Anti-Piracy Design of RF Transceivers

Alán Rodrigo Díaz-Rizo, Hassan Aboushady, Haralampos-G. Stratigopoulos

Anti-Piracy Design of RF Transceivers

Alán Rodrigo Díaz-Rizo, Hassan Aboushady, *Senior Member, IEEE*,
and Haralampos-G. Stratigopoulos, *Member, IEEE*

***Abstract*—We present a locking-based design-for-security methodology to prevent piracy of RF transceiver integrated circuits. The solution is called *SyncLock* as it locks the synchronization of the transmitter with the receiver. If a key other than the secret key is applied, synchronization and, thereby, communication fail. *SyncLock* is implemented using a novel locking concept consisting of two spatially separated mechanisms. A hard-coded error is hidden into the design to break synchronization while error correction, i.e., unlocking, takes place in another part of the design by applying the secret key. *SyncLock* offers several advantages: the secret key is unique, i.e., any incorrect key causes a denial-of-service, there is no performance penalty, it can be seemingly integrated into the digital design flow, area and power overheads are negligible, and it achieves maximum provable security thwarting all known counter-attacks. *SyncLock* is demonstrated with hardware measurements.**

***Index Terms*—Hardware security and trust, RF transceivers, wireless ICs, IC piracy, locking.**

I. Introduction

In the early days of the semiconductor industry, a single company possessed all the design know-how, tooling, fabrication facilities, and test equipment required to build end-to-end an Integrated Circuit (IC). Today, few such vertically integrated companies combining all the diverse competencies exist. We observe increasing globalization of design and manufacturing tasks and outsourcing to third parties. For instance, many companies are founded or have transitioned to be "fabless": they outsource the manufacturing step of their IC design to offshore foundries, many of which are located in separate continents. In this way, they do not need to bear the enormous costs of building, maintaining, and upgrading a manufacturing facility, which rise dramatically with each new technology node visited. Another trend is the rise of complex Systems-on-Chip (SoCs) where numerous general and specialized functions are integrated onto the same chip. Many companies do not have the know-how to design end-to-end a SoC, thus relying on third-party Intellectual Property (IP) blocks for building some of the functions.

A major security threat resulting from this globalized supply chain is piracy of IP blocks in ICs and SoCs or of the entire IC or SoC [1]. Piracy refers to cloning, overbuilding, remarking, and recycling of chips. More specifically, cloning consists of illegally copying a design and reusing it without the consent or knowledge of the design owner. It can be conducted by rogue agents in IC/SoC integration houses and foundries. It can also be conducted by an end-user through reverse-engineering of a legally purchased chip. In fact, nowadays there are increased capabilities for performing reverse-engineering of chips to extract the design netlist and other technology secrets [2]. Overbuilding can be performed by a foundry that holds the blueprint of the design and refers to producing and selling chips beyond the number agreed on in the contract with the chip design owner. Remarking can be performed by a test facility and refers to relabelling failing chips as functional. Recycling refers to scrapping a likely aged chip from a used board and re-entering it into the market as a "fresh" chip. Unauthorized chip use is often considered another form of piracy.

Piracy leads to counterfeit chips that are a serious threat to design houses (e.g., loss of know-how, sales, and brand name), governments (e.g., national security threat if counterfeit chips are used in critical infrastructure or defense), and the society as a whole (e.g., counterfeits are likely to be of lower quality and have shorter lifespan).

To defend against IP/IC piracy, IP/IC locking is considered as the strongest counter-measure [3]. Illustrated in Fig. 1, it is performed by the designer and consists in embedding a lock mechanism inside the IP/IC. The lock mechanism is a circuit that is mingled with the original circuit and is controlled by a key, which is typically in the form of a digital bit-string. The lock mechanism is transparent to the IP/IC such that upon application of the correct key the nominal functionality is restored. However, applying an incorrect key corrupts the functionality. The correct key is a designer's secret and is not shared with any potentially untrusted party, i.e., SoC integration house, foundry, or end-user. The chip is securely activated after fabrication by storing the secret key in a Tamper-Proof Memory (TPM) such that it is erased on detecting a probing attempt. In this case, the secret key is common to all chips, thus if it is leaked any chip instance can be unlocked. Alternatively, a key provisioning on-die unit can be used to ensure that each chip is unlocked only by a user key, which is unique to that chip [4]. A standard scheme [5] uses a Physical Unclonable Function (PUF) [6] to generate on-die a chip identification key, then a chip-unique user key is generated by XORing the identification key with the common key. The common key is generated internally by XORing the user and identification keys. Read access to the PUF output is disabled after recording to prevent probing attacks by end-users. Another key management scheme uses a PUF and RSA encryption to securely activate the chip remotely [7]. IP/IC

Manuscript received July 22, 2022; revised September 16, 2022; accepted September 30, 2022. This work was supported by the ANR STEALTH project under Grant ANR-17-CE24-0022-01. The work of A. R. Díaz Rizo was supported by the Mexican National Council for Science and Technology (CONACYT) through Fellowship. This article was recommended by Associate Editor D. Zhao. (Corresponding author: Haralampos-G. Stratigopoulos.)

Alán Rodrigo Díaz Rizo, Hassan Aboushady, and Haralampos-G. Stratigopoulos are with the Sorbonne Université, CNRS, LIP6, 75005 Paris, France (e-mail: Alan-Rodrigo.Diaz-Rizo@lip6.fr; hassan.aboushady@lip6.fr; haralampos.stratigopoulos@lip6.fr).

Digital Object Identifier 10.1109/TCSI.2022.XXXXXXX

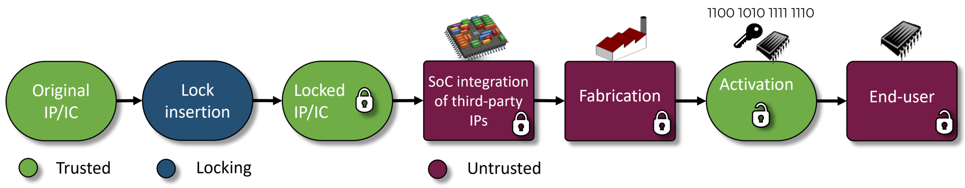


Fig. 1. Locking methodology.

locking protects an IP/IC against potential attackers located anywhere in the supply chain, as well as against malicious end-users. It can protect also against recycling facilities as long as the key is reloaded every time the IC is powered on.

In this paper, we propose a locking technique for RF transceivers. Even the most modern RF transceiver designs (for example see [8]–[12]) are not made with anti-piracy security in mind. To this end, we propose a security plug-in that can transform any design to a key-controlled version. In this way, the intellectual property of the design can be protected against piracy attempts at any point across the supply chain and its authenticity can be traced along its entire life-cycle.

For an RF transceiver, being a mixed-signal design, one can leverage existing techniques for locking part of its digital section, for locking blocks in its analog section, or for locking it at system-level, i.e., by exploiting its programmability features. These generic techniques, however, have shown to be vulnerable to attacks, as it will be described in more detail in Section II. Herein, we propose a domain-specific locking technique for RF transceivers that takes advantage of a specific part of the signal processing chain found in any RF transceiver.

More specifically, the proposed RF transceiver locking technique, called *SyncLock*, acts on the synchronization of the transmitter with the receiver. Upon application of an incorrect key, *SyncLock* disables the synchronization, thus the wireless communication link crashes. The synchronization is commonly set via a preamble that is appended to the beginning of data frames. *SyncLock* is based on two spatially separated hardware-level mechanisms. The first mechanism hides a hard-coded error into the design of the data frame generator corrupting the preamble of the data frame. The second mechanism is located upstream in the signal processing chain into the preamble generator and its goal is to corrupt the preamble so as to cancel out the downstream corruption. The corruption applied by the second mechanism is key-controlled, with a single correct key being capable of counterbalancing the two spatially separated preamble corruptions.

SyncLock is generally applicable to any RF transceiver architecture, any wireless communication protocol using correlation-based synchronization algorithms, and any modulation scheme. As the lock mechanism is embedded into the baseband Digital Signal Processor (DSP), *SyncLock* can be effortlessly integrated into the digital design flow. On the other hand, the sensitive Analog Front-End (AFE) is left intact which is an essential characteristic of *SyncLock* allowing for its wide adoption by analog IC designers. *SyncLock* elegantly achieves all locking objectives: (a) locking is totally transparent to the RF transceiver operation when the correct secret key is applied; (b) applying invalid keys breaks the operation; (c) area and power overheads are minimal; (d) all known counter-attacks in both the analog and digital domains are thwarted. We demonstrate *SyncLock* in hardware using the Software Defined Radio (SDR) bladeRF board from Nuand$^{\text{TM}}$ [13].

SyncLock was originally proposed in [14]. This paper describes a new design and implementation of *SyncLock* that offers higher security compared to its preliminary version in [14]. In addition, the paper provides an in-depth analysis of the inner workings of *SyncLock*.

The rest of the article is structured as follows. In Section II, we discuss the prior art on locking and anti-piracy design of analog and mixed-signal ICs. In Section III, we present the new implementation and design of *SyncLock*. Section III concludes by presenting the first *SyncLock* implementation in [14] as a sub-case and comparing the two. In Section IV, we present the hardware platform used for demonstrating the locking efficiency of *SyncLock* in Section V. Section VI discusses related locking and obfuscation approaches, including approaches based on the corrupt-and-correct principle utilized by *SyncLock*, and explains their differences as compared to *SyncLock*. Section VI also discusses existing counter-attacks for locking approaches based on the corrupt-and-correct principle. Section VII provides the threat model and analyzes the resilience of *SyncLock* to all known counter-attacks. Section VIII concludes this article.

II. Prior art on locking and anti-piracy design

The first locking technique was proposed originally for digital circuits [7], a.k.a. logic locking or logic encryption. Since then, several logic locking techniques were proposed aiming at reducing Power, Performance, and Area (PPA) penalties, increasing corruption for invalid keys, and circumventing counter-attacks that were developed in the meantime aiming at exposing security vulnerabilities of logic locking, i.e., finding the secret key with reasonable effort or identifying and subsequently removing the lock. For a recent review of logic locking techniques and counter-attacks the reader is referred to [3]. Leveraging logic locking to lock a mixed-signal design via locking its digital section was proposed in [15]–[17]. In [15], locking targeted the digital processor in the feedback calibration loop, while in [16], [17] locking targeted digital blocks within the signal processing chain. This latter locking approach, called *MixLock*, was demonstrated recently for RF transceivers [18]. A state-of-the-art logic locking technique called Stripped Functionality Logic Locking (SFLL)-rem [19] was employed in [18]. However, recently a counter-attack based on structural analysis of the netlist was

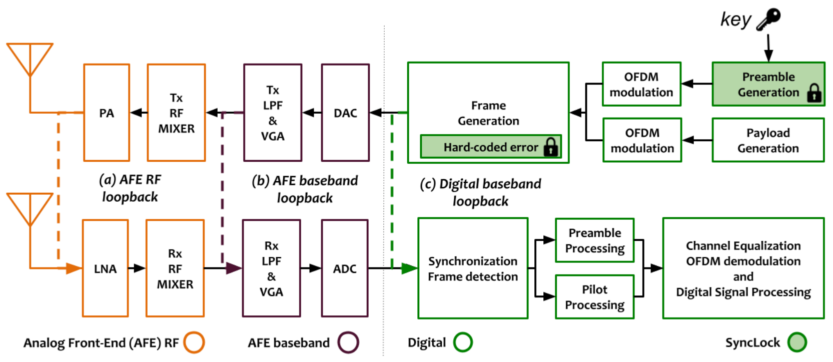


Fig. 2. Simplified architecture of a wireless device IC with *SyncLock* embedded.

shown to break SFLL-rem within seconds [20]. Essentially, there is an ongoing "cat-and-mouse" game between logic locking defenses and counter-attacks. Every newly introduced logic locking technique is considered secure until shortly after a counter-attack heuristic is demonstrated that breaks it.

For locking analog blocks the existing technique is biasing locking which aims at controlling the bias generation with the key. Unless the correct key is provided the analog block is incorrectly biased meaning that the quiescent point of transistors is not the desired one resulting in performance degradation or malfunction. For RF transceivers, one can perform biasing locking in blocks of the AFE, i.e., Low Noise Amplifier (LNA), Power Amplifier (PA), Phase-Locked Loop (PLL), data converters, etc. Several embodiments of biasing locking exist, including obfuscating the geometry of a bias transistor [21], designing key-controlled current mirrors [5], and replacing the biasing circuit with an alternative key-controlled bias generator, e.g., based on an on-chip neural network [22] or a programmable memristor crossbar [23]. Biasing locking may result in imprecise or unstable biasing and, besides, recently counter-attacks were proposed based on Satisfiability Modulo Theory (SMT) [24] and optimization [25], [26] that break this type of defense.

System-level locking can be achieved via calibration locking which makes the compensation of process variations or adaptation to different operation modes key-dependent. Techniques in this category include logic locking of the digital section of the calibration loop [15], treating digital programmability as a natural secret key [27]–[29], and making the calibration range key-dependent [30]. To be secure calibration locking requires that the calibration algorithm is complex enough to be devised or re-designed in hardware by the attacker, an assumption that is not always met.

Besides locking, other anti-piracy methods include split manufacturing [31] that protects only against an untrusted foundry and camouflaging [32] that protects only against reverse-engineering. Split manufacturing has been demonstrated for RF designs [33] and camouflaging ideas for analog and mixed-signal ICs include multi-threshold transistor design [34] and obfuscating the geometry of layout components [35]. Unlike split manufacturing and camouflaging, locking offers an end-to-end protection against all potential piracy threat scenarios.

III. *SyncLock*

A. *Principle of operation*

SyncLock is a security mechanism for preventing piracy of RF transceivers. It can be viewed as a domain-specific logic locking capitalizing on a specific digital signal processing path in RF transceivers. The underlying idea is to lock the preamble that allows the synchronization process between the transmitter and the receiver. By blocking the synchronization, wireless receivers are unable to find the start of the received frame, thus the wireless communication fails.

A simplified architecture of a wireless IC with *SyncLock* embedded is shown in Fig. 2. *SyncLock* acts on two different parts of the design. First, it modifies the frame generation block at the end of the baseband DSP chain of the transmitter by corrupting the preamble of each transmitted frame. The introduced error is hard-coded such that after logic synthesis of the DSP it is impossible to be traced and recovered by structural analysis of the netlist. Then, it modifies the preamble generation block at the beginning of the baseband DSP chain such that the output preamble is key-controlled. To enable the synchronization process, the key must neutralize the unknown to the attacker later corruption in the frame generation block.

B. *Preamble generation*

In all wireless communication protocols, the payload is transmitted along with the physical layer (PHY) specifications. The baseband DSP prepares the payload in a frame format for transmission. The PHY Protocol Data Unit (PPDU) frame format of an Orthogonal Frequency-Division Multiplexing (OFDM) IEEE 802.11 transmission consists of several OFDM symbols. These symbols are divided into three parts: preamble (a.k.a SYNC), header (a.k.a SIGNAL), and payload (a.k.a DATA). The preamble section is composed of two different training symbol sequences, namely a Short Training Sequence (STS) and a Long Training Sequence (LTS). Fig. 3 shows the PPDU of an IEEE 802.11 transmission with the above three parts as defined in the IEEE 802.11 standard [36]. The STS field consists of 10 identical short symbol repetitions and is used for timing acquisition based on the Schmidl and Cox algorithm [37], i.e., for synchronization or start of frame detection and for coarse frequency offset estimation. The LTS field consists of 2 long symbol repetitions and is used for channel estimation and fine frequency offset estimation [36].

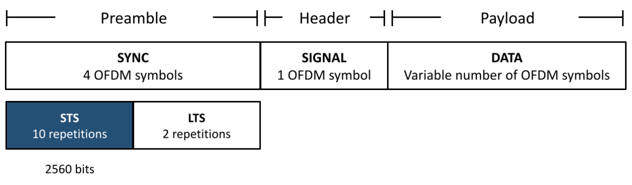


Fig. 3. PPDU frame format of an OFDM IEEE 802.11 transmission.

TABLE I
STS_{nom} AS DEFINED IN THE IEEE 802.11 STANDARD [36].

Sample (k)	Floating-point (I,Q)	Fixed-point (I,Q)
0	0.04600 , 0.04600	16’h02F2 , 16’h02F2
1	-0.13245 , 0.00234	16’hF786 , 16’h0026
2	-0.01347 , -0.07853	16’hFF23 , 16’hFAF9
3	0.14276 , -0.01265	16’h0923 , 16’hFF31
4	0.09200 , 0.00000	16’h05E3 , 16’h0000
5	0.14276 , -0.01265	16’h0923 , 16’hFF31
6	-0.01347 , -0.07853	16’hFF23 , 16’hFAF9
7	-0.13245 , 0.00234	16’hF786 , 16’h0026
8	0.04600 , 0.04600	16’h02F2 , 16’h02F2
9	0.00234 , -0.13245	16’h0026 , 16’hF786
10	-0.07853 , -0.01347	16’hFAF9 , 16’hFF23
11	-0.01265 , 0.14276	16’hFF31 , 16’h0923
12	0.00000 , 0.09200	16’h0000 , 16’h05E3
13	-0.01265 , 0.14276	16’hFF31 , 16’h0923
14	-0.07853 , -0.01347	16’hFAF9 , 16’hFF23
15	0.00234 , -0.13245	16’h0026 , 16’hF786

More specifically, as defined in the IEEE 802.11 standard [36], the nominal STS is divided into two parts, denoted here by $STS_{nom}I$ and $STS_{nom}Q$, corresponding to the real I and imaginary Q channels, respectively. $STS_{nom}I, Q$ is composed of 10 repetitions of the 16 samples of 16 bits each shown in Table I in floating-point and fixed-point representations. Thus, $STS_{nom}I, Q$ is composed of $10 * 16 * 16 = 2560$ bits in total.

Each sample of $STS_{nom}I, Q$ is generated in the baseband DSP by the preamble generation block shown in Fig. 4. There are in total 13 multiplexers (MUXes) per I/Q branch where the i-th MUX receives a constant 16-bit input DATA_i with values shown in Table II. The SEL input of the MUXes is a 4-bit word and selects the creation of one of the 16 samples of the sequence. The position of the selected bit of DATA_i that is transferred at the output of each MUX equals the decimal representation of the SEL input. The 16-bit fixed-point I and Q values of the sample are then created by concatenating the outputs of the MUXes according to the schemes shown in the second and fifth rows of Table III for the I and Q branches, respectively. The same hardware and concatenation operations are used to generate any sample k by setting the input SEL equal to k in decimal.

For example, let us consider the first sample, i.e., $k = 0$, in the I branch which has a fixed-point value of 16’h02F2 in hexadecimal representation. In this case, SEL = 4’b0000 selecting the first bit position of the DATA_i inputs of the MUXes, as shown in blue in the I branch part of Table II. The concatenation of the MUXes output is shown in blue in the third row of Table III resulting in the desired value of 16’h02F2. As a second example, let us consider the fourth sample, i.e., $k = 3$, in the Q branch with a fixed-point value of 16’hFF31 in hexadecimal representation. SEL = 4’b0011 selecting the fourth bit position of the DATA_i inputs of the MUXes, as shown in red in the Q branch part of Table II.

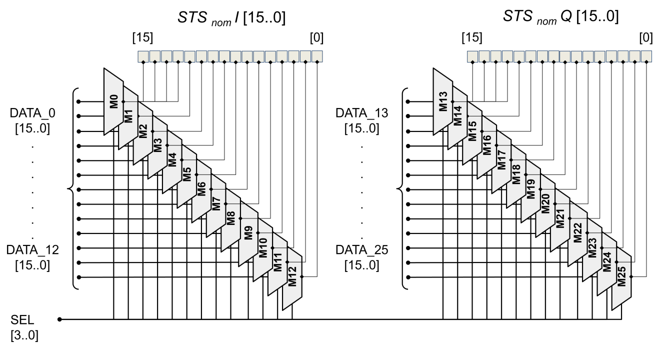


Fig. 4. Original preamble generation block.

TABLE II
INPUT VALUES OF MUXES IN THE PREAMBLE GENERATION BLOCK.

MUX	Name	Input (16-bit)
	I branch	
M0	DATA_0	16’b0110 1100 1100 011**0**
M1	DATA_1	16’b0110 1100 0110 110**0**
M2	DATA_2	16’b0010 1000 1101 011**0**
M3	DATA_3	16’b0110 1101 1100 011**1**
M4	DATA_4	16’b0010 1000 1111 111**0**
M5	DATA_5	16’b0100 0101 1001 001**1**
M6	DATA_6	16’b0100 0101 0001 000**1**
M7	DATA_7	16’b1110 1111 0111 110**1**
M8	DATA_8	16’b0110 1101 0000 000**1**
M9	DATA_9	16’b0100 0100 0000 000**0**
M10	DATA_10	16’b1000 0010 1000 001**0**
M11	DATA_11	16’b1000 0011 1111 111**1**
M12	DATA_12	16’b0110 1100 0111 110**0**
	Q branch	
M13	DATA_13	16’b1100 0110 0110 **1**100
M14	DATA_14	16’b0110 1100 0110 **1**100
M15	DATA_15	16’b1101 0110 0010 **1**000
M16	DATA_16	16’b1100 0111 0110 **1**101
M17	DATA_17	16’b1111 1110 0010 **1**000
M18	DATA_18	16’b1001 0011 0100 **0**101
M19	DATA_19	16’b0001 0001 0100 **0**101
M20	DATA_20	16’b0111 1101 1110 **1**111
M21	DATA_21	16’b0000 0001 0110 **1**101
M22	DATA_22	16’b0000 0000 0100 **0**100
M23	DATA_23	16’b1000 0010 1000 **0**010
M24	DATA_24	16’b1111 1111 1000 **0**011
M25	DATA_25	16’b0111 1100 0110 **1**100

The concatenation of the MUXes output is shown in red in the sixth row of Table III resulting in the desired value of 16’hFF31.

C. Locking mechanism

SyncLock acts specifically on the generation of the STS. The locking mechanism of *SyncLock* is divided into two parts embedded into the preamble and frame generation blocks, as shown in Fig. 5. In the frame generation block, the STS originally generated by the preamble generation block is embedded into the frame for transmission, with the final STS denoted by STS_{out}. The design owner deliberately corrupts the incoming STS to the frame generation block prior to frame creation by XORing it with the output of a nonlinear module $f(\cdot)$. This module implements a feedback loop involving STS_{out} and a hard-coded key, denoted by key_{h-c}. In the preamble generation block, an XOR operation is performed between the key-bits stored in the TPM and STS_{nom}, thus corrupting

TABLE III
CONCATENATION OPERATION AT THE OUTPUTS OF THE MUXES.

		I branch			
	Concatenation of MUXes (M#)	M0,M0,M0,M0	M1,M2,M3,M4	M5,M6,M7,M8	M9,M10,M11,M12
SEL = 4'b0000 (first sample)	Fixed-point binary value	**0000**	**0010**	**1111**	**0010**
	Fixed-point hexadecimal value	0	2	F	2
		Q branch			
	Concatenation of MUXes (M#)	M13,M13,M13,M13	M14,M15,M16,M17	M18,M19,M20,M21	M22,M23,M24,M25
SEL = 4'b0011 (fourth sample)	Fixed-point binary value	**1111**	**1111**	**0011**	**0001**
	Fixed-point hexadecimal value	F	F	3	1

STS_{nom} to a faulty value, denoted by STS_{faulty}. Herein and in the rest of the article we refer to a TPM but any other on-die key provisioning scheme can be used instead. The equations describing the operations are

$$STS_{faulty} = STS_{nom} \oplus key \tag{1}$$

$$STS_{out} = STS_{faulty} \oplus f(STS_{out}, key_{h-c}) \tag{2}$$

Combining Eqs. (1)-(2) and using the associative property $(A \oplus B) \oplus C = A \oplus (B \oplus C)$ of the XOR function, the system equation becomes

$$STS_{out} = STS_{nom} \oplus (key \oplus f(STS_{out}, key_{h-c})) \tag{3}$$

Thus, using the self-inverse property $A \oplus A = 0$ of the XOR function, $STS_{out} = STS_{nom}$ if and only if $key = f(STS_{nom}, key_{h-c})$

$$STS_{out} = STS_{nom} \iff key = f(STS_{nom}, key_{h-c}) \tag{4}$$

The *SyncLock* mechanism can be viewed as two spatially separated XOR-based stream ciphers controlled by two secret keys, one stored in the TPM and the other one being hard-coded. The generated STS_{nom} by the original preamble generation block, i.e., the plaintext, is encrypted by the key to STS_{faulty}, i.e., the ciphertext, so as to "match" the hidden hard-coded decryption that comes downstream in the DSP chain at the frame generation block. The secret correct key must be loaded in the TPM of the chip for correct deciphering. Applying incorrect keys introduces two uncorrelated STS corruptions at two distinct blocks of the DSP chain which breaks the synchronization.

As mentioned in Section III-B, for each channel I or Q, at any point in the signal processing chain, STS is composed of 2560 bits and is processed in 160 blocks with each block corresponding to one sample of 16 bits shown in Table I. Each key is composed of 512 bits divided into two parts of 256 bits for each channel. Thus, for each channel, a key is divided into 16 blocks of 16 bits each. This means that for each channel the key is repeatedly applied 10 times for every 16 blocks of STS.

The implementation specifics showing how STS_{out} converges in Eq. (2) will be described in detail in Section III-E.

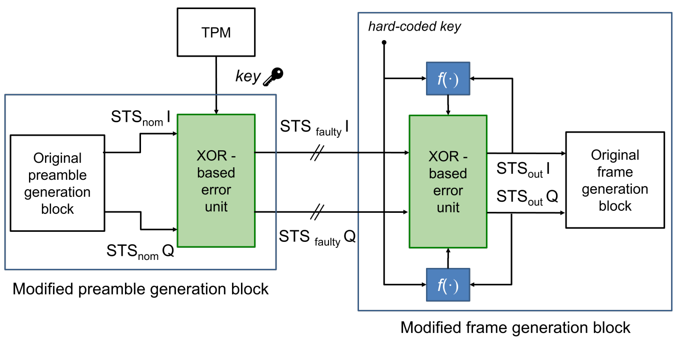


Fig. 5. *SyncLock* principle of operation.

D. Choice of function $f(\cdot)$

As will be explained in detail in Section VII-B5, the function $f(\cdot)$ is introduced to circumvent the Known-Plaintext Attack (KPA), which is a vulnerability of the preliminary version of *SyncLock* in [14]. The choice of $f(\cdot)$ is free, leaving in theory unlimited freedom to the defender. It can also change from one design to another or across design iterations to update the key for increased security.

In our current implementation, $f(\cdot)$ is a two-step function. It first performs 16-bit parallel XORing of STS_{out} with the hard-coded key key_{h-c}, then it applies to the result a circular shift operation, a.k.a. bitwise rotation, i.e., $f(STS_{out}, key_{h-c}) = (STS_{out} \oplus key_{h-c}) >> b$, where $>>$ is the bitwise rotation operation and b is the number of bit rotations to the right. Other possibilities include bitwise logical operations between STS_{out} and key_{h-c}, bit scrambling or substitution after the XORing between STS_{out} and key_{h-c}, etc.

The function $f(\cdot)$ can be executed in a single clock cycle using any of the above bitwise operations with depth equal to one. For example, in our current implementation, the XOR operation $STS_{out} \oplus key_{h-c}$ needs one clock cycle, while the rotation can be simply implemented by rotating the wiring of the 16-bit output of $STS_{out} \oplus key_{h-c}$ when it is fed into the XOR function with STS_{faulty}. To accommodate this one clock cycle delay and guarantee convergence, as we will see in Section III-E that presents the implementation specifics, we let the first 16-bit block of STS pass without being processed by the XOR stream ciphers, whereas the XOR steam ciphers come into play starting from the second 16-bit block of STS. Essentially, from this point onward, the k-th block of STS_{faulty} is XORed with the output of $f(\cdot)$, which has been computed with the $(k-1)$-th blocks of STS_{out} and key_{h-c}, to produce the k-th block of STS_{out}.

In general, if $f(\cdot)$ had a larger depth needing n clock cycles to be executed, then the XOR stream ciphers would come into play starting from the n-th 16-bit block of STS.

E. Implementation specifics

Without loss of generality, let us consider the I channel. Let $s_I[k]$ denote the 16-bit k-th sample in Table I, $k = 0, \cdots, 15$, e.g., $s_I[0] = 16'h02F2$, $s_I[1] = 16'hF786$, etc. Let also $STS_{faulty}I$ and $STS_{out}I$ denote the real parts of STS_{faulty} and STS_{out}, respectively, each composed of 2560 bits similarly to $STS_{nom}I$. As explained Section III-B, by construction, STS is divided into words of 16-bits corresponding to samples $s_I[k]$. Starting with $STS_{nom}I$, it is divided into 16-bit words $STS_{nom}I[j]$ corresponding to bit positions from $j*16$ to $(j*16+15)$, $j = 0, \cdots, 159$. $STS_{nom}I[j]$ can be expressed in terms of $s_I[k]$ as

$$STS_{nom}I[j] = s_I[mod(j,16)] \tag{5}$$

The key and hard-coded key are composed of 512-bits each and are reused in every repetition of the 16 samples. Each key can be divided into two equal 256-bit parts, with the first part corresponding to the I channel and the second part to the Q channel. For the I channel, the key and hard-coded key are denoted by $keyI$ and $key_{h-c}I$, respectively. Similar to STS values, each key is divided into 16-bit words corresponding to samples $s_I[k]$. For example, $keyI$ results from the concatenation $keyI = keyI[0]...keyI[15]$, where $keyI[n]$ is the part of $keyI$ in bit positions from $n*16$ to $n*16+15$, $n = 0, \cdots, 15$.

Using the above definitions, we can now formally explain the *SyncLock* implementation. Since the nonlinear module evaluates STS_{out} in a feedback loop, the system essentially incorporates an internal memory and a valid STS_{out} would become available starting from the second sample of the first repetition. To remove this delay, for the first sample of the first repetition both XOR-based error units in the preamble and frame generation blocks are bypassed, i.e., $STS_{faulty}I[0] = STS_{nom}[0]$ and $STS_{out}I[0] = STS_{faulty}I[0]$, that is, we force the initial condition

$$STS_{out}I[0] = STS_{nom}I[0] \tag{6}$$

From the second sample of the first repetition onward, the key and the two XOR-based error units start intervening in the computation. Specifically, using the above definitions, for $j \geq 1$ we have

$$STS_{faulty}I[j] = STS_{nom}I[j] \oplus keyI[mod(j,16)], \quad j = 1, \cdots, 159 \tag{7}$$

$$STS_{out}I[j] = STS_{faulty}I[j] \oplus f(STS_{out}I[j-1], key_{h-c}I[mod(j,16)]), \quad j = 1, \cdots, 159. \tag{8}$$

Substituting Eq. (7) into Eq. (8) we have

$$STS_{out}I[j] = STS_{nom}I[j] \oplus (keyI[mod(j,16)] \oplus f(STS_{out}I[j-1], key_{h-c}I[mod(j,16)])), \quad j = 1, \cdots, 159. \tag{9}$$

The hard-coded key is set arbitrarily by the designer. The key is then selected such that $STS_{out}I = STS_{nom}I$ which from Eq. (9) is satisfied by the identity

$$keyI[mod(j,16)]) = f(STS_{nom}I[j-1], key_{h-c}I[mod(j,16)]) \quad j = 1, \cdots, 159. \tag{10}$$

This results in

$$keyI[0] = f(STS_{nom}I[15], key_{h-c}I[0]) \tag{11}$$

$$keyI[n] = f(STS_{nom}I[n-1], key_{h-c}I[n]), \quad n = 1, \cdots, 15. \tag{12}$$

An excerpt of the computations for the first 19 real samples $j = 0, \cdots, 18$ of $STS_{nom}I$, i.e., comprising a complete first iteration and 3 samples in the second iteration, is shown in Table IV for three different key cases, namely (a) incorrect zero key; (b) random incorrect key; and (c) correct key. Bitwise rotation with $b = 1$ is used as the nonlinear function. The key is repeated every 16 samples, but the XOR operations are bypassed for the first sample of the first iteration so as to force the initial condition for the feedback loop. The locking mechanism becomes active starting from the second sample of the first iteration and stays active until the end of the STS_{nom} transmission to the frame generation block. As it can be seen, STS_{out} is generated without errors only for the correct key.

F. Key size

The above *SyncLock* implementation has the advantageous property that there is a single correct key enabling synchronization, while any other key results in no synchronization, i.e., there are no approximate keys. This property stems from the nonlinear module inside the frame generation block. More specifically, considering for example a bit rotation function, for a single bit flip of the secret key, there is a large and arbitrary number of bit flips in STS_{out}. Thus, even for an incorrect key with Hamming Distance (HD) of 1 from the correct key, STS_{out} contains a high number of errors. As a result, this *SyncLock* implementation has a full effective 512-bit key size.

So far, we have assumed full key sizes of 512 bits. However, this is a rather unnecessarily large key size from a security point view. Typically, a key size of 64 bits suffices to guarantee high resilience against brute-force and optimization attacks. Therefore, we can consider keys of smaller size that can be composed using any key-bits of the original 512-bit keys since all key-bits are effective. Reducing the keys' size has the advantage of reducing the die area of the TPM, or in general the die area of the implemented on-die key provisioning scheme, as well as the die area of the lock mechanism itself.

TABLE IV

VALUES OF THE MAIN SIGNALS OF THE *SyncLock* LOCKING MECHANISM FOR THREE KEY CASES: INCORRECT ZERO KEY, RANDOM INCORRECT KEY, AND CORRECT KEY. THE EXAMPLE CONSIDERS A BIT ROTATION FUNCTION WITH $b = 1$ AND SHOWS THE COMPUTATIONS DURING THE TRANSMISSION OF THE FIRST 19 SAMPLES OF STS_{nom} FOR THE I CHANNEL.

(a) Incorrect zero key

Preamble generation block				Frame generation block		
j	$STS_{nom}I[j]$	$keyI[mod(j,16)]$	$STS_{faulty}I[j]$	$key_{h-c}I[mod(j,16)]$	$f(STS_{out}I[j-1], key_{h-c}I[mod(j,16)])$	$STS_{out}I[j]$
0	16'h02F2	16'h0000 (Bypassed)	16'h02F2	16'h0052 (Bypassed)	(Bypassed)	16'h02F2
1	16'hF786	16'h0000	16'hF786	16'hFAFA	(16'h02F2 $\oplus$ 16'hFAFA) $>>$ b = 16'h7C04	16'h8B82
2	16'hFF23	16'h0000	16'hFF23	16'hFEA6	(16'h8B82 $\oplus$ 16'hFEA6) $>>$ b = 16'h3A92	16'hC5B1
3	16'h0923	16'h0000	16'h0923	16'h5216	(16'hC5B1 $\oplus$ 16'h5216) $>>$ b = 16'hCBD3	16'hC2F0
4	16'h05E3	16'h0000	16'h05E3	16'h5614	(16'hC2F0 $\oplus$ 16'h5614) $>>$ b = 16'h4A72	16'h4F91
5	16'h0923	16'h0000	16'h0923	16'hCAFE	(16'h4F91 $\oplus$ 16'hCAFE) $>>$ b = 16'hC2B7	16'hCB94
6	16'hFF23	16'h0000	16'hFF23	16'hFEF8	(16'hCB94 $\oplus$ 16'hFEF8) $>>$ b = 16'h1AB6	16'hE595
7	16'hF786	16'h0000	16'hF786	16'h4516	(16'hE595 $\oplus$ 16'h4516) $>>$ b = 16'hD041	16'h27C7
8	16'h02F2	16'h0000	16'h02F2	16'h0158	(16'h27C7 $\oplus$ 16'h0158) $>>$ b = 16'h934F	16'h91BD
9	16'h0026	16'h0000	16'h0026	16'hCAFE	(16'h91BD $\oplus$ 16'hCAFE) $>>$ b = 16'hADA1	16'hAD87
10	16'hFAF9	16'h0000	16'hFAF9	16'hFFAC	(16'hAD87 $\oplus$ 16'hFFAC) $>>$ b = 16'hA915	16'h53EC
11	16'hFF31	16'h0000	16'hFF31	16'hAAAA	(16'h53EC $\oplus$ 16'hAAAA) $>>$ b = 16'h7CA3	16'h8392
12	16'h0000	16'h0000	16'h0000	16'h003A	(16'h8392 $\oplus$ 16'h003A) $>>$ b = 16'h41D4	16'h41D4
13	16'hFF31	16'h0000	16'hFF31	16'h0569	(16'h41D4 $\oplus$ 16'h0569) $>>$ b = 16'hA25E	16'h5D6F
14	16'hFAF9	16'h0000	16'hFAF9	16'hFFC2	(16'h5D6F $\oplus$ 16'hFFC2) $>>$ b = 16'hD156	16'h2BAF
15	16'h0026	16'h0000	16'h0026	16'h9623	(16'h2BAF $\oplus$ 16'h9623) $>>$ b = 16'h5EC6	16'h5EE0
16	16'h02F2	16'h0000	16'h02F2	16'h0052	(16'h5EE0 $\oplus$ 16'h0052) $>>$ b = 16'h2F59	16'h2DAB
17	16'hF786	16'h0000	16'hF786	16'hFAFA	(16'h2DAB $\oplus$ 16'hFAFA) $>>$ b = 16'hEBA8	16'h1C2E
18	16'hFF23	16'h0000	16'hFF23	16'hFEA6	(16'h1C2E $\oplus$ 16'hFEA6) $>>$ b = 16'h7144	16'h8E67

(b) Random incorrect key

Preamble generation block				Frame generation block		
j	$STS_{nom}I[j]$	$keyI[mod(j,16)]$	$STS_{faulty}I[j]$	$key_{h-c}I[mod(j,16)]$	$f(STS_{out}I[j-1], key_{h-c}I[mod(j,16)])$	$STS_{out}I[j]$
0	16'h02F2	16'h2324 (Bypassed)	16'h02F2	16'h0052 (Bypassed)	(Bypassed)	16'h02F2
1	16'hF786	16'hCAFE	16'h3D78	16'hFAFA	(16'h02F2 $\oplus$ 16'hFAFA) $>>$ b = 16'h7C04	16'h417C
2	16'hFF23	16'h5249	16'hAD6A	16'hFEA6	(16'h417C $\oplus$ 16'hFEA6) $>>$ b = 16'h5FED	16'hF287
3	16'h0923	16'h3216	16'h3B35	16'h5216	(16'hF287 $\oplus$ 16'h5216) $>>$ b = 16'hD048	16'hEB7D
4	16'h05E3	16'hEFAC	16'hEA4F	16'h5614	(16'hEB7D $\oplus$ 16'h5614) $>>$ b = 16'hDEB4	16'h34FB
5	16'h0923	16'h1234	16'h1B17	16'hCAFE	(16'h34FB $\oplus$ 16'hCAFE) $>>$ b = 16'hFF02	16'hE415
6	16'hFF23	16'hAFC5	16'h50E6	16'hFEF8	(16'hE415 $\oplus$ 16'hFEF8) $>>$ b = 16'h8D76	16'hDD90
7	16'hF786	16'hDE18	16'h299E	16'h4516	(16'hDD90 $\oplus$ 16'h4516) $>>$ b = 16'h4C43	16'h65DD
8	16'h02F2	16'h0090	16'h0262	16'h0158	(16'h65DD $\oplus$ 16'h0158) $>>$ b = 16'hB242	16'hB020
9	16'h0026	16'hFE10	16'hFE36	16'hCAFE	(16'hB020 $\oplus$ 16'hCAFE) $>>$ b = 16'h3D6F	16'hC359
10	16'hFAF9	16'h3620	16'hCCD9	16'hFFAC	(16'hC359 $\oplus$ 16'hFFAC) $>>$ b = 16'h9E7A	16'h52A3
11	16'hFF31	16'h5148	16'hAE79	16'hAAAA	(16'h52A3 $\oplus$ 16'hAAAA) $>>$ b = 16'hFC04	16'h527D
12	16'h0000	16'h6696	16'h6696	16'h003A	(16'h527D $\oplus$ 16'h003A) $>>$ b = 16'hA923	16'hCFB5
13	16'hFF31	16'hA5CD	16'h5AFC	16'h0569	(16'hCFB5 $\oplus$ 16'h0569) $>>$ b = 16'h656E	16'h3F92
14	16'hFAF9	16'hB517	16'h4FEE	16'hFFC2	(16'h3F92 $\oplus$ 16'hFFC2) $>>$ b = 16'h6028	16'h2FC6
15	16'h0026	16'h9ED1	16'h9EF7	16'h9623	(16'h2FC6 $\oplus$ 16'h9623) $>>$ b = 16'hDCF2	16'h4205
16	16'h02F2	16'h2324	16'h21D6	16'h0052	(16'h4205 $\oplus$ 16'h0052) $>>$ b = 16'hA12B	16'h80FD
17	16'hF786	16'hCAFE	16'h3D78	16'hFAFA	(16'h80FD $\oplus$ 16'hFAFA) $>>$ b = 16'hBD03	16'h807B
18	16'hFF23	16'h5249	16'hAD6A	16'hFEA6	(16'h807B $\oplus$ 16'hFEA6) $>>$ b = 16'hBF6E	16'h1204

(c) Correct key

Preamble generation block				Frame generation block		
j	$STS_{nom}I[j]$	$keyI[mod(j,16)]$	$STS_{faulty}I[j]$	$key_{h-c}I[mod(j,16)]$	$f(STS_{out}I[j-1], key_{h-c}I[mod(j,16)])$	$STS_{out}I[j]$
0	16'h02F2	16'h003A (bypassed)	16'h02F2	16'h0052 (bypassed)	(bypassed)	16'h02F2
1	16'hF786	16'h7C04	16'h8B82	16'hFAFA	(16'h02F2 $\oplus$ 16'hFAFA) $>>$ b = 16'h7C04	16'hF786
2	16'hFF23	16'h0490	16'hFBB3	16'hFEA6	(16'hF786 $\oplus$ 16'hFEA6) $>>$ b = 16'h0490	16'hFF23
3	16'h0923	16'hD69A	16'hDFB9	16'h5216	(16'hFF23 $\oplus$ 16'h5216) $>>$ b = 16'hD69A	16'h0923
4	16'h05E3	16'hAF9B	16'hAA78	16'h5614	(16'h0923 $\oplus$ 16'h5614) $>>$ b = 16'hAF9B	16'h05E3
5	16'h0923	16'hE78E	16'hEEAD	16'hCAFE	(16'h05E3 $\oplus$ 16'hCAFE) $>>$ b = 16'hE78E	16'h0923
6	16'hFF23	16'hFBED	16'h04CE	16'hFEF8	(16'h0923 $\oplus$ 16'hFEF8) $>>$ b = 16'hFBED	16'hFF23
7	16'hF786	16'hDD1A	16'h2A9C	16'h4516	(16'hFF23 $\oplus$ 16'h4516) $>>$ b = 16'hDD1A	16'hF786
8	16'h02F2	16'h7B6F	16'h799D	16'h0158	(16'hF786 $\oplus$ 16'h0158) $>>$ b = 16'h7B6F	16'h02F2
9	16'h0026	16'h6406	16'h6420	16'hCAFE	(16'h02F2 $\oplus$ 16'hCAFE) $>>$ b = 16'h6406	16'h0026
10	16'hFAF9	16'h7FC5	16'h853C	16'hFFAC	(16'h0026 $\oplus$ 16'hFFAC) $>>$ b = 16'h7FC5	16'hFAF9
11	16'hFF31	16'hA829	16'h5718	16'hAAAA	(16'hFAF9 $\oplus$ 16'hAAAA) $>>$ b = 16'hA829	16'hFF31
12	16'h0000	16'hFF85	16'hFF85	16'h003A	(16'hFF31 $\oplus$ 16'h003A) $>>$ b = 16'hFF85	16'h0000
13	16'hFF31	16'h82B4	16'h7D85	16'h0569	(16'h0000 $\oplus$ 16'h0569) $>>$ b = 16'h82B4	16'hFF31
14	16'hFAF9	16'h8079	16'h7A80	16'hFFC2	(16'hFF31 $\oplus$ 16'hFFC2) $>>$ b = 16'h8079	16'hFAF9
15	16'h0026	16'h366D	16'h364B	16'h9623	(16'hFAF9 $\oplus$ 16'h9623) $>>$ b = 16'h366D	16'h0026
16	16'h02F2	16'h003A	16'h02C8	16'h0052	(16'h0026 $\oplus$ 16'h0052) $>>$ b = 16'h003A	16'h02F2
17	16'hF786	16'h7C04	16'h8B82	16'hFAFA	(16'h02F2 $\oplus$ 16'hFAFA) $>>$ b = 16'h7C04	16'hF786
18	16'hFF23	16'h0490	16'hFBB3	16'hFEA6	(16'hF786 $\oplus$ 16'hFEA6) $>>$ b = 16'h0490	16'hFF23

G. Overheads

1) Area Overhead: The hardware added by the *SyncLock* are two XOR-based modules in the preamble and frame generation blocks and one nonlinear module involving another XOR operation and a bitwise rotation in the frame generation block. To compute the area overhead of *SyncLock* we used as baseline non-locked implementation an open-source IEEE 802.11 compatible SDR VHDL modem [38]. The project is called *bladeRF-wiphy* as it implements the IEEE 802.11 PHY on the Cyclone V Field-Programmable Gate Array (FPGA) integrated on the bladeRF board [13]. More details about the bladeRF board will be given in Section IV. Starting from the non-locked implementation, we added the *SyncLock* locking mechanism into the PHY of the modem and we re-synthesized the project using Quartus II 16.0 from Intel to find the resultant overhead. Considering a full key size of 512 bits, *SyncLock* results in 1.22% area overhead for the baseband DSP section, which when projected to the entire RF transceiver is even smaller as the area is dominated by the AFE.

The overhead for the key management scheme, i.e., based on a TPM, is common to all locking schemes. Besides, the key can be shared across different blocks in a SoC. Thus, the overhead of the key management scheme is taken as fixed for any locking mechanism and is not considered.

2) Power overhead: Embedding *SyncLock* in the *bladeRF-wiphy* PHY implementation as above resulted in no noticeable power overhead.

3) Performance penalty: The *SyncLock* mechanism has no impact on the performance of the RF transceiver. The delay of $f(\cdot)$ is accommodated by just enabling the XOR stream ciphers with the same delay. Thus, the STS generation is not delayed because of $f(\cdot)$. However, the two XOR stream ciphers introduce a delay of two clock cycles in the STS generation. This results in no timing violation because the preamble, consisting of STS and LTS, and the payload are generated in parallel to compose the data frame, while the payload part is much longer than the preamble part. In other words, the preamble generation, despite being delayed, still finishes before the payload is generated. The non-instrusiveness of the *SyncLock* mechanism is confirmed with hardware measurements in Section V.

4) Design flow: The AFE is left intact, thus there is no change in the analog IC design flow. This is an important attribute of *SyncLock* since analog IC designers are often reluctant to make any alternations in the circuit once it is finalized since this would typically add parasitics that would likely degrade performance. A lock mechanism inside the analog section inevitably would have to be co-designed with the circuit, possibly increasing design iterations and failing to meet the intent specifications. In contrast, *SyncLock* is a plug-in module added to the digital section of the RF transceiver once the design is completed without requiring any change in the design flow.

H. Practicality

Since a synchronization process is present and necessary in any wireless communication protocol, *SyncLock* is applicable to any of them. For instance, Wireless Local Area Network (WLAN) IEEE 802.11 (i.e., Wi-Fi), Wireless Personal Area Network (WPAN) IEEE 802.15.1 (i.e., Bluetooth), Low-Rate Wireless Personal Area Network (LRWPAN) IEEE 802.15.4 (i.e., Zigbee), and any other standard using correlation-based synchronization algorithms, are natural candidates. Furthermore, since *SyncLock* only acts on the preamble generation, it is independent of the modulation scheme that is applied on the payload and, thereby, it is generally applicable for any modulation scheme. Finally, since *SyncLock* modifies only the DSP, it is independent of the AFE of the RF transceiver architecture. Therefore, it can be applied to conventional RF transceiver architectures, such as Zero Intermediate Frequency (Zero-IF) and Low Intermediate Frequency (Low-IF), as well as to highly-digitized RF transceiver architectures.

I. First SyncLock version [14]

The first *SyncLock* implementation in [14] is a special case of the new *SyncLock* implementation proposed in this paper. In particular the first *SyncLock* implementation does not include the nonlinear module and feedback in the part of the mechanism that is embedded inside the frame generation block. This can be expressed as

$$f(STS_{out}, key_{h-c}) = key_{h-c} \tag{13}$$

with the system equation being simplified from Eq. (3) to

$$STS_{out} = STS_{nom} \oplus (key \oplus key_{h-c}) \tag{14}$$

As will be explained in detail in Section VII-B5, the motivation for the new *SyncLock* implementation is that the first *SyncLock* implementation in [14] is vulnerable to the KPA. This new *SyncLock* implementation effectively thwarts the KPA.

Another difference is that the first *SyncLock* implementation in [14] does not provide a full effective key size. The reason is that a single bit flip in the input secret key results in a single bit flip in STS_{out}. To quantify the fraction of incorrect keys that are still capable of enabling synchronization we performed a HD test. In particular, we generated incorrect keys with increasing HD from the correct key. For a full size key of 512 bits, there are 512 incorrect keys with HD=1 and $\binom{512}{k}$ keys with HD=k. Since $\binom{512}{k}$ is very high for $k > 1$, e.g., $\binom{512}{2}$=130816, for each $k > 1$ we tested a randomly generated set of 10^3 keys. We increased k until for all tested 10^3 keys synchronization failed. The results are shown in Table V. For HD=1, only 192 incorrect keys, or 37% of the incorrect keys, did not allow the synchronization process, thus reducing the number of effective key-bits to 192. This percentage increases with k and for $k = 14$ all incorrect keys resulted in no synchronization. The number of incorrect keys that enabled synchronization can be estimated as:

$$\sum_{i=1}^{n=13} \left(1 - \frac{\omega_k}{100}\right) \cdot \binom{512}{i} \approx 10^{22}$$

TABLE V
HD TEST FOR THE FIRST *SyncLock* IMPLEMENTATION IN [14].

HD	1	2	3	4	5	6	7
Percentage of failing keys	37.5%	63.5%	72.8%	84.6%	91.7%	94.4%	96.3%
HD	8	9	10	11	12	13	14
Percentage of failing keys	98.0%	98.3%	98.9%	99.3%	99.7%	99.9%	100%

TABLE VI
COMPARISON BETWEEN THE NEW AND FIRST *SyncLock* IMPLEMENTATIONS.

	First implementation [14]	New implementation
Effective number of key-bits	192	512
Area overhead (projected to DSP)	1.12%	1.22%
Performance penalty	no	no
Attacks in the analog domain	✓	✓
Brute-force and optimization attacks	✓	✓
Input-output query attacks	✓	✓
Removal attacks	✓	✓
KPA through AFE baseband loopback	✓	✓
KPA through digital baseband loopback	✗	✓

✓: Resilient , ✗: Not Resilient

where ω_k is the percentage of failing keys for HD=k and $n = 13$ is the highest HD showing keys that enable synchronization. Thus, a negligible percentage $(10^{22}/2^{512}) \cdot 100 = 10^{-131}\%$ of incorrect keys were capable of enabling synchronization.

Table VI summarizes the comparison between the first and new *SyncLock* implementations. The description of different counter-attacks and the resilience to them will be described in detail in Section VII.

IV. Hardware Platform

SyncLock is demonstrated in hardware using a SDR bladeRF board from Nuand [13]. The board contains three main chips: (a) an RF transceiver; (b) an FPGA; and (c) a USB 3.0 peripheral controller. We implemented on the bladeRF board an IEEE 802.11 RF transceiver with a direct conversion AFE architecture for both the receiver and the transmitter. The bladeRF board has an AFE RF loopback mode as shown in Fig. 2, which allows us to perform Bit Error Rate (BER) measurements and symbol timing recovery, i.e., synchronization, using the same board. Note that this on-board loopback minimizes the impairments of the wireless communication channel, such as path loss, fading, and shadowing, and greatly simplifies the channel model. The measurements presented in Section V were obtained using this on-board loopback considering an Additive White Gaussian Noise (AWGN) channel model. The baseband DSP is designed in VHDL [38] and implemented on the FPGA of the board. The VHDL code of the preamble and frame generation blocks is modified to insert the *SyncLock* locking mechanism and is re-embedded into the same FPGA

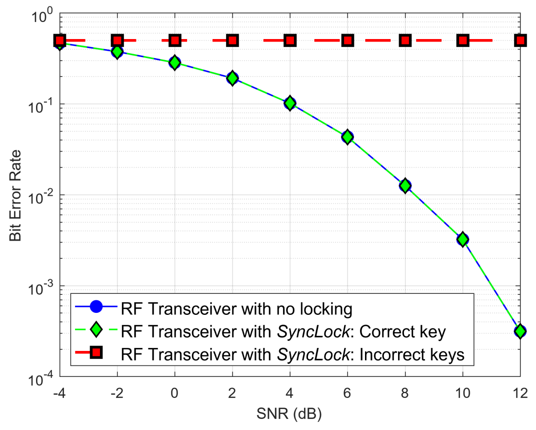


Fig. 6. Bit error rate with for RF transceiver with no locking and RF transceiver with *SyncLock* embedded using the correct and incorrect keys.

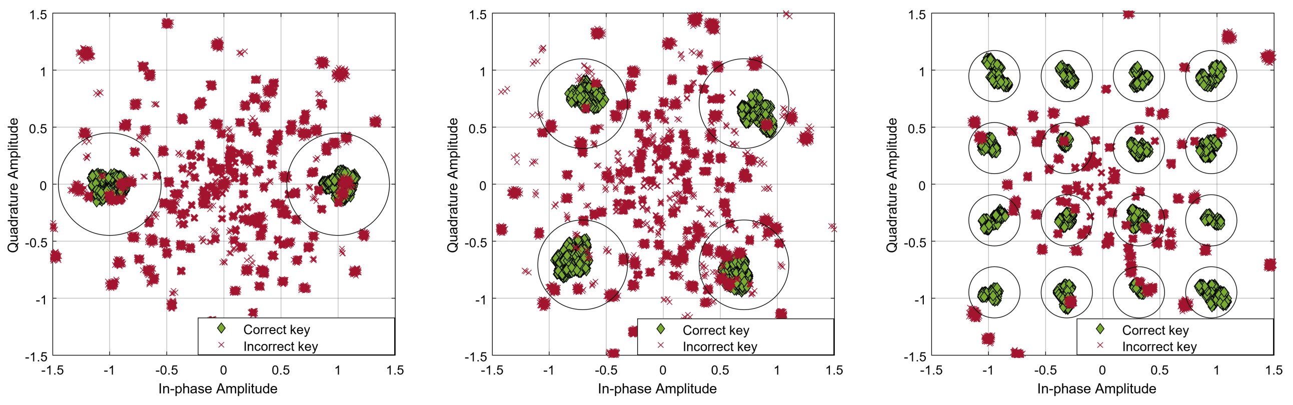
project. Detailed information on implementation overhead is presented in Section III-G1. As mentioned in Section III-H, *SyncLock* is independent of the modulation scheme. To show this, we repeat the demonstration by modulating the payload of the transmitted signal using Binary Phase-Shift Keying (BPSK), Quadrature Phase-Shift Keying (QPSK), and 16-Quadrature Amplitude Modulation (QAM), then encoding it into OFDM symbols. The frame generation block creates the PPDU frame format for an IEEE 802.11 transmission, as shown in Fig. 3. At the receiver side, the synchronization frame detection block searches for the start of the frame based on the Schmidl and Cox algorithm [37]. The received signal is processed and demodulated, and different performances of the RF transceiver are derived and visualized, such as BER and constellation diagram of the received payload.

V. Measured *SyncLock* efficiency

A. *BER performance for incorrect keys*

The hardware platform is used for assessing the impact of *SyncLock* on the nominal performance when using the correct key and for demonstrating the locking efficiency when using an incorrect key. As discussed in Section III-F, there is a single key enabling synchronization since any incorrect key, even those with HD=1 from the correct key, generate an arbitrary number of bit errors in the preamble of the outgoing data frame which impedes synchronization. For this reason, in the measurement results below we utilize a randomly selected incorrect key.

Fig. 6 shows the BER of an OFDM-BPSK transmission considering different Signal-to-Noise Ratio (SNR) values without *SyncLock*, with *SyncLock* when applying the correct key, and with *SyncLock* when applying a randomly selected incorrect key. A first observation is that when applying the correct key there is no BER penalty. The curves of BER without and with *SyncLock* are identical for all SNR values. This measurement proves that *SyncLock* is totally transparent when the correct key is used, thus there is zero performance penalty. This is expected since *SyncLock* leaves intact the sensitive AFE concentrating the lock mechanism inside the DSP. For each



(a) BPSK. (b) QPSK. (c) 16-QAM.

Fig. 7. Constellation diagram of the received payload with *SyncLock* embedded using the correct and an incorrect key.

preamble bit line *SyncLock* essentially introduces two spatially separated XOR gates in the path without causing any timing violation. A second observation is that with an incorrect key the system does not synchronize and erroneously demodulates the received signal. As a result, the BER is maximum and constant across all SNR values. It should be noted that for SNR values below -5dB the synchronization was not possible even for the device with no locking.

B. Constellation diagrams for incorrect keys

Fig. 7 shows the constellation diagrams of the received payload for three different modulation schemes, namely BPSK, QPSK, and 16-QAM, when applying the correct key and when applying a randomly selected incorrect key. The thin black circles show the reference constellation points for the modulation schemes. While the received signal lies inside the reference constellation for every modulation using the correct key, the non-synchronized signal is randomly distributed.

C. Locking efficiency for approximate keys

Finally, we tested the synchronization process for all the 512 incorrect keys with HD=1 from the correct key. All incorrect keys resulted in no synchronization with the smallest observed HD between STS_{out} and STS_{nom} being equal to 80. As discussed theoretically in Section III-F, any incorrect key will show the same behavior observed in Figs. 6 and 7, with the only difference being the randomness of distribution of payload data in Fig. 7 when different incorrect keys are loaded onto the chip.

VI. Related locking and obfuscation approaches

Herein, we describe related locking approaches and explain their differences compared to *SyncLock*.

1) Key-gates in logic locking: Traditional logic locking techniques insert key-gates into the design [7], [39]–[42]. A key-gate interrupts a digital line controlling its value with a key-bit. The first technique inserted key-gates randomly [7], while follow-up techniques targeted high output corruption for incorrect keys [39], resilience to sensitizing the key-bits to the output [40], reducing PPA overheads [41], or thwarting the

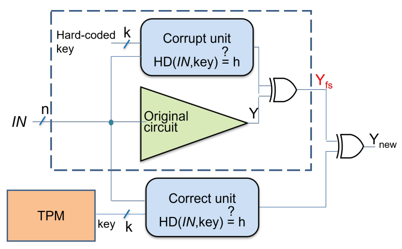


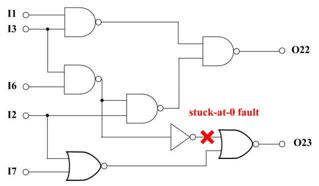
Fig. 8. SFLL-hd principle of operation.

ability of the attacker to learn the key-bit value from the key-gate type [42]. In all cases, key-gates are inserted randomly or algorithmically. In contrast, *SyncLock* inserts key-gates on fixed binary sequences, i.e., STS_{nom} inside the preamble generation block and STS_{faulty} inside the frame generation block.

2) Preamble obfuscation: The XOR-based cipher of the *SyncLock* mechanism inside the preamble generator that encrypts the nominal preamble STS_{nom} with a key was used in [43] as a PHY layer security to prevent man-in-the-middle attacks such as eavesdropping. In this different context, the preamble obfuscation is performed through unique keys that are independently generated at both the transmitter and the receiver based on channel characteristics known only to the pair. Using only the XOR-based cipher inside the preamble generator is not sufficient for anti-piracy since the attacker can identify and straightforwardly remove this XOR-based cipher by tracing the key-bits from the TPM. *SyncLock* hides a second XOR-based cipher inside the frame generation block to achieve the anti-piracy objective.

3) Corrupt-and-correct logic locking: SyncLock belongs to the family of corrupt-and-correct locking techniques. Two state-of-the-art corrupt-and-correct logic locking techniques are SFLL-hd [44] and SFLL-rem [19].

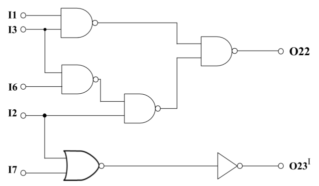
SFLL-hd, illustrated in Fig. 8, inserts a corrupt unit which compares the input to a hard-coded secret key. If the HD between the key and the input is h, then the output of the corrupt unit is 1, thus flipping the output of the circuit using an XOR gate. The inputs that satisfy this condition are called Protected Input Patterns (PIPs). The correct unit is identical



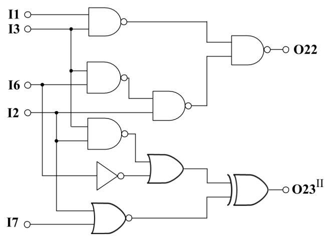
(a) Targeted logic cone from the c17 circuit having $n = 5$ inputs $\{I1, I2, I3, I6, I7\}$. The location of the stuck-at fault is shown.

Failing test patterns					(a)	(c)	(d)	(e)
I1	I2	I7	I3	I6	O23	O23^{I}	O23^{II}	O23^{III}
X	1	1	1	1	0	1	0	0
X	0	1	1	1	0	1	1	0
X	1	0	1	1	0	1	0	0

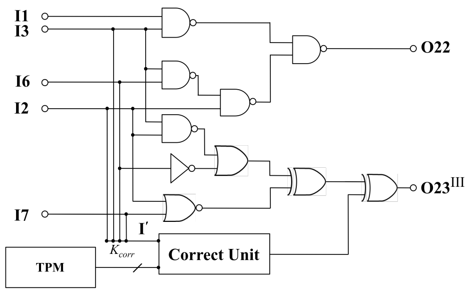
(b) Failing inputs patterns for the stuck-at 0 fault of Fig. 9a highlighting the selected PIPs.



(c) Simplified circuit after stuck-at-0 fault injection.



(d) Circuit after restoring logic in the circuit in Fig. 9c for all input patterns in Fig. 9b except the selected PIPs.



(e) Addition of correct unit for restoring functionality for the selected PIPs in Fig. 9b.

Fig. 9. SFLL-rem principle of operation explained with an example using the c17 circuit from the ISCAS benchmark suite [45].

to the corrupt unit, but in this case the key is sourced from the TPM. The correct unit flips the output a second time for the PIPs to restore correct functionality only when the correct secret key is loaded into the TPM. If an incorrect key is used, the functionality will be corrupted for all PIPs. The hypothesis is that the corrupt unit is immersed into the circuit after logic synthesis becoming indistinguishable to the attacker.

SFLL-rem, illustrated in Fig. 9 with an example reproduced from [19], corrupts the circuit functionality by injecting a stuck-at fault. Due to the fault, a number of input patterns fail resulting in incorrect output. Also due to the fault, some gates become redundant and the topology of the circuit is simplified by removing them. The circuit is then redesigned to correct functionality for all input patterns apart for a selected one that has k care bits and $n-k$ don't care bits, where n is the input size. In this way, 2^{n-k} PIPs are generated. Thereafter, a correction unit similar to SFLL-hd is used where the secret key is composed of the aforementioned k care bits and is sourced from the TPM to flip the output for all PIPs restoring functionality. Similar to SFLL-hd, if an incorrect key is used, the functionality will be corrupted for all PIPs.

SyncLock compared to SFLL-hd and SFLL-rem is conceptually different. As it will be discussed in detail in Section VII-B4, SFLL-hd and SFLL-rem were shown to be vulnerable to recently developed structural attacks [20], [46], [47], while *SyncLock* circumvents successfully these attacks.

VII. Threat Model and Security Analysis

A. *Threat model*

We consider the most demanding threat model for a defender. We assume that the attacker is in possession of the netlist and an oracle, i.e., a working chip with the correct key applied into the TPM. The goal of the attacker is either to identify a key that establishes synchronization or, alternatively, remove *SyncLock* while restoring the functionality. Next, we describe the known counter-attacks and discuss how *SyncLock* achieves resilience against all of them.

B. *Resilience to counter-attacks*

1) Attacks in the analog domain: Biasing locking is the only known locking approach working in the analog domain. Recently several attacks on biasing locking were demonstrated, some of them not requiring particular knowledge on analog design by the attacker [24]–[26]. These attacks assume the existence of an obfuscated analog component, i.e., the geometry of the mirroring transistor in a current mirror. They do not apply to *SyncLock* since *SyncLock* is not based on analog component obfuscation.

2) Brute-force and optimization attacks: The attacker searches in the key space either randomly in a brute-force manner or more efficiently by employing an optimization algorithm hoping to find a key that enables synchronization. The search is performed by simulating the design at netlist-level where the TPM is circumvented and the key inputs are accessed directly. At each iteration, instead of evaluating synchronization, a faster evaluation criterion may be devised by involving the oracle. For example a simulated transient response can be compared to that of the oracle. Resilience against this attack is achieved since: (a) the key space size, i.e., 2^{512} for a full key size, is huge; (b) a single secret key enables synchronization, thus the optimization function behaves like a delta function on the secret key and an optimization algorithm will "zig-zag" endlessly; (c) a single simulation at netlist-level can be very time-consuming, thus the attacker in practice can perform a very limited number of trials.

3) Input-output query attacks: Attacks based on Boolean satisfiability (SAT) [48] belong to this category and were shown to be very powerful, breaking traditional logic locking approaches [7], [39]–[41] by recovering the key with little effort. The SAT attack computes Distinguishing Input Patterns (DIPs), defined as inputs which produce different output for at least two different keys, and prunes down multiple incorrect keys iteratively using DIPs and querying the oracle. SFLL-hd and SFLL-rem were specifically proposed to push the limits of the SAT attack by eliminating exactly one key per iteration, thus making it equivalent to a brute-force attack in terms of attack time. As the SAT attack makes use of the scan chain, another recently proposed solution to thwart the SAT attack is to withdraw the secret key upon detection of access to the scan chain [42]. The SAT attack does not apply to *SyncLock* since the inputs to the preamble generation block, i.e., the DATA_i values and SEL, are fixed and hard-coded, thus no DIPs can be generated.

4) Structural attacks: Structural attacks, a.k.a. removal attacks, aim at identifying and removing the locking mechanism. The attacker can trace the key-bits from the TPM to straightforwardly identify and remove the first XOR-based stream cipher in the preamble generation block. In this case, the design will be left with a hard-coded error introduced by the second XOR-based stream cipher inside the frame generation block. Thus, the attacker will need to identify this second corrupt unit too to complete the removal attack. However, after logic synthesis this small circuit is immersed in the original design and the two become inseparable. The attacker has at hand a non-annotated netlist, thus identifying this small circuit is puzzling.

The fact that the corrupt unit is non-identifiable is the hypothesis of SFLL-hd too. For SFLL-hd, however, two specific attacks were developed recently that succeed in identifying the corrupt unit [46], [47]. They perform a structural analysis of the locked netlist to identify PIPs by leveraging the properties of the HD-based corrupt and correct units, and also exploiting the fact that in SFLL-hd the input feeds the corrupt unit. These attacks, apart from being specific to SFLL-hd, are not generalizable for *SyncLock* for two reasons. First, in *SyncLock* corruption is not a function of the STS_{nom} input which is fixed, i.e., *SyncLock* does not generate PIPs or stated differently all inputs are PIPs. Second, the corrupt unit hidden inside the frame generation block is spatially separated from the correct unit inside the input preamble generation block, thus the corrupt unit cannot be traced from the input.

In [20], a structural attack is proposed that defeats both SFLL-hd and SFLL-rem. It works differently by analyzing the Boolean truth table of the corrupted circuit to extract the PIPs. The SFLL-hd and SFLL-rem techniques essentially construct the corrupted circuit by adding (removing) selected minterm(s) to (from) the original circuit to create the PIPs. Then, the logic synthesis tool synthesizes the resulting corrupted circuit. The attack in [20] aims at recovering the PIPs. It demonstrates how the optimization performed to minimize the PPA overhead executed by the Electronic Design Automation (EDA) tools may expose the PIPs. This attack is not applicable to *SyncLock* either since *SyncLock* does not employ PIPs to secure the circuit, i.e., it does not add or remove any minterms, and no trace is left in the Boolean truth table. In addition, the correctness of the extracted PIPs is verified by querying the oracle with the PIPs. As explained in Section VII-B3, in the case of *SyncLock*, an attacker cannot query the oracle since the inputs to the preamble generation block, i.e., the DATA_i values and SEL, are fixed and hard-coded.

In short, it is realistic to assume that the corrupt unit inside the frame generation block cannot be distinguished within a "sea" of non-annotated digital gates. However, as we observed with other corrupt-and-correct logic locking techniques [19], [44], it leaves a backdoor that may be exploited to develop an attack based on structural analysis. Although such an attack is not known at this point for *SyncLock*, the possibility cannot be ruled out.

5) KPA: It applies to stream ciphers with symmetric encryption, i.e., when the encryption and decryption processes are performed using the same encryption key. In our context, the plaintext is STS_{nom} and is known to the attacker since it is published in the IEEE 802.11 standard [36].

Let us consider the first *SyncLock* implementation in [14]. The attacker applies a trial key, denoted by key_{trial}. From Eq. (14), using the associative and self-inverse properties of the XOR function, we obtain

$$key_{h-c} = key_{trial} \oplus STS_{nom} \oplus STS_{out} \qquad (15)$$

Knowing STS_{nom} and selecting any key_{trial}, the attacker can successfully recover key_{h-c} and, thereby, the secret key since $key = key_{h-c}$, provided that STS_{out} is measured accurately. In the oracle chip, the attacker cannot re-write the TPM to apply a trial key, thus STS_{out} has to be extracted by simulation. The attacker can simulate a transmission and try to extract STS_{out} from the transmitted frame. A loopback connection to the receiver can be used to analyse the transmitted signal. There are three different loopback modes, as shown in Fig. 2, namely (a) the AFE RF loopback that connects the output of the transmitter's PA to the input of the receiver's LNA, (b) the AFE baseband loopback right before the RF mixers, and (c) the digital baseband loopback between the DSP output and subsequent AFE data converters. In all three scenarios, the attacker should be able to manually locate the received STS_{out} bits. However, with loopback modes (a) and (b), some bits of STS_{out} will be corrupted due to analog impairments, quantization noise, and nonlinearities introduced throughout the signal processing at transistor-level. To demonstrate this, we implemented this attack using loopback mode (b) on our hardware platform, which will be described in Section IV. Fig. 10 shows the amplitude values of the first 32 samples of the measured transmitted and received STS_{out} for a given key_{trial}. As it can be seen, for every sample the amplitude values differ between the two STS_{out} signals. To quantify the number of bit errors, we translated the floating-point values into binary fixed-point values and obtained a HD of 1902 bits out of 5120 bits between the two signals. Thus, the transmitted STS_{out} will be extracted with errors and the computed key_{h-c} from Eq. (15) will be incorrect.

In contrast, if the attacker can locate the boundary between the DSP and AFE to implement loopback mode (c), then STS_{out} can be extracted accurately and KPA is completed successfully. This is a security breach of the first *SyncLock* implementation in [14] that is addressed with the second *SyncLock* implementation in this paper. As shown in the comparison Table VI, this is the main differentiation between the two implementations and the motivation for complexifying *SyncLock* giving rise to the second implementation.

In particular, introducing the nonlinear feedback inside the frame generation block helps thwarting the KPA attack even when STS_{out} is correctly extracted using a purely digital baseband loopback (c). The reason is that the identity from Eq. (3) now becomes

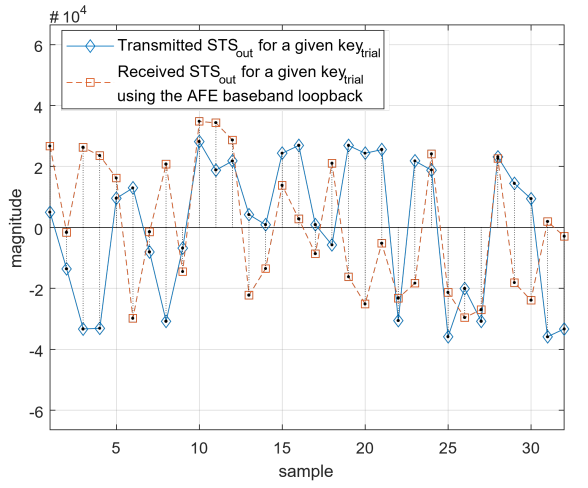


Fig. 10. Amplitude values of the first 32 samples of the transmitted and received STS_{out} for a given key_{trial} using the AFE baseband loopback.

$$STS_{out} = (STS_{nom} \oplus key_{trial}) \oplus f(STS_{out}, key_{h-c}) \quad (16)$$

making it impossible to de-embed key_{h-c} since both the nonlinear function f and key_{h-c} are unknown to the attacker. In the implementation presented herein, f is an XOR followed by a circular shift operation. However, as mentioned in Section III-D, any shift value can be used and any other function performing bitwise operations can be used instead. Thus, we presented one out of the countless implementations without endangering the security of *SyncLock*.

VIII. Conclusion

We presented *SyncLock*, an anti-piracy design technique for RF transceivers. *SyncLock* makes the synchronization between the transmitter and receiver key-dependent, while there is a single valid key that can have up to 512 effective key-bits. The *SyncLock* mechanism is hidden inside the DSP resulting in an overhead of around 1.22% of the DSP, which is negligible when projected to the entire RF transceiver since the area is dominated by the AFE. *SyncLock* is non-intrusive to the RF transceiver operation when applying the correct key incurring no performance penalty. No changes in the AFE design or the analog design flow are needed. The *SyncLock* mechanism is a simple plug-in to the DSP. *SyncLock* is a generic approach applicable to any RF transceiver architecture, communication protocol, and modulation scheme. Finally, it is shown to be resilient against any known counter-attack aiming at finding the secret key or removing the lock mechanism.

References

[1] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.

[2] M. Holler *et al.*, "High-resolution non-destructive three-dimensional imaging of integrated circuits," *Nature*, vol. 543, pp. 402–406, Mar. 2017.

[3] A. Chakraborty *et al.*, "Keynote: A disquisition on logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 1952–1972, Oct. 2020.

[4] A. Sanabria-Borbón, N. G. Jayasankaran, S. Lee, E. Sánchez-Sinencio, J. Hu, and J. Rajendran, "Schmitt trigger-based key provisioning for locking analog/RF integrated circuits," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020.

[5] J. Wang, C. Shi, A. Sanabria-Borbon, E. Sánchez-Sinencio, and J. Hu, "Thwarting analog IC piracy via combinational locking," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2017.

[6] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proc. IEEE*, vol. 102, no. 8, pp. 1126–1141, Aug. 2014.

[7] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, Oct. 2010.

[8] H.-C. Park *et al.*, "4.1 A 39GHz-band CMOS 16-channel phased-array transceiver IC with a companion dual-stream IF transceiver IC for 5G NR base-station applications," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 76–78.

[9] J. Lee *et al.*, "30.2 NB-IoT and GNSS all-in-one system-on-chip integrating RF transceiver, 23dBm CMOS power amplifier, power management unit and clock management system for low-cost solution," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 462–464.

[10] K. Shibata *et al.*, "A 22nm 0.84mm2 BLE transceiver with self IQ-phase correction achieving 39dB image rejection and on-chip antenna impedance tuning," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2022, vol. 65, pp. 398–400.

[11] R. Chen *et al.*, "A 6.5-to-10GHz IEEE 802.15.4/4z-compliant 1T3R UWB transceiver," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2022, vol. 65, pp. 396–398.

[12] E. Bechthum *et al.*, "30.6 A low-power BLE transceiver with support for phase-based ranging, featuring 5μs PLL locking time and 5.3ms ranging time, enabled by staircase-chirp PLL with sticky-lock channel-switching," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 470–472.

[13] Nuand, "SDR bladeRF 2.0 micro xA9," https://bit.ly/3z2QV1N, Online.

[14] A. R. Díaz Rizo, H. Aboushady, and H.-G. Stratigopoulos, "SyncLock: RF transceiver security using synchronization locking," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 1153–1156.

[15] N. G. Jayasankaran, A. S. Borbon, E. Sanchez-Sinencio, J. Hu, and J. Rajendran, "Towards provably-secure analog and mixed-signal locking against overproduction," in *Proc. 18th Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018.

[16] J. Leonhard *et al.*, "MixLock: Securing mixed-signal circuits via logic locking," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, p. 84–89.

[17] J. Leonhard *et al.*, "Digitally-assisted mixed-signal circuit security," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2021, early access.

[18] A. R. Díaz Rizo, J. Leonhard, H. Aboushady, and H. Stratigopoulos, "RF transceiver security against piracy attacks," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, 2022, early access.

[19] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly stripping functionality for logic locking: A fault-based perspective," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4439–4452, Jan. 2020.

[20] Z. Han, M. Yasin, and J. Rajendran, "Does logic locking work with EDA tools?," in *Proc. 30th USENIX Security Symposium*, Aug. 2021.

[21] V. Rao and I. Savidis, "Performance and security analysis of parameter-obfuscated analog circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 12, pp. 2013–2026, Dec. 2021.

[22] G. Volanis, Y. Lu, S. Govinda, R. Nimmalapudi, A. Antonopoulos, A. Marshall, and Y. Makris, "Analog performance locking through neural network-based biasing," in *Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 2019.

[23] D. H. K. Hoe, J. Rajendran, and R. Karri, "Towards secure analog designs: A secure sense amplifier using memristors," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2014, pp. 516–521.

[24] N. G. Jayasankaran, A. Sanabria-Borbón, A. Abuellil, E. Sánchez-Sinencio, J. Hu, and J. Rajendran, "Breaking analog locking techniques," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 10, pp. 2157–2170, Oct. 2020.

[25] R. Y. Acharya, S. Chowdhury, F. Ganji, and D. Forte, "Attack of the genes: Finding keys and parameters of locked analog ICs using genetic algorithm," in *Proc. IEEE Int. Symp.Hardw. Oriented Secur. Trust (HOST)*, Dec. 2020, pp. 284–294.

[26] J. Leonhard, M. Elshamy, M.-M. Louërat, and H.-G. Stratigopoulos, "Breaking analog biasing locking techniques via re-synthesis," in *Proc. 26th Asia South Pacific Design Automat. Conf.*, Jan. 2021, p. 555–560.

[27] M. Elshamy, A. Sayed, M.-M. Louërat, A. Rhouni, H. Aboushady, and H.-G. Stratigopoulos, "Securing programmable analog ICs against piracy," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 61–66.

[28] M. Elshamy, A. Sayed, M.-M. Louërat, H. Aboushady, and H.-G. Stratigopoulos, "Locking by untuning: A lock-less approach for analog and mixed-signal IC security," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 12, pp. 2130–2142, Dec. 2021.

[29] M. Tlili, A. Sayed, D. Mahmoud, M.-M. Louërat, H. Aboushady, and H.-G. Stratigopoulos, "Anti-piracy of analog and mixed-signal circuits in FD-SOI," in *Proc. 27th Asia South–Pac. Design Autom. Conf. (ASP-DAC)*, Jan. 2022, pp. 423–428.

[30] S. G. Rao Nimmalapudi, G. Volanis, Y. Lu, A. Antonopoulos, A. Marshall, and Y. Makris, "Range-controlled floating-gate transistors: A unified solution for unlocking and calibrating analog ICs," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar 2020.

[31] T. D. Perez and S. Pagliarini, "A survey on split manufacturing: Attacks, defenses, and challenges," *IEEE Access*, vol. 8, pp. 184013–184035, Oct. 2020.

[32] A. Vijayakumar, V. C. Patil, D. E. Holcomb, C. Paar, and S. Kundu, "Physical design obfuscation of hardware: A comprehensive investigation of device and logic-level techniques," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 1, pp. 64 – 77, Jan. 2017.

[33] Y. Bi, J. S. Yuan, and Y. Jin, "Beyond the interconnections: split manufacturing in RF designs," *Electronics*, vol. 4, no. 3, pp. 541–564, Aug. 2015.

[34] A. Ash-Saki and S. Ghosh, "How multi-threshold designs can protect analog IPs," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Oct. 2018, pp. 464–471.

[35] J. Leonhard, A. Sayed, M.-M. Louërat, H. Aboushady, and H.-G. Stratigopoulos, "Analog and mixed-signal IC security via sizing camouflaging," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 5, pp. 822–835, Jul. 2021.

[36] IEEE, "IEEE standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, 2016.

[37] T. M. Schmidl and D. C. Cox, "Robust frequency and timing synchronization for OFDM," *IEEE Trans. Commun.*, vol. 45, no. 12, pp. 1613–1621, Dec. 1997.

[38] Nuand, "Open-source IEEE 802.11 compatible software defined radio VHDL modem (bladeRF-wiphy)," https://github.com/Nuand/bladeRF-wiphy/, Online.

[39] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 410–424, Feb. 2015.

[40] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 9, pp. 1411–1424, Sep. 2016.

[41] K. Juretus and I. Savidis, "Reduced overhead gate level logic encryption," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, May 2016, pp. 15–20.

[42] N. Limaye, E. Kalligeros, N. Karousos, I. G. Karybali, and O. Sinanoglu, "Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 9, pp. 1740–1753, Sep. 2021.

[43] J. Chacko *et al.*, "Physical gate based preamble obfuscation for securing wireless communication," in *Proc. Int. Conf. Comput. Netw. Commun. (ICNC)*, Jan. 2017, pp. 293–297.

[44] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM SIGSAC Conf. Comput. and Commun. Security*, Oct. 2017, pp. 1601–1618.

[45] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," *IEEE Des. Test*, vol. 16, no. 3, pp. 72–80, Jul. 1999.

[46] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 936–939.

[47] F. Yang, M. Tang, and O. Sinanoglu, "Stripped functionality logic locking with hamming distance-based restore unit (SFLL-hd) – unlocked," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 10, pp. 2778–2786, Oct. 2019.

[48] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Sym. Hardw. Oriented Secur. Trust (HOST)*, May 2015, pp. 137–143.

Alán Rodrigo Díaz-Rizo is a Ph.D. candidate at the Computer Science Laboratory (LIP6) of Sorbonne Université, Paris, France. His research interests include hardware security, cognitive radio, and radio signal processing. He received the B.Sc. in Electronics and Communication Engineering from Guadalajara University, Guadalajara, Mexico, in 2015, and the M.Sc. in Electrical Engineering from the Center for Research and Advanced Studies of the National Polytechnic Institute (Cinvestav), Mexico, in 2018.

Hassan Aboushady (Senior Member, IEEE) received the B.Sc. degree in Electrical Engineering from Cairo University, Egypt, in 1993, the M.Sc. and Ph.D. degrees in Electrical Engineering and Computer Science from Sorbonne University, Paris, France, in 1996 and 2002 respectively. Dr. Aboushady is currently an Associate Professor at Sorbonne University. His research interests include Sigma-Delta modulation, Analog/RF circuit design, Analog-to-Digital and Digital-to-Analog conversion, as well as Security in analog and mixed-signal circuits. He is the author and co-author of more than 70 publications in these areas. He is the recipient of the 2004 best paper award in the IEEE Design Automation and Test in Europe Conference, as well as the recipient and the co-recipient of the 2nd and the 3rd best student paper awards of the IEEE Midwest Symposium on Circuits and Systems in 2000 and 2003 respectively. Dr. Aboushady is an IEEE-CAS distinguished lecturer and a member of the IEEE Circuits and Systems for Communications Committee (CASCOM). He also served as an Associate Editor of the IEEE Transactions on Circuits and Systems II: Express Briefs.

Haralampos-G. Stratigopoulos (Member, IEEE) received the Diploma in electrical and computer engineering from the National Technical University of Athens, Athens, Greece, in 2001 and the Ph.D. in electrical engineering from Yale University, New Haven, USA, in 2006. He is a Research Director of the French National Center for Scientific Research (CNRS) at the LIP6 laboratory of Sorbonne Université, Paris, France. Before he was Researcher with the CNRS at the TIMA Laboratory, Université Grenoble Alpes, Grenoble, France. His main research interests are in the areas of hardware security, neuromorphic computing, and design-for-test for analog, mixed-signal, RF circuits and systems. He was the General Chair of the 2015 IEEE International Mixed-Signal Testing Workshop (IMSTW), the Program Chair of the 2017 IEEE European Test Symposium (ETS), and the General Chair of the 2021 and 2022 AI Hardware: Test, Reliability and Security (AI-TREATS) Workshop. He has served on the Technical Program Committees of Design, Automation, and Test in Europe Conference (DATE), Design Automation Conference (DAC), IEEE International Conference on Computer-Aided Design (ICCAD), IEEE European Test Symposium (ETS), IEEE International Test Conference (ITC), IEEE VLSI Test Symposium (VTS), and several others international conferences. He has served as an Associate Editor of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Circuits and Systems I: Regular Papers, IEEE Design & Test, and Springer Journal of Electronic Testing: Theory & Applications.