TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BLOOM FİLTRE TABANLI BELLEK UYGULAMALARI İLE GÖMÜLÜ SİSTEMLERDE VE İŞLEMCİLERDE ETKİNLEŞTİRİLMİŞ TRUVA ATININ BELİRLENMESİ

YÜKSEK LİSANS TEZİ

Alperen BOLAT

Bilgisayar Mühendisliği Anabilim Dalı

Prof. Dr. Oğuz ERGİN

TEMMUZ 2022

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, alıntı yapılan kaynaklara eksiksiz atıf yapıldığını, referansların tam olarak belirtildiğini ve ayrıca bu tezin TOBB ETÜ Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırlandığını bildiririm.

Alperen BOLAT

ÖZET

Yüksek Lisans Tezi

BLOOM FİLTRE TABANLI BELLEK UYGULAMALARI İLE GÖMÜLÜ SİSTEMLERDE VE İŞLEMCİLERDE ETKİNLEŞTİRİLMİŞ TRUVA ATININ BELİRLENMESİ

Alperen BOLAT

TOBB Ekonomi ve Teknoloji Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tarih: TEMMUZ 2022

Donanım Truva Atları (DTAlar), modern işlemci tabanlı sistemler için birtakım güvenlik risklerini meydana getirir. İşlemcide yürütülen programın değiştirilmesi, programa ait olmayan buyrukların işlemciye gönderilmesi, program üzerinde analizler ile programcının veya donanımın kritik bilgilerinin ele geçirilmesi ve kaynaklar arasında doğrulamanın engellenmesi gibi birçok güvenlik zafiyetiyle birlikte getirilir. Bloom filtre tabanlı bellek uygulamaları ile işlemci içerisindeki yürütmenin güvenilir olmasını sağlamak mümkündür. Bloom filtrenin sağladığı sorgulama ve eğitim yeteneği sayesinde, buyruklar ve bellek adresleri veri seti ile eğitim yapılır ve yürütme sırasında bu eğitilen veriler sorgulanır. Bu sayede işlemcideki yürütmenin güvenilir ve doğrulanmış olması sağlanır. Diğer taraftan, işlemci ile derleme kaynağı arasında belirlenecek bir takım şifreleme ve güvenli ağ protokolleri ile kaynakların birbirini doğrulaması ve sisteme yapılan dış müdahalelerin oluşturacağı sorunlar engellenebilir. Bu tez çalışmasında belirtilen güvenlik zafiyetlerini engelleyecek mimari tasarımlar ile ilgili çalışmalar sunulmaktadır.

Tezde tartışılan ilk çalışma DTA aktivasyonunu tespit etmeyi öneren bir işlemci mimarisidir. Önerdiğimiz mimari, geçici olarak tasarlanmış denetleyicilere dahil edilen ve koruma mimarisine entegre edilen Bloom Filtrelerinin kullanımına dayanmaktadır. Bloom filtreler sıfır yanlış alarmı ve küçük (ve yapılandırılabilir) algılanmayan alarm oranını garanti eder. Koruma mimarisini, bir FPGA üzerinde uygulanan ve bir dizi yazılım kıyaslaması çalıştıran bir RISC-V mikroişlemcisine dayalı bir vaka inceleme

sistemine uyguladık. Sonuçlar önerdiğimiz mimarinin olası DTA aktivasyonlarının %99'undan fazlasını sıfır yanlış alarmla tespit edebildiğini gösterildi. %0,68 ile %10,52 arasında değişen bir arama tablosu ek yükü ve %0,68 ile %0,99 arasında bir flip-flop ek yükü ölçtük ve sistemin çalışma frekansında herhangi bir azalmaya neden olmadık.

Tezde tartışılan diğer bir çalışma ise işlemci içerisindeki yürütmenin güvenliğini sağlamak ve başta DTA aktivasyonu olmak üzere saldırganlardan programın kendisini ve işlemci ile derleyici arasındaki ağı korumayı önerir. Modern bulut bilişim sistemleri, tipik olarak güvenlik açısından kritik bir kümede derlenen yazılım kaynaklarını gizli tutmak için yürütülebilir yazılımları bir ağ üzerinden dağıtır. Tez ile beraber yeni, verimli ve genel bir yazılım gizleme mimarisi çerçevesi sunan ERIC'i öneriyoruz. ERIC, yazılımı (i) yazılımın nasıl dağıtıldığına bakılmaksızın, yürütülebilir yazılımların yalnızca şifrelenmiş bir sürümünü insan gözünün kullanımına sunarak statik analize ve (ii) şifreli bir yürütülebilir dosyanın yalnızca doğru şekilde olabileceğini garanti ederek dinamik analize karşı korur. Tek bir kimliği doğrulanmış cihaz tarafından şifresi çözülür ve yürütülür. ERIC, verimli yazılım gizleme desteği sağlamak için temel donanım ve yazılım bileşenlerini içerir: (i) bir donanım şifre çözme motoru (HDE), hedef cihazdaki şifrelenmiş donanımın verimli bir şekilde şifresinin çözülmesini sağlar, (ii) derleyici, yalnızca benzersiz bir cihaz verilen yazılım yürütülebilir dosyalarını sorunsuz bir şekilde şifreleyebilir tanımlayıcı. Hem donanım hem de yazılım bileşenleri buyruk mimarisinden bağımsızdır ve ERIC'i genel kılar. ERIC'in ana fikri, fiziksel klonlanamayan işlevleri (PUF'ler), benzersiz cihaz tanımlayıcılarını, yazılım yürütülebilir dosyalarını şifrelemede gizli anahtarlar olarak kullanmaktır. Hedef cihazdaki PUF'a erişemeyen kötü niyetli taraflar, şifrelenmiş ikili dosya üzerinde statik veya dinamik analizler yapamazlar. Uçtan uca değerlendirmek için ERIC'in prototipini bir FPGA üzerinde geliştiriyoruz. Prototipimiz, yazılım şifre çözme maliyetlerini en aza indirmek için RISC-V Rocket Chip'i donanım şifre çözme motoru ile genişletir. RISC-V yürütülebilir dosyalarının kısmi/tam şifrelemesini etkinleştirmek için özel LLVM tabanlı derleyiciyi genişletiyoruz. HDE, küçük FPGA kaynak giderlerine maruz kalır, Rocket Chip taban çizgisine kıyasla %2,63 daha fazla arama tablosu (*-ing.* LUT) ve %3.83 daha fazla flip flop gerektirir. LLVM tabanlı yazılım şifrelemesi, derleme süresini %15,22 ve yürütülebilir program boyutunu %1,59 artırır. ERIC'e açık kaynaklı olarak https://github.com/kasirgalabs/ERIC adresinden erişilebilir.

Anahtar Kelimeler: Donanım truva atı, Güvenli yürütme, Bellek güvenliği.

ABSTRACT

Master of Science

DETECTING AN ACTİVATED TROJAN HORSE IN EMBEDDED SYSTEMS AND PROCESSORS WITH BLOOM FILTER-BASED MEMORY APPLICATIONS

Alperen BOLAT

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Department of Computer Engineering

Date: July 2022

Hardware trojans pose a number of security risks for modern processor-based systems. It comes with many security vulnerabilities such as changing the program executed on the processor, sending non-program instructions to the processor, capturing critical information of the programmer or hardware with analysis on the program, and preventing validation between sources. With Bloom filter-based memory applications, it is possible to ensure that the execution inside the processor is reliable. Thanks to the querying and training capability provided by the Bloom filter, training is done with a data set of commands and memory addresses, and this trained data is queried during execution. This ensures that execution on the processor is reliable and verified. On the other hand, with a number of encryption and secure network protocols to be determined between the processor and the compilation source, the problems caused by the sources verifying each other and external interventions to the system can be prevented. In this thesis, studies on architectural designs that will prevent the specified security vulnerabilities are explained.

The first work discussed in the thesis is a processor architecture that proposes to detect hardware trojan horse (HWT) activation. Proposed architecture presents a protection architecture meant to shield the communication between the processor and the memory in processor based system. The architecture aims at detecting the activation on HWTs infesting the instruction and data memories of the system. Our proposal relies on the use of Bloom Filters that are included in ad-hoc designed checkers and integrated in the protection architecture. Bloom filters could guarantee zero false alarms and a small and

configurable percentage of undetected alarms. We applied the protection architecture to a case study system based on a processor RI5CY of Pulpino (RISC-V instruction set) implemented on an FPGA and running a set of benchmark programs. Bloom filter based HWT checker mechanism demonstrated to be able to detect more than 99% of possible HWTs activations with zero false alarms. Also we measured a lookup table overhead ranging from 0.68% up to 10.52% and a flip-flop overhead between 0.68% and 0.99%, and with no working frequency reduction.

Another study discussed in the thesis proposes to ensure the security of execution inside the processor and to protect the program itself and the network between the processor and the compiler from attackers, especially HWT activation. Modern cloud computing systems distribute software executables over a network to keep the software sources, which are typically compiled in a security-critical cluster, secret. We develop ERIC, a new, efficient, and general software obfuscation framework. ERIC protects software against (i) static analysis, by making only an encrypted version of software executables available to the human eye, no matter how the software is distributed, and (ii) dynamic analysis, by guaranteeing that an encrypted executable can only be correctly decrypted and executed by a single authenticated device. ERIC comprises key hardware and software components to provide efficient software obfuscation support: (i) a hardware decryption engine (HDE) enables efficient decryption of encrypted hardware in the target device, (ii) the compiler can seamlessly encrypt software executables given only a unique device identifier. Both the hardware and software components are ISA-independent, making ERIC general. The key idea of ERIC is to use physical unclonable functions (PUFs), unique device identifiers, as secret keys in encrypting software executables. Malicious parties that cannot access the PUF in the target device cannot perform static or dynamic analyses on the encrypted binary. We develop ERIC's prototype on an FPGA to evaluate it end-to-end. Our prototype extends RISC-V Rocket Chip with the hardware decryption engine (HDE) to minimize the overheads of software decryption. We augment the custom LLVM-based compiler to enable partial/full encryption of RISC-V executables. The HDE incurs minor FPGA resource overheads, it requires 2.63% more LUTs and 3.83% more flip-flops compared to the Rocket Chip baseline. LLVM-based software encryption increases compile time by 15.22% and the executable size by 1.59%. ERIC is publicly available and can be downloaded from https://github.com/kasirgalabs/ERIC.

Keywords: Hardware trojan horse, Trusted execution, Memory security.

TEŞEKKÜR

Lisans ve yüksek lisans çalışmalarım boyunca her türlü yardım ve katkılarıyla beni yönlendiren, her zaman destek olan değerli hocam ve tez danışmanım Prof. Dr. Oğuz Ergin'e, kıymetli tecrübelerinden yararlandığım, bana her konuda yardımcı olan TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği Bölümü öğretim üyelerine, araştırmalarım boyunca bana her türlü yardım ve katkılarından ötürü University of Twente (UT) öğretim üyesi Prof. Dr. Marco Ottavi'ye, eğitimim boyunca bana burs veren TOBB Ekonomi ve Teknoloji Üniversitesi'ne, destekleriyle her zaman yanımda olan arkadaşlarıma ve bu süreçte gösterdikleri her türlü destek, sabır ve yardımları için biricik annem Selma Bolat'a , babam Erdal Bolat'a ve kardeşim Alparslan Bolat'a ayrıca eğitim hayatım boyunca birlikte çalıştığımız Kasırga Mikroişlemci Laboratuvarından çalışma arkadaşlarım İsmail Emir Yüksel, Ataberk Olgun, Nisa Bostancı, Yahya Can Tuğrul, Oğuzhan Canpolat ve Şevval İzmirli'ye de teşekkürlerimi sunarım. Son olarak sevgili eşim Cansu'ya hep yanımda olduğu ve beni desteklediği için çok teşekkür ederim.

İÇİNDEKİLER

	Sayfa
ÖZET	**iv**
ABSTRACT	**vi**
TEŞEKKÜR	**viii**
İÇİNDEKİLER	**ix**
ŞEKİL LİSTESİ	**x**
ÇİZELGE LİSTESİ	**xi**
KISALTMALAR	**xii**
SEMBOL LİSTESİ	**xiii**
1. GİRİŞ	**1**
2. TEMEL KONULAR	**3**
2.1 Donanım Truva Atı	3
2.1.1 Donanım truva atı tespiti	3
2.2 Bloom Filtre	4
2.3 Fiziksel Olarak Klonlanamayan Fonksiyonlar	5
2.4 LLVM	7
3. BLOOM FİLTRE TABANLI BELLEK UYGULAMASI	**9**
3.1 Motivasyon	9
3.2 İlgili Çalışmalar	9
3.3 Tehdit Modeli	10
3.4 Önerilen Yöntem	11
3.4.1 Denetleyicilerin yapısı	12
3.4.2 Donanım Truva atı tespit mekanizmasının tasarımı	13
3.5 Deneysel Sonuçlar	14
3.5.1 Deney ortamı	15
3.5.2 Bloom filtresinin tasarlanması ve sentezlenmesi	15
3.5.3 Bloom filtrelerinin donanım uygulaması ve değerlendirmesi	16
3.5.4 Güvenlik analizi ve değerlendirmesi	17
4. ERIC	**19**
4.1 Motivasyon	19
4.2 İlgili Çalışmalar	20
4.3 Tehdit Modeli	21
4.4 Önerilen Yöntem	21
4.4.1 Yazılım mimarisi	22
4.4.2 Donanım mimarisi	25
4.4.3 Tasarım akışı	26
4.5 Deneysel Sonuçlar	27
4.5.1 Yazılım kaynağı	28
4.5.2 Hedef donanım	31
5. SONUÇ	**33**
KAYNAKLAR	**34**

ŞEKİL LİSTESİ

Sayfa

Şekil 2.1: Bloom Filtresi İşlem Akışı 6
Şekil 2.2: 5 Bit Zorluk - 1 Bit Yanıt Gecikme Tabanlı PUF Şeması 6
Şekil 2.3: LLVM/Clang Akış Şeması 7
Şekil 3.1: Önerilen işlemci koruma mimarisi 12
Şekil 3.2: Bloom filtresinin donanım uygulaması 12
Şekil 3.3: Bloom filtresi için tasarım akışı 14
Şekil 4.1: İki Taraflı Kimlik Doğrulama Akışı 23
Şekil 4.2: ERIC'in Tasarım Akışı 26
Şekil 4.3: Uygulanan Modelin Şeması 29
Şekil 4.4: Şifrelenmemiş Programın Boyut Normalizasyonuna dayalı olarak Şifreli Program Paketleri ile Şifrelenmemiş Program Paket Boyutu Karşılaştırması 30
Şekil 4.5: Sınama Programlarına Bağlı Olarak, Şifrelenmemiş Programının Derleme Zamanı Normalleştirmesine Dayalı Derleme Zamanı Karşılaştırması 31
Şekil 4.6: Her Sınama Programına Göre, Şifrelenmemiş Programının Yürütme Süresine Karşılık Şifrelenmiş Programın Yürütme Süresi Karşılaştırması 32

ÇİZELGE LİSTESİ

Sayfa

Çizelge 3.1: Deneylerde Kullanılan Sınama Programları ve Buyruk Sayıları 15
Çizelge 3.2: Farklı miktarda kullanılan Özetleme Fonksiyonu sayısı için %1 AAO'da BAD'a ait Bloom Filtrenin bit dizisinin boyutu (bit cinsinden) 16
Çizelge 3.3: Farklı miktarda kullanılan Özetleme Fonksiyonu sayısı için %1 AAO'da BTD'a ait Bloom Filtrenin bit dizisinin boyutu (bit cinsinden) 16
Çizelge 3.4: BAD ve BTD için Bloom filtrelerin donanımda uygulanmasının boyutu (Kbit cinsinden) ve buna karşılık elde edilen AAO değeri (tespit edilemeyen DTA etkinliği oranının yüzdesi cinsinden) 17
Çizelge 3.5: Uygulanan BAD'ların kaynak işgali ve çalışma sıklığı 18
Çizelge 3.6: Uygulanan BTD'lerin kaynak işgali ve çalışma sıklığı..... 18
Çizelge 4.1: Test Ortamı 28
Çizelge 4.2: FPGA Uygulama Alan Sonuçları 31

KISALTMALAR

DTA : Donanım Truva Atı
BAD : Buyruk Akış Denetleyicisi
DTD : Bellek Tahsis Denetleyicisi
AAO : Algılanamayan Alarm Oranı
DRAM : Dynamic Random Access Memory
SoC : System on Chip
CAD : Computer Aided Design
LUT : Look-up Table
FF : Flip-flop
PUF : Physically Unclonable Function
FPGA : Field Programmable Gate Arrays
LLVM : Low Level Virtual Machine

SEMBOL LİSTESİ

Bu çalışmada kullanılmış olan simgeler açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklama**
Kbit	Kilobit (veri miktarı)
KiB	Kilobayt (veri miktarı)
MHz	Megahertz (aktarım hızı)
Gb/s	gigabit saniye (aktarım hızı)
Mb/s	megabit saniye (aktarım hızı)
ns	nanosaniye
μs	mikrosaniye
ms	milisaniye

1. GİRİŞ

Entegre devreler ve çipler, modern elektronik sistemlerin vazgeçilmez parçaları haline geldi. Üretim maliyetlerini azaltmak, pazara sunma süresini kısaltmak ve tasarım sürecini kolaylaştırmak için tasarımcılar ve entegre devre üreticileri arasındaki iş birliği artmakta ve çip endüstrisi küreselleşmektedir. Küreselleşmenin bir sonucu olarak, alt modül tasarımları ve çipte bulunacak üçüncü taraf fikri mülkiyet (3TFM, -*ing* 3PIP) çekirdekleri genellikle satın alınmakta ve çipin tasarımında kara kutu olarak kullanılmaktadır. Tasarım tamamlandıktan sonra, nihai çip üçüncü taraf dökümhaneler tarafından şeffaf olmayan bir üretim süreci ile üretilir. Bu dış kaynak bağımlılıkları ve alt tasarımların tasarım ve üretim süreçleri sırasında son kullanıcıdan yalıtılması, üretilen yonga tabanlı sistemin (YTS, -*ing* SoC) güvenilirliği konusunda ciddi endişelere neden olur.

YTS'nin geliştirilmesi sırasında saldırganların yararlanabileceği birçok güvenlik açığı vardır. Bu zafiyetler kullanılarak tasarımdaki gizli bilgiler tersine mühendislik teknikleri ile yakalanabilir [1], nihai ürün klonlanabilir [2], 3TFM çekirdeğinin hakları ihlal edilebilir [3] ve bir donanım Truva Atı (DTA) YTS [4]'e eklenebilir. DTA, üretim veya tasarım aşamalarında donanıma eklenen ve belirli tetikleme mekanizmalarına göre etkinleştirilebilen kötü amaçlı donanım değişiklikleridir. Yalnızca belirli tetikleme koşulları altında tetiklenebilmesi, DTA'ların tespit edilmesini zorlaştırır. DTA'lar, tasarım aşamasında [5] sırasında kaynak kodu değiştirilerek, silikon sentezi aşaması sırasında CAD araçları kullanılarak [6] veya bant çıkışı sırasında maskeler değiştirilerek YTS'ye eklenebilir [7].

Günümüzde akademi ve endüstriyel tarafta özellikle işlemci tabanlı sistemlerde DTA bulunma ihtimali sorgulanmakta ve çeşitli güvenlik mekanizmaları önerilmektedir. DTA'ların işlemci tabanlı sistemlerdeki olası etkisi sisteme ait buyrukların (-*ing*, instruction) manipüle edilmesi ve bunun sonucunda sisteme ait olmayan, değiştirilmiş veya zarar verilmiş buyrukların sistem tarafından yürütülmesinin sağlanmasıdır. Bu saldırının donanımın çalışma zamanı içerisinde ne zaman gerçekleşeceği, donanıma etkisinin ne olacağı ve donanımın neresinde bulabileceği tam olarak bilenemediğinden tüm olası saldırıları önleyebilecek tek bir çözüm önerilememektedir. Fakat işlem gücü olan akıllı bir işlemci tabanlı yonganın saldırgan tarafından sömürülebilmesinin temel motivasyonu işlemciye ait yürütmeyi etkileyebilmek ve gerekirse saldırganın amacına uygun kod dizinini işlemci yürütme sırasına dahil edebilmektir.

Bu tezde ifade edilen çalışmalarda ilk olarak işlemci tabanlı sistemlerde, donanıma ait olmayan ve saldırgan tarafından belleklerde saklanan program makine koduna yapılabilecek değişiklikleri ve program sırasında, program sayacın davranışında herhangi bir bozulmayı/etkiyi tespit etmeyi amaçlayan çalışmamız açıklanacaktır. İkinci olarak ise programın oluşturulduğu aşama olan derleme aşasından, programın yürütüldüğü yürütme birimine ve boru hattına kadar programı şifreleme ve imza üretme yöntemlerini dinamik olarak kullanarak saldırgan faaliyetlerden koruyan çalışmamız sunulacaktır.

İlk çalışmada, donanım içerisinde bellek ile işlemciye ait boru hattı aşamalarından birisi olan getir (-*ing* fetch) aşaması arasında önerilen Bloom filtresi tabanlı eklenti ile işlemciye yürütmesi için saldırgan tarafından gönderilen zararlı, bozulmuş veya

değiştirilmiş buyrukları ve program sırasında olmadan getirilmeye çalışılan buyrukları tespit edebilen bir mekanizma önerilmektedir. Bu mekanizma programın içinde bulunsa daha sırası yani program sayacı ile eşleşmeyen bir buyruğun işlemci tarafından yürütülmesini engeller bu sayede DTA kullanılarak belleklerde veya işlemcide gerçekleştirilen ve amacı işlemcide kötü amaçlı yürütme sırasını bozmak veya saldırgan buyruk yürütmek olan operasyonları tespit eder. Bloom filtre tasarımı gereği birçok değişkene bağlıdır. Bunlar özetleme (-*ing* hash) fonksiyonun kendisi, sayısı ve filtre için ayrılan bit genişliği olabilir. Çalışmada belirli sınama (-*ing* benchmark) programları için en verimli Bloom filtre değerleri gösterilmiştir. Bununla beraber güncel bir işlemcide sistemin performansı ve yükü çalışma ile sunulmuştur.

İkinci çalışmada ise, işlemci ve program kaynağı olan derleyici arasında iki taraflı doğrulama yapılmasını sağlayan bunun yanında programın makine kodunu hedef donanım ve kaynak derleyici arasında şifreleme ve paketleme yaparak saldırgandan koruyan uçtan uca bir mimari önerilmiştir. Önerilen mimari güncel bir derleyici tasarımı ve donanım üzerinde test edilerek sonuçlar gösterilmiştir. Ayrıca çalışmanın bir prototipi de açık kaynaklı olarak paylaşılmıştır. Çalışmada önerilen şifreleme ve paketlemenin kaynak derleyici ve hedef donanım için maliyetleri gösterilmiştir. Ayrıca derlenmiş programın boyutu ve derlenme süresindeki değişim gibi durumlar çeşitli değişkenlere bağlı olarak gösterilmiş ve tartışılmıştır.

Bu çalışmalar ile literatüre aşağıda listelenen katkılar sağlanmıştır:

- Bloom filtre tabanlı sistemler işlemci mimarilerinin içerisine eklenerek program sırasına ait olmayan, saldırgan tarafından değiştirilen, bozulan veya sisteme eklenen buyruklar etkin şekilde tespit edilerek önlenebileceği gösterilmiştir.
- Farklı sınama programları için ideal Bloom filtre tasarımı değişebilir. Programın boyutu, buyrukların dağılımı ve karakteristiği gibi değişkenlere bağlı olarak ideal Bloom filtre tasarımı belirlenebileceği ve işlemciye entegre edilebileceği gösterilmiştir.
- Uçtan uca bir mimari tasarımı ile program kaynağı olan derleyici ile programı yürütücüsü olan işlemci arasında güvenli bir ağ protokolü elde edilebilir. Bu sayede program sadece güvenilir kaynaktan alınabilir ve güvenilir donanımda çalıştırılabilir. Bu sayede donanım ve yazılım arasında iki taraflı kimlik doğrulaması sağlanacağı gösterilmiştir.
- Program önerilen şifreleme, paketleme ve iki taraflı kimlik doğrulaması ile sistem içerisinde saldırgan faaliyetlerden korunarak gizlenebileceği gösterilmiştir.

2. TEMEL KONULAR

Bu bölümde tezin devamında anlatılan çalışmaların anlaşılabilir olabilmesi için gerekli temel bilgiler bulunmaktadır. Sırasıyla donanımda Truva atı (DTA) ve Bloom filtresi hakkında gerekli bilgilendirme yapılacaktır.

2.1 Donanım Truva Atı

İşlemci tabanlı donanımların ve gömülü sistemlerin güvenliğinin sağlanabilmesi için içerisindeki yazılım ve donanımın kaynağından çıktığı şekilde muhafıza edilmesi gerekir. Günümüzdeki güvenlik varsayımlarının çoğu, işlemi gerçekleştiren donanımın güvenli olması üzerine kurulmuştur. Donanım Truva atı (DTA), donanıma çeşitli aşamalarda dahil edilen veya uygulanan değişiklikler ile sistemin saldırılara açık hale getirilmesine neden olan zararlı uygulamalara denilir. Olası DTA eklentisinin hedef sisteme dahil edileceği aşamaların net olarak belirlenememesi ve hedef sistemin üretimden son kullanıcıya ulaşana kadar birçok aşamadan geçiyor olması, endüstrideki güvenlik endişelerini arttırmaktadır. DTA eklenerek oluşturulan saldırılar ile önemli güvenlik zafiyetleri gösterilmiştir.

İşlemcinin ve donanım geliştirilmesi sırasında DTA kullanacak saldırganların yararlanabileceği birçok güvenlik açığı vardır. Bu zafiyetler kullanılarak tasarımdaki gizli bilgiler tersine mühendislik teknikleri ile yakalanabilir [1], nihai ürün klonlanabilir [2], üçüncü taraf IP çekirdeğinin hakları ihlal edilebilir [3] ve bir DTA Çip Sistemine (-ing System-on-Chip) [4] eklenebilir. Yalnızca belirli tetikleme koşulları altında tetiklenebilmesi, DTA'ların tespit edilmesini zorlaştırır. DTA'ları, tasarım aşamasında [5] sırasında kaynak kodu değiştirilerek, silikon sentezi aşaması [6] sırasında CAD araçları kullanılarak veya bant çıkışı sırasında maskeler değiştirilerek SoC'ye eklenebilir [7].

DTA'lar birkaç mantık kapısı ile oluşturulabileceği gibi çok daha karmaşık bir hiyerarşi ile sisteme entegre edilmiş büyük devreler de olabilmektedir. DTA eklenen sistemde çalıştırılmasını ve etkinleştirilmesini sağlayan tetik (*-ing*, triger) bağlı olarak sınıflandırabilirler. Aynı zamanda DTA'lar eklendikleri aşamaya göre, eklendiği fiziksel katmana göre, sistem içerisinde eklendiği bölgeye ve sisteme etkilerine göre de kategorilere ayrılabilirler.

2.1.1 Donanım truva atı tespiti

Tasarım ve üretim sürecinin tam kontrolü olmadan elde edilen herhangi bir sistemin donanımı, DTA yerleştirme için yüksek risk taşır. Bu riski azaltmak için modern sistemlerde DTA aktivitesini tespit etmek için çalışmalar önerilmiştir. Önerilen DTA tespit mekanizmaları olayı iki açıdan değerlendirir: (i) üretim öncesi ve (ii) üretim sonrası tespit mekanizmaları.

SoC fabrikadan çıkmadan önce DTA tespitini hedefleyen üretim öncesi tespit mekanizmaları. Bu çalışmalar temel olarak CAD araçlarını, ayrıntılı bir çip düzenini veya donanım tanımlama dili (örneğin Verilog, Vhdl) tabanlı tasarım kodlarının

simülasyonlarını kullanarak DTA'nın eklenmesini engellemeye çalışır [8]. Çoğu zaman, üretim öncesi DTA algılama çalışmalarını uygulamak için, çipin kaynak kodu, silikon düzeni veya silikon hücre kitaplığının detayları gibi son kullanıcının erişemeyeceği ayrıcalıklı bilgilere ihtiyaç duyulabilir.

İkinci olarak, üretim sonrası DTA algılama mekanizmaları vardır. Bu çalışmalar, SoC'yi saldırganlar tarafından DTA eklenebilecek güvenilmeyen/bilinmeyen donanım olarak varsaydıkları için bu tezdeki çalışmaların da kapsamındadır. [9], [10], [11], işlemci üzerinde kimliği doğrulanmış ve güvenilir yürütme için önerilen yazılım tabanlı çözümlerdir. Ancak, DTA'nın faaliyetleri, boşta kalması ve çıktıların küçük ayak izi (-*ing* footprint) ile derleme düzeyinde güvenli bir mekanizmadan gizlenebilir. Ayrıca, buyrukların doğrudan ardışık düzene enjekte edilmesi, süper kullanıcı ayrıcalıklarına neden olabilir ve derleme düzeyindeki çözümleri atlayabilir. Bu çalışmaların DTA'ya karşı yetersiz olmasının ana nedeni, çip dışını riskli ve çipte olanı güvenilir olarak kabul etmeleridir.

Öte yandan, [12], [13] çalışmalarında bir işlemci ve bellekler arasında ekstra filtre tabanlı denetleyiciler önerilir. Bu çalışmalar, işlemci üzerindeki her buyruğun kullanıcının programına ait olmasını ve DTA tarafından belleklere talimat enjeksiyonlarının işlemci tarafından yürütülemeyeceğini garanti etmeyi önermektedir. Bu çalışmaların ana perspektifi, güvenilir tarafı CPU, güvenilmeyen tarafı ise tüm bellek hiyerarşisi olarak varsaymaktır ve SoC mimarisinde pahalı ek değişikliklere ihtiyaç duyarlar. Ancak önbellekler genellikle SoC'nin bir parçasıdır ve CPU ile birlikte SoC'de bulunur. Dolaylı olarak, SoC'nin güvenliği için DTA yerleştirilmiş sistemde bir kontrol mekanizması oluşturmayı önermek, tüm bellek seviyelerini çipin geri kalanından soyutlamak anlamına gelir.

2.2 Bloom Filtre

Bloom filtreleri, bir dizi elemanın grup seti içerisinde bulunup bulunmadığına dair bilgilerini depolayan, yaygın olarak kullanılan olasılıksal veri yapılarıdır. Bir öğenin kümeye ait olup olmadığını kontrol etmek için Bloom filtrelerinde sorgular çalıştırılabilir [14, 15]. Bloom filtrelerinin önemli bir özelliği, sorgulamanın yanlış pozitifle (-*ing* false positive) sonuçlanabilmesine rağmen, yanlış alarmların olamayacağının her zaman doğru olmasıdır. Başka bir deyişle, bir öğenin sorgusu pozitif bir değer döndürürse, kümede olmaması için bir şans vardır (bizim durumumuzda algılanmayan alarm); ancak bir negatif döndürülürse, öğenin kümede olması mümkün değildir (bizim durumumuzda yanlış alarm).

Bloom filtre, ilgilenilen kümenin her bir öğesinin bir veya daha fazla dizi konumuna eşlendiği bir bit dizisi olarak uygulanır. Her konumun adresi, öğenin kendisinde hesaplanan bir Özetleme işlevinin (-*ing* hashing) çıktısı ile ilişkilendirilir. Bir Bloom filtre üzerinde iki işlem gerçekleştirilebilir: eğitim (-*ing* load) ve sorgu (-*ing* query).

Bloom filtreyi eğitmek, ilgilenilen kümenin tüm öğelerini içinde depolamak anlamına gelir: bunu yapmak için kümedeki her bir e_i öğesi ve her bir özetleme işlevi $ozetleme_j$ için $ozetleme_j(e_i)$ adresiyle ilişkili bit dizisi konumu 1 olarak ayarlanır (başlangıçta tüm bit dizileri 0'dır).

Bloom filtreyi sorgulamak, bir öğenin ilgilenilen kümede olup olmadığını ve dolayısıyla bit dizisinin ilişkili konum veya konumlarının 1'e ayarlanıp ayarlanmadığını kontrol etmek anlamına gelir. Kontrol edilmesi gereken belirli bir e_i öğesi ve her bir $ozetleme_j$ Özetleme işlevi için, $ozetleme_j(e_i)$ adresiyle ilişkili bit dizisi konumlarından en az birinin 0 olduğu bulunursa, Bloom filtre bir alarm verir.

Bloom filtreler uygun şekilde tasarlandığında sıfır yanlış alarmı garanti edebilirler, ancak yine de algılanmayan alarmların sıfır olmayan bir yüzdesi olabilir. Bununla birlikte, bu tür algılanmayan alarmların yüzdesi, m'nin, filtrenin bit dizisinin boyutuna (bit olarak ifade edilir), n'nin, Bloom filtreye eklenen öğelerin sayısına ve k'nin, kullanılan özetleme fonksiyonları sayısına bağlı bir fonksiyon olarak hesaplanabilir. Belirtilen değişkenlere bağlı olarak bir Bloom filtre tarafından tespit edilemeyen sorguların oranı yani algılanamayan alarm oranı (AAO) hesaplama formülü şu şekildedir:

$$AAO \approx (1 - e^{-\frac{n \cdot k}{m}})^k \tag{2.1}$$

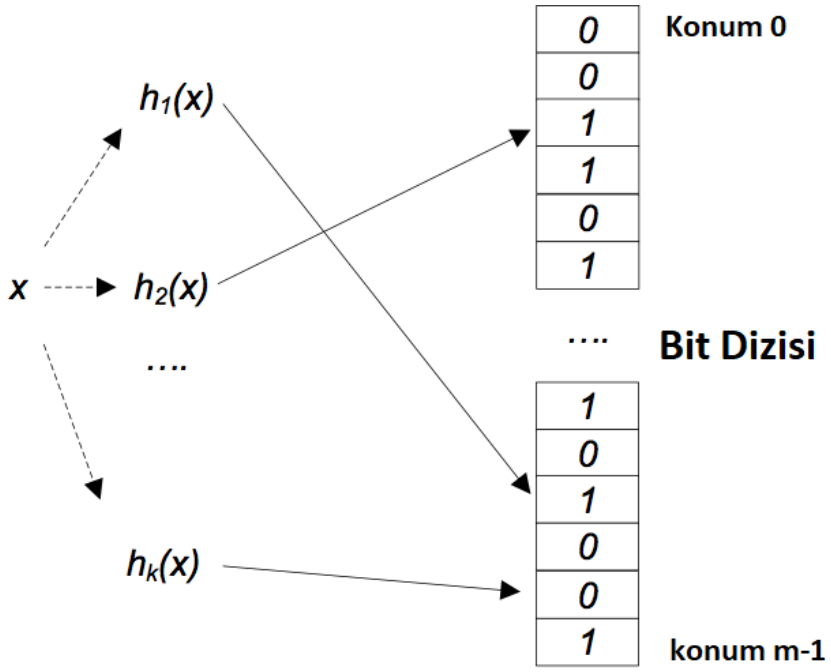
bu nedenle, m arttıkça AAO azalır. AAO'yu en aza indiren k değeri şu şekilde verilir:

$$k_{opt} = \frac{m}{n} \cdot ln2 \tag{2.2}$$

Şekil 2.1 bir Bloom filtresinin işlem akışını göstermektedir. Öncelikle ilgili veri seti ile Bloom filtresini eğitebilmek için veri setinde bulunan her eleman tüm özetleme (*-ing* hash) fonksiyonlarından geçirilir ve filtrenin kendisi olan bit dizisi üzerinden haritalandırıldığı noktalardaki bit değerlerini 1 yaparlar. Tüm veri seti özetleme fonksiyonlarından geçirildikten ve bit dizisi üzerindeki değişiklikler yapıldıktan sonra Bloom filtresi sorgulama aşamasına sokulabilir. Sorgulama aşamasında filtreye gelen bilinmeyen bir verinin filtrenin eğitimi sırasında kullanılan verilerden birisi olup olmadığı sorgulanır. Eğer bilinmeyen eleman özetleme fonksiyonlarından geçtikten sonra ilişkilendirildiği indislerdeki bit elemanları 1 ise bu elemanın filtrede olduğu sonucuna varılır (yanlış pozitif durumu da mümkündür). Eğer ilişkilendirilen indislerden birisinde 0 değeri bulunuyorsa bu elemanın veri seti içerisinde olmadığına karar verilir ve bu sonuç için hatalı karar verilmiş olma riski yoktur. Yani Bloom filtreler bir verinin veri setinde olduğunu söylüyorsa bu kesin bir şekilde doğru olmayabilir fakat o elemanın veri setinde olmadığı filtre tarafından belirlenmiş ise bu karar kesin ve doğru bir karardır.

2.3 Fiziksel Olarak Klonlanamayan Fonksiyonlar

Fiziksel olarak klonlanamayan fonksiyonlar (PUF), belirli bir girdi ve koşullar (zorluk, *-ing* challange) için fiziksel olarak tanımlanmış bir "dijital parmak izi" çıktısı (yanıt, *-ing* response) sağlayan ve genellikle bir yarı iletken cihaz için benzersiz bir tanımlayıcı olarak hizmet veren bir işlevdir [16], [17] . PUF'lar çoğunlukla yarı iletken üretimi sırasında doğal olarak meydana gelen varyasyonlara dayanır. Bu varyansların dağılımı her yarı iletken için benzersiz olduğundan, aygıtın benzersiz kimliğini elde etmek için bunu kullanır. Bugüne kadar önerilen birçok PUF yöntemi

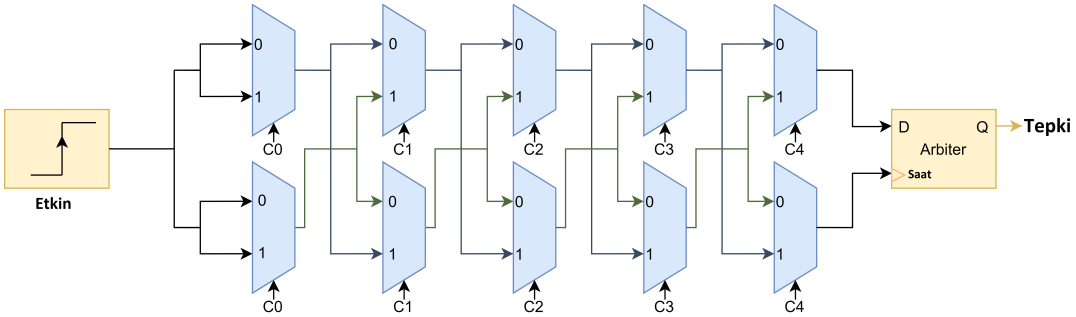


Şekil 2.1: Bloom Filtresi İşlem Akışı

vardır [18], [19], [20], [21], [22], [23], [24], [25]. Gecikme tabanlı (*-ing* Arbiter) PUF'lar, kullanılan en yaygın yöntemler arasındadır [26].

Gecikme tabanlı PUF'ları, [27] karşılaştırmasının sonucuna bağlı olarak bir '0' veya '1' biti oluşturmak için iki özdeş yolun gecikmesini karşılaştırır. Hiçbir iki yol aynı olmamasına ve aynı gecikmeye yol açmasına rağmen, üretim sürecindeki küçük öngörülemeyen farklılıklar bir yolu diğerinden daha hızlı hale getirir.

Gecikme tabanlı PUF'lar, sorgulamaya ve ona bağlı yanıt üretmeyi içeren bir kimlik doğrulaması yöntemi kullanır. Bu yöntem temelde, kaynağın gelen bir takım sorgulama sinyalini kimlik için üretilecek anahtarın tohumu olarak kullanarak çalışan bir fonksiyonu içerir. PUF tabanlı sistemler, yanıt çıktısının doğrulanmasıyla Sistem kimlik doğrulamasını kontrol edebilir. Şekil 2.2, 5 bitlik sorgulama (zorluk) ve 1 bitlik yanıt içeren gecikme tabanlı PUF modelinin şemasını göstermektedir. Şekilde görüldüğü gibi, yanıt çıktısı, donanım üzerindeki gecikmelerin yoluna ve meydan okuma değerine bağlı olarak değişmektedir.



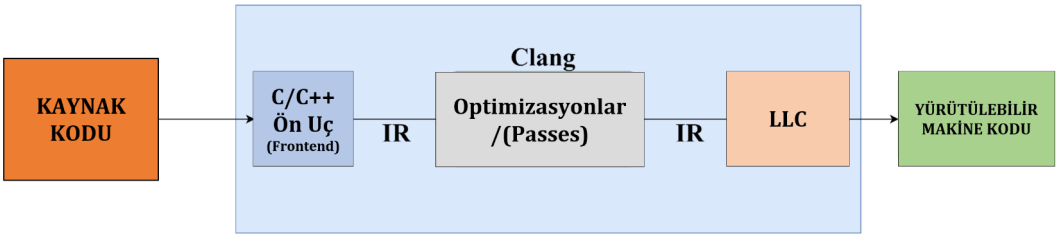
Şekil 2.2: 5 Bit Zorluk - 1 Bit Yanıt Gecikme Tabanlı PUF Şeması

2.4 LLVM

Günümüzde kullanılan programlama dilleri ve yazılım uygulamaları, daha esnek, güçlü ve güvenilir bir derleme arayüzünü, programcının üretkenliğini arttırmayı ve derleme sırasında programın hedef donanım için daha uygun hale gelmesini sağlayacak optimizasyonları desteklemeyi hedefler. Derleme sırasında kullanılan geleneksel ve genellenebilir yaklaşımlar, hedeflenen program çıktısının performansının yeterli seviyede iyileştirilmesini engellerken, birçok sistemde başarımın düşmesine neden olur.

LLVM, ön uç olarak herhangi bir programlama dilini ve arka uç olarak herhangi bir komut seti mimarisini geliştirmek için kullanılabilen bir dizi derleyici ve araç zinciri teknolojisidir [28], [29]. LLVM, Orta Düzey Temsil (IR) adı verilen dilden bağımsız bir metodoloji etrafında tasarlanmıştır. Herhangi bir dil kullanılarak derlenen program önce IR diline çevrilir. IR, birden çok geçişte çeşitli dönüşümlerle optimize edilebilen üst düzey bir derleme dilidir. IR ile derleme sırasında birçok optimizasyon ve analiz mümkündür. Programın IR gösterimi üzerinde istenilen tüm optimizasyonlar ve analizler yapıldıktan sonra, IR dilinden hedef komut seti mimarisi (ISA) için ikili (*-ing* binary) kod çevirisi üretilir. IR metodolojisi ile LLVM üzerinde özel bir derleyici geliştirmek mümkündür. Tasarlanan derleyici, IR dili üzerinde hedef derleme seçenekleri gerçekleştirilerek oluşturulabilir. Ayrıca bu derleyici, LLVM kitaplıklarının desteği sayesinde çoğu dille uyumludur. Şekil 2.3'de LLVM ve Clang akış şeması gösterilmektedir.

Bunun yanında Clang, C/C++ tabanlı diller için yapılan derlemelerde LLVM'e ön uç (*-ing* front-end) desteği sağlayan derleyici sistemdir. Ayrıca Clang GCC kütüphanelerindeki bayraklar (*-ing*, flags) ile de uyumludur.



Şekil 2.3: LLVM/Clang Akış Şeması

3. BLOOM FİLTRE TABANLI BELLEK UYGULAMASI

3.1 Motivasyon

Gerçek dünya devrelerine yerleştirmenin zorluğu ve geçmişteki önerilmiş tehditler göz önüne alındığında, DTA'lar endüstriden çok akademi tarafından bir sorun olarak kabul edilir. Bununla birlikte, son yıllarda, karmaşık yazılım tarafından sömürülebilir DTA'ları bulundu ve gerçek dünyadaki ticari mikroişlemcilere yerleştirildi. Ayrıca DTA'nın, saldırganın kendi kötü amaçlı yazılımını yürütmesine, çalışan yazılımı değiştirmesine veya kök ayrıcalıkları edinmesine izin verebileceği gösterilmiştir [30, 31, 32]. Ayrıca, 2018'de, ticari bir Via Technologies C3 işlemcisinde [33] Rosenbridge arka kapısı olarak adlandırılan bir DTA bulundu. Bu Truva atı, yönetici moduna girmek için yazılım aracılığıyla etkinleştirilebilir ve kullanılabilir.

Bu çalışmada, işlemci tabanlı sistemleri DTA'larına karşı işlemciyi korumak için sistem düzeyinde bir mimari öneriyoruz. Daha ayrıntılı olarak, önerilen mimari, bir Harvard mimarisinin hem buyruk hem de veri belleklerini istila eden DTA'larının çalışma zamanı aktivasyonunu tespit etmeyi amaçlar. Mikroişlemciyi kötü amaçlı bir kod çalıştırmaya ve/veya yetkisiz bellek konumlarında veri okumaya/yazmaya zorlayan, yazılım tarafından sömürülebilir DTA'larını tespit etmeyi amaçlıyoruz. Ayrıca, olası hizmet reddi ve bilgi hırsızlığı DTA'ların bir alt kümesi olarak kabul ederek mimarimizdeki güvenlik tehdidine dahil ediyoruz. Önerilen koruma mimarisi, buyruk belleğinden getirilen buyruklar ve hem buyruk hem de veri belleğindeki erişilen adresleri izleyen Bloom filtre tabanlı iki denetleyiciye dayanır. Önerilen çözümün tamamen sisteme entegre ederek sistemin normal işleyişi ve çalışma zamanında herhangi bir gecikmeye veya kod yürütmesinde bir kesintiye neden olunmuyor.

Önerilen mikroişlemci koruma mimarisine, bir FPGA cihazında uygulanan ve bir dizi yazılım kıyaslaması çalıştıran bir RISC-V işlemcisine dayalı bir sınama programı sistemi uyguladık. Mimari, olası DTA aktivasyonlarının %99'undan fazlasını sıfır yanlış alarmla her zaman tespit edebildi. Çalışma frekansında herhangi bir azalma olmaksızın %0,68 ile %10,52 arasında değişen bir arama tablosu ek yükü ve %0,68 ile %0,99 arasında bir flip-flop ek yükü ölçtüldü.

3.2 İlgili Çalışmalar

Modern entegre devrelerin karmaşıklığı ve DTA'ların gizlenebilir yapısı göz önüne alındığında, DTA'ları sisteme konuşlandırılmadan önce tespit etmek giderek daha zor hale geliyor. Tasarım zamanında DTA'larını tespit etmeyi amaçlayan klasik devre düzeyi taramaya dayalı teknikleriyle birlikte (mantık testi [34], resmi özellik doğrulama [35], yan kanal analizi [36], yapısal ve davranışsal analiz [37, 38, 39]) güvenilmeyen bileşenlerle oluşturulmuş güvenilir bir sistem elde etmeyi sağlayan *sistem düzeyinde* tekniklere artan bir ilgi var [40, 41, 42]. Benzer bir fikir [43, 44]'da da önerilmiştir; burada odak noktası mikroişlemci tabanlı sistemlerdir ve amaç, güvenilmeyen bir CPU ile güvenilir bir yazılım yürütme elde etmektir.

Öte yandan, son zamanlarda bellek yongalarındaki DTA'lar da incelenmiştir [45].

Aynı zamanda, [45]'da tartışıldığı gibi, mikroişlemci tabanlı sistemleri belleklere yerleştirilmiş DTA'lardan korumak için henüz yeterli çalışma yapılmamıştır.

Bildiğimiz kadarıyla, işlemci tabanlı sistemlerin belleklerinde olabilecek DTA'larına karşı sistem düzeyinde herhangi bir koruma metodolojisi henüz önerilmemiştir. Önerimize daha çok benzediğini düşündüğümüz çalışmalar, [43, 44]'da bulunan, bizim mimarimizde olduğu gibi, problemin sistem seviyesinde (devre seviyesinde değil) ele alıyor ve işlemci çekirdeği ile bellek arasına koruma ünitesi entegre edilmesini öneriyor. Fakat bu çalışmalarda bizim önerimizden farklı olarak mikroişlemciye tehdit modelinin içinde ve bellekler güvenilir varsayılır. [43]'de koruma birimi, yürütülen buyrukların işlem kodunun ve ilgili kontrol sinyallerinin geçerli olup olmadığını ve bir buyruk yürütmek için kullanılan saat çevrimi sayısının doğru olup olmadığını kontrol eder. [44]'da koruma birimi, işlemcinin hala aktif olup olmadığını ve doğru işletim sistemi modunda çalışıp çalışmadığını kontrol eder. Her iki çözüm de işlemcinin normal yönergeleri çalıştırmasını sağlayarak sistemin işlevselliğini değiştiren DTA'larını hesaba katmaz. Başka bir deyişle, bu çalışmaların hiçbiri işlemcinin istenmeyen yazılımları yürütüp yürütmediğini ve hedef bellek konumlarına erişip erişmediğini kontrol etmez.

3.3 Tehdit Modeli

Daha önce de belirttiğimiz gibi, bir DTA, çoğu zaman sessiz kalan, belirli nadir durumlarda sistemin nominal davranışını değiştiren veya bilgi çalan bir tasarımın tespit edilmesi çok zor bir modifikasyonudur. [46]'da sunulan sınıflandırmaya göre, DTA'lar tetikleme mekanizması, yükü ve ekleme aşaması temelinde sınıflandırılabilir.

Bir DTA aşağıdaki şartlara göre tetiklenebilir:

- Dahili mantıksal sinyaller (veya mantıksal sinyal dizileri) veya fiziksel nicelikler, örneğin sıcaklık veya voltaj veya bir sayaç (saatli bombalar olarak da adlandırılır) yoluyla tetiklenebilir.
- Dışarıdan veya sistemin kendisinden alınan mesajlar, örüntüler ya da fiziksel etkileşimler yoluyla tetiklenebilir.
- Sürekli uyanık şekilde sisteme güç sağlanması ile birlikte tetiklenebilir.

Yüke bakıldığında, DTA'ları şu şekilde sınıflandırılabilir:

- Etkilenen sistem tarafından yürütülen işlevleri değiştiren
- Mevcut iletişim arayüzleri veya gizli yan kanallar aracılığıyla yetkisiz bilgileri sızdıran
- Sistemin çalışmasını engelleyen ve donduran DTA'ları, örneğin, boş veya işlevi olmayan buyrukları işlemciye vererek, sistemin pillerini boşaltarak veya iletişim arayüzlerini bozarak sistemin işleyişini durduran DTA etkinliği.

Son olarak, DTA'lar, çekirdek içerisinde eklenen modül sağlayıcıları tarafından satın alınan 3TFM'lere (*-ing* 3PIP), kötü niyetli tasarımcılar tarafından ve muhtemelen tasarım akışının her aşamasında ve talaş üretimi sırasında dökümhane tarafından kullanılan devre tasarımı ve sentezlemesini sağlayan araçlar tarafından eklenebilir.

Bu çalışmada, mikroişlemci tabanlı bir sistemin komut ve veri belleğini istila eden DTA'larını ele alıyoruz. Öte yandan, mikroişlemci burada güvenilir olarak kabul edilir. Önerilen metodolojinin etkinliği tetikleyiciye bağlı değildir. Aslında, daha önce tartışılan tetikleme mekanizmalarından herhangi birine sahip DTA'lar ele alınmaktadır. Yüke baktığımızda, işlemciye istenmeyen bir kod yürütmeye zorlayarak sistemin işlevselliğini değiştiren DTA'ları tespit edebiliyoruz. Ayrıca, yetkisiz bellek konumlarında gizli bilgileri okuyan/yazan ve sistemden veri çalan DTA etkinliğini tespit edebiliyor ve işletim sistemini uyarıyoruz.

DTA ekleme açısından, önerilen algılama metodolojisi çalışma zamanında çalıştığı için, tasarım sürecinde ve tedarik zincirinde yer alan herhangi bir aktör tarafından eklenen DTA'ları tespit edebildiğinden önerilen yöntem özellikle kritik güvenlikli sistemlerde gereklidir.

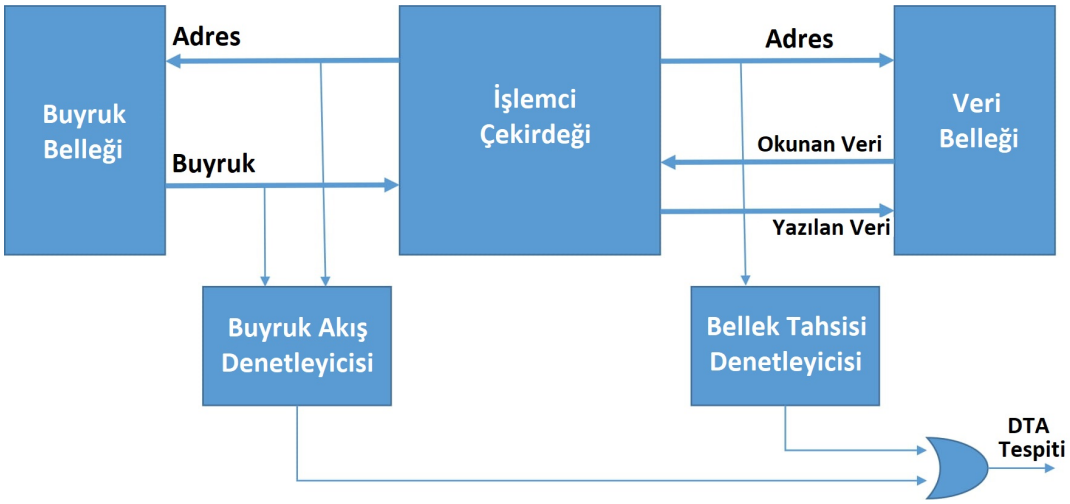
İşlemciye kötü niyetli olarak her zaman aynı buyruğu (veya buyruk dizisini) alarak sistemi durduran hizmet reddine neden olan DTA'ları, önerilen mimariye geçici boyutlu bir bekçi köpeği (*-ing* watchdog) sağlanarak tespit edilebilir. Bu, önerilen tasarımın kapsamı dışındadır. Ayrıca, önerilen metodoloji, işlemcinin getirme etkinliğini izleyen bekçi köpeklerinden yararlanarak, işlemciyi donduran ve hizmet etmesini engelleyen DTA etkinliğini tespit edebilir.

Çalınan bilgileri gizli bir kanal aracılığıyla gönderen bilgi çalan DTA'lar ve mikro mimari seviyenin dışında hareket eden hizmet reddi DTA'ları, örneğin pilleri boşaltan veya iletişim arayüzlerini sıkıştıran bu önerilen metodolojinin kapsamı dışındadır.

3.4 Önerilen Yöntem

Önerilen mimari, mikroişlemciyi talimat ve veri belleğine yerleştirilmiş DTA'lardan korur. Özellikle buyruk akışında ve bellek adres alanında anormal davranışların aktivasyonunu tespit etmeyi amaçlar. Bu, genel bir Harvard mimarisi için Şekil 3.1'de gösterildiği gibi, alınan buyrukları ve çekirdeğin buyruk ve veri belleklerine erişimini izleyen iki denetleyici eklenerek yapılır, yani buyruk belleği veri belleğinden ayrıdır.

İki denetleyici, yani Buyruk Akış Denetleyicisi ve Bellek Tahsis Denetleyicisi, çalışma zamanında sırasıyla buyruk ve veri belleğine erişimlerin yürütülmekte olan program tarafından beklenenler olup olmadığını izler. Ayrıca, Buyruk Akışı Denetleyicisi, işlemci tarafından getirilen buyrukların getir aşaması tarafından programın doğru adresine karşılık gelip gelmediğini izler. Beklenmeyen bellek erişimleri veya hatalı alınan buyruk durumunda, denetleyiciler derhal bu sorunu bildirir ve muhtemelen boru hattı temizleme, kesinti oluşturma veya diğerleri gibi hızlı karşı önlemlere yol açar. Tasarım DTA etkinliği sırasında alarm verecek şekilde tasarlanmıştır ve bu tespitten sonra sistemin uygulayabileceği prosödürler tartışılmamıştır. Beklenmeyen davranışları saptamak için denetleyiciler, çalışan programın tüm doğru adres ve buyruk kombinasyonlarını ve tüm yetkili veri adreslerini depolayabilmelidir. Bu değerleri

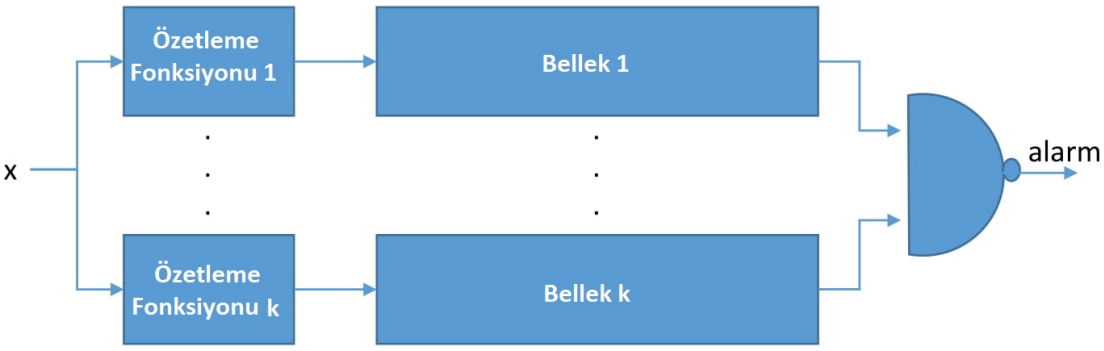


Şekil 3.1: Önerilen işlemci koruma mimarisi

verimli bir şekilde saklamak için iki denetleyiciyi Bloom filtresi tabanlı hızlı karar verebilen veri seti sorgulama yapılarını kullanmıştır. Benzer bir sorgulama yapısı farklı bir amaçla [47]'da da gösterilmiştir.

3.4.1 Denetleyicilerin yapısı

Daha önce tartıştığımız gibi, hem Buyruk Akış Denetleyicisi (BAD) hem de Bellek Tahsisi Denetleyicisi (BTD), Bloom filtre tabanlı bir tasarıma dayanır. Bir Bloom filtresi, k bellekleri kullanılarak donanımda verimli bir şekilde uygulanabilir, öyle ki her bir özetleme (*-ing* hash) fonksiyonu bunlardan biriyle eşleşir. Daha sonra, bellekten veriler paralel olarak okunabilir ve nihai sonucu elde etmek için denetleyicilerden gelen sonuçlar bir Ve Değil (-ing NAND) kapısı ile birleştirilir. Erişilen bellek konumlarından en az biri DTA tespiti yani uyumsuz veri akışı içerdiğinde sistem alarm verir. Uygulanan Bloom filtre yapısının mimari ile entegrasyonunun bir temsili Şekil 3.2'da gösterilmektedir; burada x, BAD için getirilen buyruk ve ona ait bellek adresi demetidir ve BTD için yalnızca bellek adresidir. Bu yapı, Denklem 2.1 tarafından hesaplanan değere benzer bir algılanmayan alarm oranına (AAO) ulaşır.



Şekil 3.2: Bloom filtresinin donanım uygulaması

Dikkate alınan DTA modelleri, beklenmeyen buyruk dizilerinin getirilmesine ve/veya yetkisiz buyruk ve veri bellek adreslerine erişime neden olur. Bu nedenle, BAD'daki

Bloom filtresi, analiz edilen programın tüm bellek adresi ve adrese ait buyruk demetleriyle eğitilir ve BTD, program tarafından erişilen tüm olası veri belleği adresleriyle yüklenir. Bu bilgi, tasarım zamanında simülasyon yoluyla elde edilen bir yürütme izinden çıkarılır (bkz. Alt Bölüm 3.4.1).

Çalıştırma zamanında, adresleriyle birlikte bellekten alınan talimatlar, talimatın kendisi yürütülmeden önce, yürütmeyi etkinleştirmek için denetleyicinin yanıtını beklerken BAD'da sorgulanır.

Benzer şekilde, bir veri bellek adresine erişmeden önce, talep edilen adresin çalışan program için doğru ve geçerli bir adres aralığında olup olmadığını doğrulamak için BTD ile sorgulama yapar.

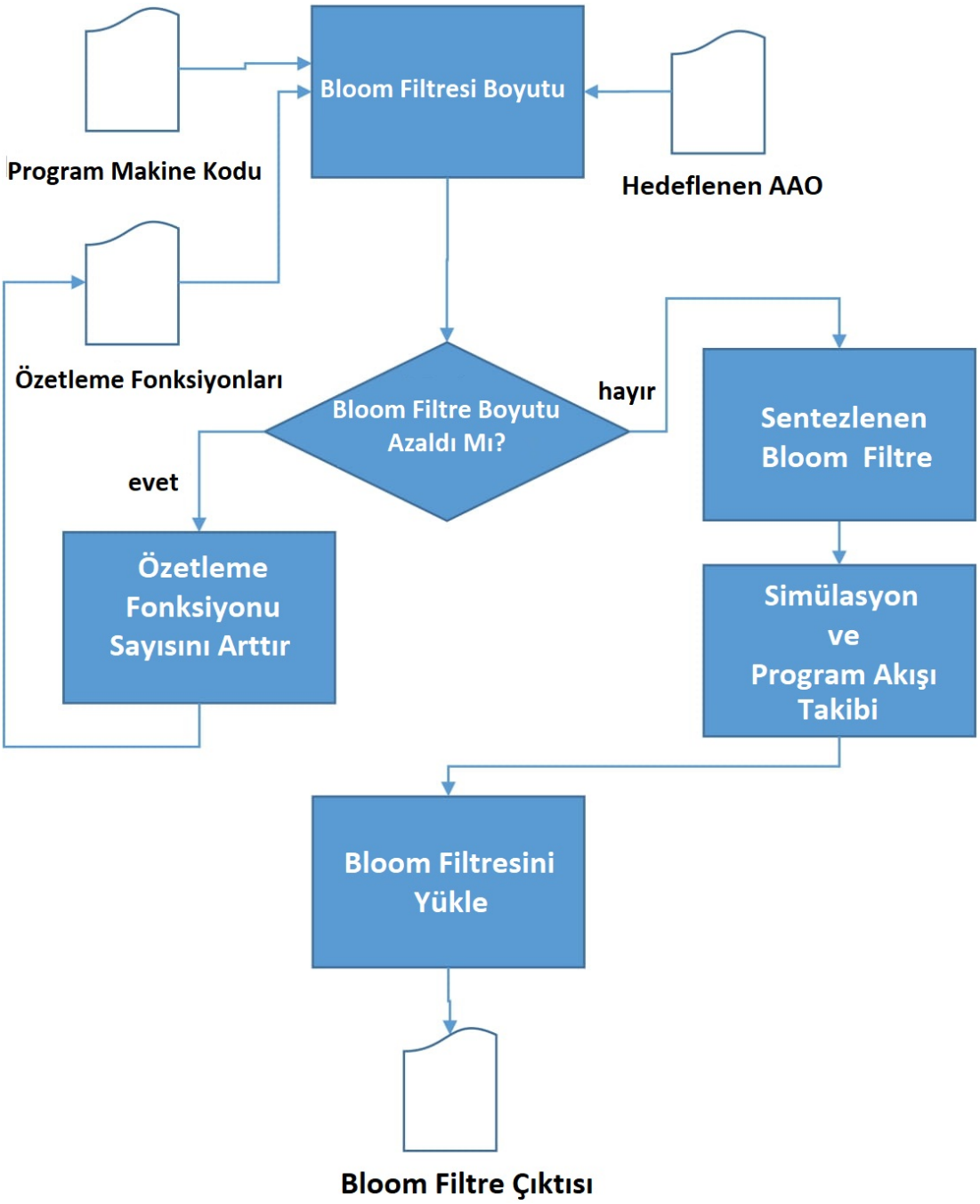
3.4.2 Donanım Truva atı tespit mekanizmasının tasarımı

Bit dizisinin boyutu ve kullanılan özetleme fonksiyonlarının sayısı açısından Bloom filtrelerinin en iyi konfigürasyonunu belirlemek ve bunları sisteme yerleştirmeden önce sentezlemek ve yüklemek için Şekil 3.3'de gösterilen tasarım akışını uyguladık. Akış, analiz edilen programın montaj (*-ing* assembly) kodunu, bir dizi özetleme (*-ing* hash) fonksiyonunu ve istenen maksimum kabul edilebilir algılanmayan alarm oranını (AAO) alır. Akışın sonunda, sentezlenen ve dağıtıma hazır olarak yüklenen Bloom filtresi uygun değişken değerlerinden oluşturulur.

Akışın ilk adımı, hedeflenen AAO'yu karşılamak için en iyi özetleme (*-ing* hash) fonksiyon sayısını ve Bloom filtresinin bit dizisinin boyutunu belirlemek ve test etmek için yapılan yinelemeli bir süreçten oluşur. Programın derlemesi, istenen AAO ve özetleme fonksiyonlarının başlangıç sayısı h göz önüne alındığında, gerekli bit dizisi boyutunu Denklem 2.1 aracılığıyla hesaplıyoruz. Kullanılan özet fonksiyonlarının sayısını artırarak gerekli bit dizisi boyutu, Denklem 2.2 tarafından tahmin edildiği gibi azalır. Bunu göz önünde bulundurarak ve özetleme fonksiyonlarının alan işgalinin hafızalarınkinden daha küçük olduğu varsayımına dayanarak (deneysel olarak Altbölüm 3.5.3'da doğrulanmıştır), bit dizisi kullanılırken kullanılan özetleme fonksiyonlarının sayısını artırmaya devam ediyoruz. Bunun sonucunda tasarlanan Bloom filtrenin bit genişliği belirli bir aralık değere kadar azalmaya devam eder. Boyuttaki değişimin azalmasındaki hız belirli bir değerin altına düştüğünde yinelemeli süreç durdurulur ve tanımlanmış bit dizisi boyutuna ve karma işlev sayısına sahip bir Bloom filtre sentezlenmesi gerçekleştirilir. Ardından, tüm adres çiftleri ve ilgili buyrukları kullanarak BAD için Bloom filtre eğitilir ve sisteme yüklemesi yapılır.

Benzer şekilde, BTD'nin Bloom filtresinin sentezlenmesi için, işlemci tarafından veri belleğine doğru verilen tüm bellek adreslerinin izi kullanılır. Bellek erişimi dinamik olduğundan ve doğası gereği değişeceğinden bellek adreslerinin değerlerine dikkat edilir ve veri içeriği kullanılmaz. Bu tür erişilen veri belleği adreslerini elde etmek için, veri belleği adres yolunda yayınlanan tüm adreslerin bir izini üretmek için analiz edilen işlemciyi Modelsim'de test ediyor ve bellek erişimini takip ediyoruz.

BAD ve BTD'nin Bloom filtreleri sentezlendikten, eğitildikten ve sisteme entegre edildikten sonra, koruma mimarisi tamamen somutlaştırılabilir ve korumalı mikroişlemci tabanlı sistem devreye alınabilir.



Şekil 3.3: Bloom filtresi için tasarım akışı

3.5 Deneysel Sonuçlar

Önerilen koruma mimarisinin etkinliğini ve verimliliğini değerlendirmek için bir dizi deney gerçekleştirdik. Burada, dikkate alınan donanım platformu ve kıyaslama programları hakkında ayrıntılar sunuyoruz; Bloom filtrelerinin uygulanmasını yönlendirmek için gerçekleştirilen simülasyon tabanlı analizden elde edilen sonuçları rapor ediyoruz ve nihayet donanım tarafından uygulanan denetleyicilerin doğruluğunu ve verimliliğini tartışıyoruz.

Çizelge 3.1: Deneylerde Kullanılan Sınama Programları ve Buyruk Sayıları

Program	Buyruk Sayısı
`Binary Sort (BinS)`	1719
`Matrix Multiplication (MM)`	1733
`Bubble Sort (BubS)`	1775
`Quick Sort (QS)`	1927
`Sudoku Solver (SS)`	3227
`Motion Detection (MD)`	4452

3.5.1 Deney ortamı

Tasarım akışının Bloom filtre boyutlandırma adımı bir C++ programı olarak uygulanmıştır. Bloom filtreleri sentezlemek için bir Xilinx Artix XC7A35T cihazını hedefleyen Xilinx Vivado'yu kullandık.

Bloom filtreler için en iyi parametreleri tanımladıktan sonra, denetleyicileri Harvard mimarisi sağlayan bir RISC-V çekirdeği çalıştıran FPGA tabanlı bir donanımda gerçekleme platformuna entegre ettik. Ağırlıklı olarak Nesnelerin İnterneti uygulamalarını hedefleyen düşük güç tüketimine sahip bir işleme platformu olan PULPINO mimarisini önerilen mimarinin gerçekleştirilmesi için kullanıldı.

4 aşamalı boru hattına sahip olan küçük bir RISC-V çekirdeği olan PULPINO'nun RI5CY [48] sürümüne entegrasyon yapuldı. İlgili çekirdek ile entegre edilen önerilen mimarideki denetleyici ve filtreler daha sonra çeşitli testlere ve analizlere sokuldu. Belirtilen işlemci çekirdeği RI5CY kendi başına Xilinx aracı üzerinde Artix XC7A35T FPGA'sı üzerinde sentezlendiğinde 14616 LUT, 8959 FF ve 16 BRAM gerektirir ve [49]'da bildirildiği gibi 50MHz'de çalışır.

Son olarak, basit sıralama algoritmalarından daha karmaşık Sudoku Çözücü ve Hareket Algılamaya kadar değişen bir dizi sınama programı ele aldık. Ayrıca kullanılan sınama programlarına ait Montaj (*-ing* assembly) buyrukları sayısıyla birlikte Çizelge 3.1'da bildirilmiştir. Burada farklı program boyutlarının kullanılması ile önerilen mimarinin farklı ölçeklerdeki performansını da gözlemlemek ve sistemin ölçeklenebilirliğini tartışmak hedeflenmiştir. Ayrıca yinelemeli programların da bulunması sayesinde sistemin tekrar eden program davranışı içerisindeki performansı da gösterilmeye çalışılmıştır. Programların hepsi standart RISC-V mimarisine uygun olarak derlenebilmekte ve çalıştırılabilmektedir.

3.5.2 Bloom filtresinin tasarlanması ve sentezlenmesi

Karşılık gelen montaj (*-ing* assembly) kodunu elde etmek için kıyaslama programları setini derledikten sonra, Alt Bölüm 3.4.1'da açıklanan akışı takip ederek Bloom filtreleri boyutlandırdık. Özellikle, algılanmayan alarm oranını (AAO) maksimum kabul edilebilir değerini %1 olarak sabitledik ve farklı sayıda kullanılan özetleme (*-ing* hash) fonksiyonu göz önüne alındığında benimsenecek minimum bit dizisi boyutunu hesapladık. Çizelge 3.2 ve çizelge 3.3, sırasıyla BAD ve BTD için 3'e

kadar 7 özetleme işleviyle bu deney hakkında bilgi verir. Her iki deneyde de bit dizisi boyutunun 3 ile 4 özetleme fonksiyonu arasında ve yine 4 ile 5 arasında önemli ölçüde azaldığı, buna karşın özetleme fonksiyonlarının sayısının daha fazla arttırılmasının bit dizisi boyutunda önemli bir azalma getirmediği fark edilebilir. Bu nedenle, gerçek algılama doğruluğunu ve eklenen ek yükü değerlendirmek için 5 özet işlevine sahip Bloom filtreleri uygulamaya dahil edildi.

Çizelge 3.2: Farklı miktarda kullanılan Özetleme Fonksiyonu sayısı için %1 AAO'da BAD'a ait Bloom Filtrenin bit dizisinin boyutu (bit cinsinden)

Program	#Özetleme Fonksiyonu				
	3	4	5	6	7
BinS	41200	24720	17512	18320	17064
MM	42816	23232	17848	18120	17848
BubS	42544	27360	19584	19304	18128
QS	46184	30488	20544	20208	19624
SS	65296	51056	33140	31648	32432
MD	68760	53552	44080	43080	42272

Çizelge 3.3: Farklı miktarda kullanılan Özetleme Fonksiyonu sayısı için %1 AAO'da BTD'a ait Bloom Filtrenin bit dizisinin boyutu (bit cinsinden)

Program	#Özetleme Fonksiyonu				
	3	4	5	6	7
BinS	1080	1016	944	816	752
MM	1192	1072	912	976	880
BubS	1336	1048	984	808	712
QS	1368	1136	1016	824	720
SS	3712	3080	2552	2864	2576
MM	5416	4912	4664	4992	4520

3.5.3 Bloom filtrelerinin donanım uygulaması ve değerlendirmesi

Önceki alt bölümde tartışılan araştırmaya dayanarak, Bloom filtreleri, 5 özetleme fonksiyonuna sahip dikkate alınan sınama programları için hem BAD hem de BTD için bir FPGA üzerinde uyguladık ve aşağıdaki boyutlara en yakın iki değerin karşılığı olan bir bellek boyutu seçtik. Burada uygulanan Bloom filterlerinin bit genişliği ise sırasıyla Çizelge 3.2 ve 3.3 içinde hesaplanır.

Her şeyden önce, uygulanan kontrolcüler ile elde edilen gerçek AAO'yu ölçtük. Bunu yapmak için, buyruk ve veri belleklerinden gelen verileri değiştirerek, dikkate alınan modellere ait bir DTA'nın aktivasyonunu taklit ettik: i) işlemcinin orijinal programda olmayan bir buyruğu yürütmesini sağlamak ii) işlemcinin orijinal programda bulunan ancak beklenmedik bir buyruk bellek adresinden getirilen bir buyruk yürütmesini sağlamak ve iii) işlemcinin beklenmedik bir veri bellek adresinden veri okumasını/yazmasını sağlamak. 100.000 rastgele oluşturulmuş DTA aktivasyon

Çizelge 3.4: BAD ve BTD için Bloom filtrelerin donanımda uygulanmasının boyutu (Kbit cinsinden) ve buna karşılık elde edilen AAO değeri (tespit edilemeyen DTA etkinliği oranının yüzdesi cinsinden)

Program	BAD		BTD	
	Bellek Boyutu	AAO (%)	Bellek Boyutu	AAO (%)
BinS	32	0.523	1	0.085
MM	32	0.520	1	0.122
BubS	32	0.572	1	0.525
QS	32	0.607	1	0.073
SD	64	0.249	4	0.134
MD	64	0.912	8	0.232

vakasını simüle ettik ve verilmeyen alarmların sayısını ölçtük. Bu deneyden elde edilen sonuçlar Çizelge 3.4'da rapor edilmiştir. Beklendiği gibi, ölçülen UAR her zaman %1'in altındadır (çoğu durumda %1'in çok altındadır). Ardından, sınama programlarını herhangi bir değişiklik yapmadan çalıştırarak, uygulanan Bloom filtrelerinin herhangi bir yanlış alarm vermediğini de kontrol ettik.

Son olarak, bir FPGA uygulamasını hedeflerken kullanılan kaynaklar ve çalışma sıklığı açısından önerilen denetleyicilerin genel etkisini değerlendirdik.

Çizelge 3.5 ve çizelge 3.6, dikkate alınan sınama programları için sırasıyla BAD ve BTD için ayrıntıları rapor eder. Kaynak kullanımı hem mutlak değerler hem de kullanılan platformdaki sistem kaynaklarına oranının yüzdesi olarak rapor edilir. Belirtilen çalışma için RI5CY işlemci çekirdeğinin uygulanan mimarinin kaynak kullanımına oranı verilmiştir. Sonuçlara göre LUT ek yükü %0,68 ile %10,52 arasında değişirken, FF ek yükü %0,68 ile %0,99 arasında değişir. Önerilen mimarinin işlemciyi çok çeşitli DTA modellerinden ve son derece küçük bir algılanmayan alarm yüzdesiyle koruyacağı düşünüldüğünde, bu tür genel giderlerin tamamen kabul edilebilir olduğuna inanıyoruz. Kaynak yüküyle ilgili bir başka ilginç veri, LUT kullanımının geniş varyasyonunun, seçilen uygulamanın LUT'u Bloom filtrenin bellek öğeleri için de kullanması ve dolayısıyla yük artışını optimize etmek için 1 Kbit ila 64 Kbit diziler uygulamak için kullanması gerçeğini açıklıyor olmasıdır. Çalışma frekansına bakıldığında, maksimum çalışma frekansı (en kötü durumda 2x) dikkate alınan RI5CY çekirdeğinden çok daha yüksek olduğu için önerilen denetleyicilerin herhangi bir yavaşlama ve gecikme getirmediği gözlemlenebilir.

3.5.4 Güvenlik analizi ve değerlendirmesi

Sunulan deneysel sonuçlar, önerilen koruma mimarisinin, işlemciye sıfır yanlış alarmla kötü niyetli kod yürütmeye zorlamaya çalışan DTA'ların çalışma zamanı aktivasyonlarının %99'undan çok daha fazlasını algılayabildiğini göstermektedir. Tasarımcının algılanmayan alarm oranını (AAO) daha da azaltmak istemesi durumunda, BF'nin bit dizisinin boyutunu artırmak, yine de sıfır yanlış alarma sahip olmak yeterlidir.

Çizelge 3.5: Uygulanan BAD'ların kaynak işgali ve çalışma sıklığı

Program	Buyruk Akış Denetleyicisi		
	LUT Sayısı	FF Sayısı	Frekans (MHz)
BinS	880 (%6,02)	84 (%0,93)	112,19
MM	880 (%6,02)	84 (%0,93)	112,19
BubS	880 (%6,02)	84 (%0,93)	112,19
QS	880 (%6,02)	84 (%0,93)	112,19
SS	1539 (%10,52)	89 (%0,99)	106,37
MD	1539 (%10,52)	89 (%0,99)	106,37

Çizelge 3.6: Uygulanan BTD'lerin kaynak işgali ve çalışma sıklığı

Program	Bellek Tahsisi Denetleyicisi		
	LUT Sayısı	FF sayısı	Frekans (MHz)
BinS	100 (%0,68)	61 (%0,68)	181,91
MM	100 (%0,68)	61 (%0,68)	181,91
BubS	100 (%0,68)	61 (%0,68)	181,91
QS	100 (%0,68)	61 (%0,68)	181,91
SS	170 (%1,16)	71 (%0,79)	154,13
MD	275 (%1,88)	76 (%0,84)	143,67

Daha ayrıntılı olarak, DTA'nın yürütmesi amaçlanan geçerli programda olmayan veya programda bulunan ancak dorğu sırada veya adresde bulunmayan, programın bellek alanından ve bellek konumlarından yüklenen buyruk yürütmesini sağlamaya çalışan herhangi bir DTA'nın . Ayrıca, işlemcinin yetkisiz veri bellek adreslerinden veri okumasını/yazmasını sağlayan DTA'lar da her zaman algılanır. Önerilen çözümün etkinliğinin, DTA'nın tetikleme mekanizmasından, yani kombinasyonel/sıralı tetikleme mekanizmaları, harici olarak etkinleştirme mekanizmaları, saatli tetiklenen mekanizmalar ve her zaman açık mekanizmalardan bağımsız olduğunu ve işlemcinin akışına dahil olduğu için tüm tetik türlerine karşı etkilini olduğunu da belirtmek gerekir.

Özetle, işlemci tabanlı sistemlerin belleklerindeki donanım truva atlarının (DTA) çalışma zamanı etkinliklerinin tespiti için bir koruma mimarisi (ve eşlik eden tasarım akışı) sunulmuştur. Önerilen mimari, sıfır yanlış alarma sahipken, istenen maksimum kabul edilebilir algılanmamış alarm oranını (AAO) elde etmek için ince ayar yapılabilir. Önerilen işlemci koruma mimarisini, bir FPGA cihazında uygulanan ve bir dizi yazılım kıyaslaması çalıştıran bir RISC-V işlemcisine dayalı bir sınama testine ve sistemine uyguladık. Mimari her zaman olası DTA etkinliklerinin %99'undan fazlasını sıfır yanlış alarmla tespit edebildi. Çalışma frekansında herhangi bir azalma olmaksızın, %0,68 ile %10,52 arasında değişen bir arama tablosu ek yükü (LUT) ve %0,68 ile %0,99 arasında bir flip-flop (FF) ek yükü ölçtük.

4. ERIC

4.1 Motivasyon

Nesnelerin interneti (*-ing* Internet of things) bağlamında elektronik cihazların kullanımının artmasıyla birlikte, giderek daha fazla cihaz birbirleriyle ağlar üzerinden iletişim kuruyor [50, 51, 52, 53]. Bir ağ üzerinden birden fazla cihaz arasında doğrulanmış ve güvenilir iletişim kanallarının sürdürülmesi, çok çeşitli uygulamalar için güvenlik ve güvenilirlik garantileri sağlamak için önemlidir [54, 55].

Gömülü sistemler tipik olarak bir programı yürüten işlemcilerden veya işlem birimlerinden oluşur. Bu sistemler genellikle belirli bir amaca hizmet eder ve performans, güç bütçesi ve hesaplama kapasitesi açısından genel amaçlı sistemler kadar yetenekli değildir [56], [57]. Gömülü sistemler tarafından yürütülen programlar genellikle daha yetenekli bilgi işlem sistemlerinde (örneğin kişisel bilgisayarlar ve bulut sistemleri) derlenir ve derlenmiş ikili dosyaları gömülü sistemlere fiziksel arabirimler (örneğin seri bağlantı) veya bir ağ üzerinden teslim edilir. [58], [59], [60], [58]. Tipik olarak, bu ikili dosyalar kritik bilgileri (ör. hedef donanımın mimari ayrıntıları ve karmaşık, ticari sır algoritma uygulamaları) [61], [62] örtük olarak kapsar. Bu bilgilerin kötü niyetli taraflara maruz kalmaması önemlidir.

Derlenmiş programın olduğu ikilik tabanlı gösterilen program dosyalarına gömülü kritik bilgilerin gizliliğini tehdit eden iki tür saldırı tanımlanabilir. İlk olarak, bir ikili yani makine kodu halinde program verileri, standart derleyici araçları (örneğin, çözücü (*-ing* decoder)) kullanılarak insan tarafından okunabilir bir forma dönüştürülebilir ve kritik bilgileri belirlemek için analiz edilebilir [63], [64], [65], [66], [67]. Bu tür saldırılara statik analiz saldırıları diyoruz. İkinci olarak, kötü niyetli taraflarca kontrol edilen bir bilgisayarda belirtilen programın makine kodu yürütülebilir ve bilgisayarın durumu (örneğin, performans sayaçları, kayıt değerleri) yürütülen program üzerinden kaynak kodunu tersine mühendislik yapmak için izlenebilir. Bu tür saldırılara dinamik analiz saldırıları [68], [69], [70], [13], [71] diyoruz. Belirtilen çalışmada amacımız hem statik hem de dinamik analiz saldırılarını önleyebilecek yeni bir çerçeve tasarlamaktır ve bu çerçevenin başarımını göstermektir.

Programın kaynağından hedef donanım platformuna nasıl aktarıldığına bakılmaksızın program bilgilerini gizli tutan yeni bir çerçeve olan ERIC'i öneriyoruz. Bunu yapmak için ERIC, program ikili (*-ing* binary) dosyalarını şifrelemek için hedef donanım aygıtına özgü bir tanımlayıcıdan (örneğin, fiziksel klonlanamayan işlevler (*-ing* Physically Unclonable Function) oluşturulan şifreleme anahtarlarını kullanır. Şifrelenmiş programın şifresi yalnızca hedef donanım cihazı tarafından çözülebilir ve kötü niyetli tarafların statik ve dinamik analizler yapmasını engeller.

ERIC iki bileşenden oluşur: Birincisi, yeni bir derleyici yazılım yürütülebilir dosyalarının (yani program ikili dosyasının) kısmi ve tam şifrelemesini destekler. İkincisi, bir donanım şifre çözme motoru, şifrelenmiş ikili dosyaların verimli bir şekilde şifresinin çözülmesini sağlar. ERIC, program ikili dosyalarının simetrik şifrelemesinde ve şifresinin çözülmesinde kullanılan kriptografik anahtarları oluşturmak için fiziksel klonlanamayan işlevleri (PUF'lar) kullanır. Bu şekilde ERIC, şifrelenmiş bir ikili programın yalnızca hedef donanım tarafından şifresinin

çözülebileceğini ve yürütülebileceğini garanti eder.

ERIC'in FPGA tabanlı prototipini geliştiriyor ve ERIC'i uçtan uca değerlendirmek için gerçek bir RISC-V sistemine entegre ediyoruz. RISC-V ikili dosyalarında şifreleme gerçekleştirmek için LLVM'yi genişleterek ERIC'in derleyicisini uyguluyoruz. Prototipimiz açık kaynak projeleri üzerine kuruludur ve kendisi `https://github.com/kasirgalabs/ERIC` adresinde açık kaynaklı olarak paylaşılmıştır. Açık kaynak prototipimizin ileride endüstri ve araştırmacılar için faydalı olacağını umuyoruz.

4.2 İlgili Çalışmalar

Bildiğimiz kadarıyla, ERIC, yazılım üzerinde şifreli derlemeden donanım üzerinde güvenilir yürütmeye kadar tamamen uçtan uca ilk yazılım gizleme ve güvenilir yürütme çerçevesidir. ERIC, programların derleyiciyi gönderirken güvenli bir şekilde paketlenmesini ve orijinal programda herhangi bir değişiklik olmaksızın yalnızca hedef donanımda yürütülmesini sağlar.

[72], [73]'da, bellek güvenliğini ve program kimlik doğrulamasını sağlamak için, tüm bellek mesaj kimlik doğrulama koduyla şifrelenir. Bu çalışmalarda bellekteki her satır AES şifrelemesi ile korunmaktadır. Bu çalışmanın yetersizliği, bellekteki her satır için AES şifrelemesi kullanması ve ana bellek için güvenli bir alan oluşturamamasıdır. Ayrıca AES'in getirdiği yüksek bellek gecikmesi bulunur. Önbellek performansı düşük olan programlar, ana belleğe erişmeye çalışırken her seferinde ekstra bir gecikme yaşar.

[74], [75], [76]'de, donanım ve yazılıma dayalı saldırıları önlemek için şifreleme tabanlı mekanizma önerilmiştir. PUF tabanlı şifreleme gerçekleştirir ve bellekte farklı seviyelerden oluşan bir güvenlik ağacı kullanır. Farklı bellek koruma düzeylerini yönetmek için güvenli bir işletim sistemi içeren mimarileri için güvenli yazılım geliştirmek için bir araç zinciri sunarlar. Ancak, önerilen mimari belirli bir SoC için oluşturulmuştur ve tüm işlemciler veya buyruk mimarileri ile uyumlu olamaz. Bununla beraber işletim sistemindeki değişiklikler nedeniyle sistemin uygulanabilirliği zorlaşmaktadır. Ayrıca, temel alınan çalışmalara göre çevrim başına buyruk performansında ortalama %30 yavaşlamaya neden olur.

Başka bir çalışma, donanımda yalıtılmış bir buyruk seti eklentisine sahip bir güvenlik katmanı önerilir. [77]'de şifreleme, PUF anahtarları ile yapılır. Hafızanın ve programın güvenliği bu anahtar ile şifrelenerek sağlanmaktadır. Donanım üreticisi ile yazılım üreticisi arasındaki protokole göre oluşturulmuş sistemleri destekler. Fakat buyruk mimarisinde önerilen değişiklikler, genel derleme araçlarının kullanılamaz hale gelmesine neden olur ve sistemin ölçeklenebilirliğini azaltır. Bu çalışma, bizim çalışmamızdan farklı olarak programı şifrelememektedir. Bunun yerine önbellekleri, ana bellek gibi çalıştıkları bir mimariye sahiptir ve programa ait iş parçacıklarının etkin sayfalar ön belleği performansına odaklanır. Buna karşılık, önerdiğimiz mimari, buyrukları işlemci çekirdeğine getirmeden şifresini çözdüğü için tüm işlemcilere ve mimarilere uygulanabilir. Yürütme sürecini doğrudan etkilemez ve standart buyruk mimarilerini desteklediği için tüm derleyiciler ve sistemler için yapılandırılabilir.

Öte yandan, önerilen güvenilir yürütme ortamı çalışmaları da vardır [78], [79]. Esas olarak, bu çalışmalar, işlem birimi içindeki yürütmeyi izole etmeye yönelik

yaklaşımlardır. Ancak, ERIC'in amacı, programın doğru yazılım kaynağından gelmesini ve doğru hedef donanıma ulaşmasını sağlarken, güvenilmeyen bir ağ ortamında geçiş sırasında programı kötü niyetli eylemlerden ve herhangi bir değişiklikten korumaktır. Ayrıca ERIC, işlem biriminin mimarisine müdahale önermez ve yürütmeye direk dahil olmaz.

Programın işlemci üzerinde güvenliği ve güvenilir programın çalıştırılması için de önceden yapılmış çalışmalar da vardır [80], [81], [82], [83], [84]. Önerilen ERIC mimarisi ise şu özellikleriyle öne çıkar: (i) tutarlı bir uçtan uca mimari sağlar ve her ikisi için düşük ek yük ile donanım değişiklikleri için derleyici desteği sunar, (ii) işlemci mikro mimarisinde değişiklik gerektirmez, bu nedenle pratikte tüm mimari ve sistemlere uygulanabilir, (iii) işlemci tasarımına değil, SoC'ye dahil olduğu için sunucular ve ağlar gibi ölçeklenebilir sistemlere uç birimler olarak eklenebilir, (iv) sadece donanımın değil aynı zamanda yazılım kaynağının da kimliğini doğrulayabilir (v)) yeterli kaynağa sahip bir sistemde önbellek ve etkin sayfalar ön belleği performansını doğrudan etkilemez.

4.3 Tehdit Modeli

Tehdit modelimizde, yürütülebilir dosyanın (program ikili dosyaları) güvenilmeyen bir ağ üzerinden iletildiğini varsayıyoruz. Kötü niyetli taraflar, sistem tasarım haklarını ihlal etmek için yürütülebilir dosyayı alabilir, yürütülebilir dosyada değişiklik yapabilir ve değiştirilmiş sürümü hedef donanıma gönderebilir. Hedef donanımın güvenilir olduğunu varsayıyoruz. Özetle, şu tür tehditlere karşı koruma sağlıyoruz: (i) Kritik programları ele geçirmek ve bunun sonucunda tersine mühendislik uygulamaları yapmak, (ii) kullanıcı donanımında bilinmeyen kaynaklı programları çalıştırmaya çalışmak, (iii) yazılım kaynağı tarafından derlenen programları lisanssız veya doğrulanmamış donanım ve (iv) sistemdeki programda kötü niyetli değişikliklerin veya yazılım hatalarının yürütülmesini sağlamak.

Temelde belirtilen tehdit modellerinin hepsi aynı zamanda donanım içerisinde DTA etkinliği ile elde edilmeye çalışılan ayrıcalıklarla eşleşmektedir. Bu nedenle belirtilen çalışma aynı zamanda sistem içerisinde bulunabilecek DTA kaynaklı veri hırsızlığına veya emin olunmayan kaynaklardan gelen program dosyalarının yürütülmesine karşı da önlem alınmasına yardımcı olur. Yazılım kaynağı ve hedef yürütücü donanım arasında sağlanabilecek iki taraflı doğrulama, DTA aracılığıyla sisteme yapılabilecek müdahalelere karşı ekstra bir güvenlik ve koruma sağlayacaktır.

4.4 Önerilen Yöntem

ERIC, kimliği doğrulanmış hedef donanıma özel yazılım derleme yeteneği sağlayan güvenilir bir yürütme ortamı oluşturmak için tasarlanmış verimli ve pratik bir mimari tasarım çerçevesidir. Böyle bir yeteneği sağlamak için, ilk olarak, ERIC, yazılımı, şifrelenmiş yazılımın yalnızca hedef donanımda şifresi çözülebilecek şekilde şifreler, yazılımı statik ve dinamik analizler biçimindeki kötü niyetli saldırılara karşı korur, ikincisi, ERIC, yazılımın güvenli bir şekilde paketleyerek ve saklayarak hedef donanıma uygun çalışacak şekilde sisteme ve ağ trafiğine entegre eder. Bu

entegrasyonun sağlanması ve yazılımın hedef donanım tarafından doğrulanabilmesini desteklemek için yazılımın şifreli sürümü içinde imza çıktısı ve çeşitli şifreleme haritalandırmaları da programın kendisi ile birlikte şifreli olarak paketlenir. ERIC yapısı gereği uçtan uca çoklu ortam mimari tasarımıdır. Bu nedenle yazılım seviyesinde derleme aşamasında başlayan mimari tasarım, boru hattı veya yürütme birimine program ikili dosyaları teslim edilene kadar devam eder. Mimarinin anlaşılabilmesini kolaylaştırmak için ERIC'i iki bileşen üzerinden tanımlamaktayız. Bunlar: (i) donanım tabanlı mimari ve (ii) yazılım tabanlı mimari.

ERIC'in donanım ve yazılım bileşenleri kolayca ayrılabilir. Örneğin, ERIC'in yazılım bileşenleri, programları derlemek ve şifrelemek için bir bilgisayarda kullanılabilir ve donanım bileşenleri, şifre çözme işlemini gerçekleştirmek için başka bir bilgisayarda kullanılabilir. Bu şekilde, şifrelenmiş bir program bir ağ üzerinden bir bilgisayardan diğerine güvenli bir şekilde aktarılabilir ve bütünlüğü, şifresini çözen ve çalıştıran donanım tarafından doğrulanabilir. Şifrelenmiş bir programın bütünlüğü doğrulanabiliyorsa, güvenilir bir kaynaktan geldiği garanti edilir, böylece şifrelenmiş programın orijinalliği de doğrulanır. Bu sayede program sadece hedef donanım üzerinde çalışır ve hedef donanım sadece kendisine yazılan programları çalıştırır. ERIC'in bu özelliğine *iki yönlü kimlik doğrulama* adını veriyoruz. Şekil 4.1, donanım ve yazılım arayüzleri arasında iki yönlü kimlik doğrulamanın nasıl çalıştığını gösterir.

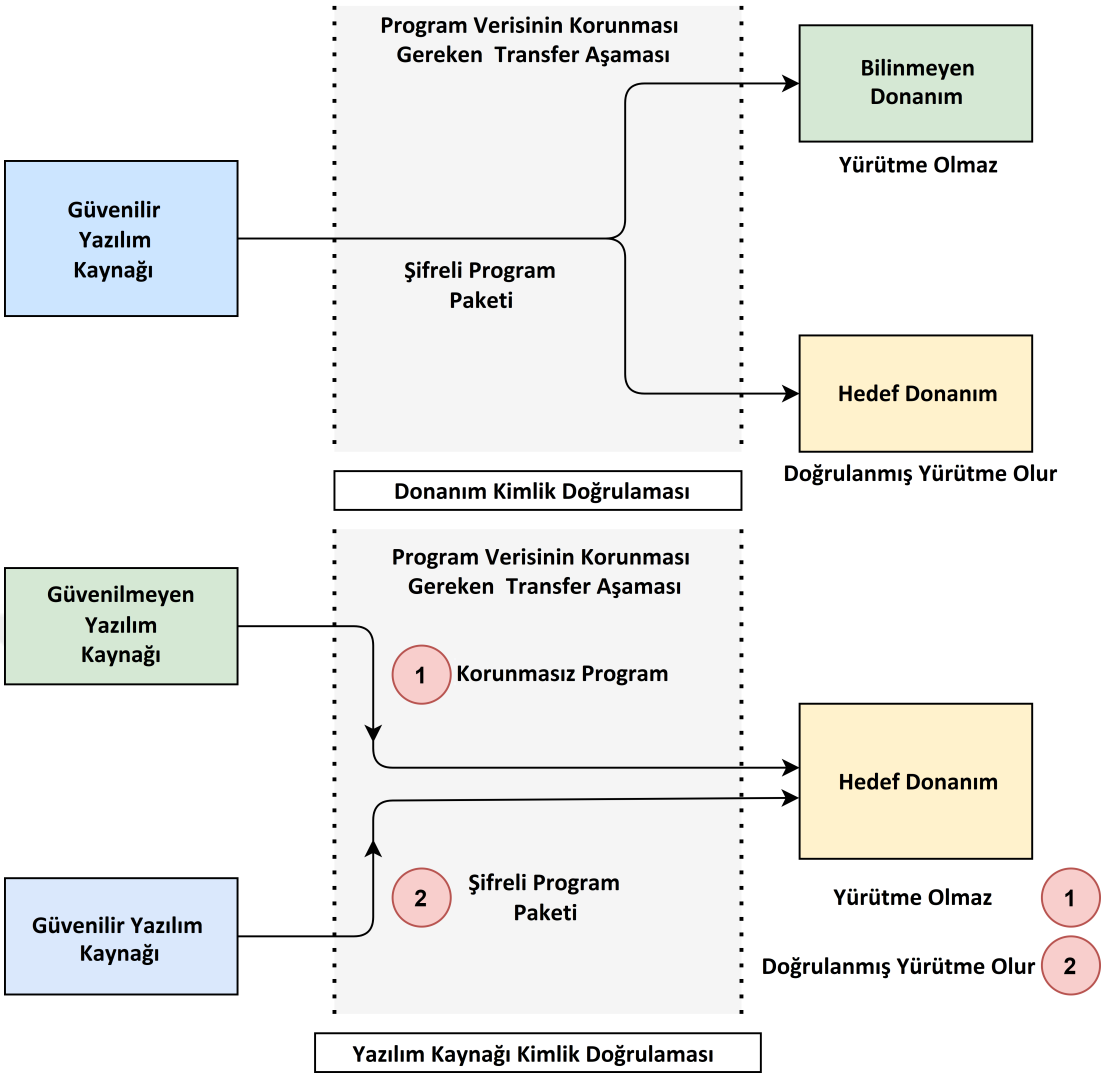
Öte yandan, ERIC, yazılım kaynağı ile hedef donanım arasında aktarım yaparken, program ikili (*-ing* binary) dosyalarının kötü amaçlı etkinliklerden gizlenmesini sağlar.

4.4.1 Yazılım mimarisi

Hedefe yönelik donanıma özel şifreleme, PUF tabanlı bir anahtar tarafından sağlanır. Bunun için derleyici donanım tarafında PUF tabanlı Anahtar Üretici Birimi ile entegre şifreleme gerçekleştirir. PUF tabanlı anahtarlar, donanımdaki PUF anahtarının Anahtar Yönetim Birimi'ndeki işlev (örneğin, güvenli karma algoritması [85]) üzerinden geçirilmesiyle elde edilir. PUF anahtarı her hedef donanımda değişmez olduğundan, PUF anahtarına doğrudan bağlı bir şifreleme mekanizması, güvenilir program kaynağına sınırsız erişim sağlar ve bu erişim ilişkisi değiştirilemez. Bunu önlemek için, hedef donanımın herhangi bir zamanda yapılandırabileceği PUF anahtarından PUF tabanlı bir anahtar alınması önerilir. PUF anahtarlarının kullanımı, fiziksel temelleri olan ve sürekli olarak bir kayıt defterinde saklanması gerekmeyen her cihaz için benzersiz bir anahtar oluşturulmasını sağlar.

Bölüm 4.3'te belirtildiği gibi, önerilen mimari, derlenen programın yürütme aşamasına kadar gizlenmesini ve güvenliğini amaçlarken, hedef donanım ve yazılım kaynağı arasında iki taraflı kimlik doğrulaması sağlar. Bu nedenle, yazılım kaynağı tarafından hedeflenen donanım için el sıkışmanın (iki tarafça bilinen iletişim örüntüsünün taraflarca gerçekleştirilmesi) zaten yapıldığı ve hedef donanımla uyumlu PUF tabanlı anahtarların yazılım kaynağı tarafından bilindiği varsayılır.

Derleyici tarafında eklediğimiz anahtar birimi hedef donanımın PUF tabanlı anahtarını girdi olarak alır ve şifreleme işlemlerinde kullanılacak anahtarları üretir. PUF anahtarı, yazılım derlemesi sırasında kullanıcıya gösterilmez. Donanımdaki PUF tabanlı anahtar



Şekil 4.1: İki Taraflı Kimlik Doğrulama Akışı

üreteci birim ile yazılımdaki anahtar üreteci arasında oluşturulan soyutlama katmanı ile, PUF anahtarı ile entegre şifreleme yapıldığında PUF anahtarına doğrudan erişime gerek yoktur. Bu, sistemde farklı anahtar yapılandırmaları sağlayarak uzun süreli anahtar kullanımına izin verir. Ayrıca bu sayede donanımdaki PUF anahtarı korunur ve yazılım geliştiricisi ile doğrudan paylaşılmadığı için farklı hedef donanımlar için kullanılabilir. Bu sayede sistem güvenilir bir yazılımcı kaynağından birden fazla hedeflenen donanıma ölçeklenebilir veya bir hedef donanım birden fazla güvenilir yazılım kaynağından gelen kodu yürütmesi için gerçeklenebilir.

ERIC'in yazılım bileşenleri, derleme aşamasında programı çeşitli konfigürasyonlarda şifreler. Şifrelenmiş programın, hedef donanıma ulaşana kadar güvende kalması amaçlanmıştır. Şifrelemenin temel amacı, program aktarılırken veya saklanırken tersine mühendislik yöntemlerinden, programa uygulanabilecek kötü niyetli eklentilerden veya yazılım hatalarından dolayı oluşabilecek bit çevirmelerinden korunmaktır. ERIC, şifreleme aşamalarını daha kolay anlamak ve programcının şifrelemeyi kendi ihtiyaçlarına göre kontrol etmesini sağlamak için bir grafik arayüz içerir. Bu arayüz üzerinden ihtiyaca göre şifreleme yapılabilmektedir.

Önerilen mimari, farklı şifreleme yöntemleriyle uyumludur. Yeni şifreleme algoritmaları, ERIC'in arayüzü ile sistemde kolaylıkla uygulanabilir. Hedef dosyalarda yapılacak değişiklikler ile kullanıcı kendi şifreleme yöntemini sisteme yükleme özgürlüğüne sahiptir. Şifreleme algoritmasına karar verildikten sonra şifreleme yöntemleri belirlenmelidir. Bunun için kullanılabilecek üç farklı şifreleme yöntemi vardır. Bunlar programın tam şifrelemesi, programın kısmi şifrelemesi ve buyruğa özel hedef bitleri belirterek programın seçilmiş buyruklarıyla kısmi şifrelenmesidir.

Tüm program şifreleme konfigürasyonuna bağlı olarak PUF tabanlı anahtarlarla şifrelenir ve hedef donanımda yürütülmeye hazır hale getirilir. Hedef donanımın hangi buyrukların şifrelendiğini algılaması için şifreleme haritasının şifrelenmiş programla birlikte karşı tarafa iletilmesi gerekir. Bu bilgi paketi de belirlenen protokole bağlı olarak şifrelenir. Kısmi şifreleme kullanarak, programcı programın kritik kısımlarını koruyabilir veya program içinde yalnızca hedef donanımda etkin olabilecek bir alan oluşturabilir. Ayrıca programcı, program içerisinde sadece lisanslı donanımlarda çalıştırmak istediği özellikleri seçebilir.

Kısmi şifreleme için hedef buyrukların seçilebileceği arayüz ERIC ile sağlanır. Sunulan arayüz aynı zamanda hedef buyruklar içinde özel parçaların seçilmesine de izin verir. Bu sayede program akışına müdahale edilmeden sadece kritik bilgiler korunabilmektedir. Örneğin, yalnızca bellek erişimlerini sağlayan buyrukların işaretçi değerleri şifrelenebilir, bu da programın bellek izini takip etmeyi zorlaştırır. Kısmi şifreleme sırasında talimatların işlem kodu (*-ing* opcode) kısımlarının şifrelenmemesi, tersine mühendislik durumunda programın şifreli olduğunu anlamayı da zorlaştıracaktır.

ERIC, birden çok hedef donanım için tek bir yazılım kaynağından derleme yapmak veya tek hedef donanım için birden çok güvenilir yazılım kaynağı oluşturmak için uygundur. Bunun için PUF tabanlı anahtarı PUF anahtarları ile eşleştirmek için sisteme yalnızca uygun dönüştürme işlevini uygulamak gerekir. Ayrıca donanım üreticisi, Anahtar Yönetim Birimi'nde dönüştürme işlevini gerçekleştirirken aynı PUF tabanlı anahtara iki veya daha fazla farklı donanımı eşlerse, tek bir derleme adımı ile birden fazla donanım üzerinde çalışacak güvenli paketlenmiş programlar oluşturulabilir. Bu, ERIC'in birden fazla hedef veya kaynak için bir ölçekleme sorunu olmadığını gösterir.

Şifrelemeye ek olarak, ERIC, programın hedefe değişiklik yapmadan ulaşmasını sağlamak için derlenmiş programlar üzerinden şifrelemeden önce imzalar üretir. Bu imza, program şifrelenmeden önce talimatlar üzerinde bir kriptografik özetleme fonksiyonu çalıştırılarak elde edilir. İmza, program şifrelenmeden önce üretilir ve imza programla şifrelenir, bu da programın şifresini çözemeyenler için imzayı işe yaramaz hale getirir. Ortaya çıkan imza, sonunda PUF tabanlı bir anahtarla şifrelenir. Bu sayede programın aktarılması veya saklanması sırasında programda oluşabilecek parazit veya hatalar donanım tarafından algılanabilir hale gelir. İmza oluşturulması sayesinde program sadece saldırganlardan değil, programın taşınması sırasında oluşabilecek hataların fark edilmesine katkı sağlar.

ERIC'in yazılım tarafında yapılan geliştirmeler, hedeflenen kişiselleştirme için tamamen uygundur. Şifreleme yöntemlerinin çeşitliliği, şifreleme işlevini değiştirebilme yeteneği ve hedef donanıma uygun ISA spesifikasyonlarının seçimi bunu mümkün kılar.

4.4.2 Donanım mimarisi

Donanımın şifreli ve imzalı programı güvenli ve verimli bir şekilde çalıştırabilmesi için yazılım mimarisi ile uyumlu çalışacak donanım mimarisine ihtiyaç vardır. ERIC'in Donanım mimarisi birkaç birim içerir. Bunlar Çözme Birimi, Doğrulama Birimi, Anahtar Yönetme Birimi, İmza Üretme Birimi ve PUF Anahtarı Üreticisidir. Bu birimlerin entegre edildiği üst düzey birime Donanım Şifre Çözme Motoru denir. İşlemci, Donanım Şifre Çözme Motoru ile sistemde bulunan işlemci, bellekler gibi diğer birimler entegre edilerek donanım mimarisi tamamlanır. Donanımdaki güvenlik modeli, alınan programların yürütülmek üzere ana belleğe yüklenene kadar şifreli tutulmasını sağlar. Şifre çözme işlemleri program belleğe yazılmadan yapıldığından önerilen donanım mimarisi ortak işlemci mimarileri ile uyumludur.

PUF Anahtarı Üretici. Bu birim üretim sırasında donanımdaki farklılıklar (örneğin, süreç varyasyonu) nedeniyle donanım aygıtı için bir kimlik görevi gören anahtarların oluşturulmasını sağlar. Bu anahtarlar, donanıma benzersiz bir kimlik numarası sağlar. Bu PUF anahtarları, iki farklı donanımı ayırt etmek için ERIC'de kullanılacaktır.

Anahtar Yönetim Birimi. Donanım tarafından kullanılan PUF tabanlı anahtarlar, gelen şifreli programın şifresini çözmek için kullanılır. Yazılım tarafında kullanılan anahtarlar ile donanım tarafında kullanılan anahtarların entegrasyonu için mevcut PUF anahtarı Anahtar Yönetim Birimi içerisinde anahtar oluşturma fonksiyonundan geçer. PUF tabanlı anahtar, PUF anahtarının işlevden geçirilmesiyle elde edilir. Bu şekilde, tek bir PUF anahtarıyla birden çok PUF tabanlı anahtar oluşturulur. Daha önce belirtildiği gibi, bu, şifreleme/şifre çözme anahtarı ile PUF anahtarı arasında bir soyutlama katmanı sağlar. PUF anahtarının soyutlama ile kullanılması, Anahtar Yönetim Birimi'nde PUF tabanlı anahtar için kullanılan işlevin yapılandırılmasına ve uyumlu yazılım kaynaklarının zamana veya tercihlere göre değiştirilmesine olanak tanır. Bu esneklikle, donanımdaki gerekli değişkenler PUF tabanlı anahtar oluşturma işlevine girdi olarak verilirse, yalnızca şifresi çözülebilen ve belirli bir zaman aralığında çalıştırılabilen bir program veya yalnızca belirli bir sıcaklıkta şifresi çözülebilen bir program, frekans veya yükseklik vb. elde edilebilir. PUF tabanlı anahtar üretiminin farklı konfigürasyonlarını, makalenin sadeliği ve gelecekteki çalışmalar için planlanması adına tartışmadık.

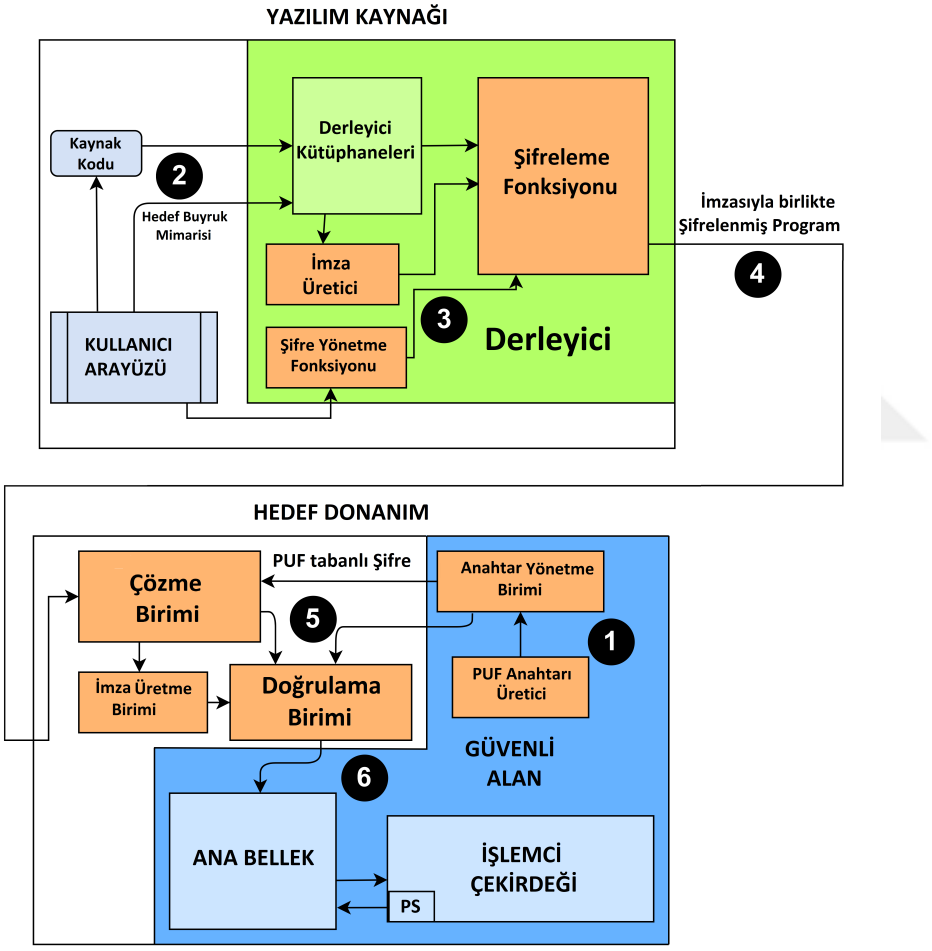
Çözme Birimi. Çözme Birimi, PUF tabanlı anahtarla şifrelenmiş SoC'ye ulaşan programın şifresini çözer. Program kısmen şifrelenmişse, Çözme Birimi, her talimat için şifrelenmiş programa eklenen ek bitleri analiz eder. Bunlar talimat tabanlı kısmi şifrelemede her komuta yeni bir bit eklenmesi iken, kısmi şifreleme seçimine bağlı olarak kullanılacak şifrelemeyi gösteren bitlerin kullanımı artabilir. Şifre çözme için uygulanacak işlevin, derleme işlemindeki şifreleme işlevinin ters çifti olması gerekir. Bu sayede aynı verilerin tekrar elde edilmesi sağlanmaktadır.

İmza Üretme BİrimI. SoC'ye gelen programlar, yazılım mimarisinde program şifrelemesinden önce elde edilen imzalarla taşınır. Bu imzalar da program ile şifrelenir. İmza Üretme Birimi, donanımdaki şifresi çözülen programdan imzayı yeniden hesaplamak için kullanılır. Bu birim, programın şifresi çözülürken aldığı komutlarla programın imzasını oluşturur. Son olarak oluşturulan imza Doğrulama Birimine aktarılır.

Doğrulama Birimi. Programın şifresinin çözülmesi tamamlandığında, kendi hesapladığı imzayı Doğrulama Birimine aktarır. Aynı şekilde programla birlikte gelen şifreli imza da Doğrulama Birimine aktarılır. Doğrulama Biriminde programla birlikte gelen imzanın şifresi çözüldükten sonra, donanımın kendisi tarafından hesaplanan imza ile programla birlikte gelen imza karşılaştırılır. Bir eşleşme varsa, şifresi çözülen programın yürütülmesi için yetki verilir.

4.4.3 Tasarım akışı

Şekil 4.2 önerilen mimarinin şemasını göstermektedir. Bu şekildeki turuncu renk, ERIC'in temel bileşenlerini temsil eder. Şekildeki sayılar, program şifrelemeden SoC'de yürütmeye kadar tipik bir ERIC iş akışını tanımlar.



Şekil 4.2: ERIC'in Tasarım Akışı

1. adım, mimarinin çalışması için gereken ilk adım olan donanımda PUF tabanlı anahtarların oluşturulmasını ifade eder. PUF Anahtar Üreticisi ile elde edilen PUF anahtarı, Anahtar Yönetim Birimi ile şifreleme ve şifre çözme için gerekli olacak PUF tabanlı anahtarı üretir. Bu şekilde SoC'un güvenliği için kritik bir veri olan PUF Anahtarı üzerinde bir soyutlama katmanı da sağlar.

2. adım, doğru şifrelemenin yapılabilmesi için gereksinimleri derleme aşamasına

vermeyi içerir. Bu adım için ERIC'in grafik arayüzü ve teknik belgeleri kullanılabilir. Programın hedeflediği ISA, şifreleme için kullanılacak fonksiyon, kısmi veya tam şifreleme kararı ve hedef donanımın anahtar bilgileri belirlenmelidir.

3. adım, programı şifrelemeyi ve imzayla paketlemeyi içerir. İlk olarak program, bir önceki adımda belirlenen gereksinimlere göre derleyici kitaplıkları kullanılarak hedef ISA için derlenir. Derleme tamamlandıktan sonra İmza Üretici ile programın imzası alınır. İkinci olarak, derleyici aşamasına aktarılan PUF tabanlı anahtarı kullanan anahtar yönetimi işlevi, şifreleme işlevine uygun anahtarlar üretir. Elde edilen imza ve anahtarlar derlenen program ile şifreleme fonksiyonuna taşınır. Bu işlevde, program önceki adımın şifreleme kısıtlamalarına göre şifrelenir ve gerekirse şifresini çözmek için gereken ekstra bilgilerle paketlenir (ERIC'in bir programın kısmi şifrelemesini gerçekleştirmesi durumunda). Ardından imzanın şifrelenmesi ile şifrelenmiş program paketi ve imza yazılım kaynağından çıkmaya hazır hale gelir.

4. adım, programın hedef donanıma ulaşana kadar güvenli bir şekilde paketlenmesini ifade eder. Program, hedef donanım tarafından talep edilmeyi bekleyen bir sunucuda saklanabilir veya dağıtılmış sistemlerde alt iş parçacıklarından biri olarak derlenebilir. Ayrıca bir program kaynağından uzak donanıma gönderilebilir. Bu durumda, programa hedef olmayan donanım veya kötü niyetli kişiler tarafından erişilirse, program şifreli program paketi ve şifreli imza ile korunur.

5. aşamada program ve donanıma ulaşan imzası, PUF Tabanlı Anahtar ile Şifre Çözme Ünitesinde çözülür. Şifre çözme işlemi, şifreleme işleminde belirlenen konfigürasyonlara uygun olarak yapılır. Daha sonra şifresi çözülen program kullanılarak İmza Üretme Biriminde tekrar imza oluşturulur. Ayrıca şifresi çözülen imza Doğrulama Birimi'ne aktarılır.

6. adım, akışın son aşamasıdır. Doğrulama Biriminde, programla birlikte gelen imza, donanımın imza üreticisinde yeniden hesaplanan imza ile karşılaştırılır. Bir eşleşme durumunda, programın herhangi bir değişiklik yapılmadan geldiği ve bu donanım için özel olarak paketlendiği anlaşılır ve şifresi çözülen program güvenli alan olarak kabul edilen bölgeye gönderilir ve işlemci üzerinde yürütülmeye uygun hale gelir.

4.5 Deneysel Sonuçlar

ERIC tarafından getirilen performans ek yükünü ölçmek için, (i) yazılım kaynağı ve (ii) hedef donanım üzerinde ERIC uygularız.

ERIC değerlendirmesini iki kısımda gerçekleştiriyoruz. İlk olarak, çeşitli yazılım kaynakları için ERIC'in şifreli derleme performansını ölçüyoruz. İkinci olarak, hedef donanımdaki şifreli yazılım ikili dosyasının şifresini çözmede ERIC'in performansını değerlendiririz. Çizelge 4.1 değerlendirmemizin yapılandırmasını gösterir. Hedef donanım için şifrelenen yazılım kaynağını özel LLVM tabanlı şifreleme derleyici tasarımımızda test ettik. Önerilen mimari için tasarladığımız SoC'yi FPGA içerisinde kurarak hedef donanımı hayata geçiriyoruz. Şekil 4.3 bu tasarımların şemasını göstermektedir. MiBench [86], sistem performansını değerlendirirken bir kıyaslama noktası olarak kullanılır. MiBench'in LLVM ve RISC-V mimarisi ile uyumlu sınama programlarını seçtik. Önerdiğimiz çerçeve, program üzerindeki iterasyonlara

dayandığından ve bellekteki program boyutu ile doğrudan ilişkili olduğundan, farklı büyüklükteki programların da kıyaslama olarak kullanılması amaçlanmaktadır.

Çizelge 4.1: Test Ortamı

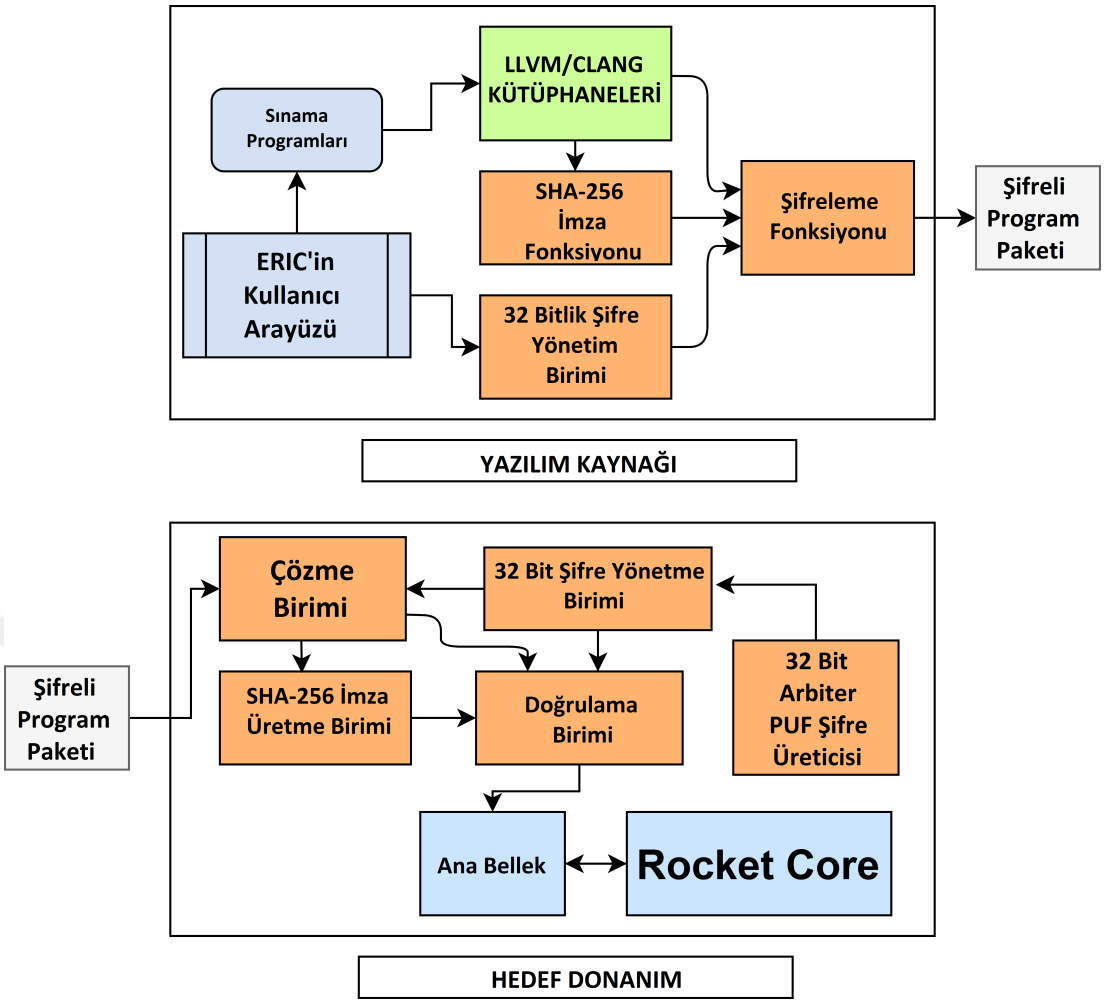
Değişken	Değer
FPGA	Xilinx Zedboard [87]
PUF Türü	Arbiter PUF
PUF Değişkenleri	32x 8-bit tohum (*-ing* challange) 1-bit cevap
İmza Fonksiyonu	SHA-256
Şifreleme Fonksyionu	XOR Cipher
İşlemci Sistemi Türü	Rocket Chip (Sıralı, 6 aşamalı) [88]
Saat Sıklığı	25 MHz
Hedeflenen Buyruk Mimarisi	RV64GC
L1 Veri Önbelleği	16KiB, 4-way, Kümeli ilişki
L1 Buyruk Önbelleği	16KiB, 4-way, Kümeli ilişki
Yazmaç Konfigürasyonu	31 Adet 64-bit

LLVM ve Clang kütüphaneleri ile ön uçtan gelen bir programın derlenmesi sırasında şifreleme fonksiyonu ile kütüphanelerin sağladığı buyruklara ve buyruk mimarisine ait bayraklar (*-ing* flags) entegre edilmiştir. Bu sayede programın derlenmesi sırasında şifreleme işlemlerinin hangi buyrukları hedef alacağı verimli bir şekilde belirlenmiştir. LLVM tabanlı kütüphanelerin ERIC'e sağladığı temel işlev derleme ve şifreleme aşamalarının entegre edilmesi sonucunda derleme ve şifreleme sürecinin daha verimli yapılmasıdır. Şifreleme derleme sürecinin bir aşaması haline getirilip, kısmi şifreleme sırasında oluşabilecek buyruk ve buyruklara ait anahtar bölgelerin seçilmesi süreci ektra bir sorgu yapılmaksınız gerçekleşir. Kullanıcı hedef buyruk mimarisi içerisindeki şifreleme konfigürasyonunu derleyicinin sağladığı seçenekler ile geliştirebilir. Bunun sonucunda çok daha kısa süren ve her mimariyle entegre edilebilir bir mimari çerçeve sağlanır. Yapılan analizlerde derleyici entegrasyonu ile yapılan şifreleme ve paketlemelerin daha düşük derleme zamanı ve daha yüksek kaynak kodu verimliliği sağladığı gözlemlenmiştir.

4.5.1 Yazılım kaynağı

Yazılım kaynağını, tasarladığımız özel derleyiciyi içeren bir sistemde değerlendiriyoruz. Özel derleyicimiz ile şifreleme ve imza oluşturma fonksiyonlarını entegre ettiğimiz bu sistemde kıyaslama programları çalıştırıyoruz. Derleyici, Clang derleyici sürücüsünden taşınır. Ayrıca, derleyici ile entegrasyon için LLVM araçlarından şifreleme ve imza oluşturma mekanizmaları taşınır.

LLVM, birçok buyruk kümelerini ve uzantılarını destekleyen işlevler sağlar, böylece buyruklar derleme sırasında doğrudan etiket değerlerine bağlı olarak bulunan bayraklara (*-ing* flags) göre seçilebilir. Ayrıca LLVM desteği ile ERIC, tercihlere göre derleme sırasında şifreli talimatların dağıtımını optimize etmeye izin verir.



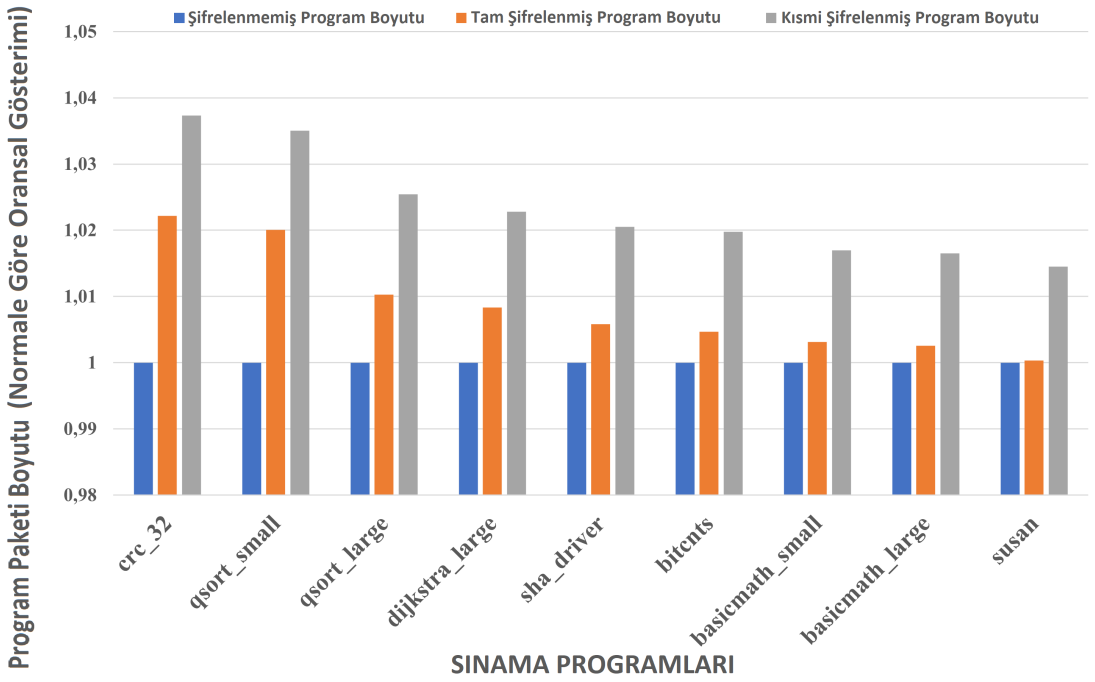
Şekil 4.3: Uygulanan Modelin Şeması

Oluşturduğumuz mekanizma, derleme ve şifreleme konfigürasyonlarının seçilebildiği ERIC arayüzü ile birleştirilmiştir.

Prototipimiz için LLVM 11.1 ve Clang 11.1 versiyonlarını kullanıyoruz. İmza mekanizmasını oluşturmak için C++'da SHA-256 işlevini ve şifreleme işlevi için XOR şifreleme işlevini uygularız. XOR şifreleme işlevi, talimatların ardışık XOR kapılarından geçirilmesiyle yapılan bir şifreleme yöntemi olduğundan, şifrelenmiş mesaja simetrik adımlarla geri ulaşılır. Bu işlevi tasarımın sadeliği için kullanıyoruz. Şifreleme işlevi, Anahtar Yönetim Birimi'nden gelen anahtarlar ile gerçekleştirilir. Anahtar Yönetim Birimi, kullanıcı arayüzünden verilen PUF tabanlı anahtarı, Şifreleme Birimi için uygun formatlarda anahtarlara dönüştürür. Bu şekilde, çoklu şifreleme yinelemeleri tek bir PUF tabanlı anahtarla devam eder. PUF tabanlı bir anahtar, bir PUF anahtarı kullanılarak donanım tarafından oluşturulan bir anahtardır ve Anahtar Yönetim Birimi tarafından donanım üzerinde uygulanır. Bu süreci nasıl uyguladığımız bir sonraki bölümde açıklanmaktadır.

Programları şifreleme yoluyla derleyen ve imzalar oluşturan bir sistemin performansını göstermek için, sonuçları derleme zamanı ve program boyutundaki değişikliklerle gösteriyoruz.

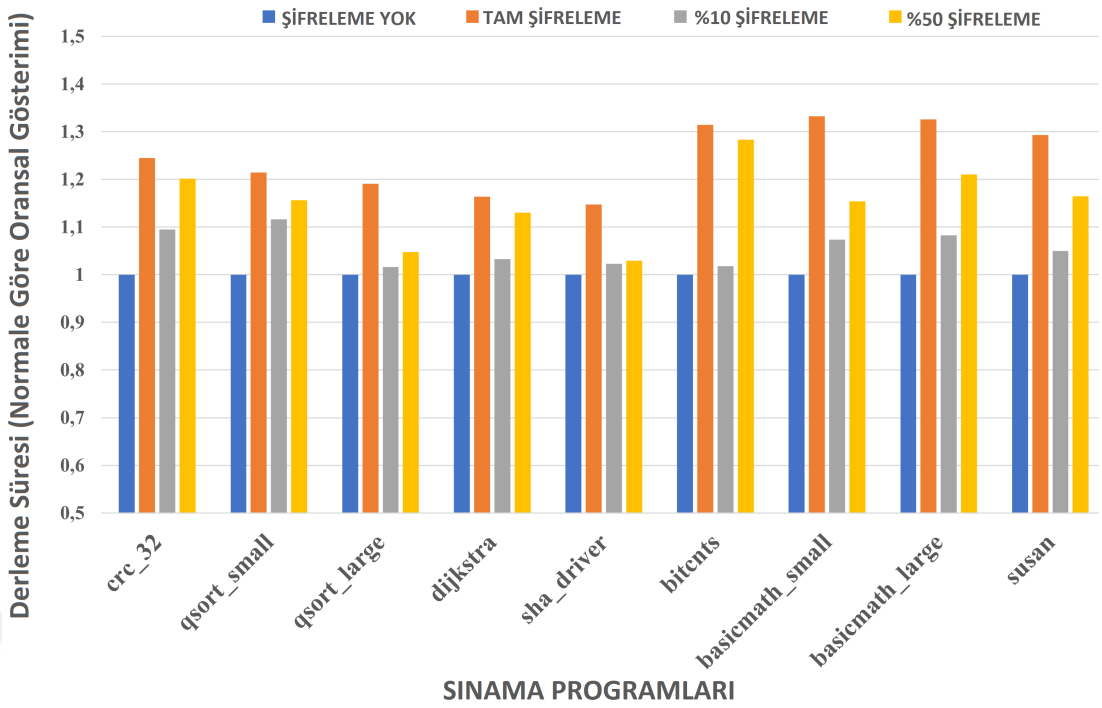
İlk olarak, program boyutundaki sürekli değişiklik, imzaların eklenmesidir. Boyutu ne olursa olsun, her program SHA-256 algoritması sayesinde 256 bit imza eklenerek paketlenir. Tüm talimatlar şifrelenmişse, program boyutundaki dinamik artış mevcut değildir. Ancak program kısmen şifrelendiğinde, programdaki her talimat için talimatın şifrelenip şifrelenmediğini gösteren bir bit eklenir. Kısmi şifreleme yapılandırması için, programdan şifreleme için rastgele belirlenen talimatlar seçilir. Bu, programdaki her talimat için program boyutunda 1 bitlik bir artış anlamına gelir. Sonuç olarak program tamamen şifrelenirse sadece 256 bitlik bir imza artışı görülecektir. Öte yandan, program talimat seçimine göre kısmen şifrelenmişse, program paketi boyutu her talimat için 1 bit ve imza için 256 bit artacaktır. Bu hesaplamaya yakınsayan program paket boyutu testlerde ölçülür. RISC-V buyruk mimarisindeki sıkıştırılmış buyrukların programda yer alması durumunda 16 bit için 1 bit ekstra bilgi alındığından program paket boyutundaki artış oranının değişebileceği gözlemlenmiştir.



Şekil 4.4: Şifrelenmemiş Programın Boyut Normalizasyonuna dayalı olarak Şifreli Program Paketleri ile Şifrelenmemiş Program Paket Boyutu Karşılaştırması

Programın boyutuna bakılmaksızın her program paketine sabit boyutlu imza bitleri eklendiğinden, program boyutu değişim oranlarının eşit olmaması beklenir. Şekil 4.4 şifreli program paketlerinin düz metin (yani şifrelenmemiş) program boyutuna göre boyut değişimini gösterir. En yüksek artış, imza ile şifrelenmiş biçimde paketlenmiş program boyutu, normal derlenmiş program boyutundan %3,73 daha fazladır ve ortalama artış %1,59'dur.

Benchmark Programlarının derleme süresindeki değişiklik Şekil 4.5'da gösterilmektedir. Her bir sınama programı, kendi temel çizgisine (*-ing* baseline) göre normalleştirilir ve grafikte gösterilir. Taban çizgisini elde etmek için, her program varsayılan Clang derleyicisiyle derlendi ve derleme süresi ölçüldü. Daha sonra aynı ortamda uyguladığımız mekanizma ile programların derlenip paketlenmesinde geçen süre ölçüldü. Elde edilen sonuçlara göre derleme süresi en kötü senaryoda %33,20 ve ortalamada %15,22 arttı.



Şekil 4.5: Sınama Programlarına Bağlı Olarak, Şifrelenmemiş Programının Derleme Zamanı Normalleştirmesine Dayalı Derleme Zamanı Karşılaştırması

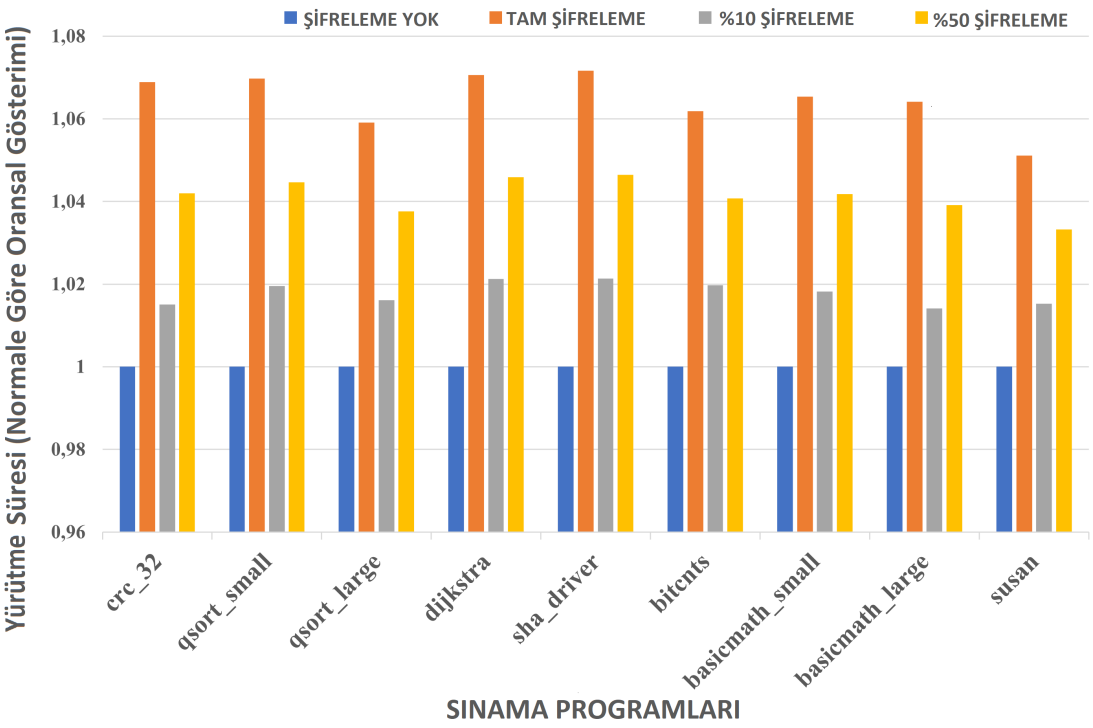
4.5.2 Hedef donanım

Rocket Chip ile güvenliği sağlayan mekanizma olarak geliştirdiğimiz Donanım Şifre Çözme Motoru (donanımda önerilen tüm mimariye verilen isim) Xilinx Zedboard üzerinde uygulayarak hedef donanımın performansını değerlendiriyoruz. Şifre Çözme Birimi için XOR Şifre tabanlı şifre çözme algoritması kullanılır. Bu üniteler ortak bir arayüze bağlanarak donanımdaki alan yükü hesaplanmıştır. Donanım Şifre Çözme Motoru ile Rocket Chip'in FPGA uygulaması ve sadece Rocket Chip uygulaması Çizelge 4.2'de karşılaştırılmıştır. Önerilen donanım mimarisi, Rocket Chip temeline kıyasla %2,63 daha fazla arama tablosu (LUT) ve %3,83 daha fazla flip-flop kullanımı gerektirir.

Çizelge 4.2: FPGA Uygulama Alan Sonuçları

	Rocket Chip	Rocket Chip + ERIC Mimarisi	Değişim Oranı (%)
Toplam LUT Sayısı	33894	34811	+2,63
Toplam Flip-Flop Sayısı	19093	19854	+3,83
Frekans(MHz)	25	25	-

SoC üzerinde şifrelenmiş programların performansını gözlemlemek için şifreli program paketlerini FPGA üzerinde çalıştırdık. Bir temel oluşturmak için, Rocket Chip ile aynı sistem konfigürasyonlarında şifrelemeden derlenen programları çalıştırdık. Yürütme süresindeki değişiklik, Şekil 4.6'deki grafikte, taban çizgisine normalleştirilmiş olarak gösterilir. Elde edilen sonuçlara göre önerdiğimiz yöntemin sistemi en fazla %7,05 ortalama %4,13 yavaşlattığı görülmektedir. ERIC tarafından önerilen mimari Rocket Chip dışında olduğu için programların çalışma performansı



Şekil 4.6: Her Sınama Programına Göre, Şifrelenmemiş Programının Yürütme Süresine Karşılık Şifrelenmiş Programın Yürütme Süresi Karşılaştırması

ve çalışma yöntemlerinin sistem performansı üzerindeki etkisi ERIC ile çalışırken doğrudan gözlemlenmez. Ancak programın dinamik boyutu ile performansı arasında doğrudan bir orantı vardır.

Özetle hem yazılım gizleme hem de güvenilir yürütme için tamamen uçtan uca ve kapsamlı bir çerçeve çözümü sunduk. Mevcut güvenilir yürütme ve yazılım gizleme mimarileriyle karşılaştırıldığında, çerçevemiz kapsamlılığı ve genelleştirilebilirliği önemli ölçüde artırır. Prototipte, donanım ek yükü ve derleyici maliyetleri, mevcut mimari çözümlerine kıyasla çok daha düşüktür.

Mimarimiz teknolojiden bağımsız iki fikri birleştirir: 1) Program derlenirken hedef donanımın PUF tabanlı anahtarı ile şifrelenir ve imzasını oluşturur. 2) Hedef donanım üzerinde çalıştırmadan önce, ERIC'in donanım mimarisi programın şifresini PUF tabanlı anahtarı ile çözer ve programın orijinal versiyonda olduğunu imza ile doğruladıktan sonra çalıştırır. ERIC çerçevesi (*-ing* framework) hem programın güvenliğini hem de donanım üzerindeki yürütmelerin kimlik doğrulamasını geliştirmede etkili olmakla birlikte, yapılandırılabilir ve uygulanması kolaydır. Gelecekteki çalışmalarımız, çerçevenin dağıtılmış sistemlerde uygulanabilmesi ve sunucularda uygulanabilmesi için mimarinin paralellik, performans ve ölçeklenebilirlik yeteneklerini geliştirmeye odaklanacaktır. Ayrıca RSA tabanlı anahtar üretimi ve kullanımını ERIC'e getirmeyi amaçlıyoruz.

5. SONUÇ

Donanım Truva atı (DTA) tehdidi ile birlikte işlemci tabanlı donanımlarda yürütme sürecinde önemli riskler bulunmaktadır. Saldırgan yürütmeyi bozabilir, programı değiştirebilir, sistemde tersine mühendislik teknikleri ile kritik bilgilerin ele geçirilmesini sağlayabilir ve programın kendisini kullanarak güvenlik açıkları ile ilgili analiz yapabilir. Bu tezde belirtilen çalışmalar ile Bloom filtresi tabanlı bellek uygulamaları ve programın saldırgandan gizlenmesi ile donanım ve yazılım kaynağı arasında doğrulanmış bir ilişkinin inşa edilebileceği, yürütmenin programın değiştirilmemiş ve orijinal hali ile gerçekleştirilebileceği ve donanım Truva atının sistem üzerindeki olası etkilerinin engellenmesi ile güvenli mimarilerin inşa edilebileceği gösterilmiştir.

KAYNAKLAR

[1] **Meade**, **T.**, **Jin**, **Y.**, **Tehranipoor**, **M.**, and **Zhang**, **S.** (2016). Gate-level netlist reverse engineering for hardware security: Control logic register identification. In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, pp. 1334–1337.

[2] **Skorobogatov**, **S.** (2013). Silicon scanning reveals hidden backdoors in semiconductor chips. In.

[3] **Shultz II**, **C. J.** and **Saporito**, **B.** (1996). Protecting intellectual property: Strategies and recommendations to deter counterfeiting and brand piracy in global markets. In: *The Columbia Journal of World Business* 31.1, pp. 18–28.

[4] **Tehranipoor**, **M.** and **Koushanfar**, **F.** (2010a). A survey of hardware trojan taxonomy and detection. In: *IEEE design & test of computers* 27.1, pp. 10–25.

[5] **Chakraborty**, **R. S.**, **Saha**, **I.**, **Palchaudhuri**, **A.**, and **Naik**, **G. K.** (2013). Hardware Trojan insertion by direct modification of FPGA configuration bitstream. In: *IEEE Design & Test* 30.2, pp. 45–54.

[6] **Wang**, **X.**, **Narasimhan**, **S.**, **Krishna**, **A.**, **Mal-Sarkar**, **T.**, and **Bhunia**, **S.** (2011). Sequential hardware trojan: Side-channel aware design and placement. In: *2011 IEEE 29th International Conference on Computer Design (ICCD)*. IEEE, pp. 297–300.

[7] **Hepp**, **A.** and **Sigl**, **G.** (2021). Tapeout of a RISC-V crypto chip with hardware trojans: a case-study on trojan design and pre-silicon detectability. In: *Proceedings of the 18th ACM International Conference on Computing Frontiers*, pp. 213–220.

[8] **Vafaei**, **A.**, **Hooten**, **N.**, **Tehranipoor**, **M.**, and **Farahmandi**, **F.** (2021). SymbA: Symbolic Execution at C-level for Hardware Trojan Activation. In: *2021 IEEE International Test Conference (ITC)*, pp. 223–232.

[9] **Bloom**, **G.**, **Narahari**, **B.**, and **Simha**, **R.** (2009). OS support for detecting Trojan circuit attacks. In: *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, pp. 100–103.

[10] **Bloom**, **G.**, **Narahari**, **B.**, **Simha**, **R.**, and **Zambreno**, **J.** (2009). Providing secure execution environments with a last line of defense against trojan circuit attacks. In: *computers & security* 28.7, pp. 660–669.

[11] **Hasan**, **M.**, **Cruz**, **J.**, **Chakraborty**, **P.**, **Bhunia**, **S.**, and **Hoque**, **T.** (2021). Software Variants for Hardware Trojan Detection and Resilience in COTS Processors. In: *arXiv preprint arXiv:2112.00304*.

[12] **Palumbo**, **A.**, **Cassano**, **L.**, **Reviriego**, **P.**, **Bianchi**, **G.**, and **Ottavi**, **M.** (2021). A Lightweight Security Checking Module to Protect Microprocessors against Hardware Trojan Horses. In: *2021 IEEE*

International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). IEEE, pp. 1–6.

[13] **Bolat**, **A.**, **Cassano**, **L.**, **Reviriego**, **P.**, **Ergin**, **O.**, and **Ottavi**, **M.** (2020). A microprocessor protection architecture against hardware trojans in memories. In: *2020 15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, pp. 1–6.

[14] **Bloom**, **B. H.** (1970). Space/time trade-offs in hash coding with allowable errors. In: *Communications of the ACM* 13.7, pp. 422–426.

[15] **Pontarelli**, **S.** and **Ottavi**, **M.** (2013). Error Detection and Correction in Content Addressable Memories by Using Bloom Filters. In: *IEEE Transactions on Computers* 62.6, pp. 1111–1126.

[16] **Maes**, **R.** and **Verbauwhede**, **I.** (2010). Physically unclonable functions: A study on the state of the art and future research directions. In: *Towards Hardware-Intrinsic Security*. Springer, pp. 3–37.

[17] **Herder**, **C.**, **Yu**, **M.-D.**, **Koushanfar**, **F.**, and **Devadas**, **S.** (2014). Physical unclonable functions and applications: A tutorial. In: *Proceedings of the IEEE* 102.8, pp. 1126–1141.

[18] **Chen**, **Q.**, **Csaba**, **G.**, **Lugli**, **P.**, **Schlichtmann**, **U.**, and **Rührmair**, **U.** (2011). The bistable ring PUF: A new architecture for strong physical unclonable functions. In: *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, pp. 134–141.

[19] **Das**, **J.**, **Scott**, **K.**, **Rajaram**, **S.**, **Burgett**, **D.**, and **Bhanja**, **S.** (2015). MRAM PUF: A novel geometry based magnetic PUF with integrated CMOS. In: *IEEE Transactions on Nanotechnology* 14.3, pp. 436–443.

[20] **Morozov**, **S.**, **Maiti**, **A.**, and **Schaumont**, **P.** (2010). An analysis of delay based PUF implementations on FPGA. In: *International Symposium on Applied Reconfigurable Computing*. Springer, pp. 382–387.

[21] **Phalak**, **K.**, **Ash-Saki**, **A.**, **Alam**, **M.**, **Topaloglu**, **R. O.**, and **Ghosh**, **S.** (2021). Quantum PUF for Security and Trust in Quantum Computing. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11.2, pp. 333–342.

[22] **Chatterjee**, **B.**, **Das**, **D.**, **Maity**, **S.**, and **Sen**, **S.** (2018). RF-PUF: Enhancing IoT security through authentication of wireless nodes using in-situ machine learning. In: *IEEE Internet of Things Journal* 6.1, pp. 388–398.

[23] **Kim**, **J. S.**, **Patel**, **M.**, **Hassan**, **H.**, and **Mutlu**, **O.** (2018). The DRAM latency PUF: Quickly evaluating physical unclonable functions by exploiting the latency-reliability tradeoff in modern commodity DRAM devices. In: *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 194–207.

[24] **Garg**, **A.** and **Kim**, **T. T.** (2014). Design of SRAM PUF with improved uniformity and reliability utilizing device aging effect. In: *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, pp. 1941–1944.

[25] **Dolev**, **S.**, **Krzywiecki**, **Ł.**, **Panwar**, **N.**, and **Segal**, **M.** (2016). Optical PUF for non-forwardable vehicle authentication. In: *Computer Communications* 93, pp. 52–67.

[26] **Morozov**, **S.**, **Maiti**, **A.**, and **Schaumont**, **P.** (2009). A Comparative Analysis of Delay Based PUF Implementations on FPGA. In: *IACR Cryptol. ePrint Arch.* 2009, p. 629.

[27] **Machida**, **T.**, **Yamamoto**, **D.**, **Iwamoto**, **M.**, and **Sakiyama**, **K.** (2015). A new arbiter PUF for enhancing unpredictability on FPGA. In: *The Scientific World Journal* 2015.

[28] **Lattner**, **C.** and **Adve**, **V.** (2004). LLVM: A compilation framework for lifelong program analysis & transformation. In: *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* IEEE, pp. 75–86.

[29] **LLVM** (n.d.). The LLVM Compiler Infrastructure. `https://github.com/llvm/llvm-project`.

[30] **Jin**, **Y.**, **Maniatakos**, **M.**, and **Makris**, **Y.** (2012). Exposing vulnerabilities of untrusted computing platforms. In: *Proc. Int. Conf. Computer Design*, pp. 131–134.

[31] **Tsoutsos**, **N. G.** and **Maniatakos**, **M.** (2014). Fabrication Attacks: Zero-Overhead Malicious Modifications Enabling Modern Microprocessor Privilege Escalation. In: *IEEE Trans. Emerging Topics in Computing* 2.1, pp. 81–93.

[32] **Wang**, **X.**, **Mal-Sarkar**, **T.**, **Krishna**, **A.**, **Narasimhan**, **S.**, and **Bhunia**, **S.** (2012). Software exploitable hardware Trojans in embedded processor. In: *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, pp. 55–58.

[33] (n.d.[b]). https://github.com/xoreaxeaxeax/rosenbridge.

[34] **Chuan**, **X.**, **Yan**, **Y.**, and **Zhang**, **Y.** (2017). An efficient triggering method of hardware Trojan in AES cryptographic circuit. In: *Proc. Int. Conf. Integrated Circuits and Microsystems*, pp. 91–95.

[35] **Zhang**, **J.**, **Yuan**, **F.**, **Wei**, **L.**, **Liu**, **Y.**, and **Xu**, **Q.** (2015). VeriTrust: Verification for Hardware Trust. In: *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* 34.7, pp. 1148–1161.

[36] **Liu**, **Y.**, **Zhao**, **Y.**, **He**, **J.**, **Liu**, **A.**, and **Xin**, **R.** (2017). SCCA: Side-channel correlation analysis for detecting hardware Trojan. In: *Proc. Int. Conf. Anti-counterfeiting, Security, and Identification*, pp. 196–200.

[37] **Salmani**, **H.** and **Tehranipoor**, **M.** (2013). Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level. In: *Proc. Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pp. 190–195.

[38] **Salmani**, **H.** and **Tehranipoor**, **M.** (2012). Layout-Aware Switching Activity Localization to Enhance Hardware Trojan Detection. In: *IEEE Trans. Information Forensics and Security* 7.1, pp. 76–87.

[39] **Bolat**, **A.**, **Çelik**, **S. H.**, **Olgun**, **A.**, **Ergin**, **O.**, and **Ottavi**, **M.** (2022). ERIC: An Efficient and Practical Software Obfuscation Framework.

In: *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, pp. 466–474.

[40] **Šišejković**, **D.**, **Merchant**, **F.**, **Leupers**, **R.**, **Ascheid**, **G.**, and **Kegreiss**, **S.** (2019). Control-Lock: Securing Processor Cores Against Software-Controlled Hardware Trojans. In: *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. GLSVLSI '19, pp. 27–32.

[41] **Shila**, **D. M.**, **Venugopalan**, **V.**, and **Patterson**, **C. D.** (2015). FIDES: Enhancing trust in reconfigurable based hardware systems. In: *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7.

[42] **Basak**, **A.**, **Bhunia**, **S.**, **Tkacik**, **T.**, and **Ray**, **S.** (2017). Security Assurance for System-on-Chip Designs With Untrusted IPs. In: *IEEE Transactions on Information Forensics and Security* 12.7, pp. 1515–1528.

[43] **Dubeuf**, **J.**, **Hély**, **D.**, and **Karri**, **R.** (2013). Run-time detection of hardware Trojans: The processor protection unit. In: *2013 18th IEEE European Test Symposium (ETS)*, pp. 1–6.

[44] **Bloom**, **G.**, **Narahari**, **B.**, and **Simha**, **R.** (2009). OS support for detecting Trojan circuit attacks. In: *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, pp. 100–103.

[45] **Hoque**, **T.**, **Wang**, **X.**, **Basak**, **A.**, **Karam**, **R.**, and **Bhunia**, **S.** (2018). Hardware Trojan attacks in embedded memory. In: *2018 IEEE 36th VLSI Test Symposium (VTS)*, pp. 1–6.

[46] **Tehranipoor**, **M.** and **Koushanfar**, **F.** (2010b). A survey of hardware trojan taxonomy and detection. In: *IEEE Design & Test of Computers* 27.1.

[47] **Atamaner**, **M.**, **Ergin**, **O.**, **Ottavi**, **M.**, and **Reviriego**, **P.** (2017). Detecting errors in instructions with bloom filters. In: *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 1–4.

[48] **Gautschi**, **M.**, **Schiavone**, **P. D.**, **Traber**, **A.**, **Loi**, **I.**, **Pullini**, **A.**, **Rossi**, **D.**, **Flamand**, **E.**, **Gürkaynak**, **F. K.**, and **Benini**, **L.** (2017). Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.10, pp. 2700–2713.

[49] **Höller**, **R.**, **Haselberger**, **D.**, **Ballek**, **D.**, **Rössler**, **P.**, **Krapfenbauer**, **M.**, and **Linauer**, **M.** (2019). Open-Source RISC-V Processor IP Cores for FPGAs — Overview and Evaluation. In: *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–6.

[50] **Madakam**, **S.**, **Lake**, **V.**, **Lake**, **V.**, **Lake**, **V.**, et al. (2015). Internet of Things (IoT): A literature review. In: *Journal of Computer and Communications* 3.05, p. 164.

[51] **Gershenfeld**, **N.**, **Krikorian**, **R.**, and **Cohen**, **D.** (2004). The internet of things. In: *Scientific American* 291.4, pp. 76–81.

[52] **Gubbi**, **J.**, **Buyya**, **R.**, **Marusic**, **S.**, and **Palaniswami**, **M.** (2013). Internet of Things (IoT): A vision, architectural elements, and future

directions. In: *Future generation computer systems* 29.7, pp. 1645–1660.

[53] **Al-Sarawi**, **S.**, **Anbar**, **M.**, **Alieyan**, **K.**, and **Alzubaidi**, **M.** (2017). Internet of Things (IoT) communication protocols. In: *2017 8th International conference on information technology (ICIT)*. IEEE, pp. 685–690.

[54] **Birman**, **K. P.** (1997). Building secure and reliable network applications. In: *International Conference on Worldwide Computing and Its Applications*. Springer, pp. 15–28.

[55] **Yan**, **Y.**, **Qian**, **Y.**, and **Sharif**, **H.** (2011). A secure and reliable in-network collaborative communication scheme for advanced metering infrastructure in smart grid. In: *2011 IEEE wireless communications and networking conference*. IEEE, pp. 909–914.

[56] **Marwedel**, **P.** (2021). Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things. Springer Nature.

[57] **Gajski**, **D. D.**, **Abdi**, **S.**, **Gerstlauer**, **A.**, and **Schirner**, **G.** (2009). Embedded system design: modeling, synthesis and verification. Springer Science & Business Media.

[58] **Gracioli**, **G.** and **Fröhlich**, **A. A.** (2008). An operating system infrastructure for remote code update in deeply embedded systems. In: *Proceedings of the 1st International Workshop on Hot Topics in Software Upgrades*, pp. 1–5.

[59] **Callaghan**, **M. J.**, **Harkin**, **J.**, **McGinnity**, **T.**, and **Maguire**, **L. P.** (2002). An Internet-based methodology for remotely accessed embedded systems. In: *IEEE International Conference on Systems, Man and Cybernetics*. Vol. 6. IEEE, 6–pp.

[60] **Kremer**, **U.**, **Hicks**, **J.**, and **Rehg**, **J.** (2001). A compilation framework for power and energy management on mobile computers. In: *International Workshop on Languages and Compilers for Parallel Computing*. Springer, pp. 115–131.

[61] **Nethercote**, **N.** (2004). *Dynamic binary analysis and instrumentation*. Tech. rep. University of Cambridge, Computer Laboratory.

[62] **Tallent**, **N. R.**, **Mellor-Crummey**, **J. M.**, and **Fagan**, **M. W.** (2009). Binary analysis for measurement and attribution of program performance. In: *ACM Sigplan Notices* 44.6, pp. 441–452.

[63] **Falcarin**, **P.**, **Di Carlo**, **S.**, **Cabutto**, **A.**, **Garazzino**, **N.**, and **Barberis**, **D.** (2011). Exploiting code mobility for dynamic binary obfuscation. In: *2011 World Congress on Internet Security (WorldCIS-2011)*. IEEE, pp. 114–120.

[64] **Kruegel**, **C.**, **Kirda**, **E.**, **Mutz**, **D.**, **Robertson**, **W.**, and **Vigna**, **G.** (2005). Automating mimicry attacks using static binary analysis. In: *USENIX Security Symposium*. Vol. 14, pp. 11–11.

[65] **Shoshitaishvili**, **Y.**, **Wang**, **R.**, **Salls**, **C.**, **Stephens**, **N.**, **Polino**, **M.**, **Dutcher**, **A.**, **Grosen**, **J.**, **Feng**, **S.**, **Hauser**, **C.**, **Kruegel**, **C.**, et al. (2016). Sok:(state of) the art of war: Offensive techniques in binary analysis. In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 138–157.

[66] **Brumley**, **D.** (2008). *Analysis and defense of vulnerabilities in binary code*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE.

[67] **Van Der Veen**, **V.**, **Göktas**, **E.**, **Contag**, **M.**, **Pawoloski**, **A.**, **Chen**, **X.**, **Rawat**, **S.**, **Bos**, **H.**, **Holz**, **T.**, **Athanasopoulos**, **E.**, and **Giuffrida**, **C.** (2016). A tough call: Mitigating advanced code-reuse attacks at the binary level. In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 934–953.

[68] **She**, **S.**, **Lotufo**, **R.**, **Berger**, **T.**, **Wąsowski**, **A.**, and **Czarnecki**, **K.** (2011). Reverse engineering feature models. In: *Proceedings of the 33rd International Conference on Software Engineering*, pp. 461–470.

[69] **Subramanyan**, **P.**, **Tsiskaridze**, **N.**, **Li**, **W.**, **Gascón**, **A.**, **Tan**, **W. Y.**, **Tiwari**, **A.**, **Shankar**, **N.**, **Seshia**, **S. A.**, and **Malik**, **S.** (2013). Reverse engineering digital circuits using structural and functional analyses. In: *IEEE Transactions on Emerging Topics in Computing* 2.1, pp. 63–80.

[70] **Popa**, **M.** (2012). Binary code disassembly for reverse engineering. In: *Journal of Mobile, Embedded and Distributed Systems* 4.4, pp. 233–248.

[71] **Coppens**, **B.**, **Verbauwhede**, **I.**, **De Bosschere**, **K.**, and **De Sutter**, **B.** (2009). Practical mitigations for timing-based side-channel attacks on modern x86 processors. In: *2009 30th IEEE symposium on security and privacy*. IEEE, pp. 45–60.

[72] **Lie**, **D.**, **Thekkath**, **C. A.**, and **Horowitz**, **M.** (2003). Implementing an untrusted operating system on trusted hardware. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 178–192.

[73] **Lie**, **D.**, **Thekkath**, **C.**, **Mitchell**, **M.**, **Lincoln**, **P.**, **Boneh**, **D.**, **Mitchell**, **J.**, and **Horowitz**, **M.** (2000). Architectural support for copy and tamper resistant software. In: *Acm Sigplan Notices* 35.11, pp. 168–177.

[74] **Suh**, **G. E.**, **Clarke**, **D.**, **Gassend**, **B.**, **Van Dijk**, **M.**, and **Devadas**, **S.** (2003). AEGIS: Architecture for tamper-evident and tamper-resistant processing. In: *ACM International Conference on Supercomputing 25th Anniversary Volume*, pp. 357–368.

[75] **Suh**, **G. E.**, **O'Donnell**, **C. W.**, **Sachdev**, **I.**, and **Devadas**, **S.** (2005). Design and implementation of the AEGIS single-chip secure processor using physical random functions. In: *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, pp. 25–36.

[76] **Suh**, **G. E.**, **O'Donnell**, **C. W.**, and **Devadas**, **S.** (2007). Aegis: A single-chip secure processor. In: *IEEE Design & Test of Computers* 24.6, pp. 570–580.

[77] **Owusu**, **E.**, **Guajardo**, **J.**, **McCune**, **J.**, **Newsome**, **J.**, **Perrig**, **A.**, and **Vasudevan**, **A.** (2013). OASIS: On achieving a sanctuary for integrity and secrecy on untrusted platforms. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 13–24.

[78] **Costan**, **V.** and **Devadas**, **S.** (2016). Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086. `https://ia.cr/2016/086`.

[79] **Pinto**, **S.** and **Santos**, **N.** (2019). Demystifying arm trustzone: A comprehensive survey. In: *ACM Computing Surveys (CSUR)* 51.6, pp. 1–36.

[80] **Zhang**, **P.**, **Song**, **C.**, **Yin**, **H.**, **Zou**, **D.**, **Shi**, **E.**, and **Jin**, **H.** (2020). Klotski: Efficient obfuscated execution against controlled-channel attacks. In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1263–1276.

[81] **Cyr**, **B.**, **Mahmod**, **J.**, and **Guin**, **U.** (2019). Low-cost and secure firmware obfuscation method for protecting electronic systems from cloning. In: *IEEE Internet of Things Journal* 6.2, pp. 3700–3711.

[82] **Awad**, **A.**, **Wang**, **Y.**, **Shands**, **D.**, and **Solihin**, **Y.** (2017). Obfusmem: A low-overhead access obfuscation for trusted memories. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 107–119.

[83] **Fletcher**, **C. W.**, **Dijk**, **M. v.**, and **Devadas**, **S.** (2012). A secure processor architecture for encrypted computation on untrusted programs. In: *Proceedings of the seventh ACM workshop on Scalable trusted computing*, pp. 3–8.

[84] **Schrittwieser**, **S.** and **Katzenbeisser**, **S.** (2011). Code obfuscation against static and dynamic reverse engineering. In: *International workshop on information hiding*. Springer, pp. 270–284.

[85] **FIPS, PUB** (2012). 180-2: Secure hash standard (SHS). In: *US Department of Commerce, National Institute of Standards and Technology (NIST)*.

[86] **Guthaus**, **M. R.**, **Ringenberg**, **J. S.**, **Ernst**, **D.**, **Austin**, **T. M.**, **Mudge**, **T.**, and **Brown**, **R. B.** (2001). MiBench: A free, commercially representative embedded benchmark suite. In: *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, pp. 3–14.

[87] **Xilinx** (n.d.). Xilinx ZedBoard Zynq-7000. `https://www.xilinx.com/products/boards-and-kits/1-8dyf-11.html`.

[88] **Chipsalliance** (n.d.). Rocket Chip. `https://github.com/chipsalliance/rocket-chip`.