A COMPARATIVE ANALYSIS OF VLSI TRUSTED VIRTUAL SENSORS

M.C.MARTÍNEZ-RODRÍGUEZ,P.BROX,I.BATURONE

Abstract

This paper analyzes three cryptographic modules suitable for digital designs of trusted virtual sensors into integrated circuits, using 90-nm CMOS technology. One of them, based on the keyed-hash message authentication code (HMAC) standard employing a PHOTON-80/20/16 lightweight hash function, ensures integrity and authentication of the virtual measurement. The other two, based on CAESAR (the Competition for Authenticated Encryption: Security, Applicability, and Robustness) third-round candidates AEGIS-128 and ASCON-128, ensure also confidentiality. The cryptographic key required is not stored in the sensor but recovered in a configuration operation mode from non-sensitive data stored in the non-volatile memory of the sensor and from the start-up values of the sensor SRAM acting as a Physical Unclonable Function (PUF), thus ensuring that the sensor is not counterfeit. The start-up values of the SRAM are also employed in the configuration operation mode to generate the seed of the nonces that make sensor outputs different and, hence, resistant to replay attacks. The configuration operation mode is slower if using CAESAR candidates because the cryptographic key and nonce have 128 bits instead of the 60 bits of the key and 32 bits of the nonce in HMAC. Configuration takes 416.8 microseconds working at 50 MHz using HMAC and 426.2 microseconds using CAESAR candidates. In the other side, the trusted sensing mode is much faster with CAESAR candidates with similar power consumption. Trusted sensing takes 212.62 microseconds at 50 MHz using HMAC, 0.72 microseconds using ASCON, and 0.42 microseconds using AEGIS. AEGIS allows the fastest trusted measurements at the cost of more silicon area, 4.4 times more area than HMAC and 5.4 times more than ASCON. ASCON allows fast measurements with the smallest area occupation. The module implementing ASCON occupies 0.026 mm^2 in a 90-nm CMOS technology.

1 INTRODUCTION

Virtual sensors estimate the value of a variable that is very difficult or costly to measure physically from others that can be measured easily with low-cost commercial sensors. The estimation usually makes use of models obtained with neural networks, fuzzy logic, or other types of approximation techniques such as piecewise affine (PWA) approaches. Among the latter, simplicial PWA (PWAS) functions were first employed for virtual sensors in [23] and, later, hyper-rectangular (PWAR) functions were used for their further simplicity in [15]. PWAR-based models partition the domain of the input variables (those easily measured) into hyper-rectangles and estimate the output (the virtual measurement) as an affine function whose parameters

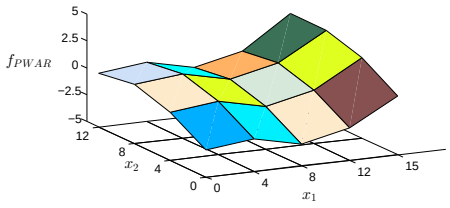

Figure 1: Bi-dimensional example of a PWAR function and its hyper-rectangular partition

depend on the hyper-rectangle which the inputs belong to [8]. Figure 1 shows an example of a PWAR model that partitions a bi-dimensional input domain into 12 hyper-rectangles.

Since the early 1980s, the use of virtual sensors has increased continuously in a wide number of applications [13, 17]. Many virtual sensors are implemented as software. However, taking into account the big size of the sensor market size, CMOS Integrated Circuit (IC) solutions are very adequate to achieve a minimum unitary cost as well as high performance in terms of area, power and response time. Another advantage of CMOS ICs that implement PWAR virtual sensors is that their SRAMs (Static Random Access Memories) can be programmed to be employed in different applications. An algorithm was presented in [15] to find the best values of the programmable parameters given a set of empirical or simulated input-output data.

Nowadays, virtual sensors are usually part of ubiquitous and distributed networks such as critical infrastructures. Hence, sensor security is becoming very important [25]. The receiver of the sensing data must trust in the data originated from the sensor. The way is to confirm that data have not been tampered with, altered or changed. Not only data integrity but also sensor integrity must be ensured. The reliability of the data should be ensured by authentication of their origin. Message authentication codes (MACs) are the usual constructions employed to ensure data integrity [21]. The MAC construction that has become more popular over the last decade is the HMAC. HMAC, standardized in [26], is an approach to implement MAC using a two-pass hash-based MAC that avoids the weakness of previous constructions. The problem with many existent HMACs is that their primary targets are high transmission rates of data with long key spaces, but do not pay so much attention to area or power consumption. This makes them not suitable for resource constrained scenarios like sensors, which have limited computational and memory capabilities. Lightweight cryptographic primitives should be employed. Specific lightweight hash functions based on sponge constructions have been proposed, as is the case of SPONGENT [7], Quark [5], and Photon families [12, 9]. The latter one has been selected for the trusted virtual sensor analyzed herein [14].

There is also a growing interest in the use of lightweight cryptographic protocols for sensors so as to ensure that data transmitted are confidential, which is required by applications using sensitive information [16, 22]. In these cases, not only message authentication but also message encryption is required. CAESAR competition is evaluating different authenticated encryption schemes that are fast in software and efficient in hardware [3]. The idea is to combine encryption and authentication into a single algorithm, called Authenticated Encryption, so as to obtain smaller area and power consumption compared to two separate algorithms. Among the CAESAR

candidates, AEGIS [27, 15] and ASCON [1, 10] have been selected for the trusted virtual sensor analyzed herein. AEGIS is based on the AES encryption round function while ASCON is based on sponge constructions like PHOTON.

The usual way to ensure the integrity of a device (in this case a sensor) is to check that the sensor knows a cryptographic key. However if the key is stored in the sensor memory, it can be attacked [24] and a counterfeit sensor can be fabricated. The problem of counterfeit chips is so significant that the Semiconductor Industry Association (SIA) maintains an anti-counterfeiting task force. A solution is the use of Physical Unclonable Functions inside the sensor hardware, which ensures that the sensor has a unique inherent identity given, in general, by the variability of its manufacturing process [11]. Among the wide number of PUFs proposed in the literature, SRAM-based PUFs are selected in this paper since the CMOS sensor requires SRAMs for its virtual sensing functionality. The start-up values of the SRAM cells are employed to generate the random seeds required by the cryptographic primitives as well as to recover the cryptographic key [6]. The key is not stored anywhere but recovered whenever needed by using a Helper Data Algorithm that employs non-sensitive data (known as Helper Data) and the response of the PUF (in this case the start-up values of SRAM cells) [11]. An impostor sensor is not able to provide a similar PUF response and, hence, is unable to recover the key while the genuine sensor is able to prove its authenticity against aging and variations of the operation conditions, namely, temperature and power supply voltage [15].

This paper analyzes three cryptographic modules suitable for trusted virtual sensors into CMOS integrated circuits. One of them, based on a lightweight HMAC construction, ensures integrity and authentication. The other two, based on CAESAR third-round candidates, ensure also confidentiality. To the best of authors' knowledge, no other work has addressed the comparison of such trusted virtual sensors into ICs. The paper is organized as follows. Firstly, the three cryptographic modules are briefly described in Section 2. Their integration in a trusted virtual sensor is presented in Section 3, describing the architecture of the resulting sensor as well as its behavioral modes related to configuration and virtual sensing. Then, Section 4 compares the three modules implemented in a 90-nm CMOS technology in terms of area and power consumption as well as sensor memory requirements and timing performance. Finally, conclusions are given in Section 5.

2 DESCRIPTION OF CRYPTOGRAPHIC MODULES

A virtual sensor provides one output variable that is virtually measured. The inclusion of cryptographic primitives allows ensuring the integrity, authenticity, and confidentiality of the virtual measurement achieving a trusted virtual sensor. Several options have been evaluated to address this goal.

In order to detect possible manipulations of measurements, a first approach includes a module to solve the problem of data integrity and authenticity based on MACs. The second approach to increase the level of security is the use of authenticated ciphers to ensure not only the integrity of the provided virtual measurement but also to encrypt it at the same time. Two authenticated encryption algorithms, which are third-round candidates of the CAESAR competition, have been considered. The three alternatives were implemented in a 90-nm CMOS technology, as part of a virtual sensor

integrated circuit, to compare their performance. Their features are briefly summarized in the following.

2.1 *Hash-based Message Authentication Code (HMAC) based on the Photon hash function*

HMAC is a specific type of MAC involving a cryptographic hash function (*hash*) and a secret cryptographic *key*. Simultaneously, it verifies both the data integrity and the authentication of a message.

The HMAC inputs are a cryptographic *key* and the message to be authenticated denoted as *text*. The HMAC provides an authentication code as follows:

$$\text{HMAC}(key, text) = hash\left[\left(K_o \oplus \text{opad}\right) || \ \ hash\left[\left(K_o \oplus \text{ipad}\right) \| \ text\right]\right] \quad (1)$$

where:

- *key* is the secret key
- *text* is the message to be authenticated
- *hash* is a cryptographic hash function
- K_o is another secret key, derived from *key*, to have a multiple of the hash input block size
- $\oplus$ denotes exclusive or (*XOR*)
- *ipad* and *opad* are, respectively, the inner and outer pads given by the standard
- $||$ denotes concatenation

If the virtual sensor only provides one output variable (y), the input message of HMAC will be the virtual measurement (y) and a number called *nonce*, which is attached to y:

$$text = nonce \ \ || \ \ y \quad (2)$$

The *nonce* is an arbitrary number that can only be used once. The use of *nonce* helps to improve the security since it ensures that old communications cannot be reused in replay attacks. This is specially important since the virtual measurement, y, can be the same for different messages.

Hash functions, which implement compression functions, can process an arbitrary-length message and produce a fixed-length output. This is achieved by segmenting the input into a series of blocks of equal size that are processed sequentially. The *hash* value of the input message is then defined as the output of the last iteration of the compression function. There are two general types: dedicated and block cipher-based *hash* functions [21]. Dedicated *hash* functions are algorithms that have been custom designed such as MD4, MD5, RIPEMD and SHA family. The alternative is to construct *hash* functions using block cipher chaining techniques using the Matyas-Meyer-Oseas construction [21].

Among dedicated *hash* functions, sponge constructions are very popular since the winner of the SHA-3 contest of the National Institute of Standards and Technology (NIST) uses them. The sponge construction has an internal state S of t bits, composed of the c-bit capacity and the r-bit bitrate (t=c+r). First, all the bits of the state are initialized to zero. The input message is

padded and cut into r-bit blocks. Then, it proceeds in two phases: the *absorbing phase* followed by the *squeezing phase*. In these phases the first r bits of the state and the remaining t-r bits or *capacity* c, are treated differently. During the absorbing phase, the message blocks (m_0, m_1, ..., m_a) are XORed with the first r bits of the state, interleaved by applying the permutation P. Once all message blocks have been handled by this absorbing phase, the sponge construction switches to the squeezing phase. The sponge construction successively outputs the final *hash* value (z_0, z_1, ..., z_b) by extracting r bits from the bitrate part of the internal state and then applying the permutation P on it. An extended version of the sponge framework allows a different number of r' bits extracted during each iteration of the squeezing process. The sponge construction is illustrated in Figure 2.

Specific lightweight hash functions based on sponge constructions have been proposed for hardware implementation into tiny devices, such as the PHOTON family [12]. The HMAC is implemented with a PHOTON-80/20/16. The input block size of this *hash* function is 20 bits. Hence, the message to authenticate must be divided into blocks of 20 bits. During absorbing steps, each 20-bit input block is XORed with 20 bits of the internal state of the hash function. The 80-bit hash output is built by concatenating five successive outputs of 16 bits obtaining after each squeezing phase. PHOTON-80/20/16 has an internal state of 100 bits (80 out of these 100 bits are never affected by the input blocks and never output the function; they correspond to the capacity of the function). After absorbing a block, a permutation operation is performed to the internal state, represented as a 5×5 matrix of 4-bit cells. The permutation operation is composed of four operations, AddConstant, Sub-Cell, ShiftRows and MixColumnsSerial, which are repeated 12 times, changing the values of the 4-bit cells. In the designed HMAC, these operations are implemented serially, following the proposal in [12], so as to obtain a module with low area and power consumption.

2.2 *AEGIS Authenticated Cipher*

There are two main approaches to authenticate and encrypt a message. One approach is to apply the encryption and authentication separately. The plaintext is encrypted with a block cipher or stream cipher, and then, a MAC algorithm is used to authenticate the ciphertext. Another approach is an

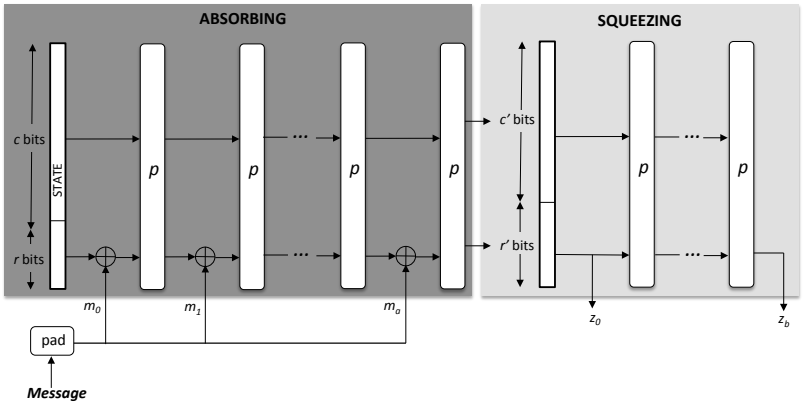

Figure 2: The sponge construction implemented by the *hash* function

authenticated encryption algorithm that shares the computation of both tasks authentication and encryption.

There are three approaches to design an integrated authentication encryption algorithm. The first approach is to use a block cipher in a special mode, for instance, CCM [18] and GCM [19] are operation modes for authenticated encryption according to NIST recommendation. The second approach is to use a stream cipher dividing keystream into two parts, one part for encryption and another part for authentication. An example of this approach is Grain-128a [4]. The third approach is to design dedicated authenticated encryption algorithms. AEGIS is a dedicated authenticated algorithm that belongs to the third type of approaches [27]. It is constructed from the AES encryption round function, providing the advantage of a computational cost about half that of AES. If the *nonce* is not reused (which should be the case for a true *nonce*), AEGIS provides a high security since the state and the *key* can only be recovered by exhaustive search.

AEGIS algorithm takes the cryptographic *key*, a *nonce*, and a plaintex, in this case the virtual measurement, y, and provides a ciphertext, C, and an authentication *tag*.

$$[C, tag] = \text{AEGIS}(key, nonce, y) \tag{3}$$

There are three variations of the algorithm. AEGIS-128 processes a 16-byte message block with five AES round functions in parallel. It consumes the least resources, but it is the slowest for large messages. AEGIS-128L processes a 32-byte message block with eight AES round functions in parallel. In terms of resource consumption, it is the biggest version, but it is also the fastest. Finally, AEGIS-256 processes a 32-byte message block with six AES round functions. Then, it offers an intermediate performance in terms of resource consumption and time response. Since the virtual measurement (y) is smaller or equal to 128-bits, only one 16-byte state has to be processed. Hence, AEGIS-128 is more adequate in this application, since it is as fast as AEGIS-128L, using less resources.

AEGIS-128 uses a 128-bit *key* and a 128-bit *nonce*. The algorithm is based on the function $StateUpdate(S_i, m_i)$ described in Algorithm 1 where the function $AESRound(A)$ is the AES encryption round function being A the state. The main operations in the AES round are SubBytes operation, ShiftRows and AES-MixColumns. The state update function of AEGIS is illustrated in Figure 3.

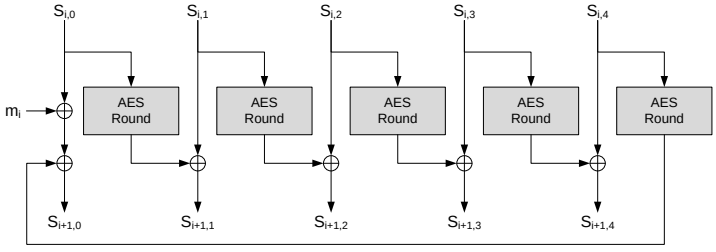

Figure 3: The state Update function of AEGIS-128

Algorithm 1 State Update function.

1: **function** StateUpdate(S_i, m_i)
2: $S_{i+1,0}$ =AESRound($S_{i,4}$) $\oplus$ $S_{i,0}$ $\oplus$ m_i
3: $S_{i+1,1}$ =AESRound($S_{i,0}$) $\oplus$ $S_{i,1}$
4: $S_{i+1,2}$ =AESRound($S_{i,1}$) $\oplus$ $S_{i,2}$
5: $S_{i+1,3}$ =AESRound($S_{i,2}$) $\oplus$ $S_{i,3}$
6: $S_{i+1,4}$ =AESRound($S_{i,3}$) $\oplus$ $S_{i,4}$
7: **return** S_{i+1}
8: **end function**

Algorithm 2 Pseudo-code of AEGIS algorithm.

Require: key, nonce, y
1: $S_{-10,0}$ = key $\oplus$ nonce
2: $S_{-10,1}$ = constant1
3: $S_{-10,2}$ = constant2
4: $S_{-10,3}$ = key $\oplus$ constant1
5: $S_{-10,4}$ = key $\oplus$ constant2
6: **for** $i = -10 : -1$ **do**
7: **if** i is odd **then**
8: m_i = key $\oplus$ nonce
9: **else**
10: m_i = key
11: **end if**
12: $S_{i+1} = StateUpdate(S_i, m_i)$
13: **end for**
14: $C_t = y \oplus S_{0,1} \oplus S_{0,4} \oplus (S_{0,2} \& S_{0,3})$
15: $S_1 = StateUpdate(S_0, y)$
16: $tmp = S_{1,3} \oplus 128$
17: **for** $i = 1 : 6$ **do**
18: $S_{i+1} = StateUpdate(S_i, tmp)$
19: **end for**
20: $tag = S_{7,0} \oplus S_{7,1} \oplus S_{7,2} \oplus S_{7,3} \oplus S_{7,3}$
21: **return** C_t, tag

The pseudo-code of the AEGIS-128 applied to the virtual measurement is described in Algorithm 2. The main steps of the algorithm are to initialize the state (line 2.1), process the *nonce* (line 2.6), process the measurement (y), and generate the ciphertext (line 2.14), and generate the authentication tag (line 2.20). *constant1* and *constant2* are two 128-bit constants fixed by the algorithm. The designed AEGIS followed the VHDL code provided in [2].

ASCON is a family of authenticated encryption designs [1]. The inputs for the authenticated encryption procedure are the plaintext, that is, the virtual measurement (y) in this case, a secret *key*, and a *nonce*. The outputs are the authenticated ciphertext C, and an authentication *tag*:

$$[C, tag] = \text{ASCON}(y, key, nonce) \tag{4}$$

The operation mode of ASCON is based on duplex sponge modes like MonkeyDuplex, but using a stronger keyed initialization and a keyed finalization function. During the initialization, the sponge state is obtained by concatenating an Initialization Vector (IV) , the *key*, and the *nonce*. The permutation functions (p^a, p^b) operate on the state with a rate of r bits and a capacity c. There is an initialization and finalization permutation (p^a) with a rounds, and intermediate permutations (p^b) with b rounds. Both permutations differ only in the number of rounds that are tunable security parameters. The permutations iteratively apply a Substitution-Permutation Network (SPN)-based round transformation. Plaintext is absorbing during intermediate rounds.

The recommended parameters for ASCON are 128 bits for the *key* and *nonce*, 64 bits for the rate r, 320 bits for the state, and then, a capacity of $c = 320 - r = 256$ bits. The selected numbers for the rounds are 12 for permutation p^a and 6 for permutation p^b.

Figure 4 illustrates the encryption operation in ASCON. The 320-bit initial state of ASCON is formed by an IV specified by the algorithm, the *key*, and the *nonce*. In the initialization, 12 rounds of the permutation are applied to the initial state, followed by an XOR with the *key*. After initialization is finished, a single bit constant is XORed with the internal state.

The padding process is applied to the plaintext (virtual measurement y) since ASCON has a message block size of r bits. The padding process appends a single 1 and the smallest number of 0s to the plaintext, obtaining a length of the padded plaintext that is a multiple of r bits and then, it is split into t blocks of r bits.

During the encryption, each block of the virtual measurement (y_1, y_2, ..., y_t) is XORed with the first r bits of the state, followed by the extraction of one ciphertext block C_i with ($i = 1, 2, ..., t$). For each block except the last one, the internal state is transformed by the permutation using 6 rounds. In the finalization, the *key* is XORed with the state and then, it is transformed by

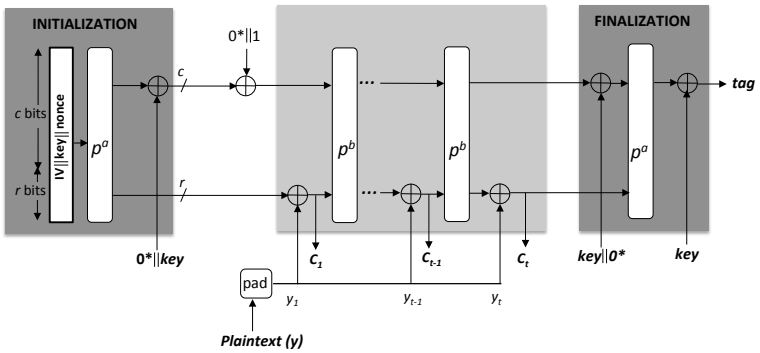

Figure 4: Encryption operation in ASCON

the permutation using 12 rounds. The *tag* consists of the last k bits of the state XORed with the *key*. ASCON returns the *tag* together the ciphertext (C_1, C_2,..., C_t). The designed ASCON followed the VHDL code provided in [2].

3 INTEGRATION ON A TRUSTED VIRTUAL SENSOR

The output of the virtual sensor depends on the cryptographic module used. If the HMAC is integrated, the sensor output is the virtual measurement, y, as well as an authentication code that ensures the integrity and the authenticity of the sensor data. Since the data can be the same for different measurements, a *nonce* is attached to the measurement in order to avoid replay attacks. Therefore, the output provided by the sensor is:

$$\text{sensor}_{\text{out\_MAC}} = \text{nonce} \parallel y \parallel \text{HMAC}\,(key, \text{nonce} \parallel y) \tag{5}$$

If an authenticated cipher is used, the output provided by the sensor is the encrypted value (C) that is attached to an authentication code that ensures the integrity, and authenticity of the sensor data:

$$\text{sensor}_{\text{out\_AEGIS\_ASCON}} = \text{nonce} \parallel C \parallel \text{tag} \tag{6}$$

The proposed sensor estimates the virtual measurement (y) from a PWA-based model. Both PWAR and PWAS forms are able to approximate any function and extract any black-box model. The PWAS form has been widely explored for virtual sensors [23, 20]. However, the PWAR form is selected herein since its implementation is simpler than PWAS implementation.

The integrity of the virtual sensor itself is ensured if the *key* employed by the cryptographic module is not stored but recovered whenever needed by using PUFs. The trusted sensor is able to recover the cryptographic key shared with the receiver of the sensing data, while any impostor is unable due to the uniqueness provided by the start-up values of the SRAM in the sensor, which is exploited as a PUF. Non-sensitive Helper Data, H, are stored to recover the key with a Helper Data Algorithm (HDA) based on an Error Correcting Code (ECC) [11]. Helper Data do not reveal anything about the cryptographic *key* because the start-up values of SRAM cells obfuscate it. Similarly, Helper Data do not reveal anything about the intrinsic nature of the sensor because the cryptographic key obfuscates it. A repetition ECC was selected in the design of the sensor.

The sensor designed has two main behavioral modes. The configuration mode is carried out whenever the sensor is powered up. It recovers the *key*, generates a seed for a *nonce*, and establishes the PWAR relation between the input data and the data to estimate. The trusted sensing mode is set once the configuration mode is finished. It generates a PWAR virtual measurement, authenticates it, and encrypts it in case of using AEGIS and ASCON.

The building blocks of the CMOS sensor designed are exposed in the following. It is composed of three main units: the PWAR, the security, and the control units. Figure 5 illustrates the block diagram of the sensor architecture, including the main signals and buses that interconnect the blocks. Note that the SRAM is shared by two units, thus saving resources since it is not used simultaneously by both units.

3.1 *PWAR Unit*

A generic PWA function with multiple inputs and one output is considered: $y(x) : D \subset \mathbb{R}^n \rightarrow \mathbb{R}$, that is represented as follows:

$$y(x) = \sum_{j=1}^{n} f_{ij} \cdot x_j + f_{i0} \; \forall \; x \in P_i \; (i = 1, \ldots, P) \tag{7}$$

where $f_i \in \mathbb{R}^{n+1}$, and $P_i \subset D$ are P non overlapped regions ($P_i \cap P_k = \emptyset \; \forall i \neq k$), called polytopes, which form a polyhedral partition of the domain D, so that $\bigcup_{i=1}^{p} P_i = D$. In the case of regular PWAR functions, the domain is partitioned into hyper-rectangular polytopes by dividing each j dimension of the domain into L_j intervals with the same amplitude, thus resulting $P = \prod_{j=1}^{n} L_j$ polytopes. Figure 1 shows a bi-dimensional example of a regular PWAR function with its domain partitioned into $12 = 4 \times 3$ rectangles.

The partition and affine functions of a PWAR function can be adjusted conveniently to provide a black-box model that approximates the relation between input variables and sensing variable to estimate. Therefore, the virtually measured output, y, is a PWA function of the inputs $x = \{x_1, .., x_n\}$.

The sensor can be configured for different PWAR functions. Therefore, it can be used to sense different variables. The configuration data are the number of inputs and the number of intervals in which each input is divided, that is, the hyper-rectangular partition is configurable. The sensor is also configurable in the affine functions by changing the value of the parameters $f_i = \{f_{i0}, f_{i1}, .., f_{in}\}$ associated to each hyper-rectangle, P_i. Prior to use the sensor, the sensor manufacturers or their authorized distribution channels program in a non-volatile memory (NVM) these configuration data and parameters.

In the configuration mode, the PWAR parameters are read from the NVM and stored in the SRAM. Each word of the SRAM contains a parameter, f_i, associated to an hyper-rectangle, P_i, of the input domain. The SRAM depth is the number of hyper-rectangles. In the trusted sensing mode, the PWAR unit receives the input data. The address generator block determines the address of the SRAM where the parameters are stored. For that purpose, it concatenates the p_k most significant bits (MSBs) of each input x_k. The MSBs determine which of the 2^{p_k} intervals the input belongs to, and, hence, determine the hyper-rectangle. Then, the arithmetic block computes in parallel the operation in (7) for the input, x, and the parameters, f_i.

3.2 *Security Unit*

The SRAM employed in the PWAR Unit is also employed in the Security Unit as a PUF. As discussed in [6], there are SRAM cells whose intrinsic conditions dominate over the external conditions. Hence, although the external condition change (such as temperature or power supply voltage), their start-up values are mostly the same. They will be named as ID cells. There are also SRAM cells whose external conditions dominate over the intrinsic conditions so that they are able to extract the noise of the external conditions as a source of entropy. They will be named as RND cells.

The NVM stores which cells of the SRAM are ID cells (an 'ID_mask' is in charge of such information) and which ones are RND_cells (an 'RND_mask' stores such information). These masks have to be stored in the NVM by the sensor manufacturers or their authorized distribution channels prior to use

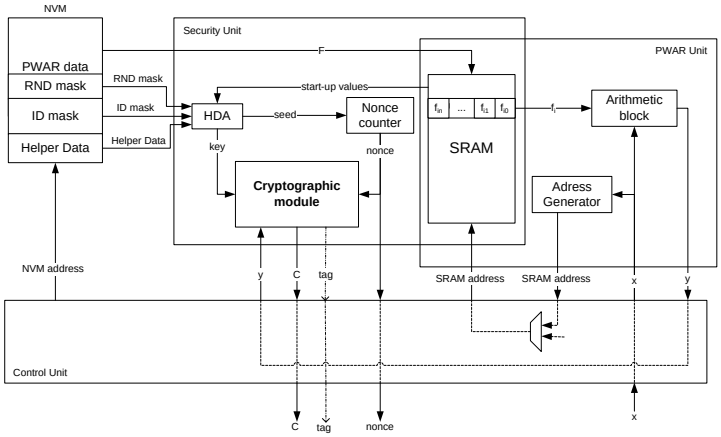

Figure 5: Architectural scheme of the proposed CMOS sensor

the sensor. These manufacturers or distributors follow a simple procedure to classify the SRAM cells: the SRAM is powered-up and down several times under different operating conditions. Each time, the start-up values are compared with the values at the previous time. If the start-up value never changes, the cell is classified as ID (it has not shown bit flipping). If the start-up value changes, the cell is classified as RND. The ID cells are used to recover the *key* while the RND cells are used to generate a seed for the nonces.

HDA is used in order to recover the *key*. The *key* is not stored in the NVM, and therefore, the attacks to obtain the *key* are forced to be done with the sensor powered. Instead of the *key*, Helper Data, H, are stored in the NVM prior to use the sensor. Helper Data do not reveal information about the cryptographic *key*, so that the NVM does not require any special security feature. The way to generate the Helper Data is as follows. Each bit of the cryptographic key $key = [k_1, .., k_a]$ is repeated r times, so that $key_r = [k_{11}, .., k_{1r}, .., k_{a1}, .., k_{ar}]$. A response, R, is obtained by concatenating $a \cdot r$ values resulting from the start-up values of $a \cdot r$ SRAM ID cells. Helper Data are obtained by XORing key_r and R, $H = key_r \oplus R$.

In the configuration mode, the *key* is recovered using the ID_mask and the Helper Data, H, as follows. The start-up values are extracted. An ID is generated with the start-up values of those cells that the ID_mask indicates as ID cells. A response, R' (slightly different to R, since even ID cells may provide some bit flipping) is obtained by taking $a \cdot r$ values of ID cells. Helper Data are XORed with R': $key' = H \oplus R' = key_r \oplus R \oplus R'$. *key* is recovered by using the decoder of the error correcting code: $key = ECC(key')$.

Besides, the seed of the nonces is generated with the start-up values of those cells indicated by the RND_mask as RND cells. Although the ID_mask, RND_mask and the Helper Data, H are stored in the NVM, this fact does not compromise security since the key cannot be recovered without the SRAM start-up values. During the configuration mode, they are loaded from the NVM into the lower part of the SRAM. The upper part of the SRAM is used in the Security Unit to extract the start-up values.

The nonce counter is initialized with the seed during the configuration mode. During the trusted sensing mode, it increases the count to generate a new nonce.

The core of the Security Unit is the cryptographic module that is one of the modules described in Section 2.

3.3 *Control Unit*

The Control Unit is a finite state machine (FSM) that enables and disables the blocks depending on the states. The main states are shown in Fig. 6. Two parts are differentiated, the left part corresponding to the configuration mode and the right part corresponding to the trusted sensing mode. When the sensor is powered up, the configuration mode starts.

3.3.1 *Configuration mode*

- The ID_mask, RND_mask, and Helper Data are read from the NVM.
- The *key* is recovered. The seed for the nonces is generated and the nonce counter is initialized with the seed.
- The PWAR configuration data and parameters are read from the NVM and stored in the SRAM.

Additionally, during the configuration mode, the *key* is absorbed if HMAC is used, or the *key* is read if AEGIS or ASCON is used.

The sensor remains in the state IDLE until it receives new input data. The sensor is set to the state OFF-ON by the FSM whenever it is powered-on, thus achieving the sensor is always configured before entering the trusted sensing mode. This also guarantees correct performance of the sensor under unexpected suspension of power supply.

3.3.2 *Trusted sensing mode*

- Input data (x) are read.
- The hyper-rectangle where the input data is located ($x \in P_i$) is determined.
- The parameters (f_i) associated to the hyper-rectangle, P_i are extracted.
- The PWAR unit output $y(x)$ is generated and the nonce counter is updated.
- The cryptographic module takes the *key*, the *nonce* and the measurement y, and provides the authentication code attached to the measurement if HMAC is used, or, the ciphertext C and the authentication *tag* in the case of AEGIS and ASCON.
- The trusted virtual sensor output is provided.

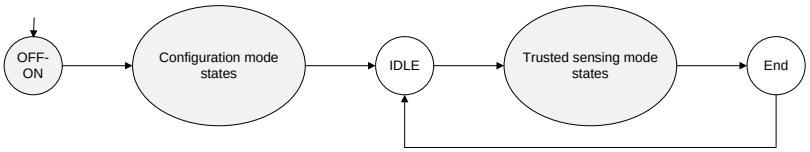

Figure 6: State diagram of the control unit

The trusted virtual sensor was synthesized in a 90-nm CMOS technology provided by Taiwan Semiconductor Manufacturing Company (TSMC). A standard low-power dual-port SRAM IP module was employed for the SRAM. The register-transfer level specifications of the other blocks were synthesized using Design Vision tool from Synopsys.

4.1 *Features of the design*

- PWAR unit. The maximum number of input variables, n, is 4. The bits of the input variables $\{x_1, , x_4\}$ and the parameters that define affine functions are 12. The bits of the output variable, y, are 26. The maximum number of hyper-rectangles, $P = 2^p$, is 4096 ($p = 12$), and the maximum number of intervals per input, $L_k = 2^{p_k}$, is 128 ($p_k = 7$). Hence, the SRAM has 4096×60 bits.
- Security Unit. Table 1 shows the size of the *key* and the *nonce* used by each cryptographic module. A binary repetition code with codeword length $r = 29$ is employed to recover the *key* in the HDA block.

Table 1: Features of the Security Unit depending on the cryptographic modules

	key (bits)	*nonce* (bits)
HMAC	60	32
AEGIS / ASCON	128	128

- NVM. This memory has an output bus with 12 bits. AEGIS and ASCON need more memory than HMAC since they employ larger *key* and *nonce* sizes, as shown in Table 1. The size of the Helper Data depends on the key size and the length of the error correcting code. A length of 29 was proven enough to cope with the maximum bit flipping measured experimentally in the SRAM employed. The sizes of the masks were proven enough according to the average ID and RND cells found experimentally in the SRAM employed. The bits required by the PWAR Unit are 245,796.

  The size of the parameters stored in this memory depends on the selected cryptographic module as shown in Table 2. The parameters for the PWAR configuration are independent of the module.

  Table 2: Features of the NVM depending on the cryptographic modules

	RND_mask (bits)	ID_mask (bits)	Helper Data (bits)
HMAC	480	2040	1740
AEGIS / ASCON	1860	4320	3720

4.2 *Comparative analysis*

Independently of the cryptographic module employed, the configuration of the sensor finishes once the NVM is completely read, since other required operations are executed in parallel. Figure 7 (a) shows how the parallelization

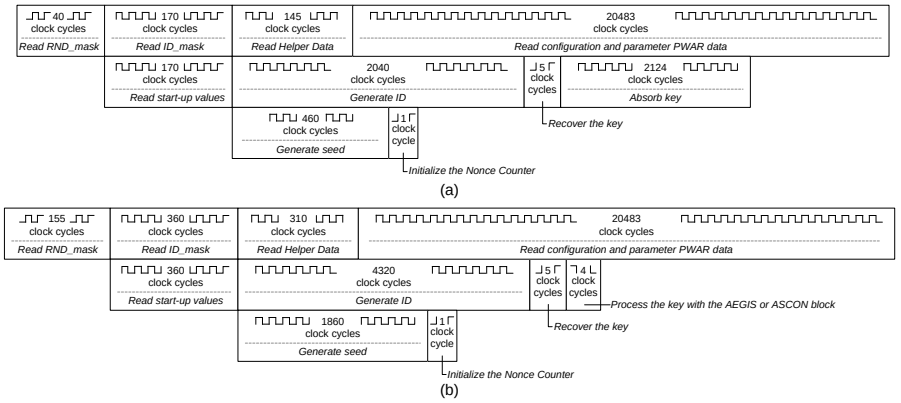

Figure 7: Timing of configuration mode for (a) HMAC, (b) AEGIS/ASCON

is carried out when HMAC is used. In total, configuration mode takes 40+170+145+20,483=20,838 clock cycles. Timing for the configuration of sensors using authenticated ciphers is illustrated in Figure 8 (b). It takes more time due to the larger number of bits required by the RND_mask, ID_mask, and Helper Data. The total number of clock cycles increases up to 155+360+310+20,483=21,308 clock cycles.

Figure 8 shows the timing of the trusted sensing mode depending on each particular cryptographic module. Again the operations have been parallelized to accelerate the trusted virtual sensing mode as much as possible. A sensor using HMAC provides the trusted measurement in 7+10,624= 10,631 clock cycles. The PHOTON-80/20/16 needs 708 clock cycles to permute a block. Since one block has 20 bits and the *key* has 60 bits, the three blocks of the *key* are absorbed in 708x3=2124 clock cycles. This is done during configuration mode, as shown in Figure 8 (a). During trusted sensing mode, since the message size is 58 bits (32 bits of the *nonce* and 26 bits of the PWA output), the HMAC absorbs 3 20-bit blocks in the first hash and squeezes 4 20-bit blocks to generate the first 80-bit output. In the second hash, 4 20-bit blocks are absorbed and squeezed. Hence, the HMAC block takes 708x(3+4+4+4)=10,620 clock cycles to provide the authentication *tag*. In case of using authenticated ciphers, the output is provided in 1+10+1+9= 21 clock cycles using AEGIS and 1+15+7+13=36 clock cycles using ASCON.

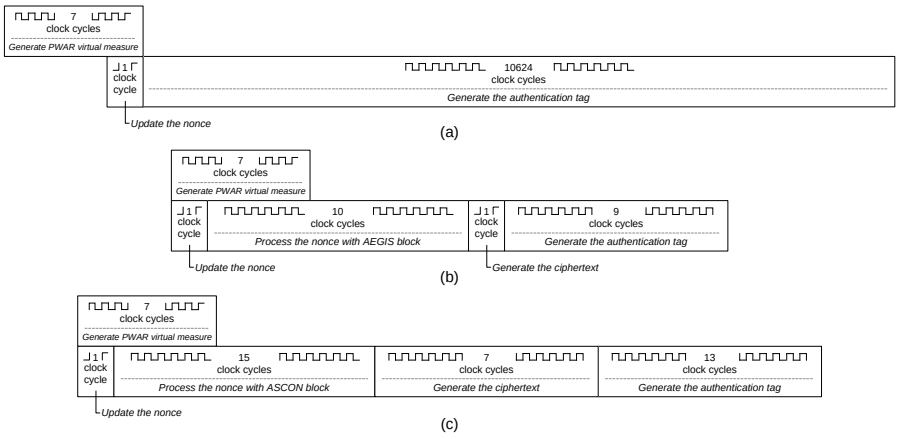

Figure 8: Timing of trusted virtual sensing mode for (a) HMAC, (b) AEGIS, (c) ASCON

Table 3 shows the area and power consumption during trusted sensing mode of the different cryptographic modules provided by Design Vision tool from Synopsys. In terms of area, ASCON offers the most competitive result. The simulations estimate a similar power consumption working at a frequency of 50MHz. Table 3 also shows the clock cycles required by each cryptographic module during trusted sensing module. In terms of speed, AEGIS offers the most competitive result.

Table 3: Silicon area of cryptographic modules and their power consumption and required clock cycles during trusted sensing mode

	Area (mm^2)	Power (mW) @50MHz	N_{clk}
HMAC	0.032	0.69	10620
AEGIS	0.140	0.77	20
ASCON	0.026	0.92	35

5 CONCLUSION

The third-round candidates AEGIS-128 and ASCON-128 of the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) are very well suited to design trusted virtual sensors into CMOS integrated circuits. They make the sensor ensure integrity, authenticity and confidentiality of the measurements at very low cost in area occupation, power consumption and clock cycles of processing time. They offer much better performance than a solution based on the standard Keyed-Hash Message Authentication Code (HMAC) that uses the lightweight hash function PHOTON 20/80/16, which ensures integrity and authenticity but not confidentiality. Considering a 90-nm CMOS technology, ASCON-128 is the smallest module (it occupies 0.026 mm^2, HMAC occupies 0.032 mm^2, and AEGIS occupies 0.140 mm^2). In terms of trusted measurement time, AEGIS-128 allows the fastest response (0.42 microseconds using AEGIS, 0.72 microseconds using ASCON, and 212.62 microseconds using HMAC, all considering an operation frequency of 50 MHz). Since the key and nonce required by these cryptographic modules can be provided by exploiting the uniqueness and randomness of the start-up values of the static random access memory (SRAM) inside the virtual sensor, the sensor is also ensured to be non-counterfeit.

ACKNOWLEDGMENT

This work was supported in part by TEC2014-57971-R project from Ministerio de Economía, Industria y Competitividad of the Spanish Government (with support from the P.O. FEDER of European Union) and 201750E010 (HW-SEEDS) project from CSIC. M.C. Martínez-Rodríguez was supported by FPI fellowship program for Students from Spanish Government. The work of P. Brox was supported by V Plan Propio de Investigación through the University of Seville, Seville, Spain.

REFERENCES

[1] ASCON: A family of authenticated encryption algorithms.

[2] Athena: Automated tools for hardware evaluation.

[3] Cryptographic CAESAR competitions.

[4] M. Ågren, M. Hell, T. Johansson, and W. Meier. Grain-128a: A new version of grain-128 with optional authentication. *International Journal of Wireless and Mobile Computing*, 5(1):48–59, 2011. cited By 50.

[5] J. P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A lightweight hash. *Journal of Cryptology*, 26(2):313–339, Apr 2013.

[6] I. Baturone, M. A. Prada-Delgado, and S. Eiroa. Improved generation of identifiers, secret keys, and random numbers from SRAMs. *IEEE Transactions on Information Forensics and Security*, 10(12):2653–2668, 12 2015.

[7] Andrey Bogdanov, Miroslav Knežević, Gregor Leander, Deniz Toz, Kerem Varıcı, and Ingrid Verbauwhede. spongent: A lightweight hash function. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, pages 312–325, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[8] F. Comashi, B. A. G. Genuit, A. Oliveri, W. P. M. H. Heemels, and M. Storace. FPGA implementations of piecewise affine functions based on multi-resolution hyperrectangular partitions. *IEEE Transactionis on Circuits and Systems I*, 59(12):2920–2933, 12 2012.

[9] S. Eiroa and I. Baturone. FPGA implementation and DPA resistance analysis of a lightweight HMAC construction based on PHOTON hash family. In *23rd International Conference on Field programmable Logic and Applications*, pages 1–4, 9 2013.

[10] Hannes Gross, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. ASCON hardware implementations and side-channel evaluation. *Microprocessors and Microsystems*, 52:470 – 479, 2017.

[11] J. Guajardo, S. S. Kumar, G. Schrijen, and P. Tuyls. Fpga intrinsic pufs and their use for ip protection. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 63–80, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[12] J. Guo, T. Peyrin, and A. Poschmann. The PHOTON family of lightweight hash functions. In *Advances in Cryptology-CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science (LNCS)*, pages 229–239, 2011.

[13] Haorong Li, Daihong Yu, and James E. Braun. A review of virtual sensing technology and application in building systems. *HVAC&R Research*, 17(5):619–645, 2011.

[14] M. C. Martínez-Rodríguez, M.A. Prada-Delgado, P. Brox, and I. Baturone. CMOS digital design of a trusted virtual sensor. In *NORCAS 2017*, 8 2017.

[15] M.C. Martínez-Rodríguez, M.A. Prada-Delgado, P. Brox, and I. Baturone. VLSI design of trusted virtual sensors. *Sensors (Switzerland)*, 18(2), 2018.

[16] S. Meguerdichian and M. Potkonjak. Security primitives and protocols for ultra low power sensor systems. In *2011 IEEE SENSORS Proceedings*, pages 1225–1227, 10 2011.

[17] J. E. Miranda-Vega, W. Flores-Fuentes, O. Sergiyenko, M. Rivas-López, L. Lindner, J. C. Rodríguez-Quiñonez, and D. Hernández-Balbuena. Optical cyber-physical system embedded on an FPGA for 3D measurement in structural health monitoring tasks. *Microprocessors and Microsystems*, 56:121 – 133, 2018.

[18] National Institute of Standards and Technology. Recommendations for Block Cipher Modes of Operation:. The ccm mode for authentication and confidentiality.

[19] National Institute of Standards and Technology. Recommendations for Block Cipher Modes of Operation:. Galois/counter mode (gcm) and gmac.

[20] A. Oliveri, L. Cassottana, A. Laudani, F. Riganti Fulginei, G. Lozito, A. Salvini, and M. Storace. Two FPGA-oriented high speed irradiance virtual sensors for photovoltaic plants. *IEEE Transactions on Industrial Informatics*, PP(99):1–1, 9 2015.

[21] C. Paar and J. Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer Berlin Heidelberg, 2009.

[22] Olaf Pfeiffer. *Implementing Scalable CAN Security with CANcrypt: Authentication and Encryption for CANopen, J1939 and other Controller Area Network or CAN FD Protocols*. Embedded Systems Academy Inc., 2017.

[23] T. Poggi, M. Rubagotti, A. Bemporad, and M. Storace. High-speed piecewise affine virtual sensors. *IEEE Trans. on Industrial Electronics*, 59(2):1228 – 1237, 2012.

[24] D. Samyde, S. Skorobogatov, R. Anderson, and J. J. Quisquater. On a new way to read data from memory. In *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, pages 65–69, Dec 2002.

[25] R. Santamarta. Go nuclear: Breaking radiation monitoring devices. In *BlackHat USA 2017, Las Vegas*, 7 2017.

[26] J. Turner. The keyed-hash message authentication code (HMAC), 2008.

[27] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In Tanja Lange, Kristin Lauter, and Petr Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013: 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, pages 185–201, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.