T.C.
BALIKESIR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

PRESENT BLOK ŞİFRELEME ALGORİTMASININ FPGA GERÇEKLEMESİ VE YAN KANAL ANALİZİ

YÜKSEK LİSANS TEZİ

YASİN REŞİT YARGICI

BALIKESIR, EYLÜL - 2019

T.C.
BALIKESIR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

PRESENT BLOK ŞİFRELEME ALGORİTMASININ FPGA GERÇEKLEMESİ VE YAN KANAL ANALİZİ

YÜKSEK LİSANS TEZİ

YASİN REŞİT YARGICI

Jüri Üyeleri : Doç. Dr. Selçuk KAVUT (Tez Danışmanı)
Dr. Öğr. Üyesi Serkan GÜRKAN
Dr. Öğr. Üyesi Sabri BİCAKCI

BALIKESIR, EYLÜL - 2019

KABUL VE ONAY SAYFASI

Yasin Reşit YARGICI tarafından hazırlanan **"PRESENT BLOK ŞİFRELEME ALGORİTMASININ FPGA GERÇEKLEMESİ VE YAN KANAL ANALİZİ"** adlı tez çalışmasının savunma sınavı 09.09.2019 tarihinde yapılmış olup aşağıda verilen jüri tarafından oy birliği / ~~oy çokluğu~~ ile Balıkesir Üniversitesi Fen Bilimleri Enstitüsü Elektrik-Elektronik Mühendisliği Anabilim Dalı Yüksek Lisans Tezi olarak kabul edilmiştir.

Jüri Üyeleri	İmza
Danışman Doç. Dr. Selçuk KAVUT	.....
Üye Dr.Öğr.Üyesi Serkan GÜRKAN	.....
Üye Dr. Öğr. Üyesi Sabri BİCAKCI	.....

Jüri üyeleri tarafından kabul edilmiş olan bu tez Balıkesir Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulunca onanmıştır.

Fen Bilimleri Enstitüsü Müdürü

Prof. Dr. Necati ÖZDEMİR .....

ÖZET

PRESENT BLOK ŞİFRELEME ALGORİTMASININ FPGA GERÇEKLEMESİ VE YAN KANAL ANALİZİ
YÜKSEK LİSANS TEZİ
YASİN REŞİT YARGICI
BALIKESİR ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI
(TEZ DANIŞMANI: DOÇ. DR. SELÇUK KAVUT)
BALIKESIR, EYLÜL - 2019

Tez kapsamında kaynak tüketimi az olan hafif siklet (lightweight) blok şifreleme algoritmalarından PRESENT algoritmasının SAKURA-X kartı üzerinde implementasyonu yapılmıştır. Şifreleme algoritması, gömüldüğü kriptografik donanımda çalışma sırasında ısı, elektromanyetik yayınım, harcadığı güç ve ses gibi yan kanal bilgileri sızdırır. Yan kanal analizi (YKA) ile saldırgan, bu bilgileri kullanarak şifreleme algoritmasının kullandığı gizli anahtarı elde etmeye çalışır. Çalışmamızda, 80-bit anahtara sahip PRESENT algoritmasının SAKURA-X gerçeklemesine yönelik en güçlü YKA tekniklerinden olan farksal güç analizi (FGA) incelenmiştir. İlk olarak Paul Kocher tarafından 1998 yılında ortaya çıkarılan farksal güç analizinin temel prensibi, gerçeklemenin yapıldığı kriptografik donanımda kullanılan bitlerin durum değiştirmesinden kaynaklanan güç tüketiminin kullanılan gizli anahtara ait bilgi içermesine dayanmaktadır.

Çalışmamız temel olarak iki konu üzerinde odaklanmaktadır. İlk olarak, PRESENT blok şifreleme algoritması ayrıntılı olarak anlatılmış ve SAKURA-X kartı üzerinde implement edilmiştir. Sonrasında, bu blok şifrenin farksal güç analizi MATLAB ile simülasyonu gerçekleştirilmiştir.

ANAHTAR KELİMELER: PRESENT, yan kanal analizi (YKA), SAKURA-X, farksal güç analizi (FGA), MATLAB.

ABSTRACT

FPGA IMPLEMENTATION AND SIDE CHANNEL ANALYSIS OF BLOCK CIPHER PRESENT
MSC THESIS
YASİN REŞİT YARGICI
BALIKESIR UNIVERSITY INSTITUTE OF SCIENCE
ELECTRICAL AND ELECTRONICS ENGINEERING
(SUPERVISOR: ASSOC. PROF. DR. SELÇUK KAVUT))
BALIKESIR, SEPTEMBER 2019

Within the scope of this thesis, we perform the implementation of block cipher PRESENT, which is one of the lightweight block ciphers designed especially for resource-constrained environments, on SAKURA-X. An encryption algorithm leaks side-channel information such as heat, electromagnetic radiation, power consumption and sound. With side channel analysis (SCA), an attacker tries to obtain the secret key used by the block cipher, exploiting those information. In this study, we examine the differential power analysis (DPA), which is one of the most powerful SCA techniques, of the implementation of PRESENT having 80-bit key on SAKURA-X. The differential power analysis, introduced for the first time by Paul Kocher in 1998, is based on the fact that the power consumed during the change of bits performed by the cryptographic hardware contains some information belonging to the secret key.

Our study mainly focuses on two subjects. First, we explain block cipher PRESENT in detail and implement it on SAKURA-X. After that, we perform the simulation of differential power analysis of this block cipher using MATLAB.

KEYWORDS: PRESENT, side channel analysis (SCA), SAKURA-X, differential power analysis (DPA), MATLAB.

İÇİNDEKİLER

Sayfa

ŞEKİL LİSTESİ

Sayfa

TABLO LİSTESİ

Sayfa

ÖNSÖZ

Yüksek lisans dönemim boyunca bana yol gösteren ve desteğini esirgemeyen hocam Doç. Dr. Selçuk KAVUT'a teşekkür ederim.

Yaşamım boyunca bana maddi ve manevi desteklerini hep hissettiren en zor zamanlarımda yükümü hafifleten canım aileme sonsuz şükran ve iyi dileklerimi sunuyorum.

Balıkesir, 2019 Yasin Reşit YARGICI

1. GİRİŞ

1.1 Kriptoloji

Yaşadığımız bilgisayar ve kablosuz haberleşme ortamında verilerin gizlenmesi ve güvenli bir şekilde üçüncü bir kişinin eline geçmeden iletilmesi önemli bir unsur haline gelmiştir. Bu durum, güvenli olarak verinin depolanması ve veri haberleşmesi ile ilgili bir bilim dalı olan kriptolojinin değerini arttırmaktadır. Kriptoloji, gizli yazma anlamına gelen kriptografi ve kod kırma olarak da bilinen kriptanalizin bir bütünüdür.

Şifreleme, veri güvenliğini sağlamanın en etkili yoludur. Günümüzde bireylerin kişisel ve finansal gizlilikleri gibi dijital mahremiyetin korunmasını sağlayan modern şifreleme algoritmaları büyük önem taşımaktadır. Bu algoritmalar simetrik (gizli anahtarlı) ve asimetrik (açık anahtarlı) şifreleme olarak iki genel sınıfa ayrılmaktadır. Simetrik şifrelemede, verilerin şifrelenmesi ve şifreli verilerin çözülmesi için aynı anahtar kullanılırken, asimetrik şifrelemede iki ayrı anahtar kullanılmaktadır. Bu iki yöntem teori ve uygulamada çok farklı olsa da, herhangi bir şifreleme sistemi için genel olarak aşağıdaki tanımlar kullanılmaktadır.

Düz (açık) metin: Şifrelenmemiş orijinal metin.

Şifreli (kapalı) metin: Şifrelenmiş, formu değiştirilmiş metin.

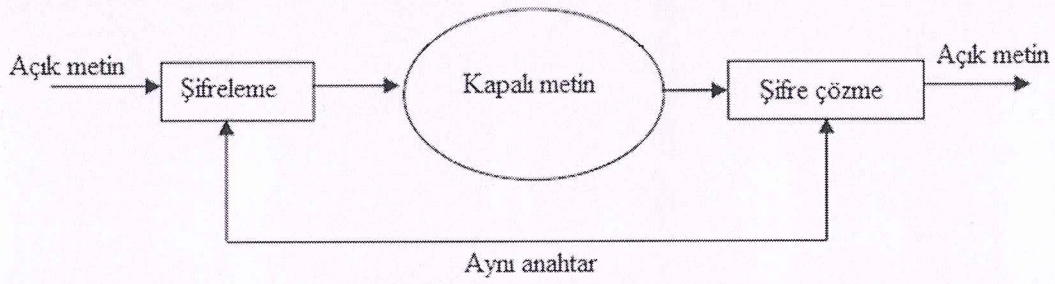
Kriptosistem: Verilerin şifrelenmesini/deşifrelenmesini gerçekleştiren sistem.

Anahtar: Açık metini kapalı metine veya kapalı metini açık metine dönüştürmek için kriptosistem tarafından kullanılan bit dizisi.

1.1.1 Kriptografi

1.1.1.1 Simetrik Şifreleme

Simetrik şifrelemede, haberleşen taraflar şifreleme ve şifre çözme işlemlerinin her ikisi için de aynı anahtarı kullanırlar (Şekil 1.1). n kişinin simetrik şifreleme yöntemi kullandığı bir haberleşme sisteminde, kullanılan toplam anahtar sayısının $\binom{n}{2}$ olduğu görülmektedir. Bu nedenle, böyle bir sistemde, kullanılan anahtar sayısı ve bu anahtarların güvenli bir şekilde paylaşımı açısından düşünüldüğünde simetrik şifreleme yöntemi verimli değildir. Bununla birlikte, simetrik şifreleme algoritmaları bir sonraki bölümde bahsedilen asimetrik şifreleme algoritmalarına göre daha basit ve daha hızlıdır. Ancak ana dezavantajı, yukarıda bahsedildiği üzere iki tarafın bir şekilde anahtarı güvenli bir şekilde paylaşması gerektiğidir. Şifrelemenin gizliliği anahtarın gizliliğini korumakla sağlanır. Anahtarı gönderenin ve alıcının haricinde saldırgan tarafından ele geçirilirse, saldırgan mesajın şifresini çözüp açık metine erişebilecektir.

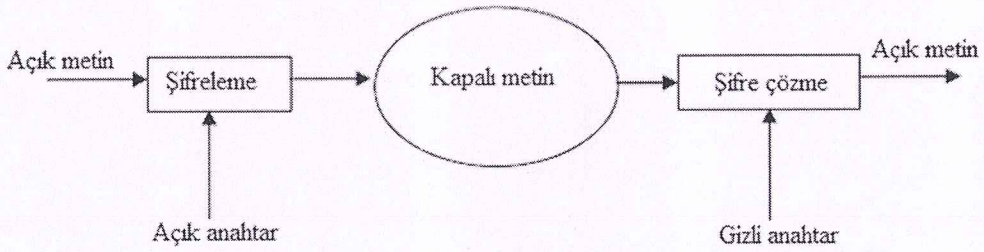


Şekil 1.1: Simetrik şifreleme.

Simetrik anahtarlı şifreleme algoritmaları blok şifreleme ve akan şifreleme olarak iki sınıfa ayrılmaktadır. Akan şifreler açık metinin her bir sembolünü kapalı metin sembolüne dönüştürürken, blok şifreler açık metin sembollerini bloklar veya gruplar halinde şifreleyerek kapalı metnin bloklarına dönüştürür. Akan şifreleme algoritmalarına RC4 [7] , blok şifreleme algoritmalarına veri şifreleme standardı (DES) [1], gelişmiş şifreleme standardı (AES) [2], Blowfish [3], PRESENT [4] örnek verilebilir. PRESENT şifreleme algoritması ikinci bölümde ayrıntılı olarak anlatılacaktır.

1.1.1.2 Asimetrik Şifreleme

Asimetrik şifrelemede haberleşen taraflar, şifreleme işlemi için ve şifre çözme işlemi için ayrı anahtar kullanırlar. Kullanılan anahtarların bir tanesi gizli, diğeri açık anahtardır. Şifreleme için kullanılan anahtar herkes tarafından erişilebilirdir. Ancak şifre çözme anahtarı yalnızca alıcı tarafından bilinmektedir. Bu sayede herkese açık haberleşme ağında güvenli bir haberleşme imkânı sağlanmaktadır. Bahsedilen anahtar çifti arasında matematiksel bir bağıntı vardır ve açık anahtardan gizli anahtar, gizli anahtardan açık anahtar elde edilemez; esasen bu zorluk, gereken işlemci gücünün çok fazla olmasından kaynaklanmaktadır. Genel olarak, simetrik şifreleme ile karşılaştırıldığında, asimetrik şifreleme güvenlik açısından daha avantajlı olmakla birlikte daha yavaştır ve her iki şifreleme türünün de güvenliği kullanılan anahtarın uzunluğuna bağladır. Bunun yanı sıra, n kişinin asimetrik şifreleme yöntemi kullandığı bir haberleşme sisteminde, kullanılan toplam anahtar sayısının n olduğu (n tane açık ve gizli anahtar çifti) görülmektedir. Simetrik şifreleme yönteminde bu sayının $\binom{n}{2}$ olduğu hatırlanırsa, kullanılan anahtar sayısı ve bu anahtarların güvenli bir şekilde paylaşımı açısından asimetrik şifreleme yöntemi daha verimlidir. Asimetrik şifrelemeye RSA [6], Diffie-Hellman [5] gibi algoritmalar örnek verilebilir. Asimetrik şifrelemenin yapısı Şekil 1.2'de gösterilmiştir.



Şekil 1.2: Asimetrik şifreleme.

1.1.2 Kriptanaliz

Kriptanaliz gizli anahtarı bilmeden şifreli metni deşifre etmek için kriptografik algoritmalarda zayıflıkları bulma ve bu zayıflıkları kullanma sürecidir.

Kriptanaliz, kriptografik algoritmaların matematiksel analizine (örneğin, doğrusal [11] ve farksal kriptanaliz [9] gibi istatistiksel saldırılar) ek olarak, gerçeklemenin yapıldığı kriptografik donanımdan elde edilen yan kanal bilgileri kullanılarak da yürütülmektedir. Gerçekte, yan kanal analizi bahsedilen matematiksel analize göre çok daha az sayıda kapalı metine ihtiyaç duyduğundan, daha güçlü ve verimli saldırı türüdür. Örneğin, DES'in doğrusal kriptanaliz yöntemiyle kırılması için 2^{43} açık ve kapalı metin çiftine ihtiyaç duyulurken [8], yan kanal analizi ile kırılması için açık metine ihtiyaç duyulmadan birkaç yüz kapalı metin yeterli olmaktadır [10].

Kriptanaliz, bununla birlikte, şifreleme sistemine saldırıların yapılması ve böylelikle kriptosistemin açıklarının belirlenip bu zayıf noktalardan gelebilecek saldırıların önlenmesi amacıyla da kullanılır.

2. PRESENT BLOK ŞİFRELEME

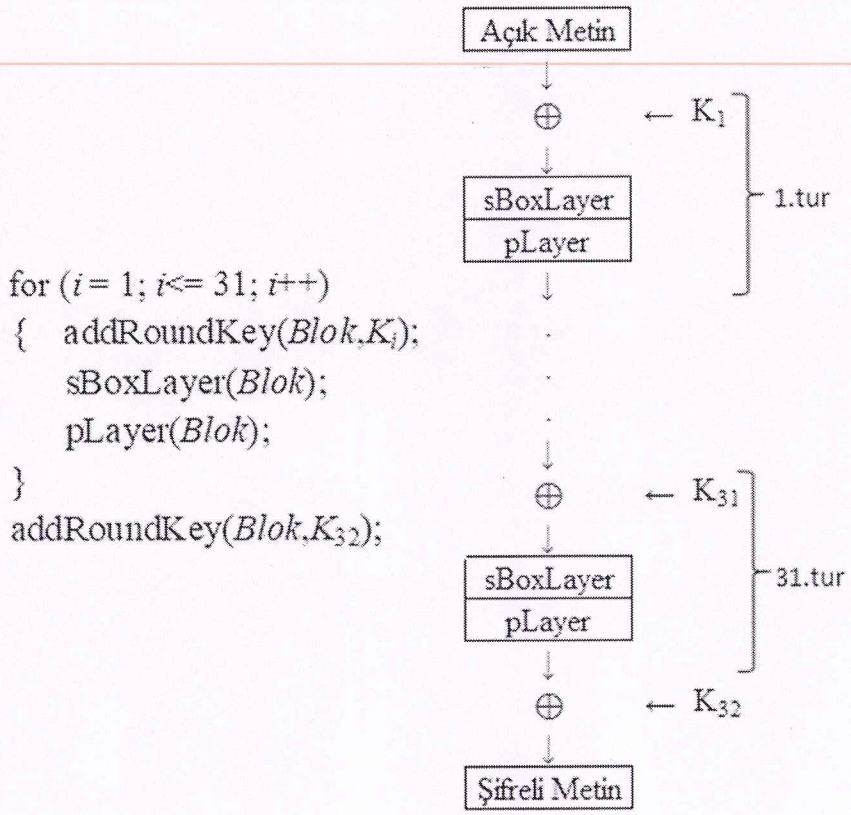
Bir önceki bölümde bahsedilen AES ve RSA gibi şifreleme yöntemlerinden birçoğu, gömülü sistemler, sensör ağları ve temassız akıllı kartlar gibi oldukça kısıtlı donanım kapasitesine sahip sistemler için uygun değildir. Bu nedenle, düşük bellek ve işlemci gücüne sahip kısıtlı ortamlarda gerçekleştirme için tasarlanmış kriptografik algoritma veya protokol anlamına gelen hafif siklet kriptografi, son yıllarda simetrik kriptografinin artan bir ilgi odağı haline gelmiştir. Çalışmamızda ele aldığımız PRESENT algoritması [4], bahsedilen özelliklere sahip kriptografik donanımlar için tasarlanmış bir blok şifreleme algoritmasıdır.

PRESENT blok şifreleme algoritması, Orange Labs (Fransa), Bochum Ruhr Üniversitesi (Almanya) ve Danimarka Teknik Üniversitesi tarafından 2007 yılında geliştirilmiştir. AES ile karşılaştırıldığında, kompakt yapısı ile dikkat çekmektedir [kapı eşdeğeri (GE - gate equivalent) AES'e göre yarısından daha azdır]. PRESENT algoritması, ISO (Uluslararası Standartlar Teşkilâtı - International Organization for Standardization) ve IEC (Uluslararası Elektroteknik Komisyonu - International Electrotechnical Commission) tarafından hafif siklet kriptografik yöntemler için yeni uluslararası standart olarak dâhil edilmiştir.

2.1 PRESENT Şifreleme Algoritması

PRESENT algoritması 31 turluk bir blok şifredir. Bu algoritmada 64-bit uzunluktaki açık veriler, 80 veya 128 bitlik anahtar ile şifrelenerek kapalı veriler elde edilmektedir. Çalışmamızda, 80 bitlik anahtar ile gerçekleme yapılmıştır. PRESENT algoritmasının şifreleme yapısı Şekil 2.1'de gösterilmiştir. Şifrelemenin her bir turu üç aşamada gerçekleştirilmektedir. Bu aşamalar sırasıyla tur anahtarı ekleme (addRoundKey), yer değiştirme (sBoxLayer) ve permütasyon (pLayer) aşamalarıdır. İlk aşamada tur girdisi olan 64 bitlik blok, PRESENT'in anahtar üretme algoritması tarafından üretilen aynı uzunluktaki tur anahtarı K_i ($i = 1, 2, \ldots, 32$) ile XOR işlemine tabi tutulur. Bir sonraki aşamada XOR işleminden çıkan 64 bitlik blok, her biri 4-bit uzunluğunda olan 16 bloğa bölünür ve bölünen bu bloklardaki her 4-bit (Bölüm

2.1.2'de tanımlanacak olan) S-kutusundan geçirilerek sBoxLayer aşamasının çıktısı elde edilir. Elde edilen bu çıktı, son aşamada (Bölüm 2.1.3'te tanımlanacak olan) permütasyondan geçirilir ve böylelikle 64-bit uzunluğundaki tur çıktısı bulunur. Bahsedilen bu tur 31 kere döngüsel biçimde gerçekleştirildikten sonra, döngü çıkışı anahtar algoritmasının ürettiği 32. tur anahtarı (K_{32}) ile XOR edilerek kapalı metin elde edilmiş olur.



Şekil 2.1: PRESENT algoritmasının şifreleme yapısı.

2.1.1 addRoundKey

Bu işlem algoritmanın 31 turu için üretilen 64-bit uzunluğundaki tur anahtarlarının aynı uzunluktaki tur girişleri ve son tur çıkışının da K_{32} ile XOR edilmesi için kullanılır. Şekil 2.1'de *Blok* ile gösterilen tur girişi veya son tur çıkışı (b_{63}, b_{62}, …, b_0), üretilen i. tur anahtarı $1 \leq i \leq 32$ için $K_i = (k^i_{63}, k^i_{62}, \ldots, k^i_0)$ olmak üzere, addRoundKey aşağıdaki işlemi gerçekleştirir:

$$(b_{63}, b_{62}, \ldots, b_0) = (b_{63}, b_{62}, \ldots, b_0) \oplus (k^i_{63}, k^i_{62}, \ldots, k^i_0) \qquad (2.1)$$

2.1.2 sBoxLayer

Bu aşamada, Tablo 2.1'de verilen 4×4 büyüklüğündeki S-kutusu kullanılmaktadır. Bu S-kutusu, 4 biti 4 bite gönderen bir fonksiyondur ve tabloda girdi ve çıktıları onaltılık tabanda gösterilmektedir. addRoundKey işleminden elde edilen 64-bit uzunluğundaki ($b_{63}, b_{62}, \ldots, b_0$) bloğu, 4'er bitlik 16 bloğa ayrılır ve bu blokların her biri Tablo 2.1'de verilen S-kutusuna sokularak sBoxLayer aşamasının çıktısı elde edilir. Diğer bir ifadeyle, $0 \leq i \leq 15$ için $\omega_i = b_{4*i+3} \,||\, b_{4*i+2} \,||\, b_{4*i+1} \,||\, b_{4*i}$ olmak üzere, bu 16 blok ($\omega_{15}, \omega_{14}, \ldots, \omega_0$) ile gösterilebilir; bu durumda, sBoxLayer aşağıdaki işlemi gerçekleştirir:

$$(b_{63}, b_{62}, \ldots, b_0) = (S(\omega_{15}), S(\omega_{14}), \ldots, S(\omega_0)) \tag{2.2}$$

Tablo 2.1: PRESENT şifreleme algoritmasının 4 bitlik S-kutusu.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

2.1.3 pLayer

PRESENT şifrelemesinde kullanılan permütasyon Tablo 2.2'de verilmiştir. Tabloda verilen i permütasyon girişindeki, $P(i)$ ise permütasyon çıkışındaki bit pozisyonlarıdır. Buna göre, bir önceki aşamadan elde edilen bloğun bitleri $0 \leq i \leq 15$ olmak üzere aşağıdaki eşitlikler vasıtasıyla permütasyona uğrar:

$$b_i = b_{4*i} \tag{2.3}$$

$$b_{i+16} = b_{4*i+1} \tag{2.4}$$

$$b_{i+32} = b_{4*i+2} \tag{2.5}$$

$$b_{i+48} = b_{4*i+3} \tag{2.6}$$

Tablo 2.2: PRESENT şifreleme algoritmasının permütasyonu.

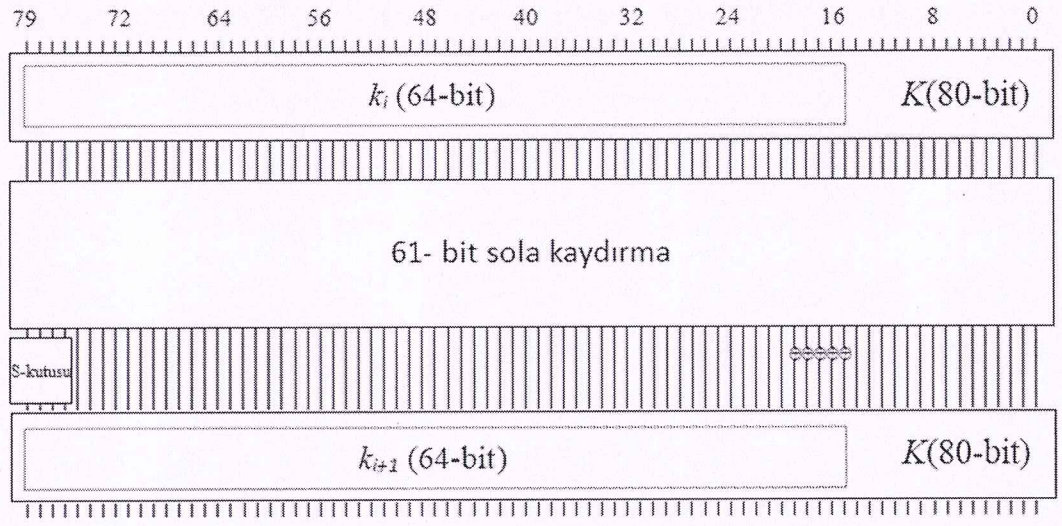
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

2.1.4 Anahtar Üretme Algoritması

Anahtar üretme algoritması, 80-bit uzunluğundaki algoritmanın gizli anahtarını kullanarak tur anahtarlarını üretmektedir. Bu gizli anahtarı $K = (k_{79}, k_{78}, \ldots, k_0)$ vektörü ile gösterelim. İlk tur anahtarı K_1, K vektörünün solundaki 64 bitdir; diğer bir ifadeyle, $K_1 = (k^i_{63}, k^i_{62}, \ldots, k^i_0) = (k_{79}, k_{78}, \ldots, k_{16})$. $2 \leq i \leq 32$ için diğer tur anahtarları K_i, aşağıdaki adımların gerçekleştirilmesi sonucunda elde edilen güncellenmiş K vektörünün solundaki 64 biti alınarak elde edilir:

1. $(k_{79}, k_{78}, \ldots, k_0) = (k_{18}, k_{17}, \ldots, k_{19})$
2. $(k_{79}, k_{78}, k_{77}, k_{76}) = S(k_{79}, k_{78}, k_{77}, k_{76})$
3. $(k_{19}, k_{18}, k_{17}, k_{16}, k_{15}) = (k_{19}, k_{18}, k_{17}, k_{16}, k_{15}) \oplus$ tur_sayacı

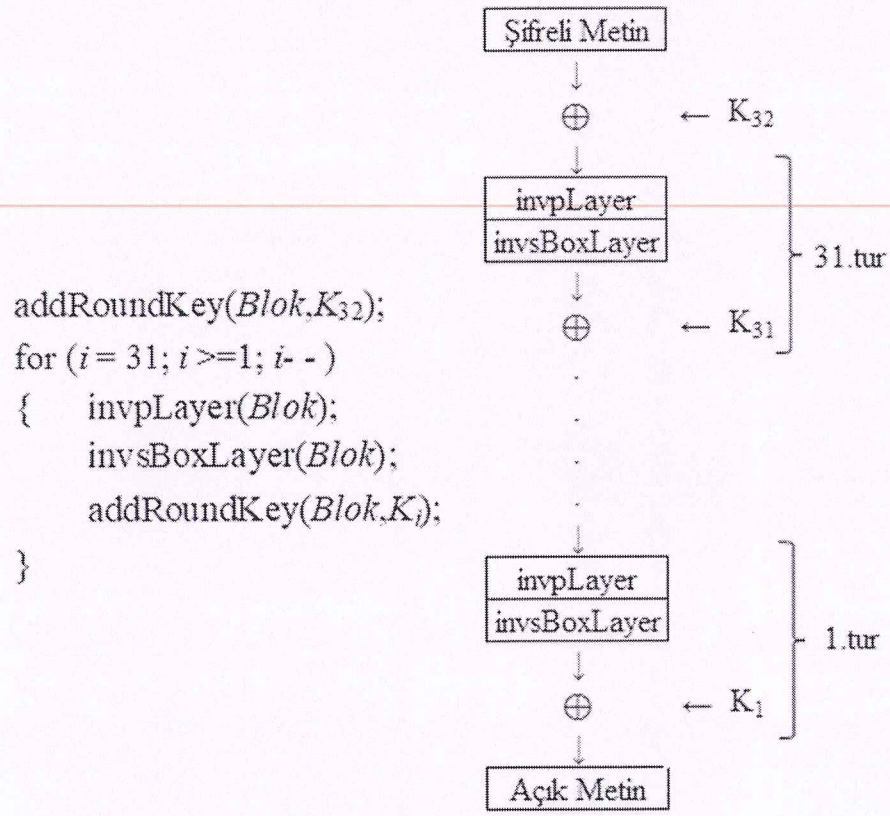
Yukardaki adımların ilkinde, 80 bitlik anahtar önce 61 bit sola kaydırılır. Bir sonraki adımda, en soldaki 4-bit S-kutusundan geçirilerek bu bitler değiştirilir. Son olarak, tur sayacı (i'nin ikili tabandaki değeri) K vektörünün (k_{19}, k_{18}, k_{17}, k_{16}, k_{15}) bitleri ile XOR edilir. Bu adımlar, görsel olarak Şekil 2.2'de gösterilmiştir.



Şekil 2.2: Anahtar üretme algoritması.

2.2 PRESENT Şifre Çözme Algoritması

PRESENT şifre çözme algoritması Şekil 2.4'te gösterildiği gibi şifreleme işlemlerinin tersten sırayla gerçekleştirilmesiyle yapılmaktadır.



Şekil 2.3: PRESENT algoritmasının şifre çözme yapısı.

Şekil 2.4'te gösterilen invsBoxLayer Tablo 2.3'te verilen (ve Tablo 2.1'deki S-kutusunun tersi olan) S^{-1} ile gösterilen S-kutusunu kullanırken, invpLayer Tablo 2.4'te verilen (ve Tablo 2.2'deki permütasyonun tersi olan) P^{-1} ile gösterilen permütasyonu kullanmaktadır.

Tablo 2.3: PRESENT şifre çözme algoritmasının 4 bitlik ters S-kutusu.

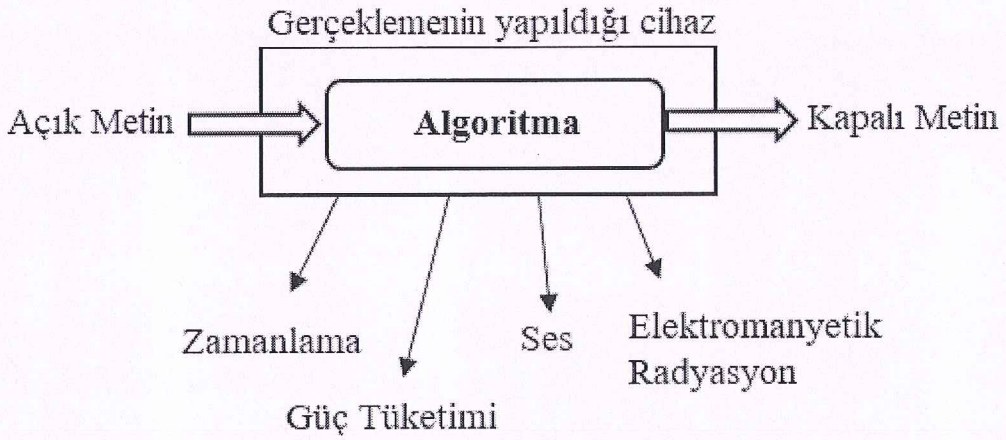
x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S^{-1}(x)$	5	E	F	8	C	1	2	D	B	4	6	3	0	7	9	A

Tablo 2.4: PRESENT şifre çözme algoritmasının ters permütasyonu.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P^{-1}(i)$	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P^{-1}(i)$	1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P^{-1}(i)$	2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P^{-1}(i)$	3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63

3. YAN KANAL ANALİZİ VE MATLAB GERÇEKLENMESİ

Anahtarı bulma işlemi sadece algoritmanın matematiksel yönden zayıflığına bağlı değildir. Gerçeklemenin yapıldığı kriptografik donanımların sızdırdığı yan kanal bilgileri sayesinde de anahtarı elde etmek veya şifrelenmiş metni çözmek mümkündür. Kriptografik donanımlar dışarıya birçok yan kanal bilgisi (zamanlama, ses, güç tüketimi ve elektromanyetik radyasyon gibi) sızdırırlar (Şekil 3.1). Bahsedilen yan kanal bilgilerini kullanarak yapılan kriptoanalize yan kanal analizi (YKA) denmektedir. Yan kanal analizi, aynı algoritma kullanılsa bile kriptografik donanıma bağlı olarak farklı biçim ve değerlerde yan kanal bilgisi sızdırdıkları için gerçekleştirildiği donanıma özeldir. Bununla birlikte, farksal [9] ve doğrusal [11] kriptanaliz gibi sadece şifreleme algoritmasının matematiksel tanımına bağlı bir kriptanaliz türü olmadığından, genellikle kriptografik donanıma uygulanarak yürütülür.



Şekil 3.1: Yan kanal sızıntıları.

Çalışmamızda, Bölüm 4'te anlatılacak olan ölçüm düzeneği vasıtasıyla elde edilen güç ölçümleri kullanılarak, en güçlü yan kanal analizi türlerinden olan Farksal Güç Analizi (FGA) [15] blok şifreleme algoritması PRESENT için uygulanacak ve atağın başarısı YKA güvenlik ölçütü olan tahmin entropisi ile değerlendirilecektir.

3.1 Farksal Güç Analizi

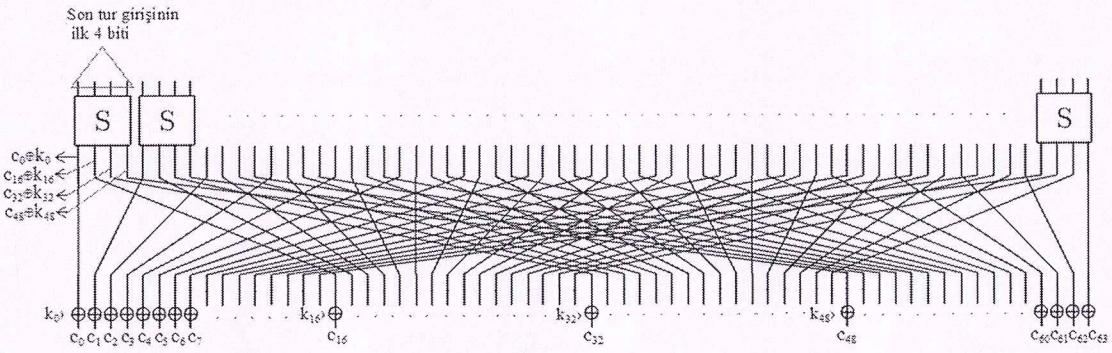
Kriptografik bir cihaz, şifreleme algoritmasını çalıştırırken algoritmanın kullandığı veriye bağlı olarak güç tüketir. Bir FGA saldırısında, bu güç tüketimleri ölçülerek cihazın kullandığı gizli veri (anahtar) elde edilmeye çalışılır. Bu saldırıda, güç ölçüm değerleri Hamming ağırlığı veya Hamming uzaklığı gibi güç modelleri kullanılarak tahmin edilir ve bu tahminin gerçek güç ölçümü ile istatistiksel ilişkisine bakılarak kriptografik donanımda kullanılan anahtar elde edilmeye çalışılır. Burada Hamming ağırlığı herhangi bir bit dizisindeki 1'lerin sayısıdır. Hamming uzaklığı ise aynı uzunlukta iki bit dizisinde aynı pozisyonda farklı olan 0 ve 1'lerin sayısıdır. Örneğin, 5-bit uzunluğundaki 01100 dizisinin Hamming ağırlığı iki iken, bu bit dizisinin aynı uzunluktaki 11001 dizisine olan Hamming uzaklığı üçtür (birinci, üçüncü ve beşinci pozisyondaki 0 ve 1'ler farklıdır).

Çalışmamızda, Hamming ağırlığı modeline göre daha verimli olan Hamming uzaklığı modeline dayanan korelasyon analizi [16] yürütülmüştür. Hamming uzaklığı modeli genel olarak herhangi bir şifreleme algoritmasının bir turu gerçekleştiğinde harcanan gücün, o turun girişindeki ve çıkışındaki blokların Hamming uzaklığı ile orantılı olduğu varsayımına dayanır. Çok bitli farksal güç analizinin verimli bir versiyonu olan korelasyon analizinde, kriptografik donanımdan elde edilen gerçek güç ölçümleri ile bu donanımda işlenen veriden elde edilen güç tüketim tahminleri arasındaki ilişkinin doğrusal olduğu kabul edilir. Bu nedenle, eğer güç tüketim tahminini elde etmek için kullanılan aday anahtar doğru anahtar ise bahsedilen korelasyon en yüksek olur.

PRESENT blok şifreleme algoritmasının son turda kullanılan tur anahtarını elde etmeye çalıştığımızı düşünelim. Bahsedilen Hamming uzaklığı ile güç ölçümü tahminini gerçekleştirebilmek için tur girişini bilmemiz gerekir. Ancak tur girişi, kullanılan tur anahtarına bağlı olduğu için, bunu tur anahtarını tahmin ederek yapabiliriz. Bu tahminler arasında doğru tur anahtarı, daha önce bahsedildiği gibi gerçek güç ölçümleri ile kullanılan model arasındaki istatiksel ilişkiye bakılarak bulunmaktadır.

Şekil 3.2'de gösterildiği gibi, son tur çıkışı olan kapalı metnin ilk dört bitini kullanarak tur anahtarının dört bitini bulmayı hedeflediğimizi varsayalım (saldırının gerçekleşebilmesi için kapalı metini biliyor olmamız gerekmektedir). Kapalı metin

$C = (c_0, c_1, c_2, \ldots\ldots, c_{63})$ ve son tur anahtarı $K = (k_0, k_1, k_2, \ldots\ldots, k_{63})$ olsun. Kapalı metine "invpLayer" permutasyonu uygulanarak elde edilen bloğun ilk 4 biti (c_0, c_{16}, c_{32}, c_{48}) olur. Bu dört bite, karşılık gelen tur anahtarının 4-biti (k_0, k_{16}, k_{32}, k_{48}) XOR işlemi ile eklendiğinde, elde edilen ($c_0 \oplus k_0$, $c_{16} \oplus k_{16}$, $c_{32} \oplus k_{32}$, $c_{48} \oplus k_{48}$) sonucunu veren S-kutusu girişi (S-kutusu dönüşümünün tersi uygulanarak elde edilebilir) son tur girişinin ilk dört bitini verir. Burada bahsedilen 4 bitlik tur anahtarını bilmediğimiz için olası bütün adaylar ($2^4 = 16$ tane) tahmin edilir. Bu ise, son tur girişinin ilk 4-biti için 16 tane tahmin yaptığımız anlamına gelmektedir. Son tur çıkışını (kapalı metni) bildiğimizi varsaydığımızdan, elde edilen tahminlerin her birinin kapalı metnin ilk 4 biti olan (c_0, c_1, c_2, c_3) vektörüne Hamming uzaklığı gerçek güç tüketimi için yapılan bir tahmin olmaktadır. Uyguladığımız Hamming uzaklığına dayanan korelasyon analizine göre, bu tahminlerden gerçek güç tüketimleri ile korelasyonu en yüksek olan aday kullanılan gerçek 4 bitlik tur anahtarı olarak elde edilir.



Şekil 3.2: Son tur anahtarı için Hamming uzaklık modelinin uygulanması.

Yan kanal analizinin uygulanacağı kriptografik donanımdan N tane güç ölçümü alındığını ve her bir güç ölçümü için donanımın ürettiği kapalı metinleri bildiğimizi varsayalım. Korelasyon analizini yürütmek için öncelikle kapalı metinlerin her birine karşılık gelen olası bütün güç ölçümü tahminleri yapılır. Örneğin, yukarıda bahsedildiği gibi son tur anahtarının 4 biti elde edilmek istendiğinde, son tur girişinin 4 biti için 16 aday olacağından her bir kapalı metinden 16 güç ölçümü tahmini bulunur. Genel olarak aday sayısına A dersek, N×A büyüklüğünde bir tahmin matrisi oluşturulur. Bu matris H olsun. Şifreleme süresi boyunca alınan güç ölçümünün B örneklemden oluştuğunu varsayarsak, donanımdan elde edilen gerçek güç ölçümleri ile de N×B büyüklüğünde bir başka matris oluşturulur. Bu matrise de T diyelim. Korelasyon analizinde, H matrisin her bir

kolonu h_i (i = 1, ..., A) için T matrisinin bütün kolonları t_j (j = 1, ..., B) ile olan korelasyonu aşağıdaki formül ile hesaplanır:

$$R_{i,j} = \frac{\sum_{k=1}^{N}(h_{k,i}-\bar{h}_i)(t_{k,j}-\bar{t}_j)}{\sqrt{\sum_{k=1}^{N}(h_{k,i}-\bar{h}_i)^2 \sum_{k=1}^{N}(t_{k,j}-\bar{t}_j)^2}} \quad (3.1)$$

Burada $\bar{h}_i$ ve $\bar{t}_j$, sırasıyla H ve T matrislerin i. ve j. kolonlarının ortalama değerleridir. Verilen denklem ile elde edilen A×B büyüklüğündeki R matrisi gerçek güç ölçümleri ve Hamming uzaklığı modeli ile tahmin edilen güç ölçümleri arasındaki istatiksel ilişkiyi veren bir korelasyon matrisidir. R matrisinin her bir satırı tahmin edilen bir anahtar (örneğin, daha önce bahsedilen PRESENT algoritması durumunda son tur anahtarının 4 biti) için korelasyon profili oluşturur. Doğru tahmin edilen anahtar için bu profil yüksek tepe değerlere sahip olacaktır. Böylelikle hangi tahminin doğru anahtar olduğuna karar verilir.

Bu çalışmada bir blok şifrenin yan kanal analizine karşı dayanıklılığını ölçen bir metrik olan tahmin entropisi kullanılmıştır. Tahmin entropisi belli bir sayıda güç ölçümü ile doğru anahtarın elde edilebilmesi için denenmesi gereken aday anahtarlarının ortalama sayısını gösteren bir ölçüt olarak tanımlanır. Bu ölçüt korelasyon profiline göre denenmesi gereken anahtarlar arasında doğru anahtarın kaçıncı sırada olduğu bulunarak hesaplanır.

3.2 MATLAB Gerçeklemesi

Yukarıda bahsedilen Hamming uzaklığına dayalı korelasyon analizini gerçekleyerek (PRESENT blok şifreleme algoritması için) tahmin entropisini bulan MATLAB kodu aşağıda verilmektedir. Kodda yer alan "ciphertexts.txt" dosyası, bilgisayar tarafından rastgele üretilen açık metinlere karşılık gelen kapalı metinleri içermektedir ve "SASEBO_G_Checker.exe" programı tarafından kaydedilir. "waveform_data.csv" dosyasında ise gerçek güç ölçümleri bulunmaktadır. Bu güç ölçümleri, ölçüm düzeneğindeki cihazların senkronize bir şekilde çalışmasını sağlayan ve bölüm 4'te verilen kodun icra edilmesi ile oluşturulur. Yaptığımız ölçümlerde bir şifreleme süresinin 15µs içinde gerçekleştirildiği görülmüştür. Kullandığımız osiloskop 2.5GS/s hızında örnekleme yaptığından, bir güç ölçümü 37500 örneklemden oluşmaktadır. Bu nedenle, kodda verilen güç ölçümlerinin

oluşturduğu TR matrisi 5000×37500 büyüklüğündedir; burada 5000 sayısı hem kapalı metinlerin hem de güç ölçümlerin sayısıdır. Ayrıca, MATLAB kodunda son tur anahtarının sadece 4 biti elde edilmeye çalışıldığından, kapalı metinlerin kullanılması ile elde edilen güç tüketim tahmin matrisi H'nin büyüklüğü 5000×16 olmaktadır. Pratikte 37500 örneklemin hepsinin kullanılması yerine, son turun tur anahtarı tahmin edilmeye çalışıldığından, şifreleme esnasında sadece son turun gerçekleştiği aralıktaki örneklemlerin kullanılması yeterlidir. Bu aralık güç ölçümleri incelenerek bulunabilir ve verilen MATLAB kodunda "t1" ve "t2" değişkenleri kullanılarak tanımlanabilir.

Bu kodda, tahmin entropisi her 100 ölçümde bir doğru anahtarın bulunması için denenmesi gereken aday anahtarların sayısını vermektedir. Doğru anahtarın tahmin entropisi için, ölçüm sayısı arttıkça bu aday anahtarların sayısının azalmasını ve sonunda sıfıra düşmesi beklenir. Diğer yanlış aday anahtarların tahmin entropileri için böyle bir durumun gerçekleşemeyeceği görülmelidir. Gerçekte, doğru anahtar böylelikle belirlenmiş olur. Verilen kodda bahsedilen tahmin entropisi "rx" dizisi yardımıyla elde edilmektedir.

```
clear all
%Kapalı metinleri yükler
load ciphertexts.txt

N=16;
n=4;

%pLayer
PR=[0 16 32 48 1 17 33 49 2 18 34 50 3 19 35 51 4 20 36 52 5 21 37 53 6 22 38 54 7
23 39 55 8 24 40 56 9 25 41 57 10 26 42 58 11 27 43 59 12 28 44 60 13 29 45 61 14
30 46 62 15 31 47 63] + 1;
%invpLayer
PI=[0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 1 5 9 13 17 21 25 29 33 37 41 45
49 53 57 61 2 6 10 14 18 22 26 30 34 38 42 46 50 54 58 62 3 7 11 15 19 23 27 31 35
39 43 47 51 55 59 63] +1;

k=0;
for i=1:5000
   k=k+1;
   CX(k,:)=ciphertexts(1+(i-1)*8:i*8);
end

%sBox
Shex='C56B90AD3EF84712';
for i=1:16
```

```
    t=hex2dec(Shex(i));
    S(i)=t;
    Sb(i,:)=dobin(t,4);% S-kutusunun ikili versiyonu
    Sinv(t+1)=i-1; %invsBox
end

% İstenilen örneklem aralığını seçer
t1=1;
t2=37500;

%Güç ölçümlerini yükler
load waveform_data.csv;
k=0;
for i=1:5000
    k=k+1;
    tx=waveform_data(1+(i-1)*37500:i*37500);
    TR(k,:)=tx(t1:t2); % Gerçek güç ölçümlerinden oluşan matris
end

% Gizli anahtar
Kth='00010203040506070809';

Kt=[];
for i=1:20
    Kt=[Kt dobin(hex2dec(Kth(i)),4)];
end

K=Kt;

% Güç ölçüm sayısı
TN=5000;

% Tahmin entropisini (eklenen) her 100 ölçümde yeniler
stp=100;

% Örneklem sayısı
Ts=t2-t1+1;

% Tur anahtarlarını üretir
R(1,:)=K(1:80);
for i=2:32
    K=[K(62:80) K(1:61)];
    t=todec(K(1:4));% 4 biti onluk tabana çevirir
    K(1:4)=Sb(t+1,:);
    rc=dobin(i-1,5);
    K(61:65)=xor(K(61:65),rc);
    R(i,:)=K(1:80);
end

Kc=R(32,PR(1:4));
```

```
% Tahmin edilmesi beklenen gerçek anahtar
kcd=todec(Kc);

% Aday anahtarlar
for i=1:N
   Ka(i,:)=dobin(i-1,n);
end

% Tahmin matrisi
H(1:TN,1:N)=0;

% Tahmin matrisini hesaplar
for I=1:TN
   Cd=CX((ST-1)*TN+I,:);
   C=[];
   for i=1:8
      C=[C dobin(Cd(i),8)];
   end

   So=C(PR(1:4));
   Soc=C(1:4);

   for i=1:N
      to=xor(Ka(i,:),So);
      t=Sinv(todec(to)+1);
      H(I,i)=sum(xor(dobin(t,4),Soc));
   end
   if rem(I,1000)==0
      [1 ST I]
   end
end

rx(1:N,1:(TN-stp)/stp+1)=0;
cnt=0;
CFa=[];

% Her 100 ölçüm için tahmin entropisini hesaplar
for pn=stp:stp:TN
   if rem(pn,1000)==0
      [2 ST pn]
   end
   for i=1:N
      h=H(1:pn,i);
      mh=sum(h)/pn;
      h=h-mh;
      hs=h'*h;
      for j=1:Ts
         t=TR(1:pn,j);
         mt=sum(t)/pn;
```

```
            t=t-mt;
            ts=t'*t;
            CF(i,j)=(h'*t)/sqrt(hs*ts); % Korelasyon matrisi
        end
        CFm(i)=max(abs((CF(i,:))));
    end
    [a ix]=sort(CFm,'descend');
    cnt=cnt+1;
    rx(ix,cnt)=(1:N)'-1;
end

plot(rx(kcd+1,:));
xlabel('Ölçüm Sayısı (10^2)')
ylabel('Tahmin entropisi')

% Onluk tabandaki K sayısını k bite dönüştürür
function Y=dobin(K,k)

A=dec2bin(K,k);
X=find(A=='1');
[m,n]=size(X);
Y(1:k)=0;
if n > 0
   for j=1:n
      Y(X(j))=1;
   end
else
   Y(1:k)=0;
end

% Bit dizisi L'yi onluk taban değerine dönüştürür
function M=todec(L)

N=0;
i=1;
for n=length(L)-1:-1:0
   N=N+L(i)*(2^n);
   i=i+1;
end
M=N;
```

4. ÖLÇÜM DÜZENEĞİ VE GERÇEKLEME

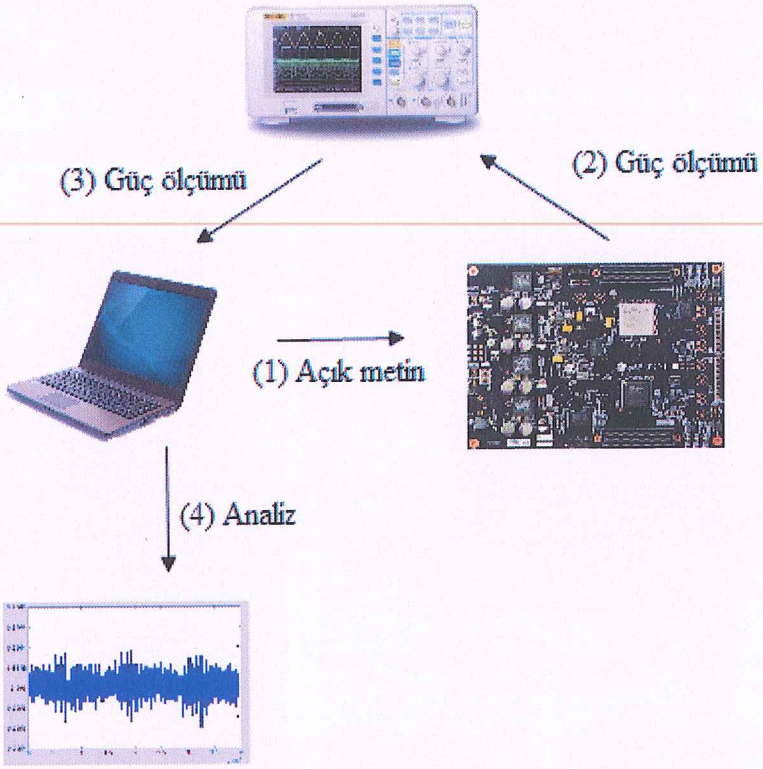
Bu bölümde, yan kanal analizini yürütmek için gereken güç ölçümlerini almak amacıyla kullandığımız SAKURA-X [12] (yan kanal analizi standart değerlendirme kartı), Keysight MSO-X 4104A osiloskop ve bilgisayardan oluşan ölçüm düzeneği ile bu ölçüm düzeneğinin uygun şekilde çalışmasını sağlayan kodlar anlatılacaktır.

4.1 Ölçüm Düzeneği

Ölçüm düzeneği, kullanılan cihazların çalışma sırası ile birlikte Şekil 4.1' de gösterilmiştir. Şekil 4.1'deki çalışma sırası döngüsel olarak tekrar edilir. Bununla birlikte, bilgisayar tarafından bu döngüye girmeden önce ölçüm parametreleri (kanal ayarları, tetikleme modu gibi) osiloskopa ve şifrelemede kullanılacak olan gizli anahtar da SAKURA-X kriptografik kartına gönderilir. Gönderilen bu başlangıç parametrelerinden sonra, Şekil 4.1'de görüldüğü gibi, öncelikle rasgele üretilen bir açık metin bilgisayardan SAKURA-X kartına gönderilir ve şifreleme işlemi hem bilgisayarda hem de kriptografik kartta gerçekleştirilir. SAKURA-X kartında şifreleme gerçekleştiği esnada oluşan güç tüketimi, osiloskop ile ölçülür ve bu güç tüketimi bilgisayara kaydedilir. SAKURA-X kartında şifreleme sonucu elde edilen kapalı metin aynı zamanda bilgisayara da gönderilerek, bilgisayarda elde edilen kapalı metin ile karşılaştırılır. Bu işlem döngüsel olarak kaç adet güç tüketim ölçümü isteniyorsa o kadar tekrar edilir.

Ölçüm düzeneğinde kullanılan SAKURA-X (SASEBO-GIII olarak da isimlendirilmektedir) kartı Spartan-6 (kontrol FPGA'i) ve Kintex-7 (kriptografik FPGA) olmak üzere iki FPGA içermektedir. Kintex-7 kriptografik algoritmayı gerçeklemek için kullanılırken, Spartan-6 kriptografik FPGA ile haberleşerek kontrol etmek amaçlı kullanılmaktadır. Düzenekte, SAKURA-X kartından osiloskop ile ölçülen güç tüketimleri, 50 Ω değerindeki SMA-BNC kablo vasıtasıyla alınmaktadır. Bu güç tüketim değerleri osiloskop ile 2.5GS/s örnekleme hızında örneklenmekte ve bilgisayara da bu şekilde kaydedilmektedir. Döngüsel olarak çalışan ölçüm

düzeneğinden elde edilen güç ölçümleri, MATLAB'da Hamming uzaklığına dayanan korelasyon analizi programı (Bölüm 3'te verilmektedir) tarafından kullanılarak, şifreleme algoritmasının kullandığı gizli anahtar elde edilmeye çalışılmıştır.



Şekil 4.1: Ölçüm düzeneği.

Ölçüm düzeneğindeki kriptografik donanım Şekil 4.2'de görülmektedir. Bu şekilde numaralandırılan kısımların kullanım amacı aşağıda belirtildiği gibidir:

1- Bilgisayar (USB) bağlantısı ve güç kaynağı : SAKURA-X kriptografik donanım ile PC arasındaki haberleşme ve SAKURA-X kartının güç beslemesi bu bağlantı ile sağlanır.

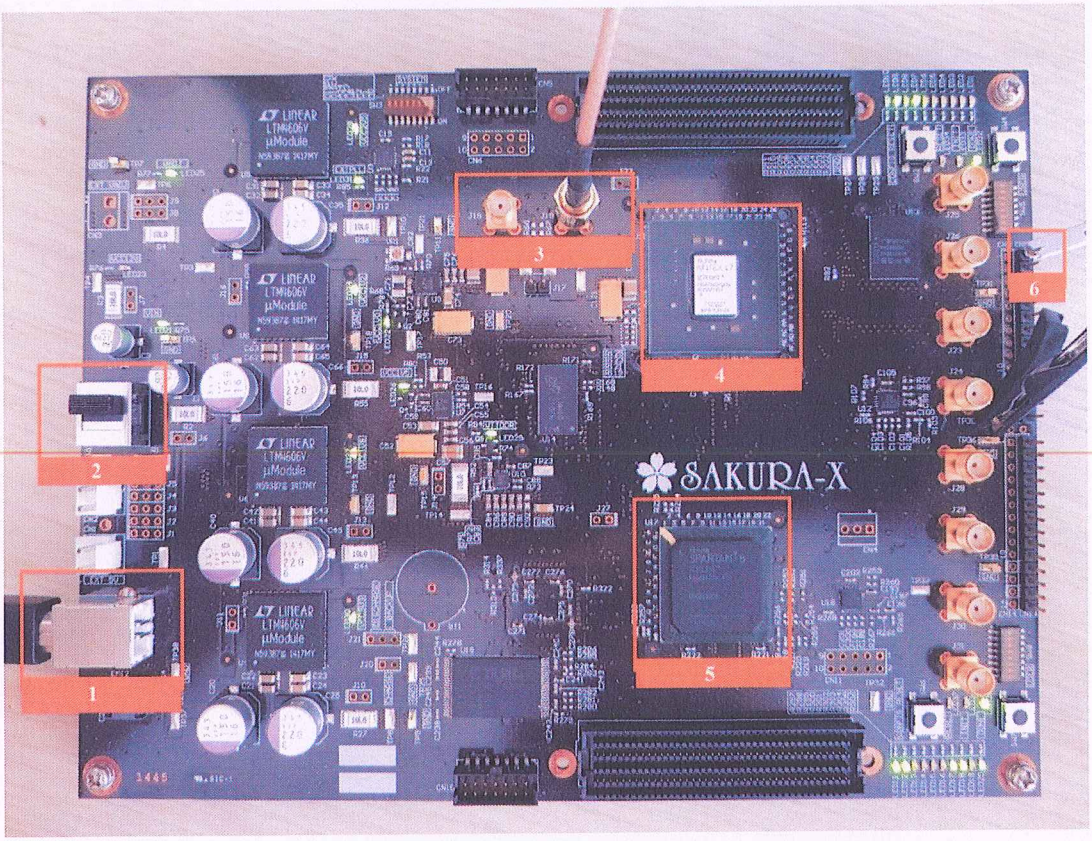
2- Açma kapama anahtarı

3- Ölçüm noktası

4- Kriptografik FPGA: XC7K160T – KINTEX-7

5- Kontrol FPGA: XC6SLX45 – SPARTAN-6

6- Tetikleme sinyalinin alındığı pin#1



Şekil 4.2: SAKURA-X kriptografik donanım.

4.2 Gerçekleme

Ölçüm düzeneğinde kullanılan osiloskopun başlangıç ayarlarının yapılması, şifreleme işlemi her gerçekleştiğinde ölçümün alınması ve bilgisayara kaydedilmesi işlemlerinin senkronize bir şekilde yürütülmesi için (osiloskopun üreticisi olan) Keysight Technologies şirketinin "Keysight Infiniium Oscilloscopes Programmer's Guide" [13] adlı dokümanında sağladığı komutlardan faydalanılmıştır. Bahsedilen hedefleri gerçekleştirmek için bu dokümanın 1732-1741 sayfaları arasında C programlama dilinde verilen örnek kod aşağıda verildiği gibi değiştirilmiştir.

```
#include <stdio.h> /* For printf(). */
#include <string.h> /* For strcpy(), strcat(). */
#include <time.h> /* For clock(). */
#include <visa.h> /* Keysight VISA routines. */
#define VISA_ADDRESS "TCPIP0::A-MX4104A-30328::inst0::INSTR"
#define IEEEBLOCK_SPACE 5000000
```

```
/* Function prototypes */
void initialize(void); /* Initialize to known state. */
void capture(void); /* Capture the waveform. */
void do_command(char *command); /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query); /* Query for string. */
void do_query_number(char *query); /* Query for number. */
void do_query_numbers(char *query); /* Query for numbers. */
int do_query_ieeeblock(char *query); /* Query for IEEE block. */
void check_instrument_errors(); /* Check for inst errors. */
void error_handler(); /* VISA error handler. */

/* Global variables */
ViSession defaultRM, vi; /* Device session ID. */
ViStatus err; /* VISA function return value. */
char str_result[256] = { 0 }; /* Result from do_query_string(). */
double num_result; /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE]; /*Result from do_query_ieeeblock().
*/
double dbl_results[10]; /* Result from do_query_numbers(). */

/* Main Program* ------------------------------------------------------------ */
void main(void)
{       /* Open the default resource manager session. */
        err = viOpenDefaultRM(&defaultRM);
        if (err != VI_SUCCESS) error_handler();
        /* Open the session using the oscilloscope's VISA address. */
        err = viOpen(defaultRM, VISA_ADDRESS, VI_NULL, VI_NULL, &vi);
        if (err != VI_SUCCESS) error_handler();
        /* Set the I/O timeout to fifteen seconds. */
        err = viSetAttribute(vi, VI_ATTR_TMO_VALUE, 15000);
        if (err != VI_SUCCESS) error_handler();
        /* Clear the interface. */
        err = viClear(vi);
        if (err != VI_SUCCESS) error_handler();

        /* Initialize - start from a known state. */
        initialize();
        /* Capture data. */
        capture();

        double x_increment;
        double x_origin;
        double x_reference;
        double y_increment;
        double y_origin;
        double y_reference;
        double tmp;
```

```
FILE *fp;
int num_bytes; /* Number of bytes returned from instrument. */
int i, I, K = 0, NWF, stp;
NWF = 150000;
stp = 30;

double *WAVE_M = (double *)malloc(40000 * sizeof(double));
for (i = 0; i < 40000; i++)          WAVE_M[i] = 0;

/* Download waveform data.
/* Set the waveform points mode. */
do_command(":WAVeform:POINts:MODE RAW");
do_query_string(":WAVeform:POINts:MODE?");
printf("Waveform points mode: %s\n", str_result);
/* Get the number of waveform points available. */
do_query_string(":WAVeform:POINts?");
printf("Waveform points available: %s\n", str_result);
/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURce?");
printf("Waveform source: %s\n", str_result);
/* Choose the format of the data returned (WORD, BYTE, ASCII): */
do_command(":WAVeform:FORMat BYTE");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);
/* Display the waveform settings: */
do_query_numbers(":WAVeform:PREamble?");
x_increment = dbl_results[4];
printf("Waveform X increment: %e\n", x_increment);
x_origin = dbl_results[5];
printf("Waveform X origin: %e\n", x_origin);
x_reference = dbl_results[6];
printf("Waveform X reference: %e\n", x_reference);
y_increment = dbl_results[7];
printf("Waveform Y increment: %e\n", y_increment);
y_origin = dbl_results[8];
printf("Waveform Y origin: %e\n", y_origin);
y_reference = dbl_results[9];
printf("Waveform Y reference: %e\n", y_reference);

ShellExecuteA(0, "open", // Operation to perform
        "SASEBO_G_Checker.exe", NULL,// Application name
        0, // Default directory
        SW_SHOW);
Sleep(2000);
/* Open file for output. */
fp = fopen("c:\\scope\\data\\waveform_data.csv", "ab");
```

```
        for (I = 0; I <= NWF; I++)
        {
                do_command(":SINGLE");
                Sleep(200);
                do_query_number("*OPC?");
                printf("\nI=%d OPC: %f", I, num_result);

                /* Read waveform data. */
                num_bytes = do_query_ieeeblock(":WAVeform:DATA?");
                if (K < stp)
                {

                        for (i = 0; i < num_bytes; i++)
                        {
        tmp=(((float)ieeeblock_data[i]-y_reference)*y_increment)+y_origin;
                                WAVE_M[i] = WAVE_M[i] + tmp;
                                K = K + 1;
                        }
                else
                {
                        for (i = 0; i < num_bytes; i++)
                {
                                /* Write voltage value. */
                                fprintf(fp, "%6f ", WAVE_M[i] / ((double)stp));
                        }
                        K = 1;
                        for (i = 0; i < num_bytes; i++)
WAVE_M[i]=(((float)ieeeblock_data[i]-y_reference)*y_increment)+y_origin;
                }
        }
        /* Close output file. */
        fclose(fp);
        printf("\nWaveform format BYTE data written to ");
        printf("c:\\scope\\data\\waveform_data.csv.\n");
        /* Close the vi session and the resource manager session. */
        viClose(vi);
        viClose(defaultRM);
        return;
}
/* Capture the waveform.
void capture(void)
{
        /* Use auto-scale to automatically configure oscilloscope. */
        do_command(":AUToscale");
        /* Set trigger mode (EDGE, PULSe, PATTern, etc., and input source. */
        do_command(":TRIGger:MODE EDGE");
        do_query_string(":TRIGger:MODE?");
        printf("Trigger mode: %s\n", str_result);
```

```
    /* Set EDGE trigger parameters. */
    do_command(":TRIGger:EDGE:SOURce CHANnel2");
    do_query_string(":TRIGger:EDGE:SOURce?");
    printf("Trigger edge source: %s\n", str_result);
    do_command(":TRIGger:EDGE:LEVel 2");
    do_query_string(":TRIGger:EDGE:LEVel?");.
    printf("Trigger edge level: %s\n", str_result);
    do_command(":TRIGger:EDGE:SLOPe NEGative");
    do_query_string(":TRIGger:EDGE:SLOPe?");
    printf("Trigger edge slope: %s\n", str_result);

    /* Change settings with individual commands:
    /* Set vertical scale and offset. */
    do_command(":CHANnel2:SCALe 1");
    do_query_string(":CHANnel2:SCALe?");
    printf("Channel 2 vertical scale: %s\n", str_result);
    do_command(":CHANnel2:OFFSet -0.275");
    do_query_string(":CHANnel2:OFFSet?");
    printf("Channel 2 offset: %s\n", str_result);
    do_command(":CHANnel1:SCALe 0.001");
    do_query_string(":CHANnel1:SCALe?");
    printf("Channel 1 vertical scale: %s\n", str_result);
    do_command(":CHANnel1:OFFSet 0.9960");
    do_query_string(":CHANnel1:OFFSet?");
    printf("Channel 1 offset: %s\n", str_result);
    /* Set horizontal scale and offset. */

    /* Set horizontal scale and offset. */
    do_command(":TIMebase:SCALe 0.0000015");
    do_query_string(":TIMebase:SCALe?");
    printf("Timebase scale: %s\n", str_result);
    do_command(":TIMebase:POSition 0.00000670");
    do_query_string(":TIMebase:POSition?");
    printf("Timebase position: %s\n", str_result);
}
```

Verilen kodda bulunan initialize, do_command, do_command_ieeeblock, do_query_string, do_query_number, do_query_numbers, do_query_ieeeblock, check_instrument_errors, error_handler adlı fonksiyonlar daha önce bahsedilen [13] kaynağında verilen fonksiyonların aynısıdır. Bu nedenle yukarıdaki kodda bu fonksiyonların içeriği ayrıca verilmemiştir. Capture fonksiyonu ise osiloskopun başlangıç ayarlarını içermektedir. Bir şifreleme süresince alınan ölçümün osiloskopta uygun bir şekilde elde edilmesini ve böylelikle bu ölçümlere dayanan yan kanal analizinin doğru şekilde yürütülmesini sağlamak amacıyla verilen kodda belirtildiği gibi değiştirilmiştir.

Ölçüm düzeneğindeki cihazların döngüsel olarak senkronize bir şekilde çalışmasını sağlamak için yazdığımız kodda bir "for" döngüsü oluşturulmuştur. Bu döngü istenilen toplam ölçüm sayısını temsil eden "NWF" değişkeninin değeri adedince çalışmaktadır. Döngüde kullandığımız diğer bir değişken "stp" ise aynı açık metin ve anahtar için alınan ölçüm sayısını belirtir. Aynı açık metin ve anahtar için "stp" adedince yapılan ölçümlerin ortalaması alınarak "WAVE_M" dizisine atılmaktadır. Sonrasında bu ortalama değer "waveform_data.csv" dosyasına aktarılmaktadır. Sonuç olarak, program sonlandığında yani istenilen toplam ölçüm sayısına ulaşıldığında "waveform_data.csv" dosyasında "NWF/stp" adedince güç ölçümü bulunur.

Şifrelemenin bilgisayarda yapılmasını ve açık metinin SAKURA-X kartına gönderilmesini sağlayan "SASEBO_G_Checker.exe" isimli çalıştırılabilir program yukarıda verilen kodda döngüye girilmeden önce bir kereliğine çalıştırılmaktadır. Açık kaynak kodu [14] adresinde bulunan bu program, yukarıda verilen kod ile sekronize olarak çalışacak şekilde değiştirilmiştir. Temel olarak bahsedilen programın bir kere çalıştırıldıktan sonra "NWF" kere şifreleme yapması (ve SAKURA-X kartına sayıda aynı açık metinleri göndermesi) gerekmektedir. C# dilinde yazılan "SASEBO_G_Checker" kodu AES blok şifreleme algoritmasının gerçeklenmesi için tasarlanmıştır. Bu nedenle bu kodun "AES.cs" adlı dosyası, AES yerine PRESENT blok şifreleme algoritmasının gerçekleştirmesi için aşağıdaki gibi değiştirilmiştir.

```
using System;
using System.IO;
namespace CipherTool
{
        //=============================================PRESENT
        public class PRESENT : IBlockCipher
        {
        //************************************* Variables
        public readonly byte[] sbox=new byte[16]{0x0C, 0x05, 0x06, 0x0B, 0x09,
0x00, 0x0A, 0x0D, 0x03, 0x0E, 0x0F, 0x08, 0x04, 0x07, 0x01, 0x02 };

                byte[]  key;
                byte[][] key_round;
```

```
            //---------------------------------------------- Constructor
            public PRESENT()
            {
   key = new byte[10];
   key_round = new byte[32][];
   for (int i = 0; i < 32; i++) key_round[i] = new byte[10];
}
//********************************************** Interface
//----------------------------------------------- setKey()
public void setKey(byte[] key)
{
   byte[] keyx;
   byte left4, right4, rcx;
   keyx = new byte[10];

   for (int i = 0; i < 10; i++)
      keyx[i] = key[i];
   for (int i = 0; i < 10; i++)
      key_round[0][i] = key[i];

   for (int I = 1; I < 32; I++)
   {
      for (int i = 0; i < 61; i++)
                  RotateLeft(keyx);

      left4 = sbox[((byte)(keyx[0] & (0xF0))) >> 4];
      right4 = (byte)(keyx[0] & (0x0F));
      keyx[0] = (byte)((left4 << 4) ^ right4);

      rcx = (byte)((keyx[8] & (0x80)) >> 7);
      rcx = (byte)(rcx ^ ((keyx[7] & (0x0F)) << 1));
      rcx = (byte)(rcx ^ I);
      keyx[8] = (byte)(keyx[8] & (0x7F));
      keyx[8] = (byte)(keyx[8] ^ ((rcx & (0x01)) << 7));
      keyx[7] = (byte)(keyx[7] & (0xF0));
      keyx[7] = (byte)(keyx[7] ^ ((rcx & (0x1E)) >> 1));

      for (int i = 0; i < 10; i++)
         key_round[I][i] = keyx[i];
   }
}
public static void RotateLeft(byte[] bytes)
{
   bool carryFlag = ShiftLeft(bytes);
   if (carryFlag == true)
   {
      bytes[bytes.Length-1]=(byte)(bytes[bytes.Length - 1] | 0x01);
   }
}
public static bool ShiftLeft(byte[] bytes)
{
   bool leftMostCarryFlag = false;
```

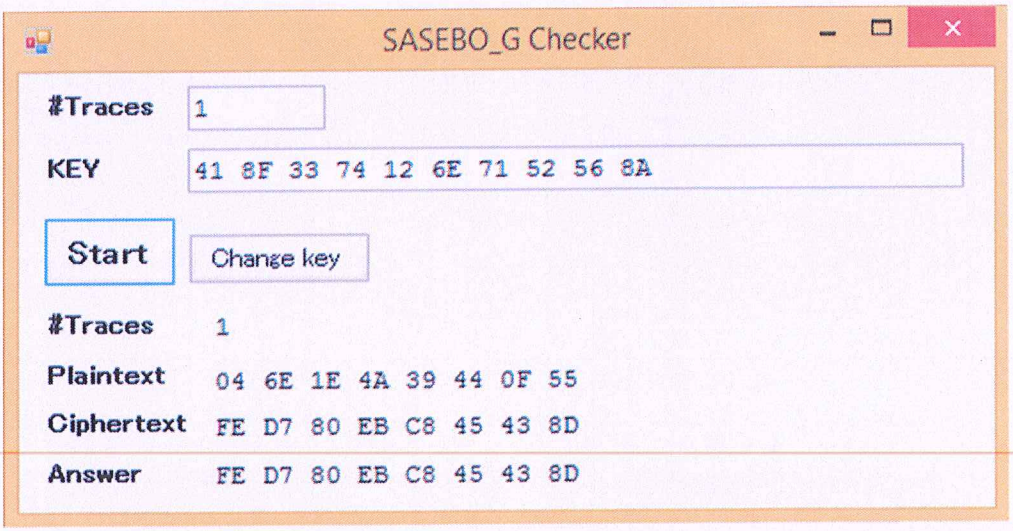
```
        // Iterate through the elements of the array from left to right.
          for (int index = 0; index < bytes.Length; index++)
          {
        //If the leftmost bit of the current byte is 1 then we have a carry.
            bool carryFlag = (bytes[index] & 0x80) > 0;
            if (index > 0)
            {
              if (carryFlag == true)
              {
        // Apply the carry to the rightmost bit of the current bytes neighbor to the left.
                bytes[index - 1] = (byte)(bytes[index - 1] | 0x01);
              }
            }
            else
            {
              leftMostCarryFlag = carryFlag;
            }
            bytes[index] = (byte)(bytes[index] << 1);
          }
          return leftMostCarryFlag;
        }
      //------------------------------------------- encrypt()
      public byte[] encrypt(byte[] pt)
      {
        byte[] state = new byte[8];
        for (int i = 0; i < 8; i++) state[i] = pt[i];
        byte[] keyt = new byte[8];
        Array.Copy(key_round[0], 0, keyt, 0, keyt.Length);
           AddRoundKey(ref state, keyt);
           for (int i = 1; i < 32; ++i)
          {
                    SubBytes(ref state);
            pLayer(ref state);
            Array.Copy(key_round[i], 0, keyt, 0, keyt.Length);
            AddRoundKey(ref state, keyt);
          }
          return state;
      }
      //------------------------------------------- decrypt()
      public byte[] decrypt(byte[] ct) { return null; }
      //******************************************** Cipher functions
      //------------------------------------------- AddRoundKey()
      public void AddRoundKey(ref byte[] state, byte[] key_round) {
        for (int i = 0; i < 8; i++) state[i] = (byte)(state[i] ^ key_round[i]);
    }
      //------------------------------------------- SubBytes()
      public void SubBytes(ref byte[] state)
      { byte sb1, sb2;
        for (int i = 0; i < 8; i++)
          {
            sb1 = sbox[((byte)(state[i] & (0xF0))) >> 4];
            sb2 = sbox[(byte)(state[i] & (0x0F))];
            state[i] = (byte)((sb1 << 4) ^ sb2);
          }
       }
```

```
//---------------------------------------------- pLayer()
public void pLayer(ref byte[] state)
{
    byte[] kp = new byte[8];
    kp[0] = 0x00;
    kp[0] = (byte)((state[0] & (0x80)) ^ ((state[0] & (0x08)) << 3));
    kp[0] = (byte)(kp[0] ^ (((byte)((state[1] & (0x80)) ^ ((state[1] & (0x08)) << 3))) >> 2));
    kp[0] = (byte)(kp[0] ^ (((byte)((state[2] & (0x80)) ^ ((state[2] & (0x08)) << 3))) >> 4));
    kp[0] = (byte)(kp[0] ^ (((byte)((state[3] & (0x80)) ^ ((state[3] & (0x08)) << 3))) >> 6));
    kp[1] = 0x00;
    kp[1] = (byte)((state[4] & (0x80)) ^ ((state[4] & (0x08)) << 3));
    kp[1] = (byte)(kp[1] ^ (((byte)((state[5] & (0x80)) ^ ((state[5] & (0x08)) << 3))) >> 2));
    kp[1] = (byte)(kp[1] ^ (((byte)((state[6] & (0x80)) ^ ((state[6] & (0x08)) << 3))) >> 4));
    kp[1] = (byte)(kp[1] ^ (((byte)((state[7] & (0x80)) ^ ((state[7] & (0x08)) << 3))) >> 6));

    kp[2] = 0x00;
    kp[2] = (byte)(((state[0] & (0x40)) << 1) ^ ((state[0] & (0x04)) << 4));
    kp[2] = (byte)(kp[2] ^ (((byte)(((state[1] & (0x40)) << 1) ^ ((state[1] & (0x04)) << 4)))
>> 2));
    kp[2] = (byte)(kp[2] ^ (((byte)(((state[2] & (0x40)) << 1) ^ ((state[2] & (0x04)) << 4)))
>> 4));
    kp[2] = (byte)(kp[2] ^ (((byte)(((state[3] & (0x40)) << 1) ^ ((state[3] & (0x04)) << 4)))
>> 6));
    kp[3] = 0x00;
    kp[3] = (byte)(((state[4] & (0x40)) << 1) ^ ((state[4] & (0x04)) << 4));
    kp[3] = (byte)(kp[3] ^ (((byte)(((state[5] & (0x40)) << 1) ^ ((state[5] & (0x04)) << 4)))
>> 2));
    kp[3] = (byte)(kp[3] ^ (((byte)(((state[6] & (0x40)) << 1) ^ ((state[6] & (0x04)) << 4)))
>> 4));
    kp[3] = (byte)(kp[3] ^ (((byte)(((state[7] & (0x40)) << 1) ^ ((state[7] & (0x04)) << 4)))
>> 6));

    kp[4] = 0x00;
    kp[4] = (byte)(((state[0] & (0x20)) << 2) ^ ((state[0] & (0x02)) << 5));
    kp[4] = (byte)(kp[4] ^ (((byte)(((state[1] & (0x20)) << 2) ^ ((state[1] & (0x02)) << 5)))
>> 2));
    kp[4] = (byte)(kp[4] ^ (((byte)(((state[2] & (0x20)) << 2) ^ ((state[2] & (0x02)) << 5)))
>> 4));
    kp[4] = (byte)(kp[4] ^ (((byte)(((state[3] & (0x20)) << 2) ^ ((state[3] & (0x02)) << 5)))
>> 6));
    kp[5] = 0x00;
    kp[5] = (byte)(((state[4] & (0x20)) << 2) ^ ((state[4] & (0x02)) << 5));
    kp[5] = (byte)(kp[5] ^ (((byte)(((state[5] & (0x20)) << 2) ^ ((state[5] & (0x02)) << 5)))
>> 2));
    kp[5] = (byte)(kp[5] ^ (((byte)(((state[6] & (0x20)) << 2) ^ ((state[6] & (0x02)) << 5)))
>> 4));
    kp[5] = (byte)(kp[5] ^ (((byte)(((state[7] & (0x20)) << 2) ^ ((state[7] & (0x02)) << 5)))
>> 6));

    kp[6] = 0x00;
    kp[6] = (byte)(((state[0] & (0x10)) << 3) ^ ((state[0] & (0x01)) << 6));
    kp[6] = (byte)(kp[6] ^ (((byte)(((state[1] & (0x10)) << 3) ^ ((state[1] & (0x01)) << 6)))
>> 2));
    kp[6] = (byte)(kp[6] ^ (((byte)(((state[2] & (0x10)) << 3) ^ ((state[2] & (0x01)) << 6)))
>> 4));
```

```
        kp[6] = (byte)(kp[6] ^ (((byte)(((state[3] & (0x10)) << 3) ^ ((state[3] & (0x01)) << 6)))
>> 6));
        kp[7] = 0x00;
        kp[7] = (byte)(((state[4] & (0x10)) << 3) ^ ((state[4] & (0x01)) << 6));
        kp[7] = (byte)(kp[7] ^ (((byte)(((state[5] & (0x10)) << 3) ^ ((state[5] & (0x01)) << 6)))
>> 2));
        kp[7] = (byte)(kp[7] ^ (((byte)(((state[6] & (0x10)) << 3) ^ ((state[6] & (0x01)) << 6)))
>> 4));
        kp[7] = (byte)(kp[7] ^ (((byte)(((state[7] & (0x10)) << 3) ^ ((state[7] & (0x01)) << 6)))
>> 6));

        for (int i = 0; i < 8; i++)     state[i] = kp[i];
    }
  }
}
```

PRESENT şifreleme algoritmasının gerçeklemesinin yapıldığı yukarıda verilen kodda, Bölüm 2'de verilen addRoundKey ve pLayer katmanları aynı isimlere sahip olan fonksiyonlarla icra edilmektedir. Algoritmanın S-kutusu kodda "sbox" dizisi ile tanımlanmaktadır. sBoxLayer aşaması ise her tur girişindeki 64-bit uzunluğunda olan bloğun her bir baytına, kodda bulunan "SubBytes" fonksiyonunun uygulanması ile icra edilmektedir. Şifreleme işlemini "encrypt" fonksiyonu gerçekleştirmektedir; burada, "pt" dizisi ile tanımlanan açık metin uygulanan 31 turun ardından "state" dizisi ile gösterilen kapalı metin dizisine dönüştürülür.

SASEBO_G_Checker programının C# kodunda yapılan değişiklerden sonra elde edilen arayüz Şekil 4.3 'de görüldüğü gibidir. Burada veriler 16'lı tabanda gösterilmektedir; anahtar (KEY) 80-bit uzunluğunda, açık metin (Plaintext), bilgisayarın ürettiği kapalı metin (Ciphertext) ve kriptografik donanımın ürettiği kapalı metin (Answer) 64-bit uzunluğundadır. Açık metin ve anahtar kriptografik donanıma bu program vasıtasıyla gönderilmektedir. Böylelikle şifreleme işlemi hem donanımda hem de bilgisayarda gerçekleştirilerek arayüzde görüldüğü üzere elde edilen kapalı metinler karşılaştırılmaktadır. Şekil 4.3'de görülen "#Traces" değeri bu programın şifreleme işlemini kaç kere icra edeceğini göstermektedir. SASEBO_G_Checker programının bir önceki verilen kod ile senkronize bir şekilde çalışabilmesi için bu değerin "NWF" değeri ile aynı olması gerekmektedir.



Şekil 4.3: SASEBO_G_Checker program arayüzü.

Açık kaynak olan verilog kodu [14], (SASEBO_G_Checker programında olduğu gibi) AES blok şifreleme algoritmasının gerçeklenmesi için tasarlanmıştır. PRESENT blok şifreleme algoritmasının gerçeklenmesi için, AES algoritmasını gerçekleyen "aes_composite_enc.v" dosyasında bulunan AES_Core, KeyExpansion ve SubBytes modülleri aşağıdaki gibi değiştirilmiştir (AES_Core modülü PRESENT_Core olarak değiştirilmiş ve diğer modüller aynı isimlerle PRESENT algoritmasını gerçekleyecek şekilde değiştirilmiştir). Orijinal dosyada bulunan diğer modüller (MixColumns, GF_MULINV_4, GF_MULINV_8, ShiftRows) PRESENT algoritmasında kullanılmadığı için silinmiştir. Burada, "KeyExpansion" modülü bir önceki tur anahtarı (kin) ve tur sayacını (rcon) kullanarak tur anahtarı çıkışını (kout) üretmektedir. "sBox" modülü girdisi olan 4-bitlik bloğa (x) karşılık gelen S-kutusu çıkışını (y) elde eder. "PRESENT_Core" modülü ise tur girişi (din) ve tur anahtarını (kin) kullanarak tur çıkışını (dout) üretmektedir. Bölüm 2'de bahsedilen PRESENT algoritmasının bir turunun tamamlanması için yürütülen addRoundKey, sBoxLayer ve pLayer fonksiyonlarının "PRESENT_Core" modülü tarafından icra edildiği görülmektedir.

```
// PRESENT
module PRESENT_Core (din, dout, kin);
input  [63:0]  din;
input  [79:0]  kin;
output [63:0] dout;
wire [3:0] st00, st01, st02, st03, st04, st05, st06, st07, st08, st09, st10, st11, st12, st13, st14,
st15;
```

```
wire [3:0]  sb00, sb01, sb02, sb03, sb04, sb05, sb06, sb07, sb08, sb09, sb10, sb11, sb12,
sb13, sb14, sb15;
wire [63:0] SB, SP;
// SASEBO_G_Checker programından gelen açık metin 4-bitlik bloklara ayrılıyor.
assign st00 = din[63:60];
assign st01 = din[59:56];
assign st02 = din[55:52];
assign st03 = din[51:48];
assign st04 = din[47:44];
assign st05 = din[43:40];
assign st06 = din[39:36];
assign st07 = din[35:32];
assign st08 = din[31:28];
assign st09 = din[27:24];
assign st10 = din[23:20];
assign st11 = din[19:16];
assign st12 = din[15:12];
assign st13 = din[11:8];
assign st14 = din[7:4];
assign st15 = din[3:0];
// 4 bitlik blokların her biri için S-kutusu çıkışı elde ediliyor (sBoxLayer).
sBox SB00 (st00,sb00);
sBox SB01 (st01,sb01);
sBox SB02 (st02,sb02);
sBox SB03 (st03,sb03);
sBox SB04 (st04,sb04);
sBox SB05 (st05,sb05);
sBox SB06 (st06,sb06);
sBox SB07 (st07,sb07);
sBox SB08 (st08,sb08);
sBox SB09 (st09,sb09);
sBox SB10 (st10,sb10);
sBox SB11 (st11,sb11);
sBox SB12 (st12,sb12);
sBox SB13 (st13,sb13);
sBox SB14 (st14,sb14);
sBox SB15 (st15,sb15);
// S-Box çıkışında elde edilen 4-bitlik bloklar 64 bit haline getirilir.
assign SB = {sb00, sb01, sb02, sb03, sb04, sb05, sb06, sb07, sb08, sb09, sb10, sb11, sb12,
sb13, sb14, sb15};
// SB bloğundaki bitlere pLayer ile tanımlanan permütasyon işlemi uygulanır.
assign SP={SB[63], SB[59], SB[55], SB[51], SB[47], SB[43], SB[39], SB[35], SB[31],
SB[27], SB[23], SB[19], SB[15], SB[11], SB[7], SB[3], SB[62], SB[58], SB[54], SB[50],
SB[46], SB[42], SB[38], SB[34], SB[30], SB[26], SB[22], SB[18], SB[14], SB[10], SB[6],
SB[2], SB[61], SB[57], SB[53], SB[49], SB[45], SB[41], SB[37], SB[33], SB[29], SB[25],
SB[21], SB[17], SB[13], SB[9], SB[5], SB[1], SB[60], SB[56], SB[52], SB[48], SB[44],
SB[40], SB[36], SB[32], SB[28], SB[24], SB[20], SB[16], SB[12], SB[8], SB[4], SB[0]};
```

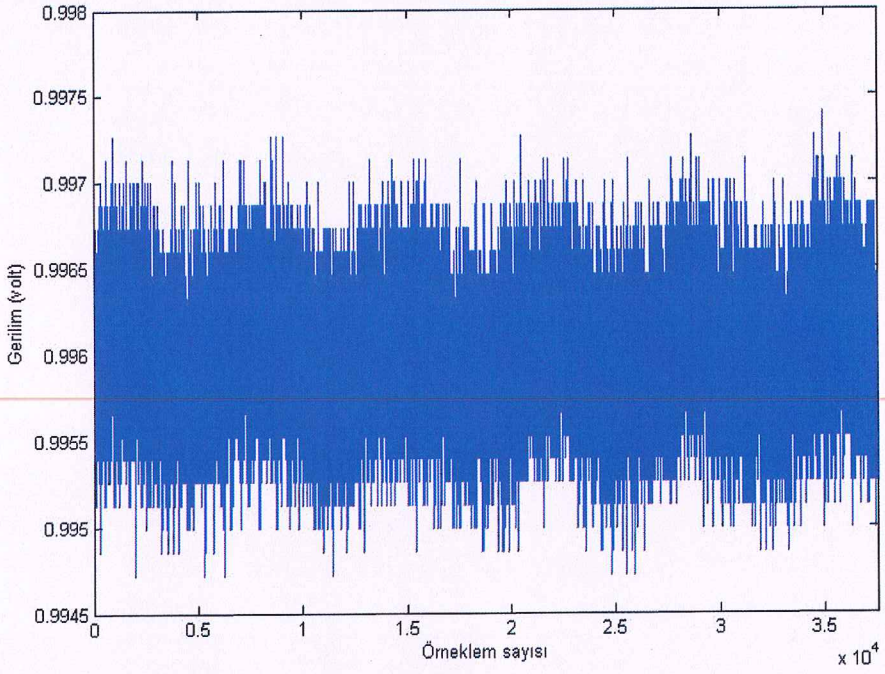
```
// Tur çıkışındaki 64-bitlik blok, SP bloğu ile aynı uzunluktaki tur anahtarının (kin[79:16])
XOR işlemi ile toplanmasıyla elde edilir (addRoundKey).
assign dout = SP ^ kin[79:16];
// Anahtar üreteci modülü
module KeyExpansion (kin, kout, rcon);
input [79:0]  kin;
output [79:0] kout;
input [4:0] rcon;
        wire [3:0] kt;
        sBox SX(kin[18:15],kt);
        assign kout = {kt,kin[14:0],kin[79:39],kin[38:34]^rcon,kin[33:19]};
endmodule // KeyExpansion
// S-Kutusu  modülü
module sBox (x, y);
        input [3:0] x;
        output [3:0] y;
        assign y=s(x);
        function [3:0] s;
                input [3:0] x;
                case (x)
                4'h0: s=4'hC;
                4'h1: s=4'h5;
                4'h2: s=4'h6;
                4'h3: s=4'hB;
                4'h4: s=4'h9;
                4'h5: s=4'h0;
                4'h6: s=4'hA;
                4'h7: s=4'hD;
                4'h8: s=4'h3;
                4'h9: s=4'hE;
                4'hA: s=4'hF;
                4'hB: s=4'h8;
                4'hC: s=4'h4;
                4'hD: s=4'h7;
                4'hE: s=4'h1;
                4'hF: s=4'h2;
                endcase
        endfunction
endmodule
```

5. GERÇEKLEME SONUÇLARI

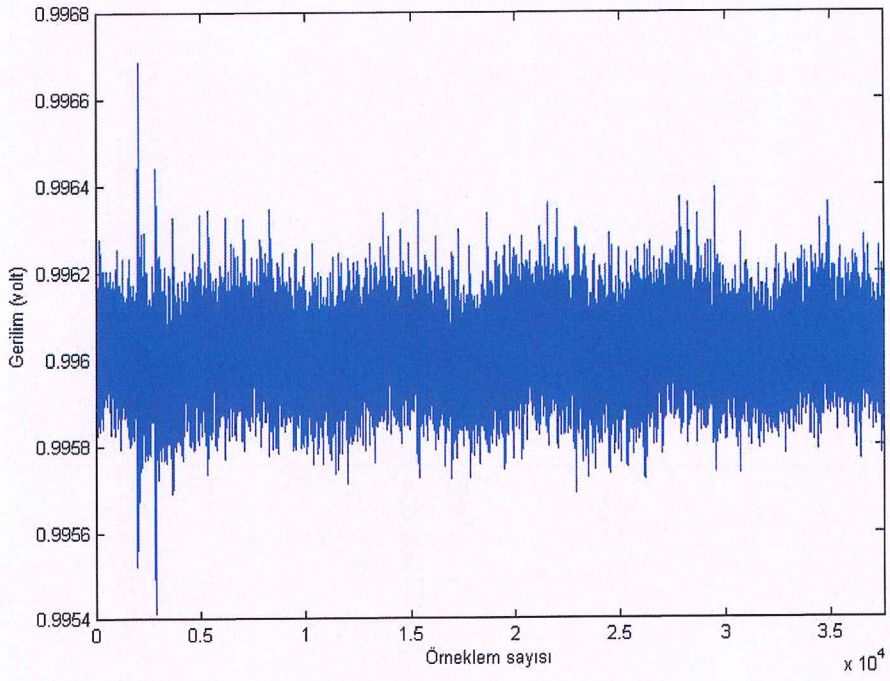
Bir önceki bölümde MATLAB kodu verilen farksal güç analizi, sabit bir anahtar için bilgisayar tarafından rasgele üretilen açık metinlerin SAKURA-X kriptografik donanımı ile şifrelenmesi sonucu elde edilen güç ölçümlerinin ve bu açık metinlere karşılık gelen kapalı metinlerin kullanılması ile gerçekleştirilmiştir. Bunun sonucunda bulunan tahmin entropileri bu bölümde ele alınacaktır.

Öncelikle PRESENT blok şifreleme algoritmasının SAKURA-X tarafından icra edilmesi esnasında bir şifreleme süresince osiloskop ile alınan tipik bir ölçüm sonucu aşağıdaki Şekil 5.1'de gösterilmektedir. Ölçümlerdeki gürültü miktarını azaltmak ve böylelikle daha sağlıklı sonuçlar elde etmek amacıyla, aynı açık metin ve anahtar için SAKURA-X kartının birden fazla sayıda şifreleme yapması sağlanmış ve her bir şifreleme için güç ölçümleri alınarak bu güç ölçümlerinin ortalaması farksal güç analizini yürütmek için kullanılmıştır. Şekil 5.2'de bahsedildiği gibi elde edilen 30 ölçümünün ortalaması gösterilmektedir.

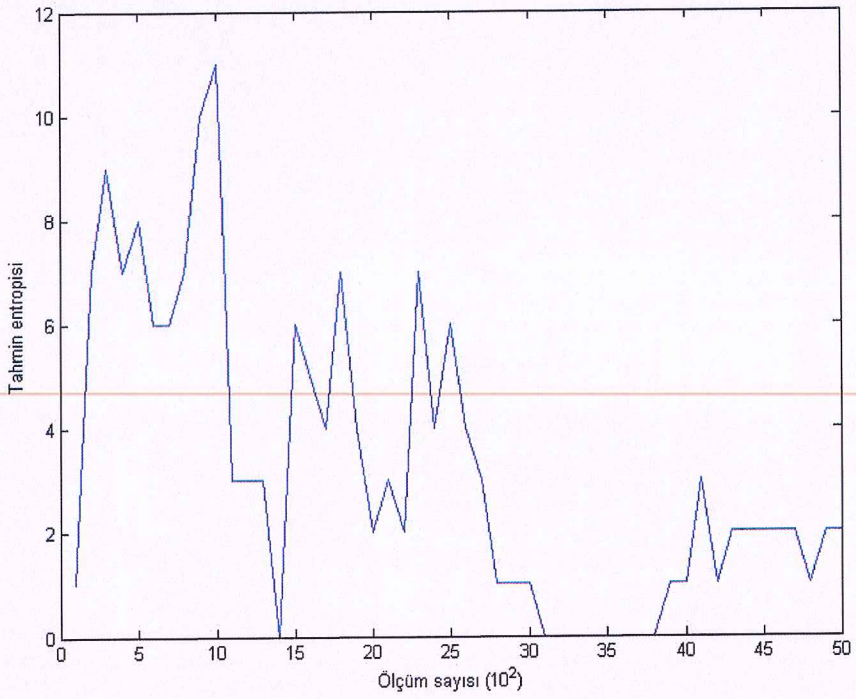
Şekil 5.2'den gürültünün nispeten azaldığı görülmektedir. Her bir güç ölçümü (aynı açık metin ve anahtar için) 30 güç ölçümünün ortalaması olan 5000 güç ölçümü ile yürüttüğümüz farksal güç analizi sonucunda elde edilen doğru anahtarın tahmin entropisi grafiği Şekil 5.3'de verilmiştir. Şekil 5.3'ten görüldüğü üzere gürültüsü azaltılmış güç ölçümleri kullanılmasına rağmen 5000 güç ölçümü sonucunda doğru anahtar elde edilememiştir. Ortalaması alınan güç ölçümlerinin sayısını 200'e çıkardığımızda bile doğru anahtarın tespit edilemediği gözlenmiştir. Gerçekte bu sonuç, ölçüm sayısını artırarak azaltamadığımız gürültünün osiloskopun kuantalama hatasından (veya dikey çözünürlüğünün yetersiz kalmasının) kaynaklandığını ve bu hatanın yürütülen yan kanal analizinin başarısını etkilediğini göstermektedir.



Şekil 5.1: PRESENT algoritması için tipik bir ölçüm sonucu.

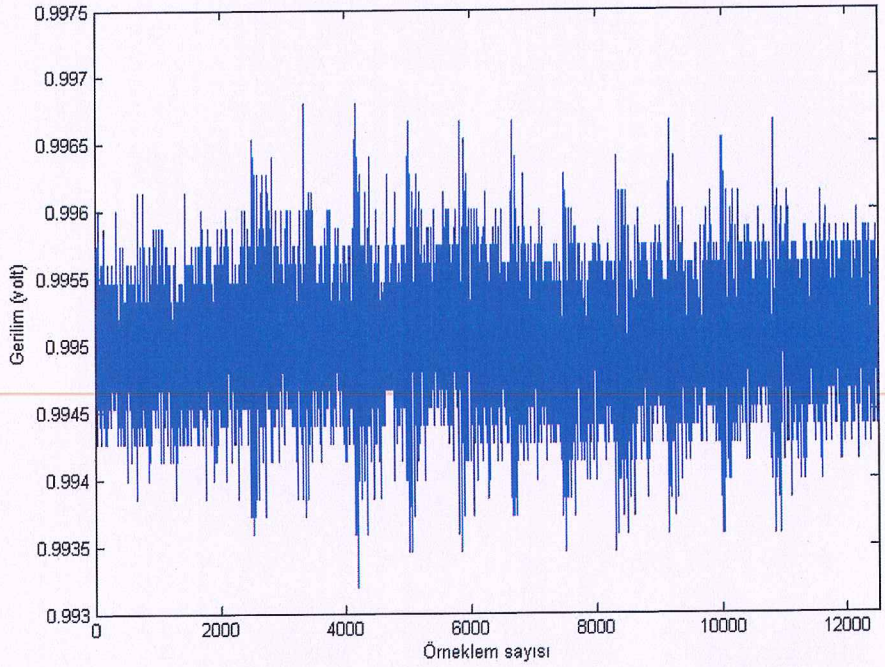


Şekil 5.2: PRESENT algoritması için 30 ölçüm ortalaması.

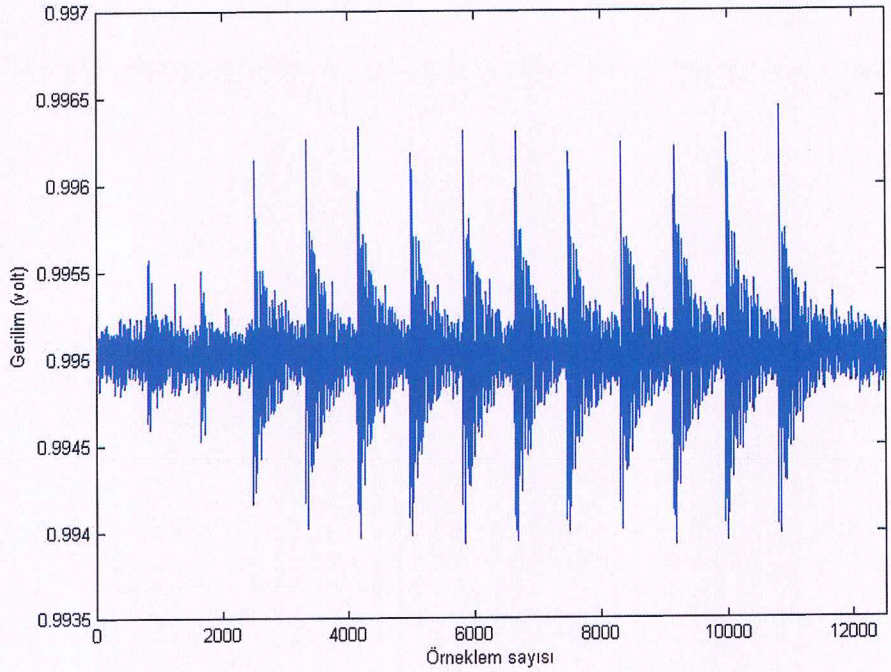


Şekil 5.3: PRESENT için ortalama ölçümler ile elde edilen tahmin entropisi.

Bir blok şifreleme algoritmasında kullanılan S-kutusunun büyüklüğünün kriptografik donanımın harcadığı gücü etkileyen bir faktör olduğu bilinmektedir. Bunu test etmek için, S-kutusu büyüklüğü 8-bit olan bir başka şifreleme algoritması AES için de farksal güç analizi gerçekleştirilmiş ve sonuçlar karşılaştırılmıştır. Şekil 5.4'te AES algoritmasının bir şifreleme süresince osiloskop ile alınan tipik bir güç ölçümü ve Şekil 5.5'te PRESENT algoritmasında olduğu gibi aynı açık metin ve anahtar için elde edilen 30 güç ölçümünün ortalaması gösterilmektedir. Bu iki şekildeki güç ölçümleri PRESENT için elde edilen güç ölçümleri (Şekil 5.1 ve Şekil 5.2) ile karşılaştırıldığında, nispeten daha az gürültülü oldukları (bir başka deyişle harcanan gücün gürültüye oranının daha fazla olduğu) ve her iki şekilde de AES algoritmasındaki turların gerçekleşme zamanlarının görülebildiği gözlenmektedir.

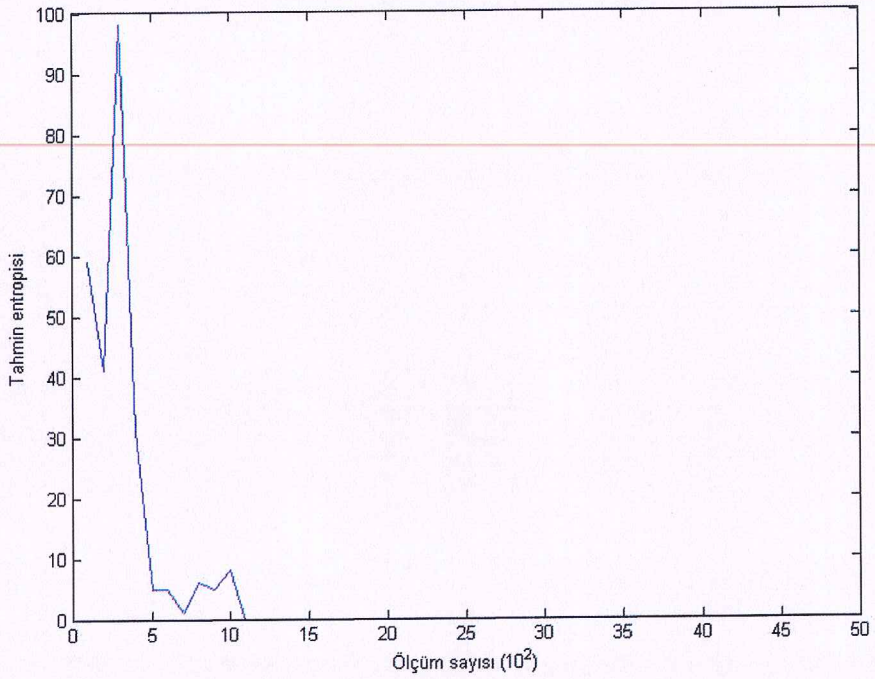


Şekil 5.4: AES algoritması için tipik bir ölçüm sonucu.



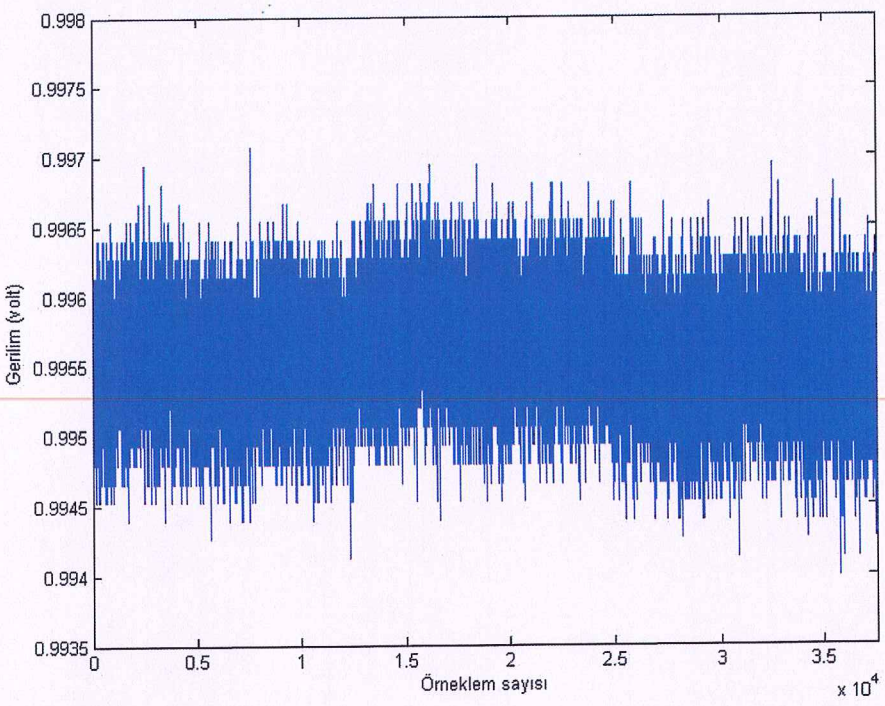
Şekil 5.5: AES algoritması için 30 ölçüm ortalaması.

AES algoritması için de, her bir güç ölçümü (aynı açık metin ve anahtar için) 30 güç ölçümünün ortalaması olan 5000 güç ölçümü ile yürüttüğümüz farksal güç analizi sonucunda elde edilen doğru anahtarın tahmin entropisi grafiği Şekil 5.6'da verilmiştir.

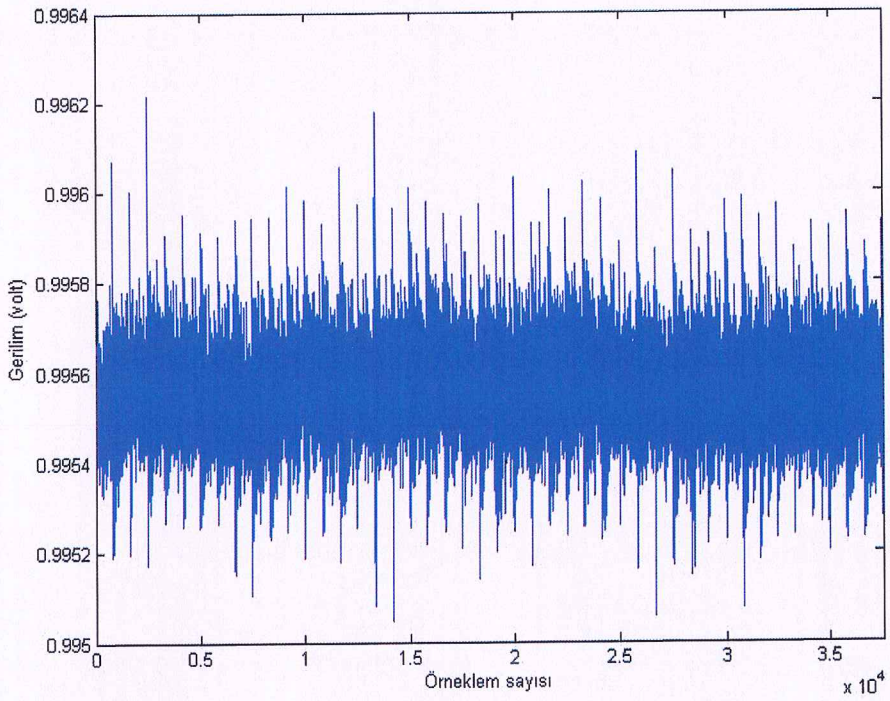


Şekil 5.6: AES için ortalama ölçümler ile elde edilen tahmin entropisi.

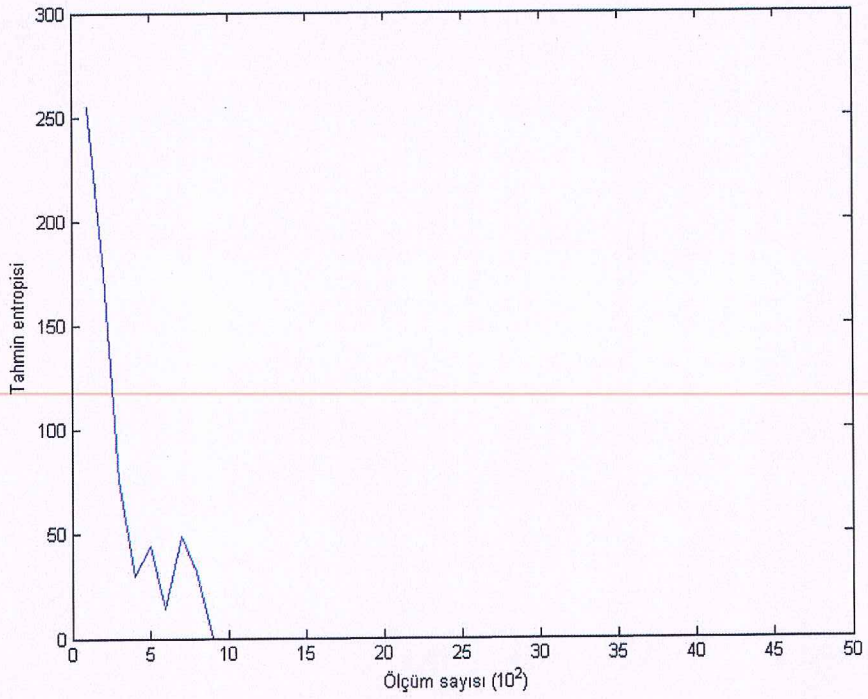
Şekil 5.6'da görüldüğü gibi 1100 güç ölçümünden sonra doğru anahtar elde edilmektedir. Bu sonuç, S-kutusu büyüklüğü arttıkça elde edilen güç ölçümlerinin gürültüden daha az etkilenebileceğini doğrulamaktadır. Ayrıca, çalışmamızda bu gerçek PRESENT blok şifreleme algoritmasında kullanılan 16 adet 4×4 büyüklüğündeki S-kutusu yerine 8 adet 8×8 büyüklüğündeki AES S-kutusu kullanılarak da doğrulanmıştır. Şekil 5.7 ve Şekil 5.8'de S-kutuları değiştirilmiş PRESENT için sırasıyla tipik bir ölçüm sonucu ve 30 ölçüm ortalaması verilmektedir. Bu ölçümler, orijinal PRESENT algoritması için verilen Şekil 5.1 ve Şekil 5.2 ile karşılaştırıldığında, özellikle Şekil 5.8'de görülen ortalama ölçümde turların çok daha net bir şekilde seçilebildiği gözlenmektedir. Gerçekte bu gözlem, harcanan gücün arttığının bir göstergesidir. Verilen kodların uygun şekilde değiştirilmesiyle bahsedilen S-kutusu değişikliği gerçekleştirildiğinde, elde ettiğimiz doğru aday anahtar için tahmin entropisi Şekil 5.9'da gösterilmektedir. Şekil 5.9'dan görüldüğü gibi, 900 ölçümden sonra doğru aday anahtar bulunabilmektedir.



Şekil 5.7: S-kutuları değiştirilmiş PRESENT algoritması için tipik bir ölçüm.

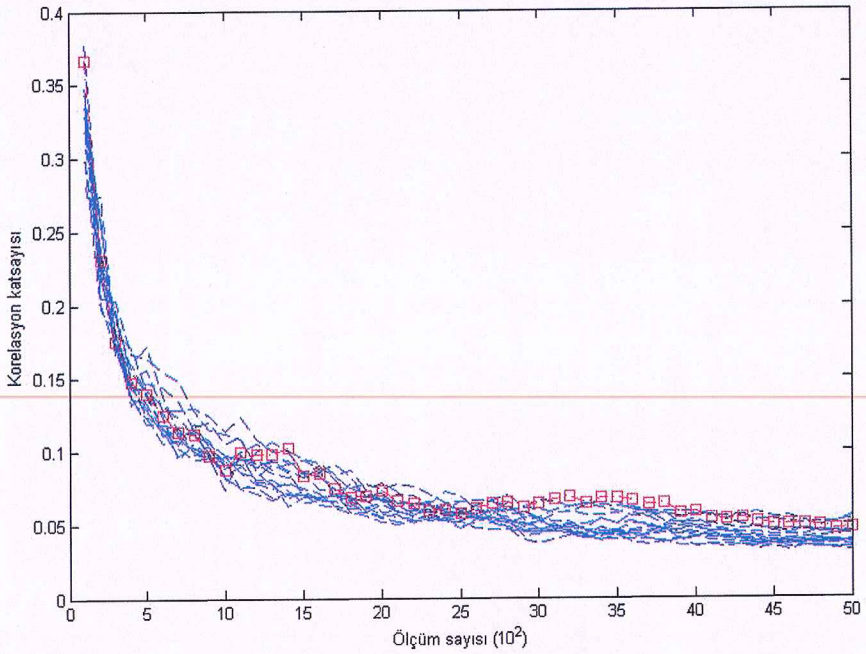


Şekil 5.8: S-kutuları değiştirilmiş PRESENT algoritması için 30 ölçüm ortalaması.

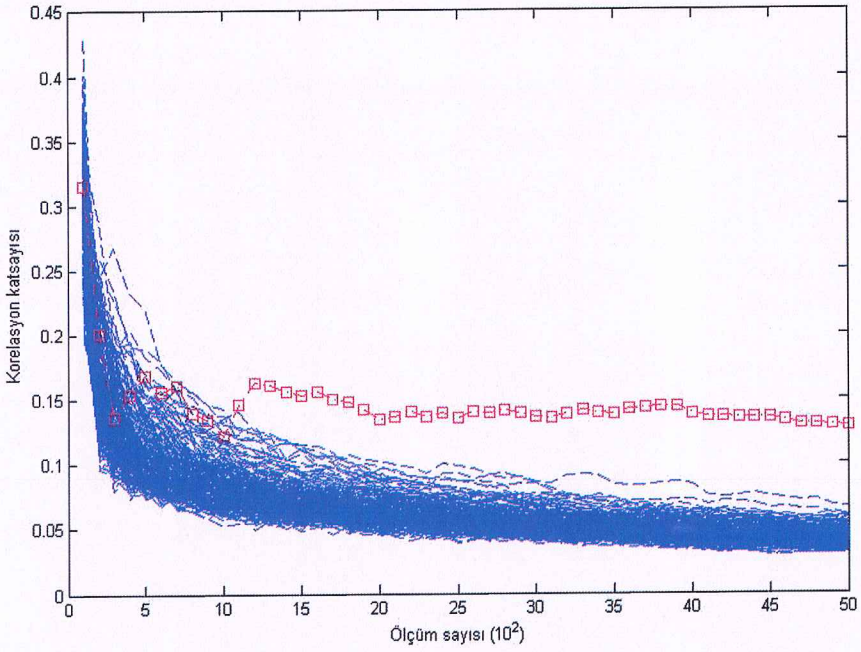


Şekil 5.9: S-kutuları değiştirilmiş PRESENT için tahmin entropisi.

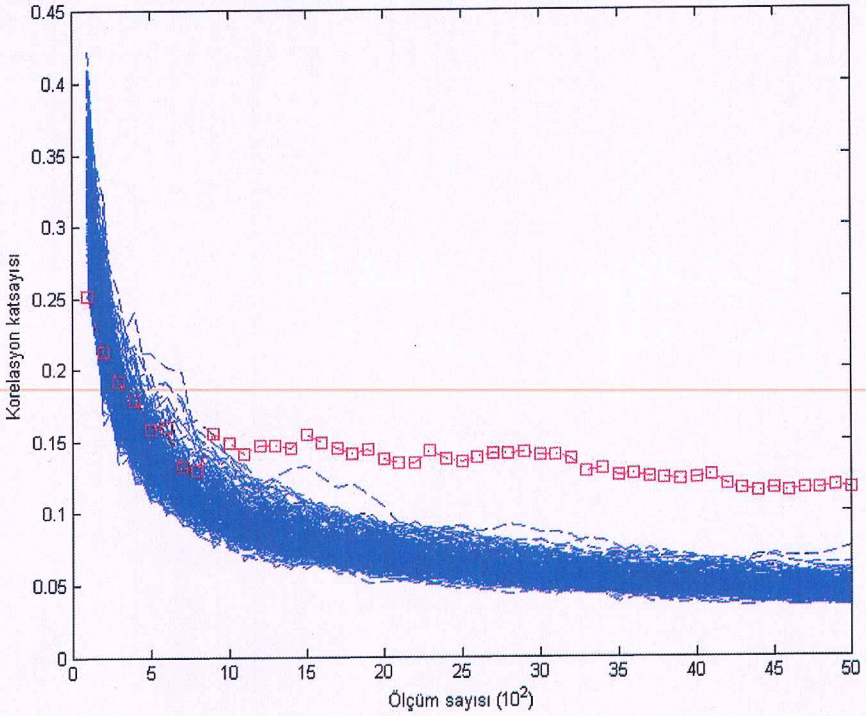
Doğru anahtar, aday anahtarların tahmin entropilerine bakılarak belirlenebileceği gibi aday anahtarlar için elde edilen korelasyon değerlerine bakılarak ta belirlenebilir. Bir önceki bölümde verilen MATLAB kodunda "CFa" dizisinin satırları aday anahtarlar için bahsedilen korelasyon değerlerini vermektedir. PRESENT, AES ve S-kutuları değiştirilmiş PRESENT algoritmaları için bulunan korelasyon değerleri sırasıyla Şekil 5.10, Şekil 5.11 ve Şekil 5.12'de gösterilmektedir.



Şekil 5.10: PRESENT için tüm aday anahtarların korelasyon değerleri.



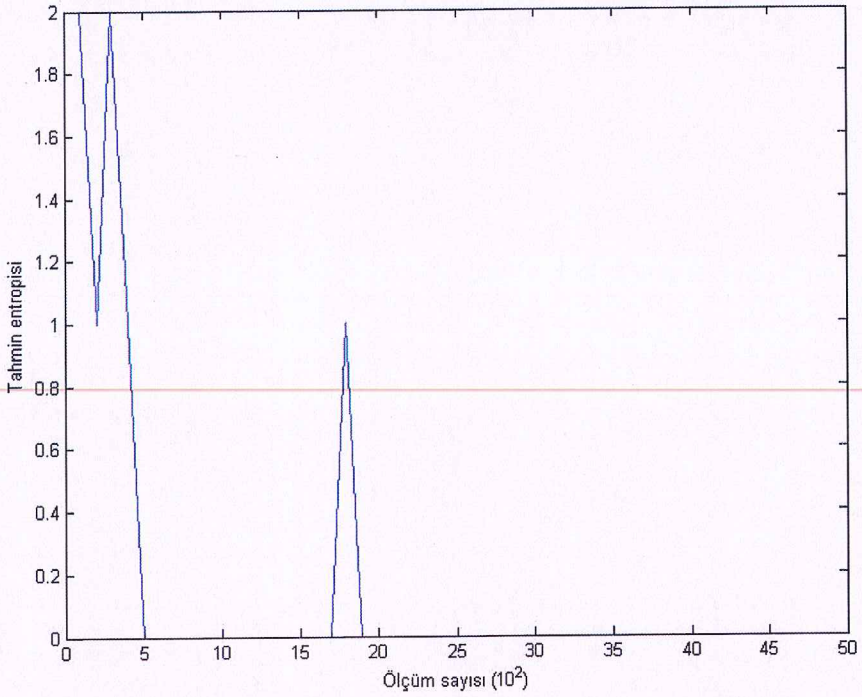
Şekil 5.11: AES için tüm aday anahtarların korelasyon değerleri.



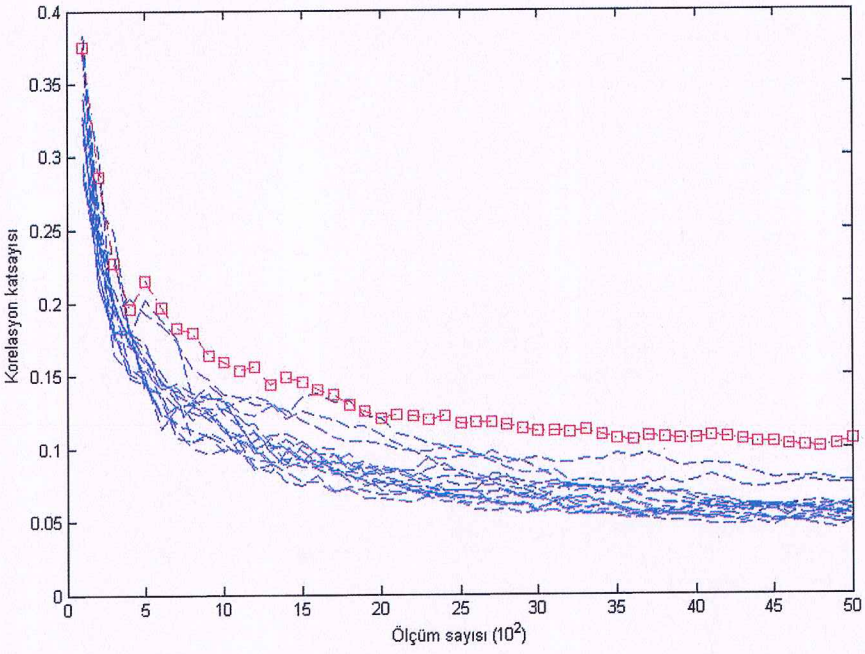
Şekil 5.12: S-kutuları değiştirilmiş PRESENT için tüm aday anahtarların korelasyon değerleri.

Bu üç şekilde de doğru aday anahtarın korelasyon değerleri kırmızı renkte gösterilmiştir. AES ve S-kutuları değiştirilmiş PRESENT için verilen Şekil 5.11 ve Şekil 5.12'de doğru aday anahtarın korelasyon değerleri en yüksektir ve kolaylıkla belirlenebilmektedir. Buna karşın PRESENT için verilen Şekil 5.10'da bu durumun gerçekleşmediği ve dolayısıyla doğru aday anahtarın tespit edilemediği görülmektedir.

Son olarak PRESENT algoritması için gerçek güç ölçümleri kullanılmaksızın, MATLAB ortamında güç ölçümleri üretilerek farksal güç analizi uygulanmış ve simülasyon sonuçları elde edilmiştir. Kriptografik donanımda oluşan gürültü, simülasyonda Gauss gürültüsü olarak modellenmiştir. Bu simülasyonu gerçekleyen MATLAB kodu EK-A'da verilmektedir. Bu kodun çalıştırılması ile elde edilen tahmin entropisi grafiği ve korelasyon değerlerinin grafiği sırasıyla Şekil 5.13 ve Şekil 5.14'te verilmektedir. Bu grafiklerin her ikisinden de doğru aday anahtarın tespit edilebildiği açıkça görülebilmektedir.



Şekil 5.13: PRESENT için simülasyon ile elde edilen tahmin entropisi.



Şekil 5.14: PRESENT için simülasyon ile elde edilen korelasyon değerleri.

6. SONUÇ VE ÖNERİLER

Bu tezde 80-bit anahtar uzunluğuna sahip PRESENT blok şifreleme algoritmasının SAKURA-X kriptografik donanım üzerinde FPGA gerçeklemesi yapılarak, yan kanal analizi için gerekli olan güç ölçümleri bir ölçüm düzeneği vasıtasıyla elde edilmiştir. SAKURA-X kartı, bilgisayar ve osiloskoptan oluşan bu ölçüm düzeneği, bahsedilen cihazların senkronize olmasını ve güç ölçümlerinin döngüsel halde elde edilmesini sağlayacak şekilde çalıştırılmıştır. Elde edilen güç ölçümleri kullanılarak en etkili yan kanal analizi yöntemlerinden birisi olan farksal güç analizi PRESENT blok şifreleme algoritması için uygulanmıştır. Bunun sonucunda, bahsedilen güç ölçümlerindeki kuantalama hatasından kaynaklı gürültünün doğru aday anahtarın tespitini zorlaştırdığı görülmüştür. Güç tüketimi PRESENT algoritmasına göre daha fazla olan AES algoritması ve S-kutuları değiştirilmiş PRESENT algoritması için bu hatanın etkisinin daha az olduğu ve farksal güç analizi ile doğru aday anahtarın daha kolay elde edildiği gözlenmiştir. Ayrıca, PRESENT algoritması için (gürültülü) güç ölçümleri MATLAB ortamında simüle edilerek farksal güç analizi uygulanmış ve doğru aday anahtar başarılı bir şekilde elde edilmiştir.

Yaptığımız çalışma, yan kanal analizini yürütmek için kriptografik donanımdan elde edilen ölçümlerdeki kuantalamadan kaynaklı hatanın etkisini göstermektedir. Dikey çözünürlüğü yüksek bir osiloskop kullanmak veya uygun bir yükselteç vasıtasıyla ölçüm almak gibi farklı yöntemler ile oluşan kuantalama hatasının azaltılması bu çalışmanın devamı niteliğinde önerilmektedir.

7. KAYNAKLAR

[1] National Bureau of Standards. Data Encryption Standard. U. S. Department of Commerce, (1977).

[2] Daemen, J. and Rijmen, V., *The Design of Rijndael, AES – The Advanced Encryption Standard*, New York, Berlin Heidelberg: Springer-Verlag, (2002).

[3] Schneier, B., "Description of a New Variable - Length Key, 64 Bit Block Cipher (Blowfish)",(ed: R. Anderson), *Fast Software Encryption*, 809, Heidelberg, Springer-Verlag, 191–204. (1994).

[4] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B, Seurin, Y. and Vikkelsoe, C., "PRESENT: An Ultra-Lightweight Block Cipher", *Cryptography Hardware Embeded System-CHES 2007*, 450–466, (2007).

[5] Diffie, W. and Hellman, M.E., "New Directions in Cryptography", *IEEE Transactions on Information Theory*, 6, 644-654, (1976).

[6] Rivest, R.L., Shamir, A. and Adleman, L.M., "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, 21, 120-126, (1978).

[7] Ronald R.L. "RSA security response to weaknesses in key scheduling algorithm of RC4", Technical note, RSA Data Security, Inc., (2001).

[8] Matsui, M., "The first experimental cryptanalysis of the Data Encryption Standard", *In Advances in Cryptology - CRYPTO 1994*, volume 839 of LNCS, Springer-Verlag, 1–11, (1994).

[9] Biham, E. and Shamir, A., "Differential Cryptanalysis of DES-like Cryptosystems", *Journal of Cryptology*, 4 (1), 3–72, (1991).

[10] Biham, E. and Shamir, A., "Differential Fault Analysis of Secret Key Cryptosystems", *Advances in Cryptology—CRYPTO 1997 Proceedings*, Springer-Verlag, 513–525, (1997).

[11] Matsui, M., "Linear Cryptanalysis Method for DES Cipher", *Advances in Cryptology—EUROCRYPT 1993 Proceedings*, Springer-Verlag, 386–397, (1994).

[12] Hori, Y., Katashita, T., Sasaki, A. and Satoh, A., "SASEBO-GIII: A hardware security evaluation board equipped with a 28-nm FPGA," in Consumer Electronics (GCCE), 2012 IEEE 1st Global Conference on, 657-660, doi: 10.1109/GCCE.2012.6379944, (2012).

[13] Keysight Technologies, "Keysight Infiniium Oscilloscopes Programmer's Guide [online]". (27 August 2019), https://www.keysight.com/upload/cmc_upload/All/Infiniium_prog_guide.pdf, (2019).

[14] National Institute of Advanced Industrial Science Technology (AIST), "SASEBO_G_Checker [online]". (27 August 2019), http://www.risec.aist.go.jp/project/sasebo/download/sasebo_giii_materials.zip, (2019).

[15] Kocher, P.C., Jaffe, J. and Jun, B., "Differential Power Analysis", In: Wiener, CRYPTO 1999. LNCS, 1666, 388–397. Springer, Heidelberg (1999).

[16] Brier, E., Clavier, C. and Olivier, F., "Correlation Power Analysis with a Leakage",(eds. J.J. Quisquater and M. Joye), *CHES 2004*, 3156, 16–29. Springer, Heidelberg (2004).

EKLER

8. EKLER

EK A MATLAB Farksal Güç Analizi Simülasyon Kodu

```
clear all
%pLayer
PR=[0 16 32 48 1 17 33 49 2 18 34 50 3 19 35 51 4 20 36 52 5 21 37 53 6 22 38 54 7 23 39
55 8 24 40 56 9 25 41 57 10 26 42 58 11 27 43 59 12 28 44 60 13 29 45 61 14 30 46 62 15
31 47 63] + 1;

%invpLayer
PI=[0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 1 5 9 13 17 21 25 29 33 37 41 45 49 53 57
61 2 6 10 14 18 22 26 30 34 38 42 46 50 54 58 62 3 7 11 15 19 23 27 31 35 39 43 47 51 55
59 63] +1;

% sBox
Shex='C56B90AD3EF84712';
for i=1:16
   t=hex2dec(Shex(i));
   S(i)=t;
   Sb(i,:)=dobin(t,4);% S-kutusunun ikili versiyonu
   Sinv(t+1)=i-1;% invsBox
end

% Gizli anahtar
Kth='00010203040506070809';
Kt=[];
for i=1:20
   Kt=[Kt dobin(hex2dec(Kth(i)),4)];
end
K=Kt;

% Güç ölçüm sayısı
TN=5000;

% Tahmin entropisini (eklenen) her 100 ölçümde yeniler
stp=100;

%Gürültünün standart sapması
std=18;

% Örneklem sayısı (Simüle edilen güç ölçümü uzunluğu)
Ts=1000;

% Tur anahtarlarını üretir
R(1,:)=K(1:80);
for i=2:32
   K=[K(62:80) K(1:61)];
```

```
    t=todec(K(1:4));% 4 biti onluk tabana çevirir
    K(1:4)=Sb(t+1,:);
    rc=dobin(i-1,5);
    K(61:65)=xor(K(61:65),rc);
    R(i,:)=K(1:80);
  end
  Kc=R(32,PR(33:36));

  % Tahmin edilmesi beklenen gerçek anahtar
  kcd=todec(Kc);

  % Aday anahtarlar
  for i=1:16
    Ka(i,:)=dobin(i-1,4);
  end

  % Tahmin matrisi
  H(1:TN,1:16)=0;

  % Tahmin matrisini hesaplar
  for I=1:TN
    C=round(rand([1 64])); % Rasgele üretilen açık metin
    C=xor(C,R(1,1:64));
    for i=2:31% PRESENT'in ilk 30 turu
      for j=1:16
        t=todec(C(4*(j-1)+1:4*j));
        C(4*(j-1)+1:4*j)=Sb(t+1,:);
      end
      C(PR)=C;
      C=xor(C,R(i,1:64));
    end
    % Son tur
    tx=C; % Son turun girişi
    for j=1:16
      t=todec(C(4*(j-1)+1:4*j));
      C(4*(j-1)+1:4*j)=Sb(t+1,:);
    end
    C(PR)=C;
    C=xor(C,R(32,1:64));
    ty=C; % Son turun çıkışı

    % Simüle edilen güç ölçümü
    TR(I,:)=sum(xor(tx,ty))+std*randn([1 Ts]);
    So=C(PR(33:36));
    Soc=C(33:36);
  for i=1:16
      to=xor(Ka(i,:),So);
      t=Sinv(todec(to)+1);
      H(I,i)=sum(xor(dobin(t,4),Soc));
    end
    if rem(I,1000)==0
      [1 ST I]
    end
  end
```

```
rx(1:16,1:(TN-stp)/stp+1)=0;
cnt=0;
CFa=[];

% Her 100 ölçüm için tahmin entropisini hesaplar
for pn=stp:stp:TN
   if rem(pn,1000)==0
      [2 ST pn]
   end
   for i=1:16
      h=H(1:pn,i);
      mh=sum(h)/pn;
      h=h-mh;
      hs=h'*h;
      for j=1:Ts
         t=TR(1:pn,j);
         mt=sum(t)/pn;
         t=t-mt;
         ts=t'*t;
         CF(i,j)=(h'*t)/sqrt(hs*ts); % Korelasyon matrisi
      end
      CFm(i)=max(abs((CF(i,:))));
   end
   CFa=[CFa CFm'];
   [a ix]=sort(CFm,'descend');
   cnt=cnt+1;
   rx(ix,cnt)=(1:16)'-1;
end
plot(rx(kcd+1,:));
xlabel('Ölçüm Sayısı (10^2)')
ylabel('Tahmin entropisi')

% Onluk tabandaki K sayısını k bite dönüştürür
function Y=dobin(K,k)
 A=dec2bin(K,k);
X=find(A=='1');
[m,n]=size(X);
Y(1:k)=0;
if n > 0
  for j=1:n
    Y(X(j))=1;
  end
else
  Y(1:k)=0;
end

% Bit dizisi L'yi onluk taban değerine dönüştürür
function M=todec(L)
N=0;
i=1;
for n=length(L)-1:-1:0
  N=N+L(i)*(2^n);
  i=i+1;
end
M=N;
```