd b-gcc
../gcc-4.7.2/configure --prefix=${TOOLS}/usr \
--target=sparc-leon3-linux --with-sysroot=${TOOLS} \
```

```
--disable-nls --enable-shared --disable-multilib \
--disable-libgomp --disable-libmudflap \
--enable-threads=posix --enable-languages="c,c++" \
--with-gnu-as --disable-libitm \
--with-cpu=v8 --with-float=soft
make
make install-strip
cd ..
rm -r b-gcc
ln -s sparc-leon3-linux-gcc \
${TOOLS}/usr/bin/sparc-linux-gcc
ln -s sparc-leon3-linux-ld ${TOOLS}/usr/bin/sparc-linux-ld
ln -s sparc-leon3-linux-objdump \
${TOOLS}/usr/bin/sparc-linux-objdump
ln -s sparc-leon3-linux-objcopy \
${TOOLS}/usr/bin/sparc-linux-objcopy
ln -s sparc-leon3-linux-readelf \
${TOOLS}/usr/bin/sparc-linux-readelf
```

GCC'nin çapraz derlenmesinden sonra Leon3 işlemcisi için Linux işletim sisteminin derlenmesinde kullanılacak olan "sparc-linux-" önekli araçlar üretilmiş olur. Leon3 üzerinde koşacak Linux işletim sistemi için yazılacak C ve C++ programlarının derlenmesinde de yine aynı araçlardan sparc-linux-gcc ve sparc-linux-g++ derleyicileri kullanılacaktır.

5.3 Kök Dosya Sisteminin Oluşturulması

Linux işletim sisteminde önemli sistem dosyaları ve kütüphaneleri kök dosya sisteminde tutulur. Aynı zamanda tüm dosya sistemleri kök dosya sisteminde bir dizine bağlanmış olarak bulunur. Kök dosya sistemi [49]'da bölüm 2'de anlatılan aşağıdaki adımlar uygulanarak oluşturulur:

- Aşağıdaki komutlar çalıştırılarak kök dosya sistemi için çalışılan bilgisayarda bir klasör oluşturulup gerekli kütüphaneler bu klasöre kopyalanır:

```
mkdir /somewhere/sysroot
```

```
export SYSROOT=/somewhere/sysroot
cp -a ${TOOLS}/lib ${SYSROOT}
cp -a ${TOOLS}/usr/sparc-leon3-linux/lib/*.so* \
${SYSROOT}/lib
cd ${SYSROOT}/lib
ln -s ld-uClibc*.so ld-linux.so.2
```

- Busybox'ın 1.20.2 versiyonu [51] adresinden indirilir. Busybox komut satırından çalıştırılan birçok komutu içeren tek bir çalıştırılabilir dosyadır. Aşağıdaki komutlarla busybox kök dosya sistemine kurulur:

```
tar xjvf busybox-1.20.2.tar.bz2
cd busybox-1.20.2
make menuconfig
make busybox
make CONFIG_PREFIX=${SYSROOT} install
ln -s bin/busybox ${SYSROOT}/init
```

  Yukarıdaki komutlardan 'make menuconfig' çalıştırılınca busybox'ın konfigürasyonunu yapabileceğimiz bir menüyle karşılaşılır. Bu menü üzerinden Şekil 5.9'deki ayarların yapılması gerekmektedir.
- [56] adresinden kök dosya sisteminde bulunan ve çeşitli ayarları içeren dosyalar indirilir ve aşağıdaki şekilde kök dosya sistemine kopyalanır:

```
tar xzvf leon3-linux-stuff-20130228.tar.gz
cp -a leon3-linux-stuff-20130228/etc ${SYSROOT}
```

```
BusyBox Settings:
    General configuration:
        Don't use /usr                          :   yes
        Support Unicode                         :   no
    Build Options                               :
        CONFIG_CROSS_COMPILER_PREFIX            :   sparc-leon3-linux-
        CONFIG_SYSROOT                          :   /somewhere/sparc-linux
        CONFIG_EXTRA_CFLAGS                     :   -D_XOPEN_SOURCE
    Library Tuning                              :
        systemd support                         :   no
        Use clock_gettime(CLOCK_MONOTONIC)      :   yes
        Support infiniband                      :   no
    Print Utilities                             :   hepsi engellenecek
    Mail Utilities                              :   hepsi engellenecek
    Shell Utilities                             :
        hush                                    :   no
```

Şekil 5.9 : Busybox ayarları.

5.4 Linux İşletim Sisteminin Derlenmesi

Linux işletim sistemi açık kaynak kodlu ve birçok platformu destekleyen bir işletim sistemidir. Linux işletim sistemi çekirdeği Sparc V8 uyumlu olan Leon3 işlemcisini de desteklemektedir. [49]'da bölüm 3'de anlatılan aşağıdaki adımlar izlenerek Linux işletim sistemi Leon3 için derlenmiştir.

- Aşağıdaki komutlar ile LEON3 için linux kaynak kodlarına yapılması gereken iki yama yapılır:

```
cd linux-3.8
patch -p1 < ../leon3-linux-stuff-20130228\
/patches/linux-3.8_sparcfp.diff
patch -p1 < ../leon3-linux-stuff-20130228\
/patches/linux-3.8_apbps2-added-\
GRLIB-APBPS2-PS-2-Keyboard-and-Mouse-d.patch
```

- Aşağıdaki komut çalıştırılarak linux çekirdeği için konfigürasyon işlemi açılan menü üzerinden yapılabilir:

```
make ARCH=sparc \
CROSS_COMPILE=sparc-leon3-linux- menuconfig
```

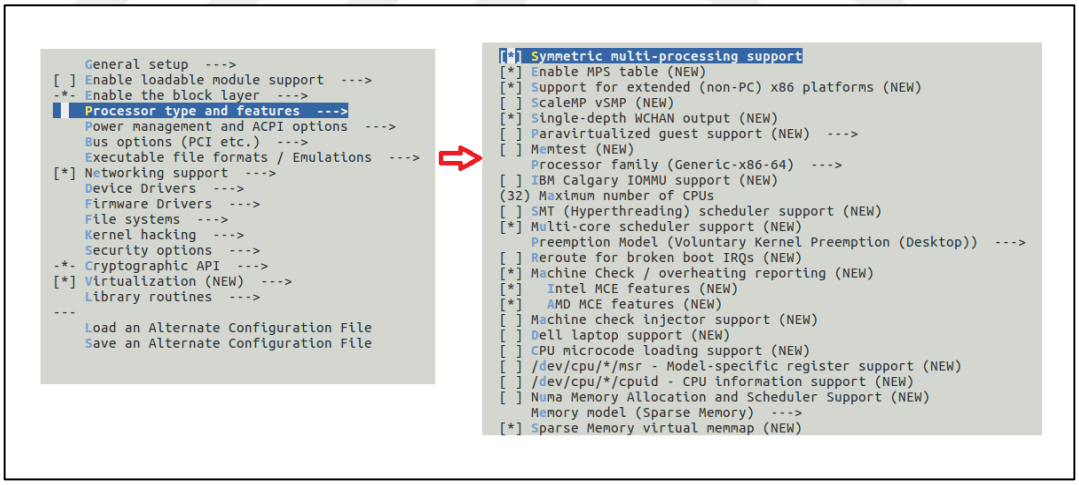
Kongfigürasyon işlemi için 'leon3-linux-stuff-20130228/linux-3.8-config' dosyası '.config' dosyası üzerine kopyalanarak LEON3 için gerekli ayarların

yapıldığı bir konfigürasyon üzerinden de devam edilebilir. Tez kapsamında bu yol izlenmiştir. 'leon3-linux-stuff-20130228/linux-3.8-config' dosyası '.config' dosyası üzerine kopyalandıktan sonra yine menü üzerinden Linux işletim sisteminin SMP modunu desteklemesi için Şekil 5.10'daki gibi "processor types and features" sekmesinden "symmetric multiprocessing support" özelliği aktif hale getirilmelidir.

- 'leon3-linux-stuff-20130228/make_initramfs.sh' betik dosyası çalıştırılarak kök dosya sistemine girecek dizinler oluşturulur.
- Aşağıdaki komutlar çalıştırılarak linux işletim sistemi derlenerek Linux kaynak kodlarının bulunduğu dizinde 'vmlinux' isimli linux imaj dosyası oluşturulmuş olur:

```
cd ${LINUXSRC}
make ARCH=sparc CROSS_COMPILE=sparc-leon3-linux-
```

Bu adımda "arch_jump_label_transform" isimli fonksiyonunun bulunamaması nedeniyle hata alınmıştır. Bu hatanın çözümü için Linux çekirdeğine [52] adresinden indirilen yama uygulanmıştır.



Şekil 5.10 : Linux çekirdeği için SMP modunun aktifleştirilmesi.

Leon3 üzerinde çalışacak olan Linux imajı içerisinde olmasını istediğimiz bir program var ise kök dosya sistemine eklenmiş olmalıdır. Linux çekirdeği derlendiğinde kök dosya sistemiyle birlikte vmlinux isimli imaj dosyası oluşur. Bu dosyaya yeni bir program eklemek istendiğinde, program kök dosya sisteminde "bin" ya da "sbin" dizinlerine konur. Daha sonra son iki adım tekrar edilir. Bu adımların tekrar edilmesi ilk yapılmalarına göre çok kısa sürmektedir. Çünkü Linux çekirdeği

zaten derlenmiş durumdadır. Sadece kök dosya sistemi ile Linux çekirdeği birleştirilip vmlinux imaj dosyası oluşturulacaktır. Tez kapsamında yazılan RSA gerçeklemesi “sbin” dizinine eklenerek vmlinux imaj dosyası oluşturulmuştur.

Tek başına vmlinux imaj dosyası Leon3 işlemcisi üzerinde çalışabilmesi için yeterli değildir. Linux imajının RAM’de belirli bir bölgeye yüklenmesi ve platform spesifik ilklendirmelerin yapılması için bir önyükleyici program gerekmektedir. Bunun için SPARC OPENPROM(Scalable Processor Architecture Programmable Read Only Memory) önyükleyicisi kullanılmaktadır. Gaisler firmasının mklinuximg [53] aracının 2.6.36-2.0.3 versiyonu [57] adresinden indirilip iki adet yama uygulanır:

```
tar xjvf mklinuximg-2.6.36-2.0.3.tar.bz2
cd mklinuximg-2.6.36-2.0.3
patch -p1 < ../leon3-linux-stuff-20130228 \
/patches/mklinuximg-2.6.36-2.0.3_compat.diff
patch -p1 < ../leon3-linux-stuff-20130228 \
/patches/mklinuximg-2.6.36-2.0.3_fixld.diff
```

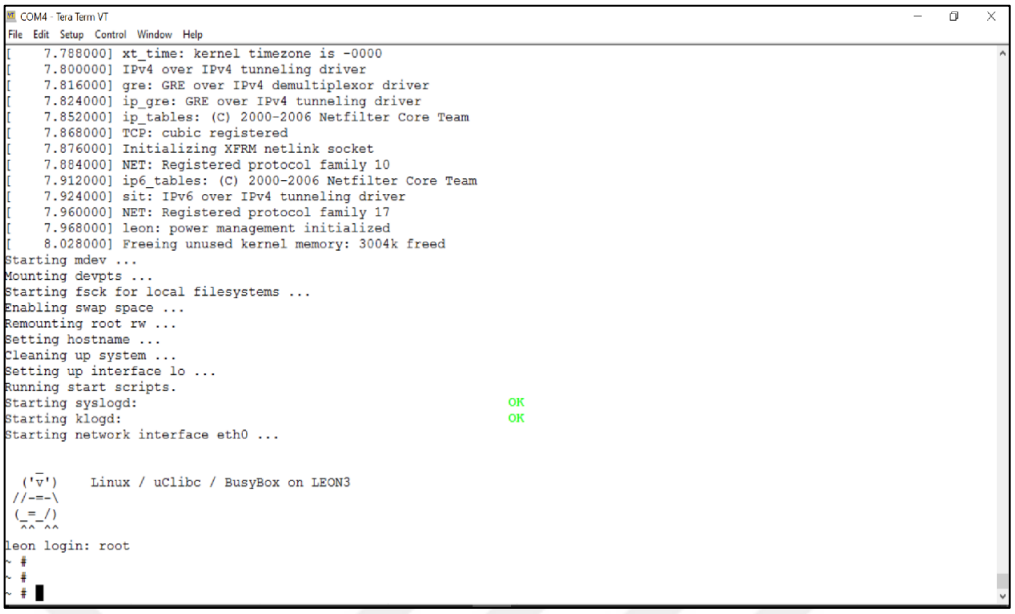
Aşağıdaki komut kullanılarak LEON3 için SPARC OPENPROM ön yükleyicisini de içeren çalıştırılabilir 'linux.dsu' oluşturulur:

```
mklinuximg-2.6.36-2.0.3/mklinuximg ${LINUXSRC}/vmlinux linux.dsu
```

Oluşturulan imaj dosyası(linux.dsu) doğrudan GRMON aracılığıyla LEON3 sisteminin DDRAM’ine aşağıdaki komutlarla yüklenip çalıştırılabilir:

```
load linux.dsu
run
```

Linux işletim sisteminin Leon3 üzerindeki açılış ekranı Şekil 5.11’de gösterilmiştir. Linux işletim sistemine Nexys4DDR kartının RS232 arayüzünden bağlanılmaktadır. RS232 arayüzüne bağlanabilmek için bilgisayar üzerinde Tera Term[54] isimli uygulama kullanılmıştır.



Şekil 5.11 : Linux terminali açılış ekranı.

6. RSA ALGORİTMASININ HATA ENJEKTE ETME ATAĞINA DAYANIKLI GERÇEKLENMESİ

Bu bölümde hata enjekte etme atağına dayanıklı RSA algoritması gerçekleme adımları anlatılmıştır. Sırasıyla RSA algoritmasının C++ programlama dili ile gerçeklenmesi, hata enjekte etme atağının yapılması ve RSA algoritmasının hata enjekte etme atağına karşı dayanıklı hale getirilmesi detaylı bir şekilde anlatılmıştır. Son olarak önerilen yöntemin analizi yapılmış ve elde edilen sonuçlar değerlendirilmiştir.

6.1 RSA Algoritmasının C++ Programlama Dili İle Gerçeklenmesi

Tez kapsamında RSA algoritması C++ dili ile yazılmış ve sparc-linux-g++ derleyicisi kullanılarak Leon3 işlemcisi üzerinde çalıştırılmıştır. RSA algoritmasında kullanılan açık ve kapalı anahtarlar ile şifrelenen ve şifresi çözülen veri [7]'de belirtilen seçeneklerden 1024 bit olarak alınmıştır. C++ dilinde işletim sistemine de bağlı olarak kullanılabilecek en büyük sayıların 128 bitlik olduğu göz önüne alındığında 1024 bitlik sayılarla aritmetik işlemlerin standard C++ değişken tipleriyle gerçekleştirilemeyeceği açıktır. Bu durumda GMP [55] kütüphanesi kullanılabilir veya büyük sayılarla işlem yapabilmek için bir C++ sınıfı yazılabilir. Bu çalışmada büyük sayılarla işlem yapabilmek için BigInt isimli bir C++ sınıfı yazılmıştır. BigInt sınıfının bazı özellikleri şunlardır:

- İstenen büyüklükteki sayılar üzerinde çarpma, bölme, toplama, çıkarma ve mod işlemi gibi aritmetik işlemler
- Büyük sayılar üzerinde sağa ve sola kaydırma işlemleri
- Mod işlemine göre bir sayının tersinin bulunması
- İstenen büyüklükte rastgele sayı üretimi
- Bit bit erişim olanağı

Şekil 6.1'da BigInt sınıfı ile yapılan örnek işlemler ve sonuçları görülmektedir. Tüm değerler onaltılık tabanda gösterilmektedir.

Şekil 6.1 : BigInt sınıfı ile yapılan aritmetik işlemler.

BigInt sınıfı RSA algoritması gerçeklemesinde kullanılan temel değişken tipi olarak kullanılmıştır. RSA algoritmasını gerçeklemek üzere RSA_1024 isimli bir C++ sınıfı yazılmıştır. RSA_1024 sınıfı şifreleme ve şifre çözme işlemlerini yapan fonksiyonlara sahiptir. Anahtar üretimi ve anahtarların saklanması için KeyContainer isimli bir C++ sınıfı yazılmıştır. KeyContainer sınıfı verilen p ve q asal sayılarını kullanarak açık ve kapalı anahtarı oluşturur. Anahtar üretim ve saklama işlemleri için RSA_1024 sınıfının kullanılmama sebebi aynı anahtarların birden fazla nesne tarafından kullanılabilmesine imkan tanımak içindir. Bir KeyContainer nesnesi oluşturulup birden fazla RSA_1024 nesnesine verilip aynı anahtarların kullanılması sağlanabilir.

RSA algoritmasında kullanılacak olan açık ve kapalı anahtar parametreleri olan n, e ve d parametrelerinin belirlenmesinde kullanılan p ve q asal sayıları sabit değerler olarak önceden belirlenerek kullanılmıştır. n, e ve d parametreleri Şekil 6.2'de görülen C++ kod parçacıklarıyla yapılmıştır. Görüldüğü üzere önce e parametresi 5 değerden biri rastgele olacak şekilde seçilir. Daha sonra d değerini bulmak için e'nin $mod\ Q$ işlemine göre tersi hesaplanır. Şekil 6.2'de gösterilmemekle birlikte e'nin $mod\ Q$ işlemine göre tersi bulunamazsa tekrar rastgele seçim yapılır.

```
1 - m_n = m_p * m_q;
2 - m_Q = (m_p - 1) * (m_q - 1);

3 - const int eMap[] = {3, 5, 17, 257, 65537};

4 - m_e = BigInt(eMap[rand()%5]);

5 - m_d = new FaultInjectableBigInt(BigInt::modInv(m_e, m_Q));
```

Şekil 6.2 : Açık ve kapalı anahtarın oluşturulması.

RSA algoritmasının şifreleme ve şifre çözme işlemlerinin aynı ve modüler üs alma olduğundan Bölüm 4.2'de bahsedilmişti. Çalışma kapsamında kare al ve çarp yöntemi kullanılarak şifreleme ve şifre çözme işlemleri gerçeklenmiştir. Kare al ve çarp yöntemindeki çarpma ve mod alma adımları için Bölüm 3.4'de anlatılan Montgomery modüler çarpma işlemini kullanan Algoritma 3.4 kullanılmıştır.

Algoritma 6.1 : Montgomery modüler çarpma işlemi tabanlı kare al ve çarp algoritması [35].

GİRİŞ : $M = (m_{n-1}, m_{n-2}, \dots., m_1, m_0)_b$
$X = (x_{n-1}, x_{n-2}, \dots., x_1, x_0)_b$
$E = (e_{n-1}, e_{n-2}, \dots., e_1, e_0)_b$ b tabanında sayılar
$1 \leq X < M,\ R = b^n$

ÇIKIŞ : $X^E \ mod\ M$

1. $X' = Mont(X, R^2)\ , A = R \ mod\ M$
2. For i from $n-1$ downto 0
   2.1. $A = Mont(A, A)$
   2.2. İf $E_i = 1$ then $A = Mont(A, X')$
3. $A = Mont(A, 1)$
4. Return A

Birinci adımdaki A değeri önceden hesaplanarak algoritmaya giriş parametresi olarak verilebilir. Benzer şekilde X' hesaplanırken Montgomery modüler çarpma işlemine giren R^2 değeri de önceden $R^2 \ mod\ m$ olarak hesaplanıp algoritma için giriş olarak kullanılabilir.

Yöntemin 1. adımındaki önceden hesaplanarak algoritmaya giriş parametresi olarak verilebilecek olan değerlerin hesaplanmasında Bölüm 3.5'de anlatılan Barrett mod alma yöntemi kullanılmıştır.

6.2 RSA Algoritmasının 3 Çekirdek Üzerinde Paralel Çalıştırılması

Leon3 işlemcisi üzerinde paralel çalışacak şekilde kod yazabilmek için Pthreads kütüphanesi kullanılmıştır. Bu kütüphane aynı program içerisinde birden fazla iş parçacığı çalışacak şekilde kod yazma ve iş parçacıkları arası senkronizasyon mekanizmaları gibi olanaklar sunar. Aynı zamanda istenen iş parçacığı istenen işlemci çekirdeğinde çalışacak şekilde konfigürasyon yapılmasına da izin vermektedir. Aşağıda verilen kod parçacıkları ile Pthreads kütüphanesinin kullanımı gösterilmiştir.

```
1 - pthread_attr_t attr;
2 - cpu_set_t cpu;
3 - pthread_t thread;

4 - pthread_attr_init(&attr);

5 - CPU_ZERO(&cpu);
6 - CPU_SET(i, &cpu);
7 - pthread_attr_setaffinity_np(&attr, sizeof(cpu_set_t), &cpu);

8 - pthread_create(&thread, &attr, ThreadFunction, NULL);
9 - pthread_join(thread, NULL);
```

Şekil 6.3 : Pthreads kütüphanesinin kullanımı.

Şekil 6.3'de gösterilen kod parçacığında 1,2 ve 3 numaralı adımlarda sırasıyla iş parçacığının sahip olacağı nitelikleri gösteren pthread_attr_t tipinde bir değişken, iş parçacığının hangi işlemci üzerinde çalışacağını gösteren cpu_set_t tipinde bir değişken ve iş parçacığı için eşsiz bir tanımlayıcı numarayı tutan pthread_t tipinde bir değişken tanımlanmıştır. 4,5,6 ve 7 adımlarında iş parçacığının hangi işlemci üzerinde çalışacağı (örnek kodda i numaralı işlemci) belirlenmektedir. 8 numaralı adımda iş parçacığının ThreadFunction isimli fonksiyonu çalıştırmak üzere oluşturulması için gerekli fonksiyon kullanımı gösterilmiştir. Pthread_create fonksiyonunun ilk parametresinde iş parçacığının adresi, ikinci parametresinde ise hangi parametrelerle başlatılacağı bilgisini içeren değişkenin adresi verilmiştir. İş parçacığı oluşturulurken pthread_create fonksiyonunun ikinci parametresine göre

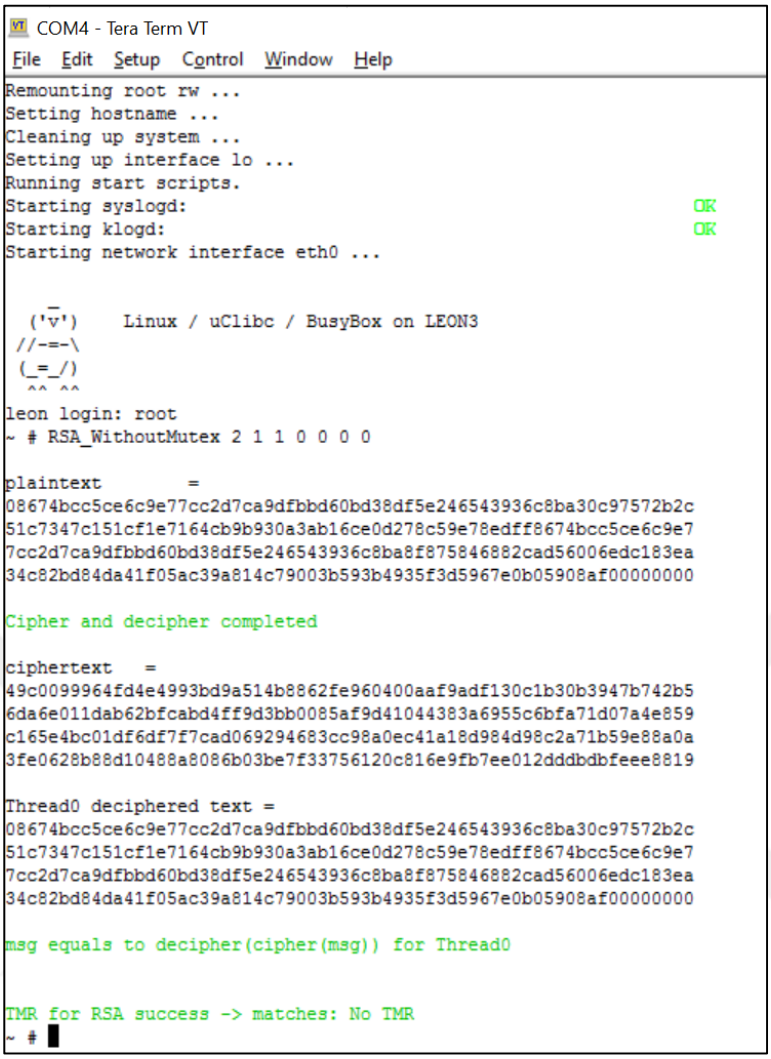
başlatılır ve birinci parametresindeki değişkene iş parçacığına özgü olan bir numara atanır. İş parçacığına özgü olan bu değer daha sonra 9 numaralı adımda gösterilen pthread_join fonksiyonuna parametre olarak verilip iş parçacığı bitene kadar beklenebilir.

6.3 Yazılan Programın Kullanılması

Bu çalışma kapsamında yazılmış olan RSA algoritması gerçeklemesi programı Linux terminalinden çalıştırılırken verilen parametrelere göre çalışmaktadır. Parametreler programın adı yazıldıktan sonra sırasıyla yazılır ve program çalıştırılır. Bu parametreler şunlardır:

1. Mod alma işlemi algoritması(reductionMod): aşağıdaki yöntemlerden biri seçilebilir.
   Klasik yöntem : 0 değeri yazılarak seçilir.
   Barrett mod alma yöntemi : 1 değeri yazılarak seçilir.
   Montgomery modüler çarpma yöntemi : 2 değeri yazılarak seçilir.
2. Çalışacak thread sayısı(threadCnt): 1-3 arasında olabilir.
3. Çalışacak thread'lerin başlatılmaları arasındaki zaman farkı(msThreadStartGap): 0'dan büyük olmalıdır.
4. Hata enjekte etmeyi aktifleştirme(FAEnabled) : Aktifleştirmek için 1, pasifleştirmek için 0 değeri yazılır.
5. Hata enjekte edilen bit pozisyonu(FABitPos) : 0-1023 arasında bir değer olmalıdır.
6. Enjekte edilen hata değeri(FAVal) : 0 ya da 1 olabilir.
7. Enjekte edilen bit hatasının devam etme süresi(msFATime) : sıfırdan büyük bir değer girilmelidir.

Şekil 6.4'de gerçeklenen RSA algoritması ile veri şifreleme ve çözme işlemi sonucu görülmektedir. Görüldüğü üzere gerçeklenen RSA algoritması tek iş parçacığı çalışacak şekilde herhangi bir hata enjekte etmeksizin test edildiğinde şifreli veri çözüldüğünde açık veri tekrar elde edilebilmektedir.



Şekil 6.4 : RSA şifreleme ve şifre çözme işlemi sonucu.

RSA algoritmasını gerçeklemek için yazılan programın mod alma işlemi için giriş parametresine bağlı olarak Montgomery modüler çarpma işlemi veya Barrett mod alma yöntemi kullanması durumları test edilmiştir. Şekil 6.5'de Montgomery yöntemi, Şekil 6.6'da Barrett yöntemi kullanılarak şifreleme ve şifre çözme işlemleri yapılmış ve ne kadar sürede bittikleri ölçülmüştür. Görüldüğü gibi Montgomery modüler çarpma işlemi ile şifreleme ve şifre çözme yapıldığında işlemin daha kısa sürede bittiği görülmektedir.

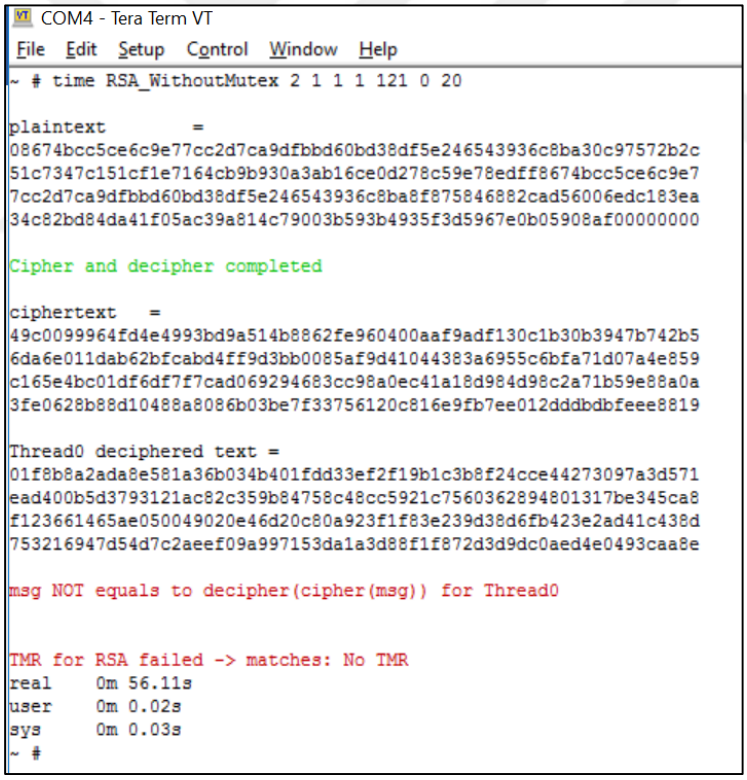
Şekil 6.5 : Montgomery modüler çarpma yöntemi ile şifreleme ve şifre çözme süresi.

Şekil 6.6 : Barrett mod alma yöntemi işlemi ile şifreleme ve şifre çözme süresi.

6.4 RSA Algoritmasına Hata Enjekte Etme Atağı Gerçekleme

RSA algoritmasına hata enjekte etme işlemi için BigInt sınıfından türetilen FaultIntectableBigInt isimli bir sınıf yazılmıştır. Bu sınıfın hata enjekte etme fonksiyonu kullanılarak istenen bitte ilk kez okunmasından başlamak üzere istenen süre boyunca hatalı okuma yapılması sağlanabilir. Hata enjekte etme işlemi kapalı anahtar üzerinden yapılacağı için sadece kapalı anahtar FaultInjectableBigInt sınıfı tipinde oluşturulmuştur.

Şekil 6.7'de kapalı anahtarın 121. bitinin 20 milisaniye boyunca 0 olarak okunmasına neden olan geçici hata enjekte edilmiştir. Sonuçta açık veri ile şifre çözme işlemi sonucunda elde edilen verinin farklı olduğu ve şifre çözme işleminin başarısız olduğu görülmektedir. Bu durumda hatalı sonuç alınmış olması kapalı anahtarın 121. bitinin gerçek değerinin 0 olduğu anlamına gelmektedir.



Şekil 6.7 : Kapalı anahtarın 121. bitinin 0 okunması için hata enjekte etme işlemi sonucu.

Şekil 6.8'de ise yine kapalı anahtarın 121. bitinin 1 20 milisaniye boyunca 1 olarak okunmasına neden olan geçici hata enjekte edilmiştir. Bu durumda ise hatalı sonuç alınmamış olması 121. bitin gerçek değerinin 1 olduğu anlamına gelir.

Şekil 6.8 : Kapalı anahtarın 121. bitinin 1 okunması için hata enjekte etme işlemi sonucu.

Bu yöntemle tüm bitler üzerinde hata enjekte etme işlemi yapıldığında kapalı anahtarın bütün bitlerinin bulunup kapalı anahtarın elde edilebileceği görülmektedir.

6.5 RSA Algoritmasının Hata Enjekte Etme Atağına Dayanıklı Hale Getirilmesi

Bir önceki bölümde gerçeklenen hata enjekte etme atağı ile kapalı anahtarın tamamen elde edilebileceği görülmektedir. Bu bölümde RSA algoritmasının TMR tabanlı bir yöntem kullanılarak hata enjekte etme atağına dayanıklı hale getirilme adımları ve sonuçları detaylı bir şekilde anlatılacaktır.

Hata enjekte etme atağından korunabilmek için gerçeklenen Leon3 sisteminin 3 çekirdeğinin paralel bir şekilde çalıştırılarak TMR yönteminin [16]' da anlatılan yönteme benzer bir şekilde kullanılması yönteminden yararlanılmıştır. Aynı şifreleme ve şifre çözme işlemini 3 işlemci çekirdeği paralel olarak ancak aralarında parametrik olarak belirlenen bir süreyle fark olmak şartıyla yapmaktadır. Bu durumda enjekte edilen geçiçi hatanın oluştuğu süre boyunca hata oluşan biti okuyan işlemci çekirdekleri hata yapacak, hata geçtikten sonra okuyan işlemci çekirdekleri ise hata yapmamış olacaktır. Her üç işlemci çekirdeği de şifreleme ve şifre çözme

işlemlerini bitirdikten sonra 3 sonuçtan çoğunlukta olan seçilecek ve şifre çözme işlemi sonucu olarak kabul edilecektir. Bu durumda enjekte edilen hatanın süresi ve işlemci çekirdeklerinin şifre çözme işlemine başlamaları arasındaki zaman farkı önemli olmaktadır. Şekil 6.9'da her bir işlemci üzerinde çalışacak iş parçacıkları arasında zaman farkı olmaması durumunda 150. bitin 1 olarak okunmasını sağlayan geçici hata enjekte etme durumu gösterilmektedir. Bu durumda TMR yöntemine göre belirlenen sonuç hatalı olmaktadır. Bu bilgiye dayanarak yapılacak olan 150. bitin 0 olduğu bilgisi doğrudur. Görüldüğü üzere TMR yöntemiyle işlem sonucunun doğru olduğu garanti edilememektedir. Bu nedenle iş parçacıklarının çalışmaya başlama süreleri arasındaki fark ve hatanın süresi önemli olmaktadır.

Şekil 6.9 : İş parçacıkları arasında zaman farkının az enjekte edilen hata süresinin uzun olması durumu.

Şekil 6.10'da ise iş parçacıklarının çalışmaya başlamaları arasında 100 milisaniye ve enjekte edilen hatanın süresinin 20 milisaniye olduğu durum gösterilmektedir. Bu durumda TMR yöntemiyle elde edilen işlemin doğru olduğu ve 150. bitin 1 olduğu yönünde yapılan tahminin yanlış olduğu görülmektedir.

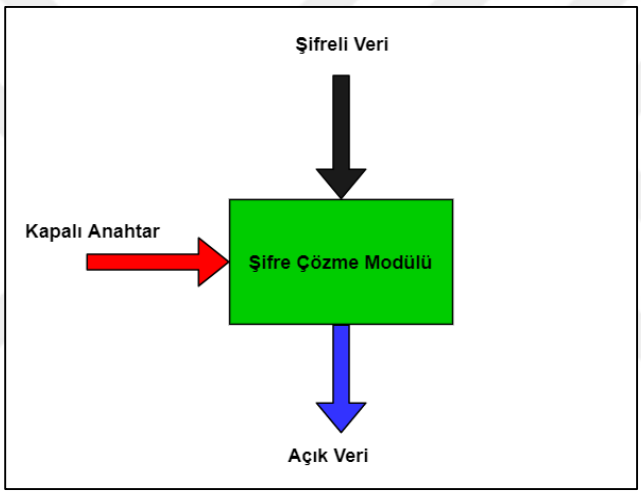
Şekil 6.10 : İş parçacıkları arasında zaman farkının çok enjekte edilen hata süresinin kısa olması durumu.

Yukarıdaki iki örnekten de görülmektedir ki iş parçacıklarının işleme başlama zamanları arasında enjekte edilen hatanın süresine göre makul bir süre olduğu durumda TMR tabanlı yöntem işe yaramakta ve geçici süreli hata enjekte etme işlemi ile kapalı anahtarın elde edilmesi mümkün olmamaktadır.

6.6 Önerilen Yöntemin Analizi

Bölüm 6.5'de verilen örnekler ile RSA algoritmasının hata enjekte etme atağına dayanıklı hale getirilmesi için önerilen yöntemin Leon3 üzerinde çalıştırılıp başarılı olduğu gösterilmiştir. Ancak Leon3 sisteminin performansının çok yüksek olmaması nedeniyle önerilen yöntemin detaylı analiz çalışmaları PC üzerinde koşan Linux işletim sisteminde yapılmıştır. Ayrıca analiz işlemlerinde 128 bitlik anahtar kullanılmıştır. Yapılan analiz çalışmasında Bölüm 1.2'de de bahsedilen şu üç durum karşılaştırılmıştır:

Durum 1: Şekil 6.11'de gösterilen tek bir modülün çalıştığı, yani TMR yönteminin kullanılmadığı durum.



Şekil 6.11 : Tek modül ile şifre çözme.

Bu durum için analiz programı şu şekilde çalıştırılmalıdır:

Programın-adı 1 minimum-test-sayısı maksimum-test-sayısı test-adım-sayısı minimum-CRC-periyodu maksimum-CRC-periyodu CRC-adım-sayısı

Programın adından sonraki "1" parametresi analizin tek modül ile yapılacağını göstermektedir. Sonraki parametrelerin ne için kullanıldığı bilgisi aşağıda verilmiştir:

Minimum test sayısı: Analizin her adımında yapılacak şifre çözme sayısının ilk değeridir.

Maksimum test sayısı : Analizin her adımında yapılacak şifre çözme sayısının son değeridir.

Test adım sayısı : Analizin her adımında bir önceki adıma göre test sayısı bu değer kadar arttırılır.

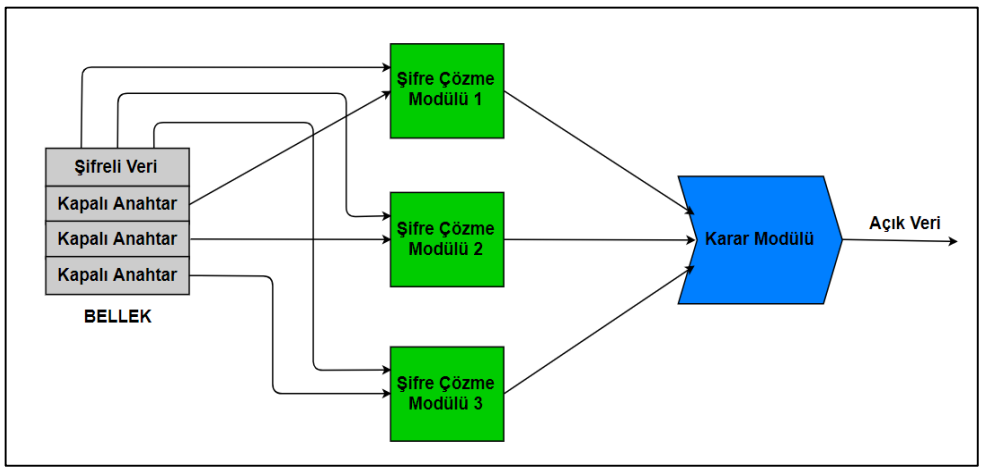
Minimum Döngüsel Artıklık Denetimi (Cyclic Redundancy Check - CRC) periyodu : Analizin kapalı anahtar üzerinde periyodik olarak CRC kontrolü yapıldığı bölümünde kullanılacak CRC periyodunun milisaniye cinsinden ilk değeridir.

Maksimum CRC periyodu : Analizin CRC kullanılan bölümünde kullanılacak CRC periyodunun milisaniye cinsinden son değeridir.

CRC adım sayısı : Analizin CRC kullanılan bölümünde her adımda bir önceki adıma göre CRC periyodu bu değer kadar arttırılır.

Analiz programı öncelikle CRC kullanmadan test yapmaktadır. Test sayısı verilen parametreler kullanılarak belirtilen ilk değerden başlayarak her adımda test sayısını belirtilen adım sayısı kadar arttırıp şifre çözme işlemi yapar. Maksimum test sayısına ulaşınca CRC kullanılmayan test bitmiş olur. Testin her adımında tek modül ile şifre çözme işlemi için maksimum süre olan 2 ms üst sınır olmak üzere rastgele bir hata enjekte etme anı ve süresi seçilerek hata hata enjekte etme işlemi yapılır. CRC'nin kullanılmadığı test bittikten sonra aynı testlerin tamamı kapalı anahtar üzerinde periyodik CRC kontrolü yapılarak tekrarlanmaktadır. CRC periyodu belirtilen ilk değerden başlatılıp her adımda bir önceki adıma göre belirtilen miktarda arttırılıp belirtilen maksimum değere ulaşana kadar test tekrarlanır.

Durum 2 : Şekil 6.12'de gösterilen her üç TMR modülünün kapalı anahtarın bir kopyasını tuttuğu fakat modüllerin çalışmaya başlama zamanları arasında zaman farkı olmadığı durum.



Şekil 6.12 : TMR ile şifre çözme – 3 kopya anahtar – gecikme yok.

Bu durum için analiz programı şu şekilde çalıştırılmalıdır:

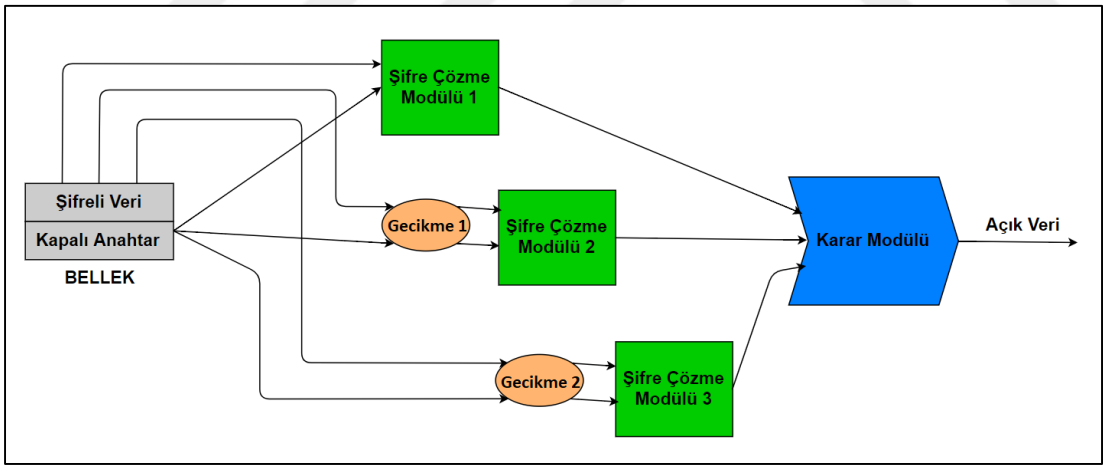
Programın-adı 2 minimum-test-sayısı maksimum-test-sayısı test-adım-sayısı minimum-CRC-periyodu maksimum-CRC-periyodu CRC-adım-sayısı

Programın adından sonraki "2" parametresi analizin TMR ile yapılacağını ve her modülün aynı kapalı anahtarın birer kopyasını tutacağını göstermektedir. Durum 2 için analiz durum 1 ile aynı şekilde yapılmaktadır. Maksimum şifre çözme süresi 3.5 ms'dir. Dolayısıyla hata enjekte etme işleminde hatanın anı ve süresi bu değer üst sınır olmak üzere seçilmektedir.

Durum 3 : Şekil 6.13'de gösterilen her üç TMR modülünün kapalı anahtarın tek bir kopyasını ortak kullandığı ve çalışmaya başlama zamanları arasında zaman farkı olduğu durum.

Bu durum için analiz programı şu şekilde çalıştırılmalıdır:

Programın-adı 3 test-sayısı minimum-CRC-periyodu maksimum-CRC-periyodu CRC-adım-sayısı iş-parçacıkları-arası-minimum-süre iş-parçacıkları-arası-maksimum-süre iş-parçacıkları-arası-süre-adım-sayısı



Şekil 6.13 : TMR ile şifre çözme – tek anahtar kopyası – gecikme var.

Programın adından sonraki "2" parametresi analizin TMR ile yapılacağını ve her modülün aynı kapalı anahtarı kullanacağını göstermektedir. CRC periyodu ile ilgili parametreler durum 1 ve 2'deki gibi kullanılmaktadır. İş parçacıkları arası süre ile ilgili parametreler aşağıdaki şekilde kullanılmaktadır:

İş parçacıkları arasındaki minimum süre : Testin her adımında iş parçacıkları arasında oluşturalacak en küçük gecikmenin milisaniye cinsinden değeridir.

İş parçacıkları arasındaki maksimum süre : Testin her adımında iş parçacıkları arasında oluşturalacak en büyük gecikmenin milisaniye cinsinden değeridir.

İş parçacıkları arasındaki süre adım sayısı : Testin her adımında iş parçacıkları arasında oluşturalacak gecikme bir önceki adıma göre bu değer kadar arttırılır.

Analiz programı verilen parametreleri kullanarak durum 1 ve 2'dekine benzer şekilde önce kapalı anahtar üzerinde CRC kullanmadan test yapar. Daha sonra kapalı anahtar üzerinde periyodik CRC kontrolü yaparak test yapar. Her iki durumda da iş parçacıkları arasındaki gecikme şu şekilde belirlenir:

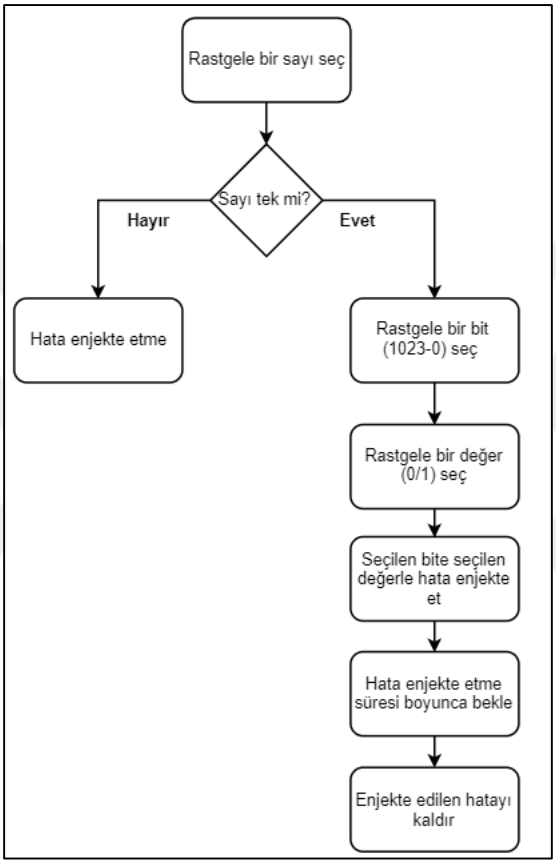
1. ve 2. iş parçacığı arasındaki süre minimum gecikme değerinden başlayarak her testte bir öncekine göre belirtilen adım sayısı kadar arttırılıp maksimum gecikme değerine ulaşana kadar test yapılır. Her test adımında 2. ve 3. iş parçacıkları arasındaki süre benzer şekilde minimum gecikme değerinden maksimum gecikme değerine kadar arttırılıp her seferinde belirtilen sayıda şifre çözme işlemi yapılır. Hata enjekte etme anı ve süresi durum 1 ve 2 ile aynı şekilde rastgele belirlenir. Hata enjekte etme anı ve süresi için üst sınır değeri iş parçacıkları arasında gecikmenin olmadığı durum yani durum 2'nin süresi ile iş parçacıkları arasındaki sürelerin toplamı olacak şekilde belirlenmektedir.

Her üç durumda da CRC kontrolü yapılması hata enjekte etme süresine bir sınır getirmektir. CRC kontrolü yapılmadığı takdirde kapalı anahtarda kalıcı hatalar veya TMR yöntemini işlevsiz kılacak kadar uzun süren geçici hatalar ile atak yapılabilecektir. Sadece CRC yönteminin kullanılması ile hata enjekte etme işlemi tespit edilebilir fakat hata düzeltme için ek bir yöntem kullanılması gerekmektedir. Bu çalışmada hem hata tespiti hem de hata düzeltme amacıyla TMR kullanılmaktadır.

Bu aşamada kullanılan hata enjekte etme modelinden bahsetmemiz gerekmektedir. Yapılan her şifre çözme işlemiyle beraber hata enjekte etme işlevini gerçekleştiren bir iş parçacığı çalışır. Bu iş parçacığına 1. ve 3. durum için bir anahtar, 2. durumda ise 3 adet kapalı anahtar verilir. Bu iş parçacığı kendisine giriş olarak verilen kapalı anahtarlara sırasıyla Şekil 6.14'de gösterilen akışa göre hata enjekte eder. Hata enjekte etme anı ve süresi için verilmiş olan bir üst sınır değerini aşmamak üzere

rastgele değer seçilir. Hata enjekte etme süresinin sonunda enjekte edilen hata düzeltilir.

Hata enjekte etme akışına bakıldığında rastgele sayı seçme işleminin tam olarak rastgele olması varsayımı altında hata enjekte etmeye karar verme olasılığı %50 ve seçilen bitin değerinin değişme olasılığı %50 olacağından %25 ihtimalle seçilen bite hata enjekte edilmiş olacaktır.



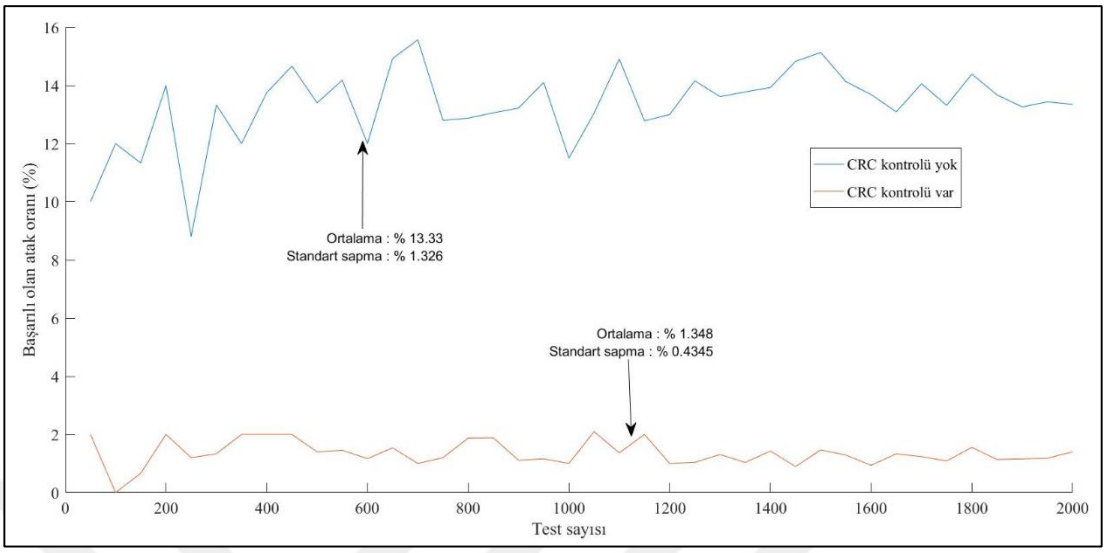
Şekil 6.14 : Hata enjekte etme işlemi.

Birinci durumun yani tek bir modülün (iş parçacığı) şifre çözme işlemi yaptığı durumun analizi için analiz programı şu şekilde çalıştırılmıştır :

Programın-adı 1 50 2000 50 1 1 1

Bu parametreler ile analiz programı 50 ile 2000 arasındaki 50'nin katı tüm değerler kadar şifre çözme işlemi yapacaktır. Daha sonra aynı işlemleri kapalı anahtar üzerinde 1 ms periyotlu CRC kontrolü yaparak tekrarlayacaktır. Analiz sonuçları Şekil 6.15'de gösterilmiştir. CRC'nin kullanıldığı durumda kullanılmadığı duruma

göre hata enjekte etme işleminin başarılı olma oranının %89.89 azaldığı hesaplanabilir.



Şekil 6.15 : Tek modül ile şifre çözme analiz sonuçları.

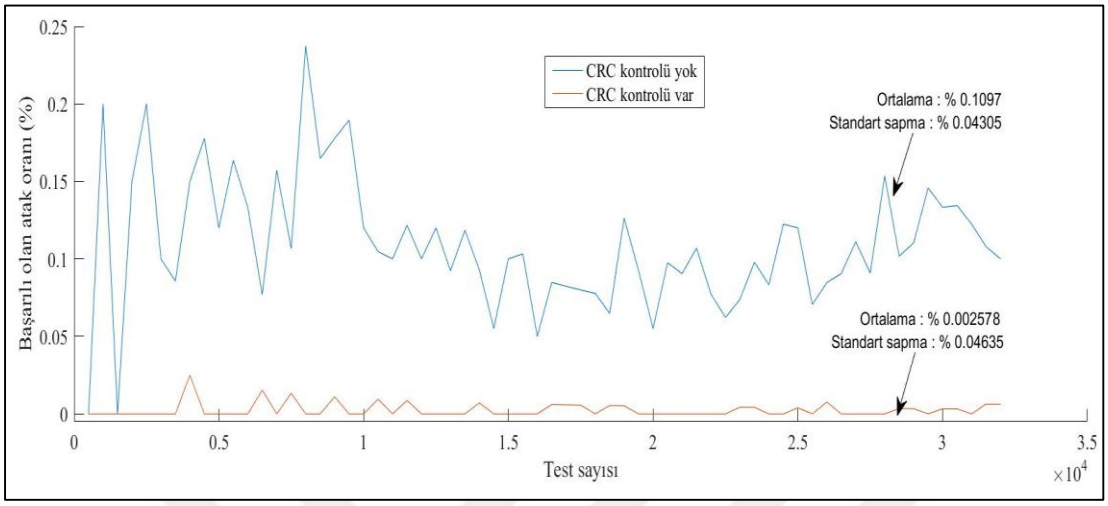
2 ve 3. durumlar yani TMR yönteminin kullanıldığı durumlar için TMR yönteminin başarılı olup sonucun yanlış olduğu durum üzerinde ve TMR yönteminin başarısız olduğu yani her üç iş parçacığının farklı sonuç ürettiği durum üzerinde durulmuştur. Çünkü ilk durumda hata enjekte etme işlemi başarılı olmuş, ikinci durumda ise hata enjekte etme işlemi başarılı olmamasına rağmen doğru sonuçta üretilememiştir.

2. durum yani TMR yönteminin üç iş parçacığının şifre çözmeye başlama zamanları arasında süre farkı olmadığı ve her bir iş parçacığının aynı kapalı anahtarın bir kopyasını tuttuğu durum için analiz programı şu şekilde çalıştırılmıştır:

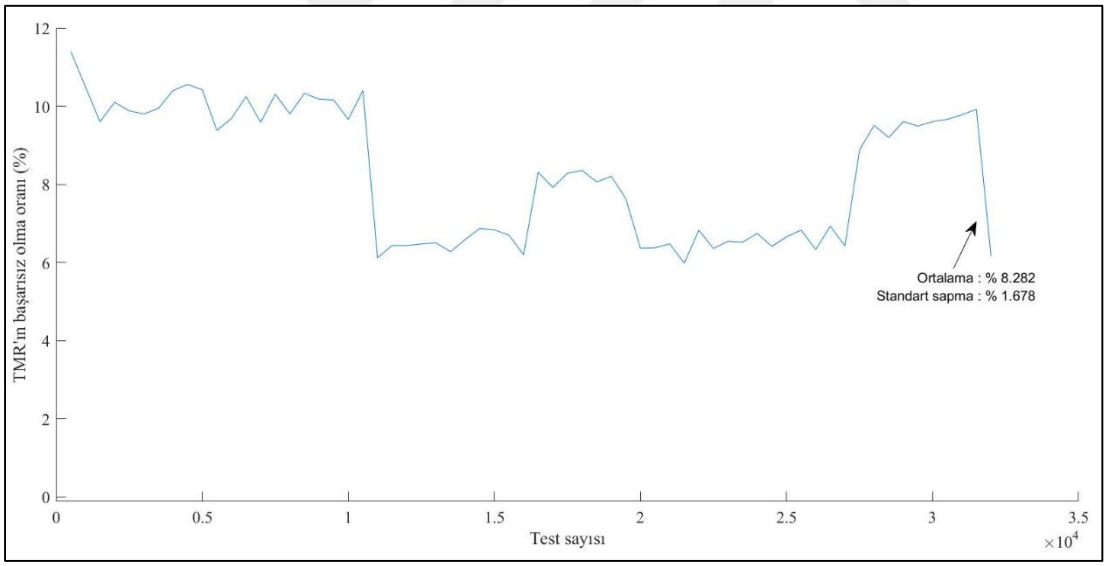
```
Programın-adı 2 1000 32000 500 1 1 1
```

Bu durumda 500'den başlayıp 32000'e kadar 500'ün katı olan tüm sayılar test sayısı olarak kullanılarak şifre çözme işlemi yapılacaktır. Daha sonra aynı işlemleri her üç kapalı anahtar üzerinde 1 ms periyotlu CRC kontrolü yaparak tekrarlayacaktır. Hata enjekte etme işleminin başarılı olduğu durum için sonuçlar Şekil 6.16'da, TMR yönteminin başarısız olduğu durum için sonuçlar Şekil 6.17'de verilmiştir. CRC kullanılmadığı durumda hata enjekte etme işleminin ortalama başarılı olma oranı %0.1097 iken CRC kullanıldığı durumda bu oran %0.002578'e düşmüştür. Sonuçta kapalı anahtar CRC kullanılmasının hata enjekte etme işleminin başarısını %97.65 oranında düşürdüğü hesaplanabilir. Şekil 6.17'ye bakıldığında ise TMR yönteminin

%8.282 oranında başarısız olduğu görülmektedir. Bu durumda hata enjekte etme işlemi başarısız olmasına rağmen şifre çözme sonucunda açık veri elde edilememiştir.



Şekil 6.16 : TMR ile şifre çözme (iş parçacıkları arasında gecikme yok) - hata enjekte etme atağı başarı oranları.



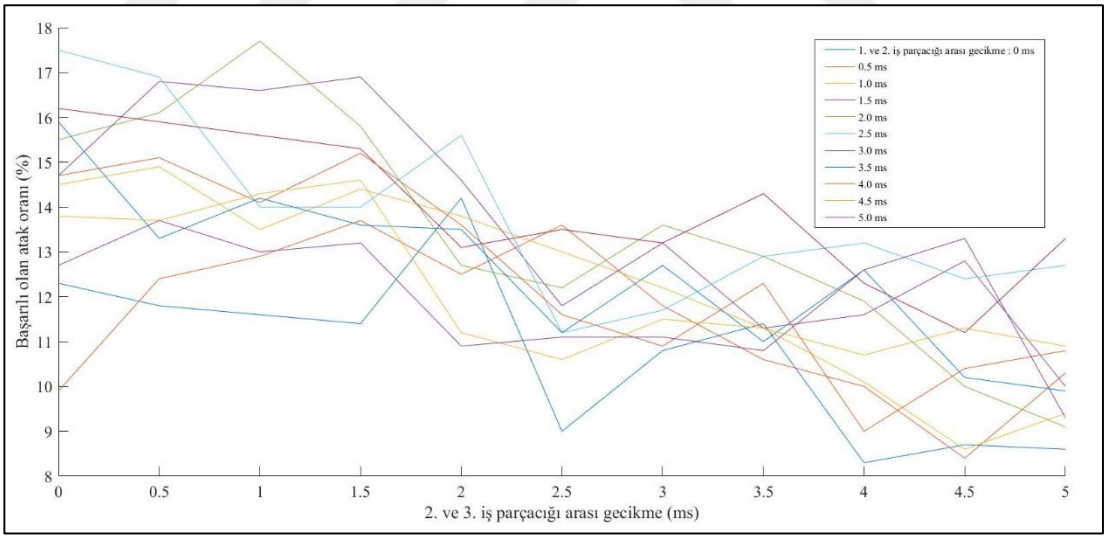
Şekil 6.17 : TMR ile şifre çözme (iş parçacıkları arasında gecikme yok) – TMR'ın başarısız olma oranları.

Önerilen yöntem olan 3. durum için analiz programı şu şekilde çalıştırılmıştır:

Programın-adı 3 1000 1 5 1 0 5 0.5

Bu durumda öncelikle 0 ile 5 ms arasındaki 0.5 ms'nin tam katı olan tüm gecikme değerleri 1. ile 2. iş parçacığı ve 2. ile 3. iş parçacığı arasındaki gecikme olarak kullanılarak her gecikme çifti için 1000 şifre çözme işlemi yapılır. Daha sonra aynı

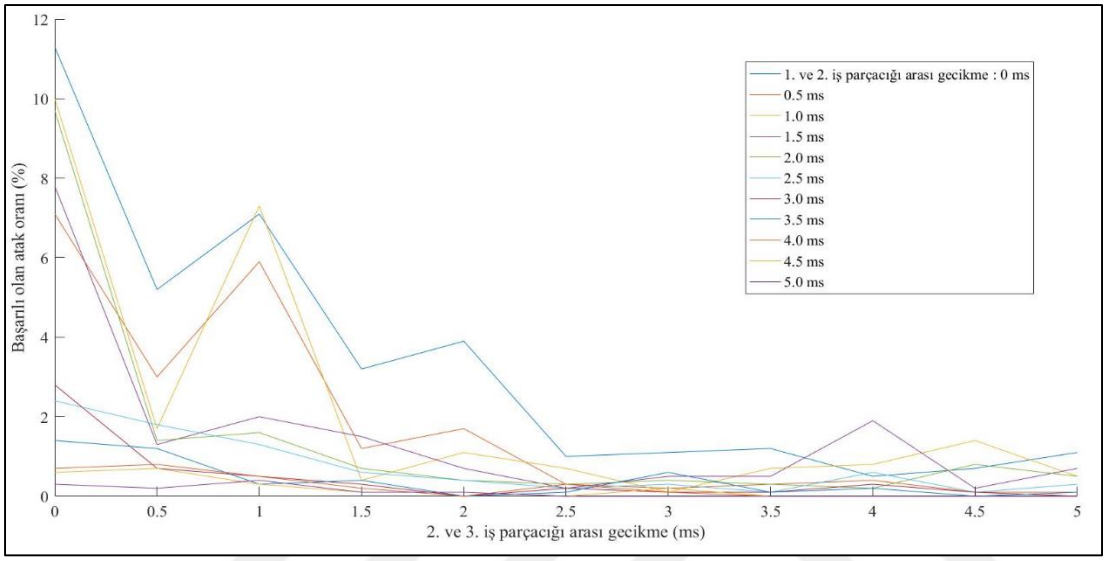
testler kapalı anahtar üzerinde 1, 2, 3, 4, ve 5 ms periyotlu CRC kontrolü yapılarak tekrarlanır. Şekil 6.18'de kapalı anahtar üzerinde CRC kontrolü yapılmadığı durumda hata enjekte etme işleminin başarılı olma oranları verilmiştir. Şeki 6.18'den görüldüğü üzere hata enjekte etme atağının başarı oranı iş parçacıkları arası gecikme arttıkça düşmüştür. Ayrıca yapılan analizde TMR'ın başarısız olduğu bir durum gözlenmemiştir. Ancak 1. ile 2. iş parçacığı arasındaki gecikme ve 2. ile 3. iş parçacığı arasındaki gecikme maksimum değer olan 5 ms olduğu durumda bile hata enjekte etme işleminin başarılı olma oranı ancak %9.3 olmuştur. İş parçacıkları arası gecikmenin artmasının hata enjekte etme işleminin başarılı olma oranını TMR yönteminin kullanıldığı fakat iş parçacıkları arası gecikmenin olmadığı durumdaki seviyelere indiremediği görülmektedir. Bunun nedenleri arasında hata enjekte etme anı ve süresinin alabileceği rastgele değerin iş parçacıkları arasındaki gecikmeler arttıkça artması gösterilebilir. Ayrıca durum 2'de anahtarın 3 kopya halinde tutulması 3 anahtarda birden aynı bit üzerinde hata enjekte edilmesini zorlaştırdığından hata enjekte etme işleminin başarılı olma oranı önerilen yönteme göre daha düşük çıkmaktadır.



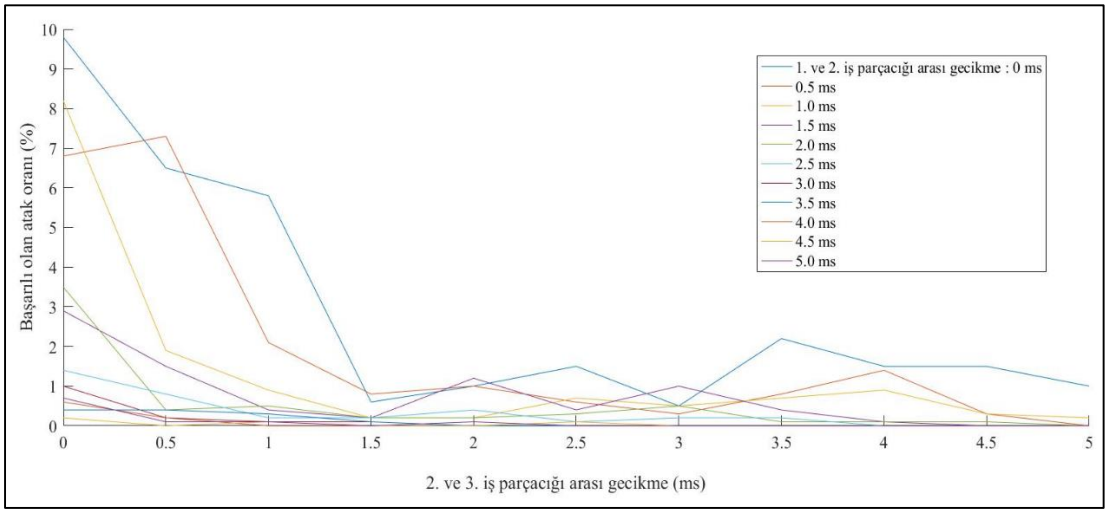
Şekil 6.18 : Önerilen yöntem – hata enjekte etme atağı başarı oranları.

Önerilen yöntemin ayırt edici özelliği olan iş parçacıkları arasında gecikme olması durumunun tek başına hata enjekte etme atağına karşı başarılı sonuçlar vermediği görülmektedir. Bu durumda önerilen yönteme kapalı anahtar üzerinde periyodik CRC kontrolünün eklenerek kullanılması gerekliliği ortaya çıkmaktadır. Kapalı anahtar üzerinde periyodik CRC kontrolü ile hata enjekte etme süresinin iş

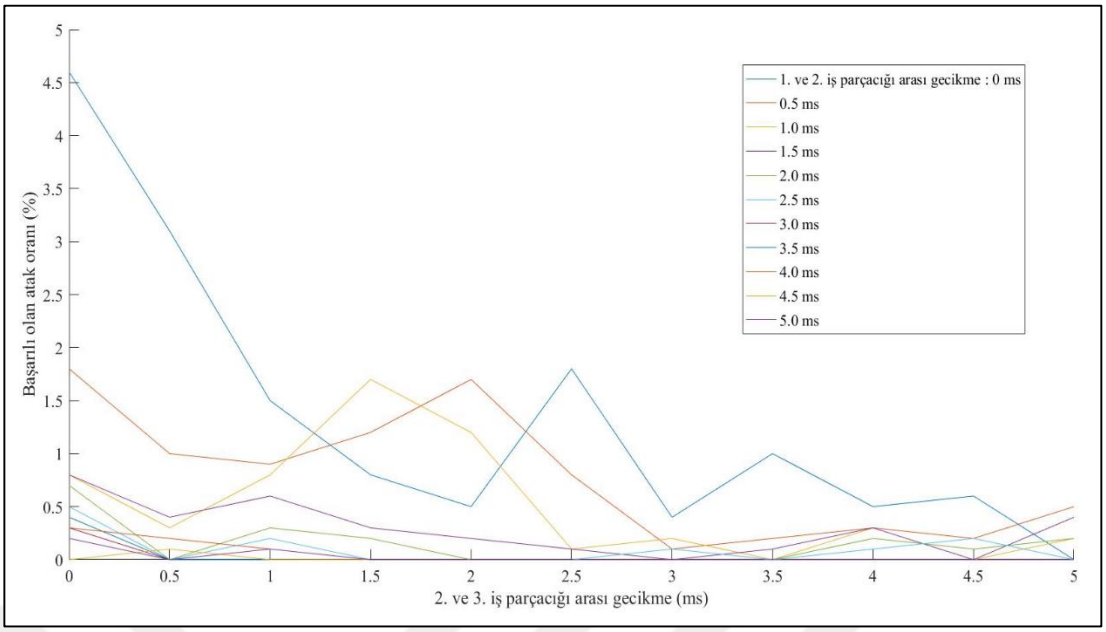
parçacıkları arası gecikmeden fazla olup atağın başarılı olacağı durumlarda atağın tespit edilmesi amaçlanmaktadır. Önerilen yönteme CRC kontrolü eklendiği durumda analiz sonuçları Şekil 6.19, 6.20, 6.21, 6.22 ve 6.23'de gösterilmiştir. CRC periyodunun 1 ms olduğu durumdaki Şekil 6.23'de görülen grafik diğer CRC periyodu grafiklerinden farklı olarak 3 boyutlu olarak gösterilmiştir. Bunun nedeni 2 boyutlu grafik oluşturulduğunda anlaşılır olmamasıdır.



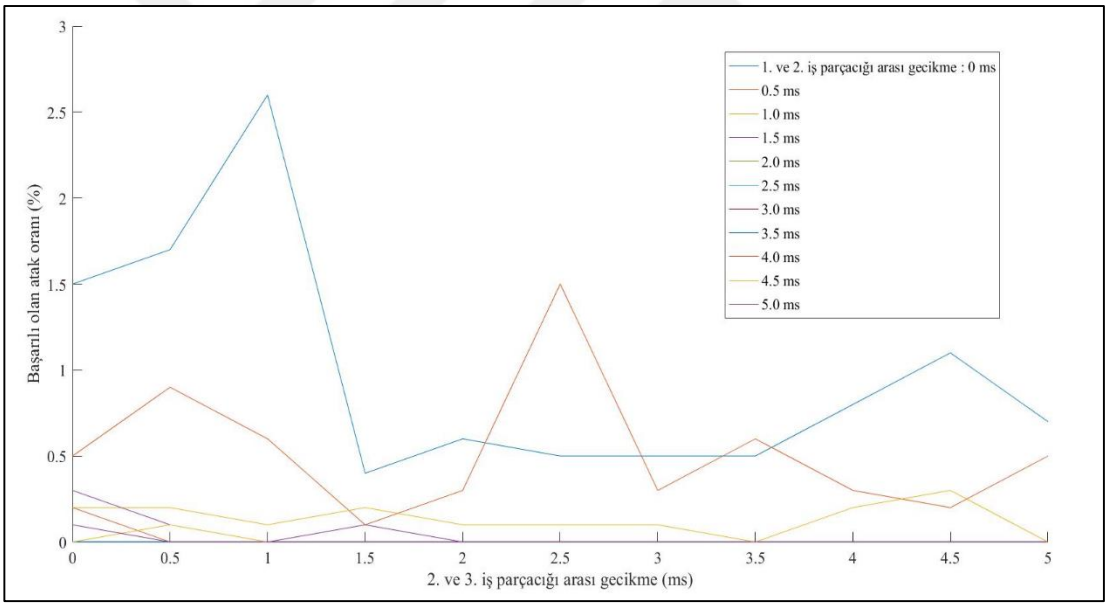
Şekil 6.19 : Önerilen yöntem, CRC periyodu 5 ms – hata enjekte etme atağı başarı oranları.



Şekil 6.20 : Önerilen yöntem, CRC periyodu 4 ms – hata enjekte etme atağı başarı oranları.



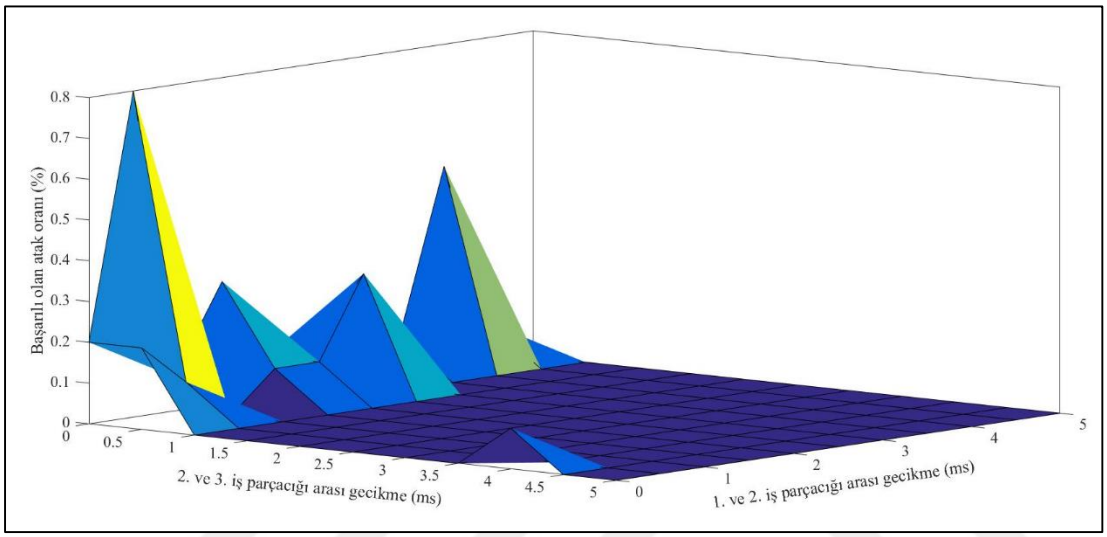
Şekil 6.21 : Önerilen yöntem, CRC periyodu 3 ms – hata enjekte etme atağı başarı oranları.



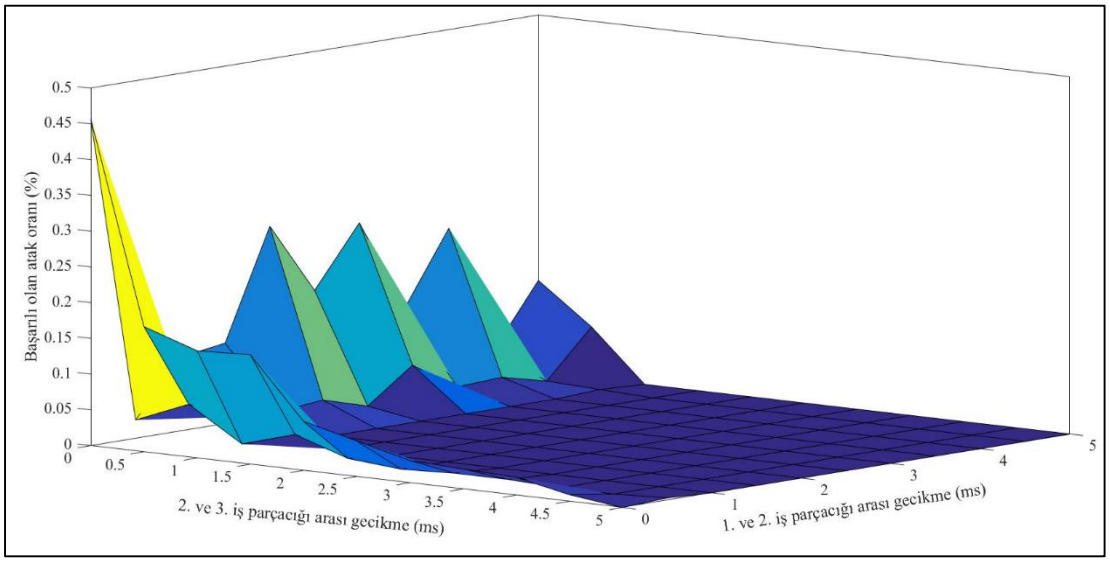
Şekil 6.22 : Önerilen yöntem, CRC periyodu 2 ms – hata enjekte etme atağı başarı oranları.

Önerilen yönteme kapalı anahtar üzerinde periyodik CRC kontrolü eklendiği durumda iş parçacıkları arasındaki gecikmeler arttıkça hata enjekte atağının başarı oranının düştüğü görülmektedir. Ayrıca CRC periyodunun küçültülmesinin de hata enjekte etme atağının başarı oranını düşürdüğü görülmektedir. Durum 2'nin CRC kullanılmadığı durumda önerilen yönteme göre hata enjekte etme atağına karşı daha başarılı olduğundan daha önce bahsedilmişti. Son olarak önerilen yönteme ek olarak

kapalı anahtar üzerinde 1 ms periyotlu CRC kontrolü uygulandığı durum için test sayısı 1000 yerine durum 2 için kullanılan maksimum test sayısı olan 32000 olarak analiz yapılmış ve durum 2 ile karşılaştırılmıştır. Test sayısının arttırılması daha doğru sonuçlar alınmasını sağlamak içindir. Bu durumda elde edilen sonuçlar Şekil 6.24'de görülmektedir. Elde edilen sonuçlara bakıldığında 1. ile 2. iş parçacığı ve 2. ile 3. iş parçacığı arasında gecikmenin 1.5 ms ve daha büyük olduğu durumlarda hata enjekte etme atağının başarı oranının %0 olduğu görülmektedir.



Şekil 6.23 : Önerilen yöntem, CRC periyodu 1 ms, test sayısı 1000 – hata enjekte etme atağı başarı oranları.



Şekil 6.24 : Önerilen yöntem, CRC periyodu 1 ms, test sayısı 32000 – hata enjekte etme atağı başarı oranları.

Önerilen yöntem ile 2. durumu karşılaştırdığımızda kapalı anahtarın kopyasının olmaması nedeniyle hem bellek hem de anahtarın güvenliği açısından daha avantajlı olduğu söylenebilir. Ayrıca TMR'ın başarısız olduğu durum açısından da önerilen yöntemin daha verimli olduğu görülmektedir. Öte yandan önerilen yöntemde iş parçacıklarının çalışmaya başlama süreleri arasında süre farkı olması nedeniyle şifre çözme işleminin daha uzun sürmesi bir dezavantaj olarak görülmektedir. Kapalı anahtar üzerinde periyodik CRC kontrolü ile birlikte önerilen yöntemin TMR kullanılmadan şifre çözme yapılan durum 1 ve TMR yönteminin kullanılıp iş parçacıkları arasında gecikme olmayan durum 2'ye göre hata enjekte etme atağına karşı daha başarılı olduğu görülmektedir.

7. SONUÇLAR

Bu çalışmada 3 çekirdekli LEON3 işlemcisi üzerinde RSA algoritmasına yapılan hata enjekte atağına karşı dayanıklı bir RSA algoritması gerçeklemesi yapılmıştır. Bu amaçla bir kere şifre çözme işlemi yerine 3 ayrı işlemci çekirdeği üzerinde koşan birer iş parçacığının şifre çözme yaptığı bir yöntem kullanılmıştır. Her iş parçacığı aynı şifre çözme işlemini birbirlerine göre zamanda ötelenmiş olarak yapmaktadır. Böylece eğer kapalı anahtarın herhangi bir bitinde oluşan veya hata enjekte etme atağı ile herhangi bir bitinde oluşturulan geçici hatalardan şifre çözme işlemi sonucunun etkilenmemesi sağlanmaya çalışılmıştır. Önerilen yönteme ek olarak kapalı anahtar üzerinde periyodik CRC kontrolü eklenerek yöntemin başarı oranı arttırılmıştır.

Önerilen yöntem tek bir iş parçacığının şifre çözme işlemi yaptığı durum ve yine TMR tabanlı ancak iş parçacıklarının şifre çözmeye başlama zamanları arasında süre farkı olmadığı durumla karşılaştırılmış, avantaj ve dezavantajları ortaya konmuştur. Önerilen yöntemin enjekte edilen hatanın süresine ve kullanılan yöntemdeki iş parçacıklarının şifre çözmeye başlama zamanları arasındaki süreye bağlı olarak kapalı anahtarın elde edilmesini engelleyebileceği gösterilmiştir. Bu çalışma LEON3 işlemcisini ülkemizde ilk defa çok çekirdekli olarak gerçekleyen çalışma olması nedeniyle önem taşımaktadır. Aynı zamanda literatürde TMR yönteminden yararlanılarak RSA algoritmasının hata enjekte etme atağına dayanıklı gerçeklenmesi açısından da bir ilk olma niteliği taşımaktadır.

KAYNAKLAR

[1] **Lone, A., & Moin Uddin, P.** (2016). Common Attacks on RSA and its Variants with Possible Countermeasures, 2278–9359.

[2] **FIPS 46-3** (2005). Data Encryption Standard (DES). National Institute of Standarts and Technology (NIST).

[3] **FIPS 197** (2001). Advanced Encryption Standard (AES). National Institute of Standarts and Technology (NIST).

[4] **Diffie, W., & Hellman, M.** (1976). New Directions in Cryptography. IEEE Trans. Inf. Theor., 22(6), 644–654. https://doi.org/10.1109/TIT.1976.1055638

[5] **Rivest, R. L., Shamir, A., & Adleman, L.** (1978). A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Commun. ACM, 21(2), 120–126. https://doi.org/10.1145/359340.359342

[6] **Koblitz, N.** (1987). Elliptic curve cryptosystems. Mathematics of Computation, 48(177), 203–209.

[7] **Barker, E. B.** (2013). Digital Signature Standard (DSS). Federal Inf. Process. Stds. (NIST FIPS) - 186-4. https://doi.org/914162

[8] **Cid, C. F.** (2003). Cryptanalysis of RSA: A survey. SANS Institute.

[9].**Url-1** <*http://www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino_sys_arch%2Fsmp.html*>, erişim tarihi 12.11.2017.

[10] **Wilson, C., Sabogal, S., George, A., & Gordon-Ross, A.** (2017). Hybrid, adaptive, and reconfigurable fault tolerance. In 2017 IEEE Aerospace Conference (pp. 1–11). https://doi.org/10.1109/AERO.2017.7943867

[11] **Garrels, M.** (2010). Introduction to Linux. Fultus Corporation.

[12] **Tian, X., & De Supins, B. R.** (2014). Explicit Vector Programming with OpenMP 4.0 SIMD Extension. Primeur Magazine 2014.

[13] **Url-2** <*http://man7.org/linux/man-pages/man7/pthreads.7.html*>, erişim tarihi 12.11.2017.

[14] **Latif-Shabgahi, G., Bass, J. M., & Bennett, S.** (2004). A taxonomy for software voting algorithms used in safety-critical systems. IEEE Transactions on Reliability, 53(3), 319–328. https://doi.org/10.1109/TR.2004.832819

[15] **Paharsingh, R., & Das, O**. (2011). An Availability Model of a Virtual TMR System with Applications in Cloud/Cluster Computing. In 2011 IEEE 13th International Symposium on High-Assurance Systems Engineering (pp. 261–268). https://doi.org/10.1109/HASE.2011.11

[16] **She, X., & McElvain, K. S.** (2009). Time Multiplexed Triple Modular Redundancy for Single Event Upset Mitigation. IEEE Transactions on Nuclear Science, 56(4), 2443–2448. https://doi.org/10.1109/TNS.2009.2021656

[17] **Gaisler,** http://gaisler.com/doc/leon3_product_sheet.pdf, alındığı tarih: 12.11.2017.

[18] **VxWorks,** https://www.windriver.com/products/product-overviews/2691-VxWorks-Product-Overview.pdf, alındığı tarih: 12.11.2017.

[19] **Massa, A. J.** (2003). Embedded software development with e-Cos. Upper Saddle River, NJ: Prentice Hall.

[20] **RTEMS,** https://docs.rtems.org/releases/rtems-docs-4.11.2/user.pdf, alındığı tarih: 12.11.2017.

[21] **Gaisler,** http://www.gaisler.com/products/grlib/grlib.pdf, alındığı tarih: 12.11.2017.

[22] **Gaisler,** http://www.gaisler.com/products/grlib/grip.pdf, alındığı tarih: 12.11.2017.

[23] **Gaisler,** http://www.gaisler.com/products/grlib/guide.pdf, alındığı tarih: 12.11.2017.

[24] **ARM,** https://developer.arm.com/products/architecture/amba-protocol/amba-2, alındığı tarih: 12.11.2017.

[25] **Digilent,** https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-4-ddr/nexys4ddr_rm.pdf, alındığı tarih: 12.11.2017.

[26] **Cygwin,** Retrieved November 12, 2017, from https://cygwin.com/cygwin-ug-net/cygwin-ug-net.pdf

[27] **Gaisler,** http://www.gaisler.com/doc/grmon.pdf, alındığı tarih: 12.11.2017.

[28] **Solovay, R., & Strassen, V.** (1977). A Fast Monte-Carlo Test for Primality. SIAM Journal on Computing, 6(1), 84–85. https://doi.org/10.1137/0206006

[29] **Miller, G. L.** (1976). Riemann's hypothesis and tests for primality. Journal of Computer and System Sciences, 13(3), 300–317. https://doi.org/10.1016/S0022-0000(76)80043-8

[30] **Pollard, J. M.** (1974). Theorems on factorization and primality testing. Mathematical Proceedings of the Cambridge Philosophical Society, 76(3), 521–528. https://doi.org/10.1017/S0305004100049252

[31] **Rabin, M. O.** (1980). Probabilistic algorithm for testing primality. Journal of Number Theory, 12(1), 128–138. https://doi.org/10.1016/0022-314X(80)90084-0

[32] **RSA And Primality Test**. Retrieved November 12, 2017, from http://www.imada.sdu.dk/~joan/projects/RSA.pdf

[33] **Jahani, S., Samsudin, A., & Subramanian, K. G.** (2014). Efficient Big Integer Multiplication and Squaring Algorithms for Cryptographic Applications [Research article]. https://doi.org/10.1155/2014/107109

[35] **Montgomery, P. L.** (1985). Modular multiplication without trial division. Mathematics of Computation, 44(170), 519–521. https://doi.org/10.1090/S0025-5718-1985-0777282-X

[35] **Menezes, A. J., Vanstone, S. A., & Oorschot, P. C. V.** (1996). Handbook of Applied Cryptography (1st ed.). Boca Raton, FL, USA: CRC Press, Inc.

[36] **Shand, M., & Vuillemin, J.** (1993). Fast implementations of RSA cryptography. In Proceedings of IEEE 11th Symposium on Computer Arithmetic (pp. 252–259). https://doi.org/10.1109/ARITH.1993.378085

[37] **Wiener, M. J.** (1990). Cryptanalysis of short RSA secret exponents. IEEE Transactions on Information Theory, 36(3), 553–558. https://doi.org/10.1109/18.54902

[38] **Boneh, D., Durfee, G., & Frankel, Y.** (1998). An Attack on RSA Given a Small Fraction of the Private Key Bits. In Advances in Cryptology — ASIACRYPT'98 (pp. 25–34). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-49649-1_3

[39] **Kocher, P. C.** (1996). Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Advances in Cryptology — CRYPTO '96 (pp. 104–113). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-68697-5_9

[40] **Kocher, P. C., Jaffe, J., & Jun, B.** (1999). Differential Power Analysis. In Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (pp. 388–397). London, UK, UK: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=646764.703989

[41] **Bao, F., Deng, R. H., Han, Y., Jeng, A., Narasimhalu, A. D., & Ngair, T.** (1998). Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. In B. Christianson, B. Crispo, M. Lomas, & M. Roe (Eds.), Security Protocols: 5th International Workshop Paris, France, April 7–9, 1997 Proceedings (pp. 115–124). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/BFb0028164

[42] **Boneh, D., DeMillo, R., & Lipton, R.** (1997). On the importance of checking cryptographic protocols for faults. In Advances in Cryptology—EUROCRYPT'97 (pp. 37–51). Springer.

[43] **Popp, T., Mangard, S., & Oswald, E.** (2007). Power Analysis Attacks and Countermeasures. IEEE Design Test of Computers, 24(6), 535–543. https://doi.org/10.1109/MDT.2007.200

[44] **Xilinx,** Retrieved November 13, 2017, from https://www.xilinx.com/support /documentation/ip_documentation/ug086.pdf

[45] **Url-3** <*https://www.gnu.org/software/binutils/*>, erişim tarihi 12.11.2017.

[46] **Url-4** <*https://www.kernel.org/*>, erişim tarihi 12.11.2017.

[47] **Url-5** <*https://gcc.gnu.org/*>, erişim tarihi 12.11.2017.

[48] **Url-6** <*https://uclibc.org/*>, erişim tarihi 12.11.2017.

[49] **Url-7** <*http://jorisvr.nl/article/leon3-linux*>, erişim tarihi 12.11.2017.

[50] **Url-8** <*https://gcc.gnu.org/viewcvs/gcc/branches/gcc-4_7-branch/gcc/ira-int.h?view=markup&pathrev=191605*>, erişim tarihi 12.11.2017.

[51] **Url-9** <*https://busybox.net/*>, erişim tarihi 12.11.2017.

[52] **Url-10** <*https://patchwork.kernel.org/patch/3647171*>,erişim tarihi 12.11.2017.

[53] **Gaisler,** Retrieved November 13, 2017, from http://www.gaisler.com/anonftp /linux/linux-2.6/doc/mklinuximg-2.0.7.pdf

[54] **Tera Term,** Retrieved November 13, 2017, from https://ttssh2.osdn.jp/

[55] **GNU MP,** Retrieved from https://gmplib.org/gmp-man-6.1.2.pdf

[56] **Url-11** <*http://jorisvr.nl/files/leon3/leon3-linux-stuff-20130228.tar.gz*>, erişim tarihi 12.11.2017

[57] **Url-12** <*http://www.gaisler.com/index.php/downloads/linux*>, erişim tarihi 12.11.2017

[58] **Cederman, D., Hellstrom, D., Sherrill, J., Bloom, G., Patte, M., & Zulianello, M**. (2014). RTEMS SMP for LEON3/LEON4 Multi-Processor Devicies. In DASIA 2014-Data Systems In Aerospace (Vol. 725).

[59] **Müller, M., Braun, A., Gerlach, J., Rosenstiel, W., Nienhüser, D., Zöllner, J. M., & Bringmann, O.** (2010). Design of an automotive traffic sign recognition system targeting a multi-core SoC implementation. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010 (pp. 532–537). IEEE.

[60] **Serres, O., Narayana, V. K., & El-Ghazawi, T.** (2011). An Architecture for Reconfigurable Multi-core Explorations (pp. 105–110). IEEE. https://doi.org/10.1109/ReConFig.2011.10

[61] **Hernandez, C., Abella, J., Cazorla, F. J., Andersson, J., & Gianarro, A.** (2015). Towards making a LEON3 multicore compatible with probabilistic timing analysis. Data Systems In Aerospace (DASIA).

[62] **Martins, P., & Sousa, L.** (2014). On the Evaluation of Multi-core Systems with SIMD Engines for Public-Key Cryptography (pp. 48–53). IEEE. https://doi.org/10.1109/SBAC-PADW.2014.10

[63] **Tan, X., & Li, Y.** (2012). Parallel Analysis of an Improved RSA Algorithm (pp. 318–320). IEEE. https://doi.org/10.1109/ICCSEE.2012.286

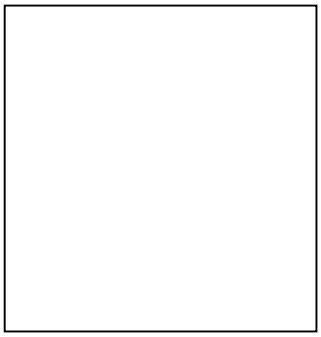
[64] **Asaduzzaman, A., Gummadi, D., & Waichal, P.** (2015). A promising parallel algorithm to manage the RSA decryption complexity. In SoutheastCon 2015 (pp. 1–5). IEEE.

[65**] Rivain, M.** (2009). Securing RSA against fault analysis by double addition chain exponentiation. In Cryptographers' Track at the RSA Conference (pp. 459–480). Springer.

[66] **Giraud, C.** (2006). An RSA implementation resistant to fault attacks and to simple power analysis. IEEE Transactions on Computers, 55(9), 1116–1120.

[67] **Nicolaidis, M., & Velazco, R.** (n.d.). Team 3/Robust Integrated Systems (RIS).

ÖZGEÇMİŞ



Ad-Soyad **: İsmail Demir**

Doğum Tarihi ve Yeri **: 1989 Malatya**

E-posta **: demirismail89@gmail.com**

ÖĞRENİM DURUMU:

- **Lisans :** 2012, İstanbul Teknik Üniversitesi, Elektrik Elektronik Fakültesi, Elektronik Mühendisliği