Capitolo 1

Scientific Background and Related Work

qui va il cappello introduttivo, direi che lo aggiungo alla fine che ho descritto tutti gli argomenti dei related.

1.1 Physically Unclonable Function

Physically Unclonable Function (PUF) are a physical entity designed to be easy to evaluate but hard to predict which uniquely identify a physical component. PUF were first time presented in 2001 by Pappu *et al.* namely Physical One-Way Functions (POWFs) in [**?**] and [**?**]. Usually, a malicious entity tries to steal information by mean of a physical attack to retrieve sensitive information from RAM or ROM, or more generally on a memory chip. Secure Storing of digital information from physical attacks is difficult and expensive where safe solutions are expensive. PUF is suitable to realize a low-cost solution for ICs authentication [**?**] while it can generate a secret volatile keys for cryptographic operations without store them. PUF outputs persistent and unpredictable results, called *responses* , without the possibility to discover the input applied as a one-way function, namely *challenges*. It is possible to define *Challenge-Response Pairs (CRPs*): an unique set of couples for

each device that contains a PUF. These can be implemented with several physical systems. In this thesis, we focus to Silicon PUFs, introduced by Gassend *et al.* in [?] and [?].

These and other works offer a way for improving security in a system with a low costs [?].

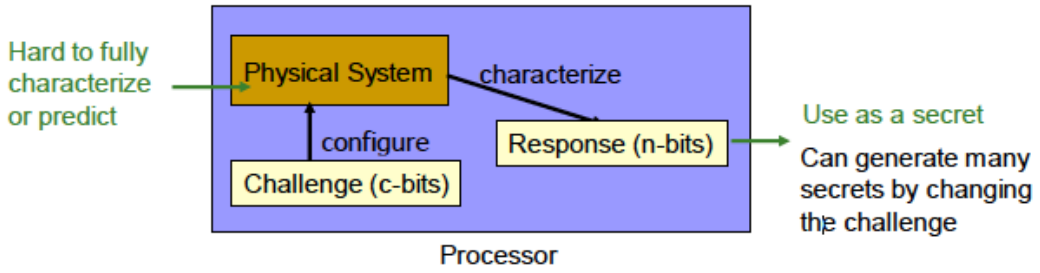


Figura 1.1.1: A logical PUF example

The physical phenomena which mainly characterize PUF are the random variation in dies across a wafer, and from a wafer to wafer due to, i.e. process temperature and pressure variation, during the various manufacturing steps [?]. Even a small variation in these variables door two chips to be different. Silicon PUFs have an exponential number of possible pair between challenges and response such that it is not possible to realize a software cloning of all the possible CRPs. PUFs can be realized for both Programmable Field Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). PUF's output looks like a random function that is unpredictable even for an attacker with physical access, this happen because, due to process variations, two integrated circuits are not identical [?].

1.1.1 PUF Properties

Considering PUF as a mathematical function

$$\prod : X \to Y : \prod(x) = y \tag{1.1.1}$$

It is possible to highlight the relationship between challenge and response and present the following properties as described in [?] and [?]:

- **evaluable:** given $\prod$ and x, y can be evaluate within polynomial time;
- **unique:** $\prod$ can uniquely identify a device that embedds $\prod$.
- **reproducible:** $y = \prod(x)$ is reproducible up to a small error[1].
- **unclonable:** $\prod$ must be:
  - *physical unclonable*: it must be difficult to build a device containing another PUF $\prod_2 \neq \prod$ such that $\prod_2(x) \approx \prod(x), \forall x$;
  - *mathematically unclonable*: it must be difficult to build a mathematical function f_r such that $f_r \approx \prod(x), \forall x$;
- **unpredictable**: given only a Challenge Response Pair $\Phi = \{(x_i, y_i = \prod(x_i))\}$ it is hard to predict value $y_c \approx \prod x_c$ up to a small error, $\forall x_c$ being a random challenge do not exist Φ;
- **one-way**: given $yand \prod$ is difficult to find x such that $\prod(x) = y$;
- **tamper-evident**: if a malicious entity try to alter the device containing the function $\prod$ he obtain $\prod'$ with the transformation $\prod \longrightarrow \prod'$ but $\prod(x) \neq \prod'(x), \forall x$.

1.1.2 Quality Metrics

PUF validity can be estimated verifying that it produces responses respecting specific evaluation parameter, for this reason, the following quality metrics are presented, introduced by Maiti *et al* in [?]:

[1]For this reason, *reproducibility*, that PUF is different from TRNG (True Random Number Generator)

- **Uniqueness**: This factor estimates how PUF responses can distinguish chips each other. The Hamming Distance (HD) is the difference of bits between two bits string. Uniqueness is estimated using hamming distance between all the CPRs extracted from different devices. Uniqueness mathematical relationship is shown below:

$$U = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{HD(R_i, R_j)}{n} \cdot 100\% \tag{1.1.2}$$

  K indicates the chips that implementing the same PUF design, each pair (i,j) of chips have a n-bit response R_i and R_j respectively. The ideal value of U is 50%.

- **Reliability**: This factor gives an estimation about how is stable a PUF response under environmental changing, i.e. temperature or supply voltage. It specifically indicates the number of bits that change their value using the same challenge as input under different environmental conditions. It is called *robustess* or *intra-PUF variation*. Let R_i be the n-bit baseline reference of the PUF with a specific design P, and k the number of generated measurements $R'_{i,j}(j = 1, 2, ..k)$. The reliability can be defined as:

$$HD_{intra} = \frac{1}{k} \sum_{j=1}^{k} \frac{HD(R_i, R'_{i,j})}{n} \cdot 100 \tag{1.1.3}$$

$$Reliability = 100 - HD_{intra} \tag{1.1.4}$$

  Reliability is the opposite of *intra-chip Hamming Distance* and its ideal value is 100%.

- **Security/Attack Resilency**: PUF ability to prevent stealing of secrets to an adversary, like PUF CRPs.

- **Uniformity**: This metric represents the distribution of 0 and 1 in the PUF

response or in a PUFs sets of responses and can be mathematical described as:

$$RU(i) = \frac{1}{n} \sum_{t=1}^{n} R_{i,t} \cdot 100 \tag{1.1.5}$$

The variable, named n is the number of response bits while i is the i-th response. and can be correctly defined in percentage: 100% means that all R_i are 1. A true random bits is characterized by an uniformity close as possible 50%. Let R be the number of considered response,RU is the *average uniformity*, represented as:

$$RU = \frac{1}{R} \sum_{i=1}^{R} RU(i) \tag{1.1.6}$$

- **Bit-Aliasing (BA)** : indicates that some response bits are biased towards either 0 or 1 on all devices. It can be defined as:

$$BA(t) = \frac{1}{k} \sum_{i=1}^{k} R_{i,t} \cdot 100 \tag{1.1.7}$$

where the variable t indicates the t-th response bit, while k is the number of devices. $R_{i,t}$ is the t-th response bit of the i-th device. If the result of this function is 100% the t-th response bit is 1 for all the devices which execute the PUF. The ideal value is 50%. There is a strong relationship between uniformity and bit-aliasing, conversely, it is important to notice that even if BA of each response is around 50% does not mean that bits are independent. On the other hand, independent bits can lead to an uniformity percentage of 50%.

1.1.3 Use of PUF as Cryptographic Key

Silicon PUF are characterized by noise responses due to circuit delays sensitive to environmental variations. In [**?**] is noticed that PUF can be used as cryptographic key, hence, for PUF responses that are not fully random. Thus, in order to

extract keys suitable for cryptographic approaches [**?**] post-processing techniques, as majority voting of fuzzy extraction approaches is required have been presented in literature. Post-processing techniques allow to produce binary strings, eligible for cryptographic operations, from un-stable PUF responses. Majority voting is a post-processing technique based on multiple measurements over a PUF circuit [**?**]. The measurement repetition can be done by averaging N sequential measurements on a PUF, using the Temporal Majority Voting (TMV) technique, or conversely, averaging N simultaneous measurements on different PUFs executing the Spatial Majority Voting (SMV). Both TMV abd SMV evaluate different measurements in order to average extracted values and decide for the most frequent. Hence, such values need to be compared with pre-defined thresholds to establish the result of the voting. For this reason, majority could be not reliable in the case of average values near thresholds. Conversely, the fuzzy extractor technique involves only one measurement and exploits an error correction code (ECC) in order to retrieve stable binary strings from noisy responses. A formal definition of fuzzy extractor is provided in [**?**] and [**?**]. Fuzzy extractor is composed of two basic primitives :

1. **Information Reconciliation (or Error Correction)**: which removes the error generated by the noise. This phase requires the adoption of error correcting codes [**?**] [**?**] [**?**], to correct bit errors in the PUF output obtaining a stable response.

2. **Privacy Amplification (or Randomess Extraction)**: which enhances the information bit entropy (randomness). An Extractor extracts random bits from PUF that are more uniform as possible. This is implemented using a set of universal hash function with a uniform random seed.

Information Reconciliation and Privacy Amplification primitives are used to perform the following two phases:

1. **Generation Phase**: a *reference* PUF response is extracted and used to generate an Helper data. This response is generated starting from a specific challenge and can be considered as uniform random bit-string. This operation is generally executed during manufacturing time. The helper data is stored into an external memory.

2. **Reproduction Phase**: extracting a *noisy response* R from the PUF during normal execution time its combination together with the Helper Data generates a response equivalent respect the reference response generated into previous phase. Error correction code and randomness extraction functions are used in this phase.

A more detailed description of this two procedures (*Gen* and *Rep*) is made in [**?**]. Therefore, fuzzy extractor guarantees a right reconstruction of the extracted bit-string, enduring a small environment variations. It is important to notice that Helper Data, have to be stored into an external storage accessible during Reproduction phase. Moreover, it is not possible to obtain PUF response from Helper Data and, for this reason it is not required to store it ino a secure storage area.

1.1.4 Types of Silicon PUFs

The literature introduces a wide set of Silicon PUFs [**?**] , [**?**] which present the following features:

- *Tight integration*: several applications in execution on the same platform of the PUF can exploit its functionalities for different scopes,
- *Manufacturing Variability*: which ensures the implicit difference among PUFs,
- *Hiding responses*: the output signal of the PUF is well shielded inside the chip, thus the response is hidden and can be used as secret key in cryptography applications.

Furthermore, PUFs intrinsic randomness, on which this work is focused, are often inherently present in a device due to its manufacturing process, which do not requires additional manufacturing processes, and, for this reason, highly attractive [**?**]. PUFs can be manufactured in a standard CMOS IC technology based on the inevitable random manufacturing variability among all the ICs. A PUF classification can be realized analysing how responses are obtained.

- **Weak PUFs**: Considered as a digital *fingerprint* of the circuit, this type of PUFs are characterized by a small number of challenges linearly related with a stable and robust responses into different environmental conditions and multiple readings [**?**]. This type of PUF, if paired with a specific crypto-hardware, can be involved into an authentication process with its response as security identification key.

- **Strong PUFs** [**?**]: A huge number of challenges can be used to generate the corresponding responses . Then, it can be authenticated directly without using any cryptographic hardware. A malicious entity cannot reasonably enumerate all CRPs even if him owns a sample of them, with the response stable to environmental changes and multiple readings

Now is presented a wide background on PUF that have been presented in literature. The description of each presented PUF is correlated by some examples and the following two performance factors:

- *Intra-PUF variation* μ_{intra}: Defined as the number of PUF response bits that vary when is given the same challenge repeatedly in environmental changes. This variation is due to environmental changes and it can be represented as *statistical noise*. Thus this variation indicates the reproducibility of response from individual PUF Circuit,

- *Inter-PUF variation* μ_{inter}: Defined as the number of bit in PUF response that changes between different devices which share a set of challenges. This

variations is due to IC production variability that cause physical difference between silicon devices. Thus this value is a measure of the uniqueness of an individual PUF circuit.

PUFs differs in construction and signal propagation. This happen due to limited resolution of lithographic and chemical processes and non-uniform fabrication conditions. Furthermore, the signal propagation into IC is not the same for all ICs with same design. Using this feature as a source of intrinsic randomness it is possible to build PUFs that are characterized by different signal propagation delay, namely Delay Based PUFs. Otherwise, manufacturing variations can produce different state into ICs on which relies Memory based PUFs. Finally, a mixture of the following two types of PUF permits to realize Composite PUFs. The following three subsections describes respectively Delay Based, Memory based and Composite PUFs.

Delay Based PUFs

This type of PUFs is based on signal delay propagation. It changes from IC to IC because MOSFET channel lengths, widths and threshold voltages etcetera, are subject to manufacturing variability as described before.

Arbiter PUF

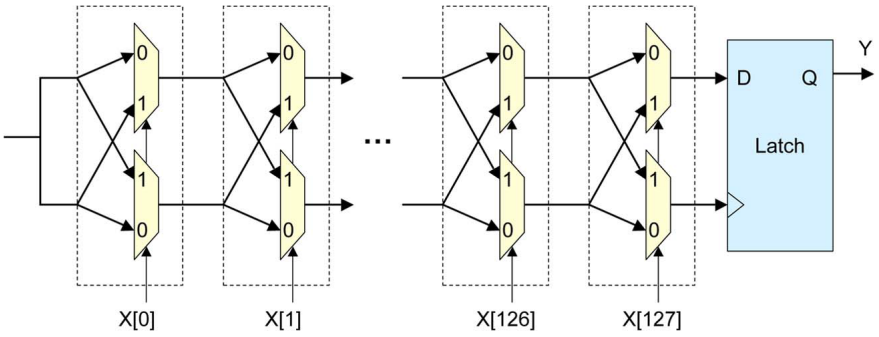


Figura 1.1.2: Arbiter PUF example circuit

As explained in [?] it is possible to implement manufacturing gate delay drawing out a race condition in a symmetric circuits. Arbiter PUF, in 1.1.2, exploited this

phenomenon. It is presented by *Lim et al* in [?].

Figure 1.1.2 shows a path composed by multiple pairs of multiplexers, called switch blocks, which can become straight or switched depending on the selection bit. Selection signal comes from the challenge: the path of the signal will vary depending of these values, and despite identical layout, due to manufacturing variability into gates causes the delay variate for each different IC. At the end of path, a latch component , called *arbiter* will receive the signal and the input 'D' signal with a certain order and time. As explained, the response value depends on the challenge bits: in particular, figure 1.1.2 shows a challenge of 127 bit that have 1 response bit. This layout can change, for example, a parallel structure with 128 response bit can be build duplicating 128 times the design

It must be specified that if the delays are close enough, the latch's set-up time is not enough giving back a non stable output. This phenomenon is called *metastability* and generates noise in the PUF response [?] . The design is symmetric and does not contain any secret information: even if an adversarial manufacturer owns the design he cannot obtain the same CRPs due to manufacturing variation of each IC. Furthermore, the high costs required to evaluate gate delays make the individual delay extraction expensive for invasive attackers. Environmental changes effect on Arbiter PUF is analyzed in [?]. in which is empathized a main drawback in this logic: each delay is independent from other and all the delays are added linearly to each other making possible to create a linear system to model gate delays and generates few CRPs [?]. A possible solution consists in the introduction of other *non-linear* effects in order to give a certain error to modelling attacks, i.e. *XOR Arbiter PUFs* or *feedforward Arbiter PUFs* [?], [?], which anyway are vulnerable against strong machine learning and side-channel attacks as explained in [?] and [?].

An Arbiter PUF implementation is developed in [?] using 23 FPGAs achieving $\mu_{inter} \approx 23\%$ and $\mu_{intra} \approx 99\%$. It is interesting to notice the high reliability

and low uniqueness obtained specifying that edge-triggered flip-flops are not fair as arbiter while flip-flops D are a more reasonable choice.

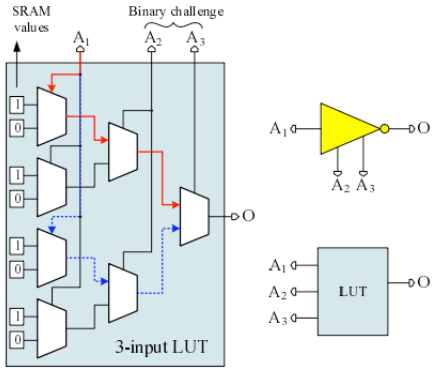


Figura 1.1.3: Programmable Delay Lines Example

Programmable Delay Lines As described before, in order to obtain a response from an IC suitable for PUF use, the signal routing problem have to be controlled. Thus, in [**?**] *programmable delay logic* are presented. PDL can be implemented into LUT internal structure. Moreover, PDLs perform fine tuning to cancel out delay skews caused by asymmetries in routing and systematic variations, and them can be used for an efficient Arbiter-PUF implementation.

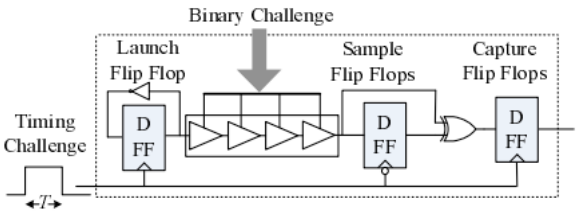


Figura 1.1.4: Delay Characterization Circuit

Figure 1.1.3 shows LUT configured with A_1, A_2, A_3 as input. LUT output is A_1 inverted value where A_2, A_3 are use to determine the path. In figure 1.1.3 two paths are shown: blue path provides ($A_2A_3 =' 00'$) and is longer than the red path with ($A_2A_3 =' 01'$). Thus, blue path causes a larger propagation delay than red one.

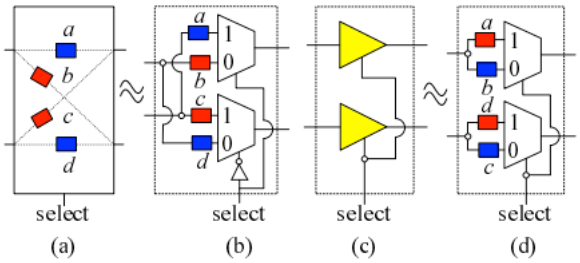


Figura 1.1.5: Path Swapping Switch with their delay abstraction (a)(b), PDL based switch and its delay abstraction (c)(d)

In order to evaluate LUT propagation delay on different inputs, a delay characterization circuit is considered, as shown in figure 1.1.4. This circuit is composed of: a *launch* Flip-Flop, a *sample* Flip-Flop, a *capture* Flip Flop, a *XOR gate* and a *Circuit Under Test* (CUT).

At the rising edge of the clock a signal is sent through the CUT by the launch flip-flop. At the falling edge of the clock, the output of the CUT is sampled by the sample flip-flop. If the signal arrives at the sample flip-flop before that sampling takes place, the correct value is sampled. Later, the XOR compares the sampled value with steady state output of the CUT and produces a zero if they are the same. Otherwise, the XOR output rises to '1', indicating a *timing violation*. If the signal arrives almost simultaneously with when sampling occurs, the sample flip-flop enters into a metastable condition and produces a non-deterministic output. By sweeping the clock frequency and monitoring the rate at which timing errors happen, the CUT delay can be measured with a very high accuracy. Resuming, this circuitry can evaluate LUT delay with low-overhead in order to tune and calibrate the delay bias caused by asymmetries in signal routing. Thus structure can mitigate arbiter routing problems. Moreover majority voting and redundancy are introduced to mitigate arbiter metastability problem.

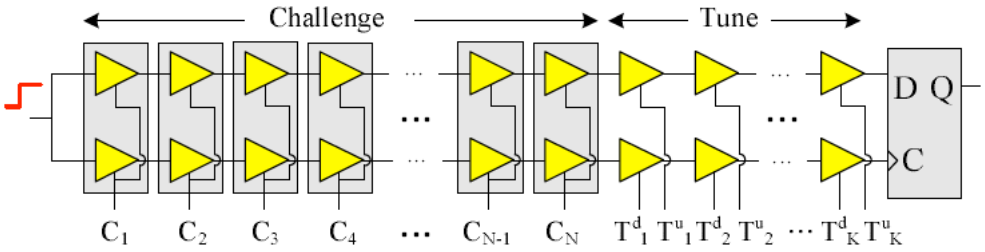


Figura 1.1.6: Arbiter-based PUF proposed by Manjzoobi et al

The Arbiter-PUF structure previously presented uses patch swapping switches, shown in 1.1.5 (a)(b). An alternative arbiter-PUF scheme requires perfect layout symmetry, hence, non-swapping switch blocks are introduced 1.1.5 (c)(d). Basing on these assumptions, an arbiter PUF structure which relies on new switch structure and the tuning blocks is shown in Figure 1.1.6: this structure is composed of N non-swapping switches and K PDLs. PDL blocks can be used to distinguish challenges that cause a large delay difference under each different temperature or voltage marking them as robust challenges. But responses of robust challenges can have lower entropy.

Ring Oscillator PUF Ring Oscillator (RO) [**?**] is a delay-based PUF, shown in 1.1.7. RO design consists of odd inverting gates which permit the maintenance of a continuous oscillation. Signal frequency is related to each inverting stages, which are the cause of delay too. The RO PUF can be synthesized onto Field Programmable Gate Array (FPGA) or an Application-Specific Integrated Circuit (ASIC).

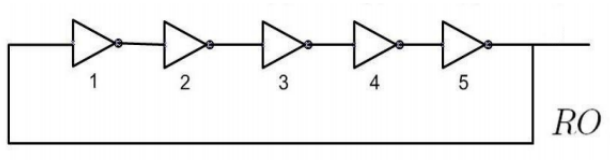


Figura 1.1.7: An example of Ring Oscillator composed of 5 stages

Ring oscillator behaviour is characterized by different gate delays that involves different frequencies between the oscillators. The frequency of a pair of oscillators are

measured and compared: this comparison for each pair generates PUF response bits. Given N oscillators, the overall possible pairings are $\frac{N(N-1)}{2}$. A paring strategy may be the following: given N ring oscillators, not all the obtained bits by pairing are independent. So the entropy[2] is less than $\frac{N(N-1)}{2}$. For maximizing entropy $\frac{N}{2}$ couples are considered, yielding to a limit upper bound of O(N) challenges.

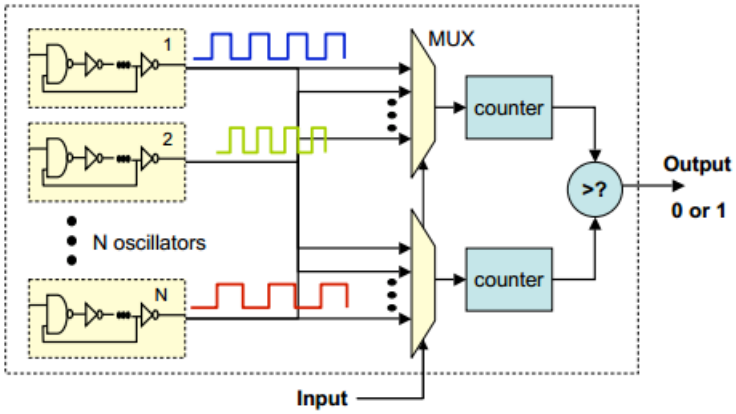


Figura 1.1.8: Ring Oscillator PUF

Ring Oscillator belongs to weak PUF category: once the PUF is fabricated, the ring oscillator's frequency is set generating an output string which remains constant. Moreover, RO PUF are vulnerable to environment variation and possible workaround to this drawback have to be considered: ring oscillators frequencies far apart are paired each other, avoiding the coupling of neighbouring frequencies. RO PUF is composed by oscillators with close frequencies have bigger error probability than oscillators characterized by distant frequencies. In [?] is described a possible Ring Oscillator PUF architecture. In this architecture, shown in 1.1.9, several ring oscillator are divided in k pairs, where each pair have a certain frequency difference δf_i. All the δf_i can be summed and the challenge bits determine the sign of the δf_i into the linear combination. The scheme final block compares results of the sum with zero and produces response bits. Using this structure, the number of possible Challenge-Response Pairs increases o 2^k. In [?], [?], [?], [?], RO (Ring Oscillator)

[2]The term *entropy* indicates the number of independent bits that can be generated from a circuit.

PUFs have valid good μ_{intra} while the reliability do not. Ring Oscillator PUFs relies on cryptographic hardware/software approaches useful to protect security of response bit. There are several malicious approaches that could threaten the RO PUF performances: the most common type of attack does not try to steal the key, but to invalidate the response generation.

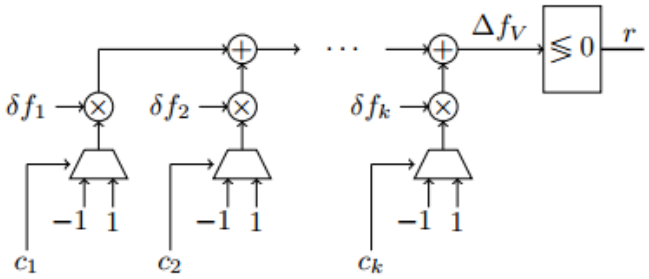


Figura 1.1.9: Ring Oscillator Sum PUF

In [?] is shown a technique in which sinusoidal signal is driven on the ground plane of a RO in order to lock the output signal: this kind of approach can be also used to invalidate a TRNG behaviour.

In addiction, [?] shows a method in which due to electromagnetic radiation, RO PUF output bits are stolen: a parallelized execution of oscillators can be used to face this attack, in this way, an attacker cannot easily understand which bits snoop.

Anderson PUF

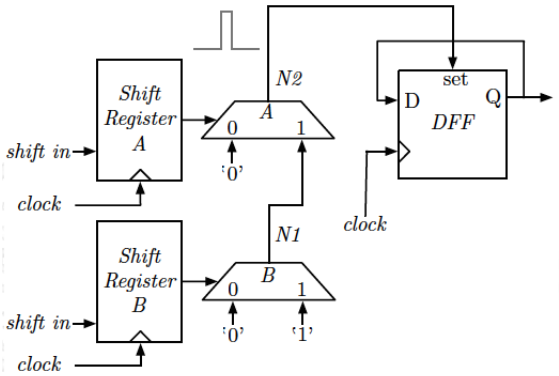


Figura 1.1.10: Anderson Cell

Anderson PUF is presented in [?]: it is based on logic gates available into FPGAs, but it needs a specific implementation on each FPGA family. Anderson PUF is

composed of Anderson PUF cells, which consist of two shift-registers A and B , two 2:1 multiplexers, namely mux-A, mux-B, and one flip-flop D. Figure 1.1.10 shows Anderson PUF cell scheme: each cell produces a logic output of '0' or '1', which depends on manufactoring difference between ICs. The mixture of several cells achieves a valid PUF behaviour. Lets now describe how Anderson PUF works: referring to 1.1.10, the two shift register, called A and B into the figure, are driven by a clock signal and initialized by means of a signal in opposite phase, i.e. $0x5555$ and $0xAAAA$.

The two multiplexer are characterized by selection signals respectively driven by the Shift Register B and A. The multiplexer with selection signal driven by B's output have *logic0* and *logic1* respectively as in 0 and 1 input, while the other one receives the first multiplexer output, $N1$ as input for port '1' and a *logic0* value into '0' input port. The second multiplexer output is defined as $N2$.

Therefore, the flip-flop D has as input a feedback for D port, N2 for set port and a clock signal. Flip-flop D holds the value which depends on signal propagation and delay. Shift registers A and B outputs are opposite: so, when A outputs is *logic1*, B outputs is *logic0*. The shift registers A and B ara characterized by the same design but a different signal propagation delay, and, this phenomena generates PUF unpredictable response. This design requires some additional consideration:

- When the delay between A and mux-A is shorter than betwenne B and mux-B, A switches from *logic0* to *logic1*, so mux-A selects the input *logic1* but N1 has not changed its value due to B which does not switch from *logic1* to *logic0* yet. Therefore, the mux-A output, N2, is determined by the value of N1 which is '1' until shift register transits to '0' as detailed before. During this interval, a *glitch* will appear on N2.

- When the delay between couple A is greater than couple B, B output switches from *logic0* to *logic1* while selection value for mux-A does not change. Since

N2 value depends on mux-B output, that is N1, a glitch will appear on N2 until the mux-A and A couple switches from *logic1* to *logic0*.

The differences between delays determine the glitch shape: Anderson explains that the glitch should have a certain duration. If it is too short, the routing network output becomes predictable and generates the same outputs for several FPGAs. Conversely, the glitch must be greater than the flip flop output set-up. In order to achieve a working Anderson PUF implementation, a *glitch tuning* is required. Without this operation Anderson cells output may be not random and the design can not be targeted as PUF. Anderson PUF is a collection of Anderson cells which work together to generate an unique response that can be used as IC signature.
Anderson PUF is considered as **FPGA-dependent.**, because depending on the FPGA family used, the cell combination design changes: different implementations have been presented in literature.
In [**?**] Anderson designs its PUF on *Xilinx Virtex-5 65 nm* shown in figure 1.1.11. Anderson PUF cell can theoretically be synthesized on a single slice [3], however, adopting glitch tuning is evaluated that a valid glitch on Virtex-5 technology requires a cell design seized two FPGA slices.

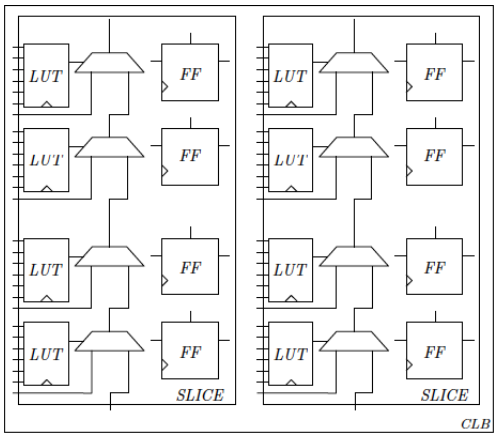


Figura 1.1.11: Anderson PUF on Virtex-5

[3]The FPGA perfom logic function by mean logic resource replicated into CLBs. Each CLB can contain slices, a logical elements which contains a set number of LUTs, flip-flops and multiplexer. LUT is a collection of logic gates hard-wired on the FPGA. The slices can be of two types: SLICEL, which implements logic and aritmetic funcions and SLICEM, which provides memory function and operation as a shift-register and RAM

Anderson measures its PUF, obtaining $\mu_{inter} = 48.28\%$ and $\mu_{intra} = 3.6\%$ considering 128-bit for response PUF and six FPGAs in six different FPGA regions. In [?] is shown an Anderson PUF implementation on Xilinx Spartan-3E 90 nm FGPA both in classical form that enhanced. Anderson PUF is characterized by $\mu_{inter} = 47.18\%$ and $\mu_{intra} = 0.15\%$ for the normal version and $\mu_{inter} = 49.37\%$ and $\mu_{intra} = 0.09\%$ for the enhanced one. Enviromental test temperature is 24^o. When enviromental temperature raises to 70^o *reliabilty* stands around 90.79% in normal form and 98.14% in enhanced form. In a successive work evaluates, the impact of the voltage variation on Anderson PUF, implemented on Xilinx Spartan-3E family [?]. Experimental results states that the supplied voltage value can dramatically change the quality of PUF responses. Other Anderson PUF implementation is made on Xilinx Zynq-7000 28 nm FPGAs [?], Xilinx Virtex-6 40 nm [?]. The Anderson PUF is characterized by the absence of challenge-response pair returning an unique hardware signature. In [?] is presented and improved version of the Anderson PUF called *Enhanced Anderson PUF*, shown in figure 1.1.12. This variation adds logic components to Anderson PUF design: one shift register and two 2-to-1 multiplexer in order to receive input challenges.

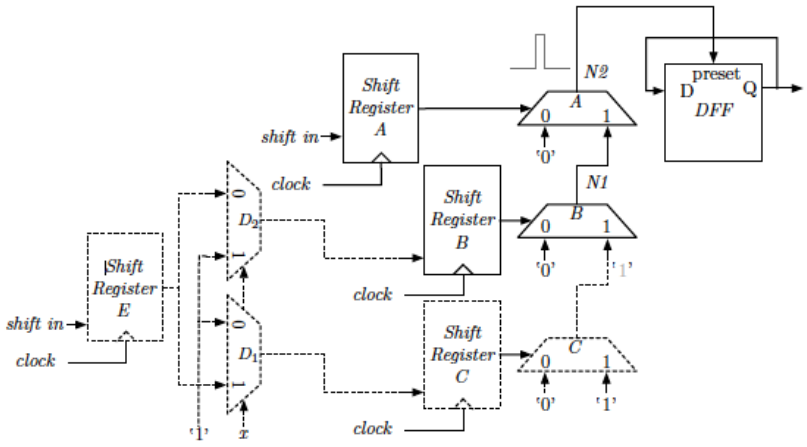


Figura 1.1.12: Enhanced Anderson PUF Architecture

ALU PUF In [?] ALU PUF is presented. This type of PUF belongs to delay-based category exploiting delay differences between two different Arithmetic

Logic Units (ALUs) due to IC manufacturing variations. An ALU can be easily find into an processing architecture design and in order to be adapted for PUF response generation, requires minimal component addictions: delay gates and an arbiter. The former is composed of programmable delay lines, shown in 1.1.4 with a different delay for each IC due to manufacturing variation and the latter which choiche a particular bit as response. Figure 1.1.13 shows an ALU PUF design example with 4-bit operands. An ALU is composed of integrated circuits arithmetic and logic computing. Given two ALUs, each takes the same 4-bit input, first addend is x_0, x_1, x_2, x_3 and second one is represented by x_4, x_5, x_6, x_7. Each ALU generates output signals $o = (o_1...o_3)$ and $o' = (o'_1..o'_3$: this bits are directed to arbiter generating the response, as shown in figure 1.1.13. In order to rightly realize the mechanism, both ALUs are characterized by a *Syncronization Logic* which guarantees same computation time for ALU operations.

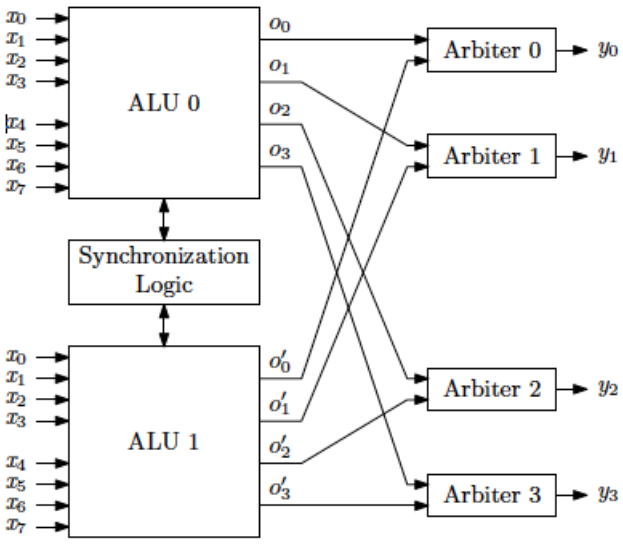


Figura 1.1.13: ALU PUF example

The ALU PUF can be activated running an assembler *add* operation.

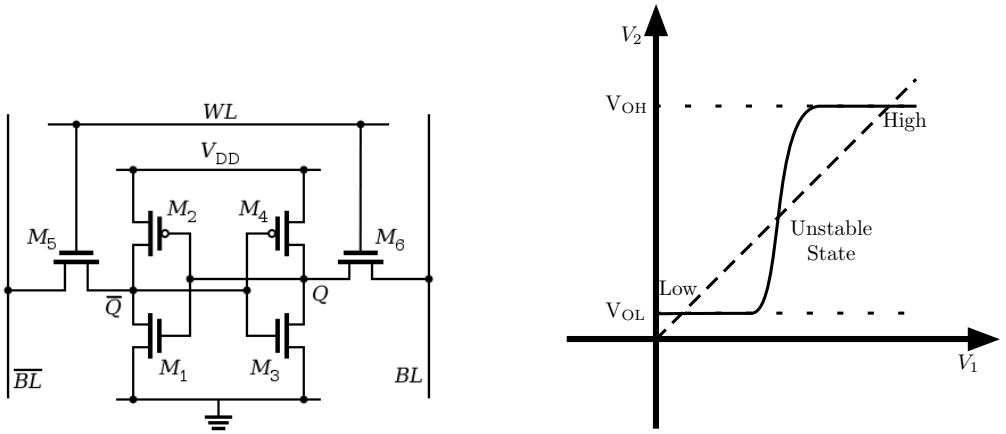
ALU PUF is experimentally tested on Xilinx Virtex 5 XC5VLK110T FPGA device with a 16-bit model. Two symmetric ALU are implemented together with programmable delay lines (PDLs) realized by mean 64 stages. PDLs occupy considerable area

on FPGA. In terms of security, ALU PUF shows drawbacks against machine learning attacks: a solution is the addition of XOR-based obsfuscation network. Using XOR obfuscation approach, an adversary cannot directly read the PUF responses. Performance evaluation on ALU PUF show $\mu_{inter} = 41.3\%$ with XOR obfuscation and $\mu_{inter} = 18.8\%$ without it . Intra-chip Hamming Distance is around $\mu_{intra} = 18.6\%$. Furthermore, ALU PUF response can be suit the role of cryptographic key associated with an error correction mechanism. In [**?**] the author uses BCH error correction code.

Memory Based PUFs

In [**?**] another category of PUF is presented: the memory-based PUFs. During power-Digital memory storage ,during its power-up, is characterized by a random initial state: it can be demonstrated that this initial state can be considered as a PUF. In fact, digital memory implementation is based on bi-stable logic cells, which store one binary digit: once a memory is powered-up in a short time converge to one of its stable state. Each cell tends to stabilize its state to one in particular: memory-based PUFs relies on this behaviour and each memory can generates an unique signature that can be considered as PUF. .

SRAM PUF SRAM PUF exploits intrinsic properties of memories. SRAM cell is a volatile memory technology commonly used into ICs which loses its content after power-down. The SRAM cell is a symmetric circuit composed of two halves and each one contains an inverter cross-coupled with the other one. Figure 1.1.14a shows a schematic for the SRAM six-transistor cell, realized in CMOS technology. SRAM cell is characterized by two stable states whereby it can store '0' or '1', and by one unstable state as illustrated in Figure 1.1.14b. In particular, the SRAM cell keeps the unstable state only when it is powered-up. Due to mismatches between two halves, each cell is forced to a preferred stable state by the inherently positive feedback,



(a) SRAM six-transistors cell schematic. (b) SRAM input-output voltage graph.

Figura 1.1.14: Details of SRAM implementation and input-output characteristic.

realized through the cross-coupling, which amplifies even small voltages variations. As the mismatch between the two halves is imprinted by the manufacturing process, the initial state of each cell cannot be predicted.

We can classify SRAM cell mismatch configurations by their behaviour at start-up as presented in [**?**]:

- **non-skewed cells**, which exhibit a random behaviour due to manufacturing variation effects that are mutually nullified;
- **partially-skewed cells**, which are characterized by a mismatch such that they have a preferred state, but it can be influenced by external conditions, such as the voltage;
- **fully-skewed cells**, which have a start-up preferred state that, unlike the partially-skewed cells, does not change, even under different external condition variations.

Additionally, as stated in [**?**], the amount of cells that very often produce a bit error, i.e. partially-skewed and non-skewed cells, is lower than the number of cells that very rarely produce a bit error, i.e. fully-skewed cells. The SRAM PUF is based on the extraction of a bit sequence stored into a SRAM, which has been powered-up

without any writing operations on it. Hence, we can claim that such bit-sequence has attractive for secure purposes.

The idea about the generation of a fingerprint from memory derives from [**?**] and subsequently several experiments are archived to verify its feasibility [**?**], [**?**]. In 1.1.14a is shown an example of SRAM cell electronical structure.

In [**?**] Guajardo *et al.* built a SRAM PUF using an FPGA with embedded block RAM memories. Power-up values of a SRAM cells group is considered as PUF response. They fixed the environmental temperature around 20^o obtaining $\mu_{inter} = 49,97\%$ and $\mu_{intra} = 3357\%$. Once of the main issue in the adoption of the SRAM PUF is the need to power-up memory to obtain PUF response. In Xilinx FPGA families this problem is addressable by power gating feature [**?**]. SRAM PUF is revealed strong against side-channel attacks, and modelling attacks if the key is secretly stored. Since SRAM PUF is sensitive to noise, in order to produce a stable fingerprint an error correction mechanism is required: in [**?**] is shown a low-overhead solution. The Main security drawback of SRAM RAM is level of access: in the event of a change to an environment variable, such as the temperature, the PUF may fail. Otherwise, an attacker who can access to the power channel may modify the stability of the SRAM cells and consequently the PUF response. An interesting study is made by Helfmeier *et al.* in [**?**] where is shown that SRAM power-on state can be observed with *near-infrared imaging* during transient operation. This is a side-channel attack that can steal the PUF response: an attacker reveals the bit sequence and after stores permanently it.

Butterfly and Flip-Flop PUF The feature offered by SRAM PUF also affect other memory devices based on the cross-coupled circuits. Supposing to have a circuit useful as a memory such as latch, flip-flops and SRAM cells, figure 1.1.16 shows an example of a single latch build with two cross-coupled inverters. Due to its nature this circuit transits from an unstable state to a stable state forced by

an external inputs. The two coupled gates should be as similar as possible, and any differences are only due to factory random variations. These differences cause a mismatch which leads to an unstable state at the memory circuits start-up. Then, Flip-Flop and Butterfly PUF have been introduced.

In [?] and [?] is implemented a *a flip-flop PUF* which exploits a specific power-up behaviour. These studies examined that, during power up, flip-flops on Xilinx FPGA are set with an initial specific random-state which is independent from information stored into the bit-stream. [?] demonstrates that this behaviour can be considered as PUF useful to realize an unique hardware identifier, evaluating $\mu_{inter} \approx 50\%$ and μ_{intra} min 5%, with the adoption of post-processing computation.

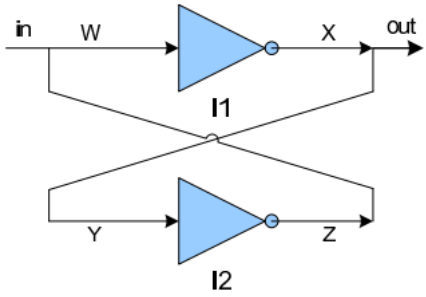


Figura 1.1.15: Cross-Coupled Inverter

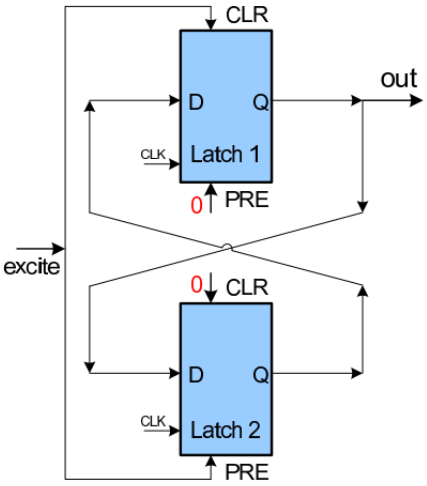


Figura 1.1.16: Butterfly PUF

At the same way, Butterfly PUF [?], is introduced in order to address SRAM PUFs drawbacks. Its design is simile than *latch PUF* [?] design with the difference that

Butterfly PUFs can be efficiently implemented on an FPGA while latch PUFs do not. In 1.1.16 is shown the Butterfly PUF logic structure composed by two latch, each with a signals D,Q,CLK,PRE and CLR respectively as input,output, clock, preset and clear signal. In particularly, if the PRE signal value is *logically high* the Q value is set to '1' while if the CLR signal value is *logically high*, Q value is set to '0'. For Butterfly PUF design, the CLR input of Latch 2 and the PRE input of Latch 1 are always set to low ('0') and the exepected behavior consider that if CLK signal is high the D value is propagated to Q.

Figure 1.1.16 shows how *excite* signal that when it is high brings latches having opposite input/output signals and it involves an unstable state into circuit. When the excite is '0', coupled latch casually assumes one of two possible states with a certain probability. This phenomenon depends on physical differences between gates and wires. Butterfly PUF can be efficiently implemented on FPGA, in which perfect wiring and gates symmetry is not reachable. In [**?**] Butterfly PUF on Xilinx Virtex-5 is implemented achieving $\mu_{inter} \approx 45\%$ and $\mu_{intra} \min 6\%$ as performance values.

Composite PUFs

Several PUF can be mixed designing a larger one called Composite PUF. Composite PUFs has to respect a series of rules in order to obtain a novel PUF design with efficient quality parameters. A formal definition [**?**] define:

> Composite PUF: $\Gamma = \{\gamma_i | \gamma_i : C_i \rightarrow R_i, C_i, R_i \subseteq 0, 1^+, i = 1, ..., m\}$ is a PUF that is defined by repeatedly applying one or more of the following rules:
>
> - $\gamma_i \triangleleft \gamma_j = \gamma_i(\gamma_i(x))$ where $x \in C_j$
> - $\gamma_i || \gamma j(x, y) = \gamma_i(x).\gamma_j(y)$ where $x \in Ci, y \in C_j$ and "." indicates binary string concatenation operation.

- $\gamma_i \otimes \gamma_j(x, y) = \gamma_i(x) \otimes \gamma_j(y)$ where $x \in Ci, y \in C_j$ and $\otimes$ indicates bit-wise XOR operation.

An example of PUF composition can be founded measuring the space required into different PUF combination. However, this solution is not practically feasible and an alternative approach must be adopted.

In [**?**] is proposed a quality evaluation framework which given quality metrics for a single component grants the possibility to evaluate quality of composite PUF. Howevers, PUF outputs must be independent. The work design a series of composite PUF on Spartan-3 Xilinx FPGA family using as component RO PUF and Arbiter PUF: some are shown in figure 1.1.17.

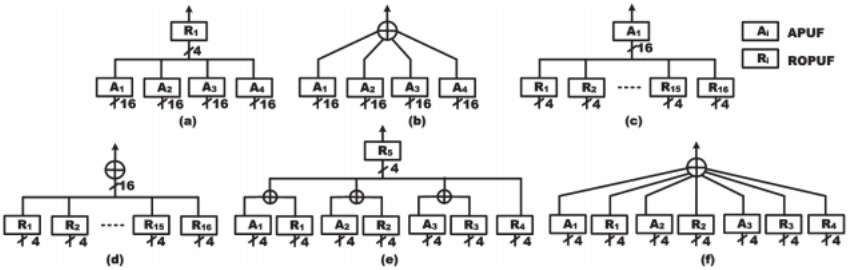


Figura 1.1.17: Composite PUF examples

1.1.5 Known Attacks Against PUFs and Possible Countermeasures

During the previous description of different PUFs, several weaknesses are denoted: this section offer a detailed description of the type of attacks on PUFs. Depending on the specific type of PUF, an attack can be dangerous or not: a PUF a immune to all attacks does not exist and, at same time, there is not an attack that could threaten all the types of PUFs.

Model Based Attack

Model based attacks consist in the creation of a model in order to guess response related to specific challenges: this type of attack is mainly directed to strong PUF. Indeed, it is a mistake think that strong PUF with a large number of CPRs can easily avoid attacks. Adopting Machine Learning (ML) approaches, specially algorithms can infer PUF response requiring only few number of CPRs. An attacker can observe a succession of CRPs acquiring a lot of CRPs: this is a realistic situation where a strong PUF is used. In [**?**] several experiments are made showing that with machine learning approach is possible to reveal Arbiter PUF and RO PUF responses with an accuracy near to 99%. Instead, these attacks do not affect weak PUFs because them are not characterized by a large number of useful CRP useful to create a malicious model.

A Countermeasure against Model based Attacks All the malicious attacks are characterized by certain costs: depending on the situation trying an attack can be a reasonable idea or don't.

Model-based attacks do not require great costs and it are a valid choice to attack strong PUFs. In [**?**] strong PUFs drawback against model-based attack is solved combining PUF response with an hash function: the result of this work is the *Contolled PUF* 1.1.18. A random hash function is placed before PUF, this countermeasure does not allow a malicious attacker to analyse CPRs.

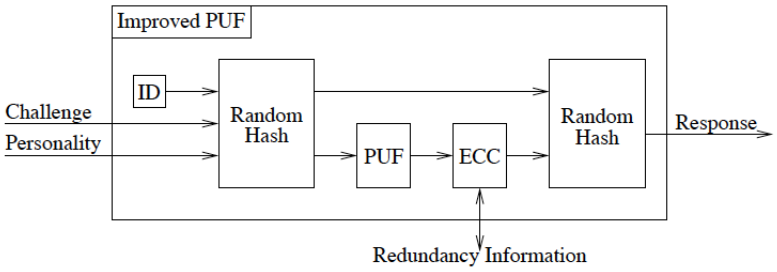


Figura 1.1.18: Controlled PUF generic scheme

CPR are not used directly: a random function h is used, hence considering x ,$f(x)$ and h respectively as the challenge, the response and the hash functions we have:

$$g(x) = h(x, f(x)) \tag{1.1.8}$$

where $g(x)$ represents the improved PUF response.

Hash function application allows to obtain a different response from the original one, and the one-way property of the hash function produces a challenge-response pair which is hard to analyse, even with a large number of couples. A PUF response combination can improve reliability : an attacker cannot see real challenges and additionally, error correction code technique can be considered to transform a noisy PUF response in a more stable value, as shown in figure 1.1.18. It is important to notice that controlled PUF can be useful to generate is multiple personalities: as we can see in figure 1.1.18 there is a user parameter called "Personality". Personality parameter can be used to distinguish PUF responses with same challenge in different applications: the owner effectively has many different PUFs at his disposal and third parties cannot determine if they interacted with the same device on which a PUF is built.

Furthermore, in [**?**] Gassend *et al* is proposed a method based on controller PUF useful to prevent man-in-the-middle attacks for chip attestation which.

Side Channel Attack

Side channel attacks (SCA) are a specific attack based on information gained from physical implementation of a security system, i.e a crypto-system in which theoretical strength of implemented cryptographic algorithms is circumvented. For example, timing information, power consumption, electromagnetic leaks or even sound can provide an extra source of information, which can be exploited to break the system. Therefore, even the PUF design presents vulnerabilities against side channel at-

tacks. Countermeasures focus to eliminate or reduce the release of side information, as power consumption of timing information or breaking the connection between physical effects and the secret data that security module handles. However, even if the PUF is protected from SCA, additional post-processing operations do no, i.e. fuzzy extractor. Literature presents some evaluation of side channel attacks on ICs and PUFs: Power consumption in CMOS technology is caused by state changes hat is the transition of a bit value into IC from 1 to 0 or vice versa. Power analysis attacks exploits this feature: in [**?**] this attack is conducted on Xilinx Virtex FPGA family devices. In [**?**] is shown SCA which exploited side channel information from a post-processing algorithm. Other work into literature, show how RO PUF response can be recognized using electromagnetic emission relevation. [**?**] [**?**]. At the same time, side channel attacks, which belongs to the category of physical attacks, even though the possibility to affect a wide range of ICs, requires high costs for realization, which reveals a very difficult bond for their implementation.

1.1.6 PUF in Commercial Devices

Commercial development of PUF is emerging in recent years with security companies that are evaluating inserting them into IC in order to raise device security.

Intrisic-ID [4] was founded in 2008 as a spin-out of Royal Philips Electronics. They patented a PUF technology deployed in Philips production environments applying it in several technology fields as SmartCards, automotive, set-top box, Pay TV applications, networking and communications, mobile, as well as government and military applications. In 2013 NXP announces a partnership agreement with Intrisic-ID in order to deploy Hardware Intrinsic Security (HIS) adopting PUFs into their next SmartMX SmartCard generation [5] available for commercial use within 2014.

[4] https://www.intrinsic-id.com/

[5] http://www.nxp.com/news/press-releases/2013/02/nxp-strengthens-smartmx2-security-chips-with-puf-anti-cloning-technology.html

Furthermore, NXP announces that for the Chinese Market their components will be built into Samsung Galaxy S6 [6].

Altera puts PUF inside their Stratix-10 family FPGA as well as Microsemi[7] presents their FPGA families for IoT devices with a built-in PUF. PUFFIN project [8] intends to study and show the existence of SRAM PUFs and other types of PUFs in standard PCs, laptops, mobile phones and consumer electronics.

Verayo [9] is a US company which produces secure components for Internet of Things, Trusted Computing and others applications. This company produces low cost authentication solutions using a proprietary PUF technology for digital authentication of products, people and machine. Since 2005 Verayo proposed solutions including unclonable chips, authentication software, consumer engagement, back-end data analysis enabling platforms and key generation. Verayo has ongoing collaborations with US Department of Defense agencies and are based in California.

1.2 Trusted Computing

The Trusted Execution Environment (TEE) is a secure area in which some primitives can be executed. Therefore, TEE is an isolated execution environment which provides security functionalities such as isolated execution of trusted applications characterized by integrity and confidentiality guaranteeing more security feature than a Normal Operating Systems operating on a device. GlobalPlatform [10] and Trusted Computing Group(TCG), non-profit associations, are working to standardize TEE specification: in particular, TCG standardized the Trusted Platform Module (TPM), a security technology which guarantees security features to users. Concur-

[6] http://www.nasdaq.com/press-release/nxp-secure-element-adopted-by-samsung-galaxy-s6-in-worlds-largest-smartphone-market-20150507-00194ixzz3dyQG5aui
[7] http://www.microsemi.com/
[8] http://puffin.eu.org/
[9] http://www.verayo.com/
[10] http://www.globalplatform.org/

rently, industries are working on their own trusted platform as ARM which develops **TrustZone** for its A-processors family.

Trusted Execution Environment In [?] GlobalPlatform describes TEE architecture concepts and principles. An Execution Enviroment (EE) is a set of hardware and software components which typically consist of an hardware processing unit, physical volatile memory and non-volatile, peripheral interfaces and connection between processing unit and other hardware resource. An EE can be characterized by security functionalities or not: in former case we speak about Trusted Execution Enviroment (TEE) in the latter of Rich Execution Environment (REE).

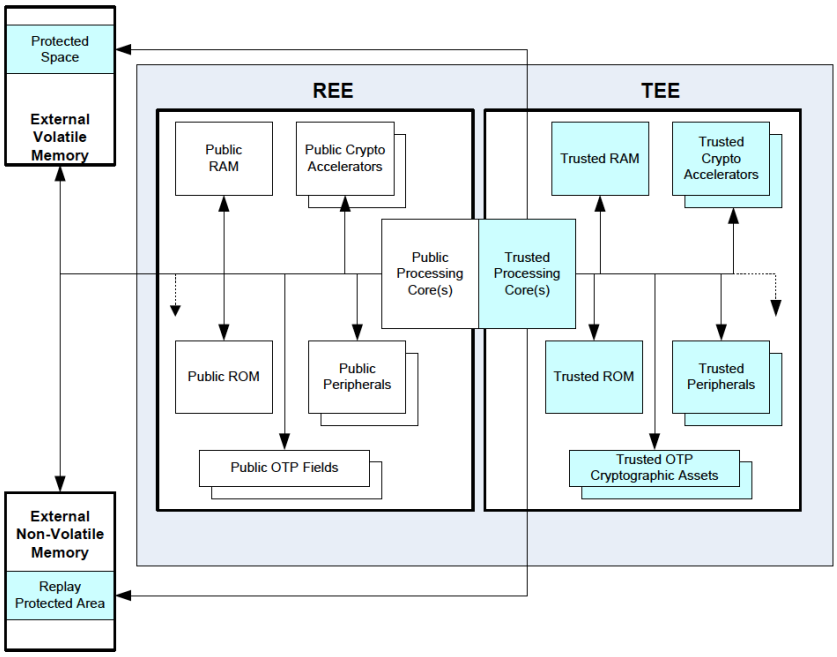


Figura 1.2.1: Hardware Architectural View of REE and TEE

Typically client applications run on REE while trusted application on TEE. A Trusted OS is an operating system running on TEE isolating a secure world from a non-secure world. The secure OS manages secure hardware sources, and dispatches the several services which consist in security-sensitive code, while the untrusted word can be considered a mobile operating system such as iOS, Android or Windows 8. If the systems provides one processor, it can only execute one world at time, context

switch is required to change execution world. This is done via a special istruction called *Secure Monitor Call (smc)*: when the smc instuction is triggered, the processor switches into a monitor mode that performs the context switch allowing messages exchange between worlds. In order to facilitate secure primitive invoking for normal world application, GlobalPlatform consortium develops TEE client API specification [**?**]. Software in the REE must not be able to call directly to TEE functions or Trusted Core Framework. The REE can request operation to TEE but it have to go through a protocol verifying the acceptability of the requested operation. Figure 1.2.1 shows TEE and REE architectural view. REE cannot access to trusted resource, its access is controlled. Trusted resource should be physical isolated from untrusted ones. Figure 1.2.1 represents a logical scheme to present separation of secure world from non-secure.

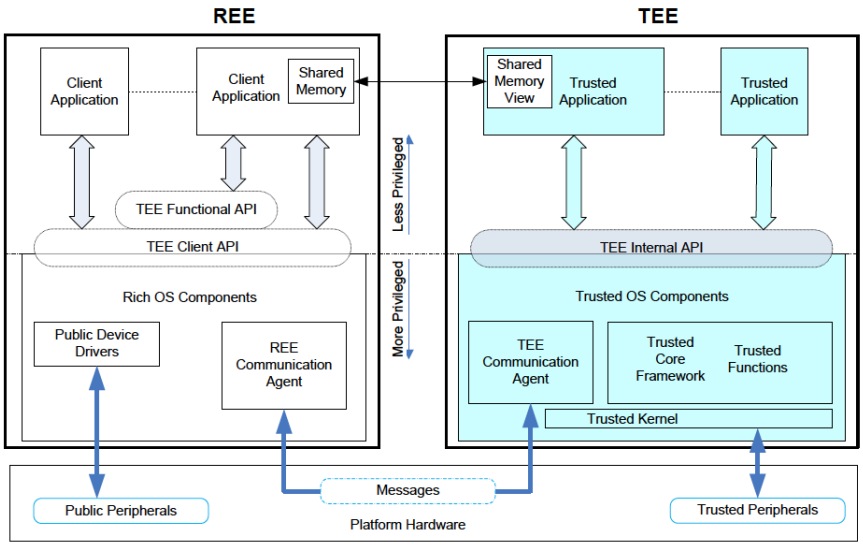


Figura 1.2.2: TEE System Architecture

Therefore, TEE offers several APIs to REE allowing it to exploit TEE functionalities.

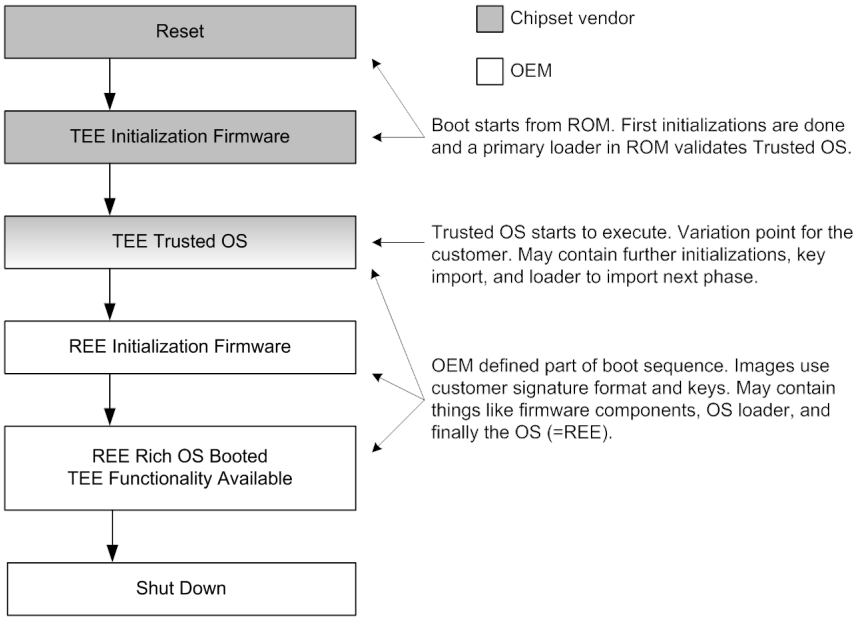


Figura 1.2.3: Simplified TEE Boot Sequence

The relationship between the major software systems components is outlined in the figure 1.2.2. During *Boot Time* the overall hardware powered-up and EE operating system is initialized and loaded into hardware system. For a TEE is required a trusted boot code. Trusted boot includes firmware code that takes control of execution after resetting. A simplified boot sequence flow diagram is shown in figure 1.3.3. In order to maintain the chain of trust, TEE can be instantiated either during secure boot sequence or later by software component already instantiated and characterized by trust level similar to TEE.

1.2.1 Trusted Platform Module

Trusted Platform Module (TPM) is a chip designed to guarantee security feature on an user platform. The technology was in the early days around 2003 known by the name TCPA (Trusted Computing Platform Alliance), or by the project (and team) names in Microsoft, “Palladium”, or “Next Generation Secure Computing Base”. The Trusted Computing Group (TCG) was announced as the successor of TCPA, with the partnership of big players in the PC ecosystem: Intel, IBM, HP, AMD and

Microsoft. TPM can today be considered a global effort to bring trusted computing, especially a hardware root of trust, into the personal computing ecosystem

The first specification of the TPM was in 2001, currently there is a draft version of the TPM 2.0 specification [?]. TPM can be considered as a stand-alone discrete chips which provides a set of security functions for the operating system and its applications. The TPM interface is a set of functional APIs for security services carried out in isolation, i.e., inside a Trusted Computing Base (TCB). TCG specifies TPM requirements for desktop PC motherboards plug-ins and recently releases a specification of trusted module suitable to embedded device, the Mobile Trusted Module [?]. For resource-constrained devices, it is considered a software implementation of security functionalities without the adoption of a dedicated hardware component. Furthermore, the MTM specification adds some mobile specific functionalities and declares (mandatory) TPM features optional in order to minimize the footprint of the module. In addition, TCG proposes trusted computing specification for several applications, as for automotive [?] or eHealth [?]. In [?] *a trusted entity* can be defined as: *an entity which always behaves in the expected manner for the intended purpose.*

Figure 1.2.4 shows TPM structure as presented in 1.2 TPM Specification [?] where the main components are:

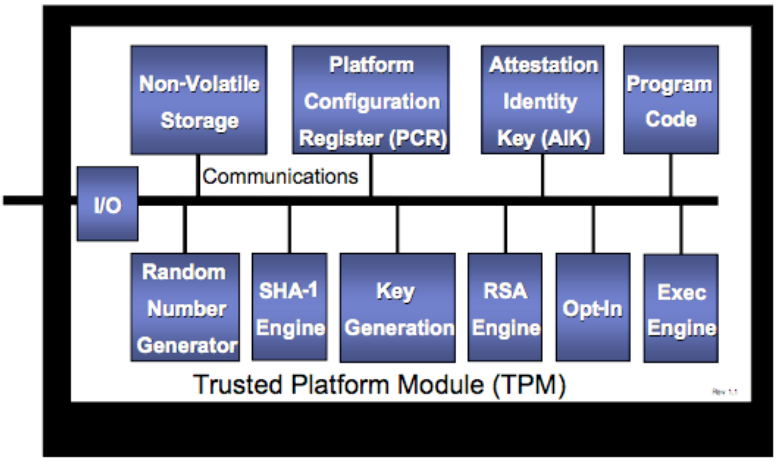


Figura 1.2.4: TPM component structure

- Input/Output (I/O): This component manages information flow over the communication bus. It routes messages to appropriate component and vice versa, when is necessarily adapting message encoding/decoding them;

- Non-Volatile Storage: A component which stores asymmetric RSA [**?**] keys used only inside TPM;

  - Endorsement Key (EK): an TPM unique identifier embedded during manufacturing phase.

  - Storage Root Key (SRK): To reduce the amount of non-volatile memory needed inside the TPM, only one key, namely the SRK, needs to be permanently stored inside the TPM. Other keys maintained by the TPM can be "wrapped" (encrypted) under the SRK or by another storage key that is already maintained by the TPM. These wrapped keys are maintained outside the TPM by the TSS, which typically stores the keys on hard disk. This allows the TPM to maintain a virtually unlimited.

- Platform Configuration Registers (PCRs): a trusted memory area in which actual configuration of platform are stored. The information of platform consists in 160-bit hash values. PCRs can be realized by beans of either volatile or non-volatile storage.

- Program Code: The firmware for measuring platform devices.

- Secure Hash Algorithm (SHA-1) [**?**]: Generates message, that are digest for the PCRs that is the response of an Hash function.

- True Random Number Generator(TRNG): This components provides, random number for key generations or authentication operations.

- RSA Key Generation: TCG standardizes the use of RSA algorithm in TPM modules. This component creates signing key (EK) and storage keys (SRK).

- RSA Engine: This component is used for signing with EK and encryption/decryption of the data with SRK.
- Opt-In: This component implements TCG policy requirements for TPM modules shipping and it determines the possibility of using or not a certain module by the customer.
- Execution Engine: a component which runs program code.
- Hashed Message Authentication Codes (HMAC): a component which checks correctness and integrity of a received message for authentication implementing signature mechanism.

TPM implementations in commercial devices In spite of this description, TCG does not specify a technology infrastructure to fully utilize this architecture. However, TPM is implemented by several vendors, ATMEL manufactures TPM devices compliant to TCG specification while Infineon provdes both TPM chips and software deliveres as OEM version in new computers. Acer, Wipro, Asus, Elitegroup, Dell, Inc., Gigabyte Technology, IBM, LG, Fujitsu, HP, Lenovo, MSI, Panasonic, Samsung, Supermicro, Sony, Eurocom Corporation, and Toshiba provide TPM integration on their devices and Google includes TPMs in their Chromebook Line-up [11]. Microsoft has announced that from January 1, 2015 all computers will have to be equipped with a TPM 2.0 module in order to pass Windows 8.1 hardware certification.[24] However, in a December 2014 review of the Windows Certification Program this requirement was changed to optional. However, TPM 2.0 is required for connected standby systems.

The taiwanese manufacturer MSI, lauches its tablet Windpad 110W that is composed of AMD CPU and Infineon Security Platform TPM. Default settings disabling the component but with pre-installed software it is allowed enabling and using it [12].

[11] http://chrome.blogspot.se/2011/07/chromebook-security-browsing-more.html
[12] http://www.msi.com/product/windpad/WindPad-110W.html#hero-overview

MTM specification does not consider an external dedicated trusted module chip, for this reason, for the field of mobile devices the common approach for building a trusted area on the devices relies on software implementation in secure areas of the memory. For example, TrustKernel's T6 secure operating system simulates TPM functionalities for mobile devices using the ARM TrustZone technology [13]

Trusted Module Remote Attestation

Integrity measurement During the description of the TPM components, has been presented that the role of the PCR for the maintenance of platform integrity measurement: this term as definite by TCG is the *the process of obtaining metrics of platform characteristics that affect the integrity (trustworthiness) of a platform, and putting digests of those metrics in shielded locations (called PCRs)"*. Hence, PCRs allow to store measurements of the platform state which consists in a cryptographic hashes of all software components loaded during the boot process. The Core Root of Trusted of Measurement (CRTM) , the first code executed by the platform, measure (compute hashes) the code and extend the first PCR register with this measurements and next, another executing code, i.e. the BIOS measure the binary image of the bootloader before transferring execution control to it, storing measured data into another PCR. Therefore, PCRs represented an accumulated measurement of the history of all the code that has executed from the power-up of the platform. CRTM is considered as a *root-of-trust*, a component that must always behave in the expected manner, because its misbehaviour cannot be detected. CRTM together with Root of Trust for Storage (RTS), a computing engine capable of maintaining an accurate summary of values of integrity digests and the sequence of digests , and Root of Trust for Reporting (RTR), a computing engine capable to reliably reporting information held by the RTS, are the root-of-trusted required to build a Trusted Platform.

[13] https://www.trustkernel.com/

Attestation with Privacy CA The roots of trusted described above require certification useful to demonstrate that the platform can be considered trusted. The trusted platform stores keys and other security data object into Protected Location within its own architecture and for this reason it must guarantee its secure nature to build a root-of-trust allowing the building of reliable services for both the user and the provider. In particular, integrity measurement can be used to ensure platform trustworthiness by logging changes to platform state: it records logged entries in PCR for validation. Some certifications are required to establish trust, i.e. for demonstrating that the key creation and protection procedure meets some security criteria. Some keys embedded into TPM's non-volatile memory requires a valid certification to attest their trustworthiness. Hence, in [**?**], TCG describes how the TPM can provide a remote attestation of the complete platform configuration relying on a Privacy Certificate Authority. In particular, with Trusted Platform Module (TPM), a device can remotely *attest* its trustworthy nature and prove to a remote-third party verifier that the platform has a specific configuration. The first attestation method presented by TCG relied on a *Privacy Certification Authority* [**?**], or Privacy CA, a trusted third party in charge of guaranteeing trust among several parties. The platform identity is handled adopting the Attestation Identity Key (AIK) which is created by means of a key certification protocol between user and Privacy CA. TCG recommends to store AIK in a persistent internal storage outside TPM where its purpose is providing user privacy when communicating with different sources. T he AIK adoption is realized to avoid the use of EK with communication with other entities: since the EK is uniquely identifies the TPM, it is a sensitive data that should be shared outside its secure perimeter only when strictly required . An EK theft could lead to attest as trusted a malicious entity that is not. The AIK credential is a certification containing the AIK public key which proves that the corresponding private key is bound to a genuine TPM. This proof is guaranteed by a signature on the credential created by the Privacy CA. The AIK key pair generation requires se-

veral step: (i) the user send a request to Privacy CA together with the endorsement credential, (ii) if the endorsement credentials verification succeeds the Privacy CA creates and signs AIK credentials encrypting them by means on public endorsement key of the platform, in this way the AIK credentials can only be used by platform who required them. The AIK can be used to sign PCR registers contest and for attesting platform trustworthiness: a third party which is aware of public AIK credentialn can verify the platform identity requesting an integrity report signed the the private part of AIK.

This approach suffers from two main issues: the Privacy CA must actively participate to each attestation, becoming a bottleneck, and if the Privacy CA establishes a malicious relationship with a third party, the TPM privacy may be compromised. This can happen because each third party entity can obtains identity information about the specific platform when communicates with it: some workaround on the adoption on Applications Identity Keys are proposed in literature, but, any pseudonym used to communicate permits to trace the platform.

Direct Anonymous Attestation Starting from 1.2 specification [**?**], TCG proposed an additional attestation mechanism, the *Direct Anonymous Attestation* (DAA), introduced by Brickell, Camenish, and Chen [**?**] to overcome the drawbacks of the previous attestation mechanism. The attestation approach based on Privacy CA establishing a trusted relation among parties relying a trusted part, conversely, DAA allows to set up a trusted relatioship among parties without the mandatory participation of a trusted one and respecting the user's privacy. DAA is based on group signature [**?**], enabling users to authenticate their participation in a group of trusted members without disclosing their own identity. Later, several DAA-based schemes were suggested in the literature.

The DAA scheme involves three players, namely: the Issuer, (TTP), the Device and the Verifier (AS). The DAA protocol requires the Join phase: the Issuer grants

credentials allowing Devices to generate anonymous signatures. Once the Device is able to anonymously certificate its own platform, the Sign phase can take place: messages are signed by the Device and sent to the Verifier using the DAA credentials. During the sign verification, the Device privacy is guaranteed as only the Issuer knows its identity. The Verifier checks if the signature belongs to the DAA group, and consequently, ensures that it came from a trustworthy Device. In order to face the real-word scenario in which TM can be compromised, DAA consider the credential revoking to rogue TM.

The DAA firstly proposed in [**?**] relies on RSA primitives to construct the anonymous group signature. However, in order to increase computing efficiency and reduce the length of the involved group signature, a DAA scheme based on ECC (i.e. ECC-DAA) and bilinear maps is proposed in [**?**]. In particular, the research efforts have been focused on balancing the computational load between hosts, an high-end platform, such as a system based on x86/x64 processors, and an embedded resource-constrained TPM. In order to limit the computation load on the TPM, a few alternative ECC-DAA schemes were proposed [**?**, **?**]. Indeed, DAA schemes can be also adapted to resource constrained devices as proposed by Ge et al. [**?**]. In the last TPM specification, the 2.0 draft [**?**], the TCG includes multiple ECC-DAA schemes. A comprehensive description of DAA implementation details and its steps which leads to establish an anonymous trusted attestation is presented in [INSERIRE RIFERIMENTO APPENDICE].

1.2.2 Research Efforts on Trusted Platform into Embedded Systems

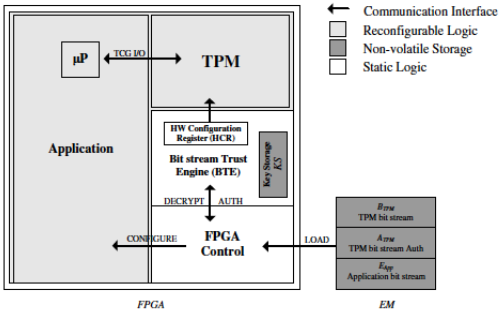


Figura 1.2.5: FPGA Trusted Architecture proposed by Eisenbarth et al

In order consider a certain component as trusted also the boot process and the loading of the software will be. The chain of trust is implemented by inserting an additional component, the *Core Root of Trust for Measurement*, that into PC systems is usually the BIOS. Its task is to check the bootloader and load a digest into PCR component presented above.

In [**?**] a TPM in a reconfigurable FPGA is proposed. Eisenbarth *ed al.* introduces a BitStream Trust Engine which permits loading, decryption and authetication bitstream in order to include configuration bitstream into chain of trust. Proposed architecture is shown in figure 1.2.5.

BTE provides a protected and non-volatile key storage (KS) and it, together with an Access Control Logic (ACL) have to build a trusted zone. ACL may be implemented as a non-volatile memory that using BTE credential decide to lock or not a bitstream syntethization. If all the possibile checks are made successfully bitstream is loaded, decrypted and autheticated.

Eisenbarth specifies that FPGA architecture needs some modification in order to support trusted computing and this is not possible to do in the most of FPGA family. Moreover in [**?**] is explained that complete TPM integration into a FPGA is not possibile. The main reason is the non-permament nature of this device. To achieve an FPGA-based TPM, modified configuration is required.

In literature there are various proposals and below some of there are shown. Besides some general proposals on FPGA, solution PUF-based are presented too.

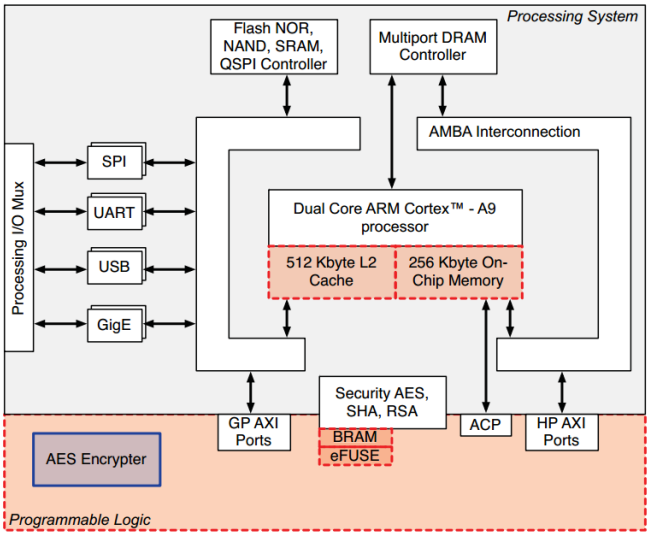


Figura 1.2.6: Xilinx Zynq-7000 Architecture

In [?] a TPM implementation in FPGA technology is made. As explained before, TPM needs a secure hardware perimeter and FPGA could not satify this requirement alone. So a Xilinx Zynq 7000 FPGA/SoC family key takes care of generating/storing keys and of cryptographic functions. Zynq includes: Artix-7 FPGA and an ARM Cortex-A9 processor with several BUSes and I/O systems.

The processing systems is static, not sythetized into FPGA. Figure 1.2.6 shows Zynq internal architecture that natively supports secure system deployment: the orange part indicates the mentioned secure hardware area in which keys are stored and decrypted. This components build a chain of trust from hardware to software.

A secure boot procedure is the root of trust. When ARM processor starts, it executes code into ROM: this code is provided by Xilinx and is unmodificable. The chip provides two secure storage services useful to store master key: BBRAM and eFUSE. During start operatons processor verify that the key in eFUSE or BBRAM matches with key into Boot header, if check fail the device transits into a secure lockdown state.

Then, the First Stage Boot Loader (FSBL) is parsed to retrieve boot-code location and actual boot configuration. The storing operation can be encrypted or not, the former case needs decryption operation with AES/SHA modules before loading, the latter do not.

FSBL loads bitstream which configure programmable logic and the user application. All this mechanism together with TPM guarantees a chain of trust, shown in figure 1.2.7.

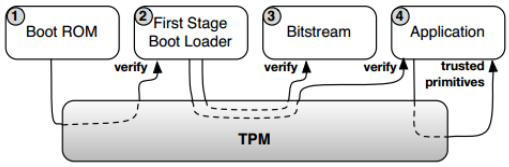


Figura 1.2.7: Xilinx Zynq-7000 Chain of Trust with TPM

Thus, the proposal in [**?**] is a valid one, but FPGA combination with a Processor System does not exploit PUF proprieties.

Glas *et al* propose a system design built using a trusted platform on reconfigurable hardware. They start from previus Glas work [**?**]. This appoach mantains flexibility advantage of reconfigurablity and at same time respecting TC requirements. In particular Glas focuses his work on Xilinx Devices that offer partial bitstream configuration. Boot operation is different from case of TPM into processing system because first hardware configuration must be loaded. The main issue is to build a trusted boot system that loads a configuration bitstream containing TPM components. In the proposal, Hardware Configuration Registers (HCRs) are presented, these registers work together with PCRs presented into TPM basic structure. Proposed design is composed of FPGA architecture with an attached *Trust Block* and a modified JTAG interfacing ($\mu JTAG$).

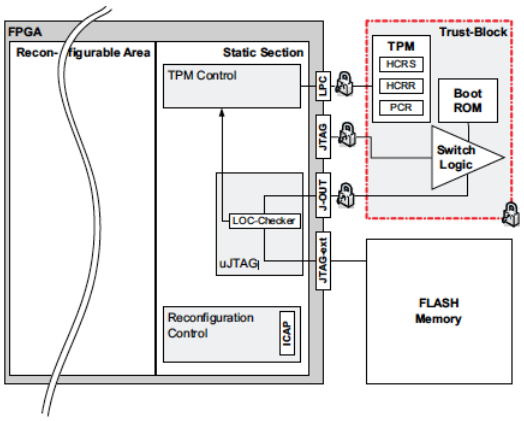


Figura 1.2.8: TPM on FPGA with Trust Block

Trust-Block cointains security relevant parts of the system and it have to be implemented with an external hardware component. Trust-Block interacts with FPGA and its interface must be reliable. Figure 1.2.8 shows TPM architecture: LPC is the Low Pin Count bus interface and connects TPM to FPGA.

Into Trust-Block, *secure BootROM* contains the initial bitstream for the device, it is assumed that it cannot be altered after its storing. Moreover, it has been previously validated and certified. Furthemore *integrated switch logic* allows configuration during power-up only from BootROM.

As shown in figure 1.2.8 there is a Static Section and a Reconfigurable Area upon TPM on FPGA. The former contains the necessary components to build trusted platform that are: TPM Control Block, FPGA-readout functionality and the uJTAG implementation. The proposed JTAG implementation provides the user with an external accessible programming port. it is called JTAG-ext. All data coming from this port are scanned and a targeted reconfigurable area of FPGA is detected. System operations follow:

1. At Power-up the FPGA is configured with configuration bitstream from BootROM (acting as CRTM).

2. HW configuration of both the static and reconfigurable part is read out, stored and hashed into HCRS and HCRR respectively. The same for the software

part as specified by TCG.

3. Using uJTAG port a partial configuration may be made. All incoming bitstreams are monitored: if an illegal area of FPGA is targeted the TPM is immediately disabled.

4. When trust authentication is required the TPM sends the requested contents of its secure registers, HCRR, HCRS and appropriate PCRs signed with a secret key.

In [?] the work shown above is extended with additional and more detailed specifications on the architecture and structure of the components for the static area and the Trust-Block. Microblaze processor plays the role of TPM Control together with a SHA-1 engine and LOC checker module: only Static Section is been implemented. This synthesis occupies 50% of Virtex-II Pro area.
Choi shows a PUF-based TPM chip [?] (figure 1.2.9) . This architecture do not use FPGA. The proposed scheme uses PUFs to generate keys used by RSA/AES module.

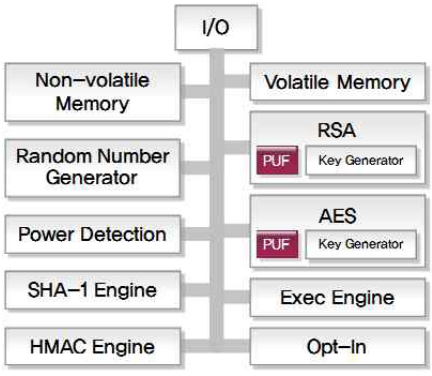


Figura 1.2.9: Choi's TPM architecture using PUF

So far several TPM implementation are presented, [?] is described an hybrid one that combines previous approaches.

TinyTPM

1.3 An example: Tiny TPM

Feller et al. proposed a cryptographic model called TinyTPM which enforces trustworthy operation and IP Core protection for embedded systems. TinyTPM can be synthesized on FPGA achieving two main operation: attestation of embedded system state and IP Core protection by providing authenticated and encrypted update procedures for FPGAs. TinyTPM has been designed to require a low quantity of resources well-adapting to the field of low cost FPGA-based embedded system.

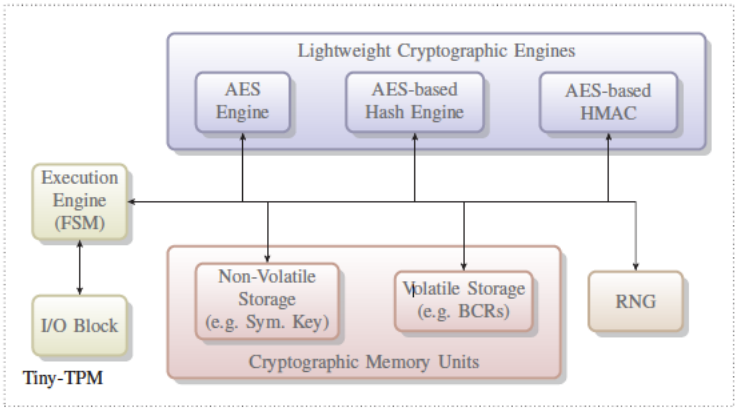


Figura 1.3.1: TinyTPM Architecture

TinyTPM follows canonical TPM architecture but to save an resources allocated uses cryptographic engines for encryption/decryption as well as for hash and HMAC computation, which are all based on a single block cipher, AES. This choice makes TinyTPM a lightweight module in comparison to a conventional TPM guaranteeing same security efforts (forse un po troppo generico). Into TinyTPM architecture Bitstream Configuration Register replace the Platform Configuration Register playing the same role with the difference that them contains integrity measurements (hashed value) of dynamically loaded bitstreams. An additional BCR is available to record the initial bitstream configuration of the FPGA considering both, static and dynamic logic.

This solution provides keys stored into NVM. NVM is used alto to store bitstreams

for partial reconfiguration. Key management has been kept deliberately simple and only one key is used for the protection of all IP Cores. Real world applications require individual keys for each IP Core. TinyTPM makes integrity measurements with hash computation, storing calculated values into BCRs. Attestation is made using hashed value of IPCores stored into BCRs issuing these value to a challenger that want to verify system trust.

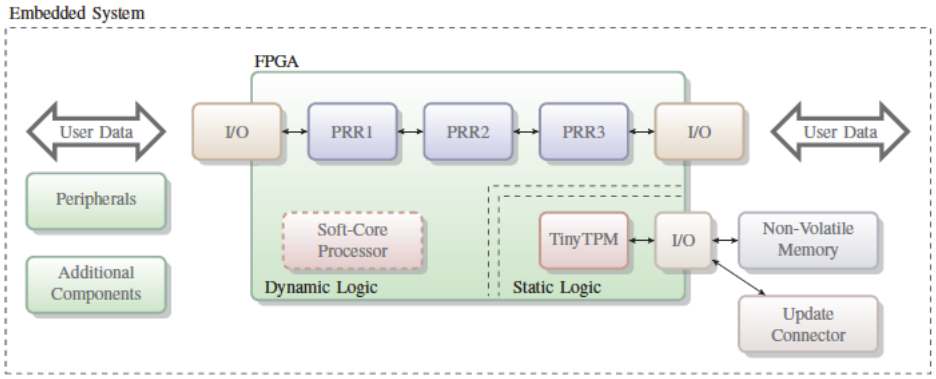


Figura 1.3.2: Complete Architecture combined with TinyTPM

TinyTPM bootstrap is simplificated respect canonical TPM requirements: a the initial bitstream is loaded into static part of the FPGA's logic, this initial state is called TinyTPM INIT STATE. Then, the FPGA is configured and the trusted module starts operation by executing a self test first. The integrity of the initially loaded bitstream is verified by comparison to reference value stored into NVM, thus building the root of trust. The NVM is accessible if and only if the initial bitstream is authentic. The result integrity measurement is then stored in BCR_0 for later reference. If this test goes well, the state of platform moves to INIT Dynamic Logic state, else the state return to TinyTPM INIT. [Mostrare figura e stati di TinyTPM]. In INIT Dynamic Logic State, all the partial bitstream are loaded. For each PRR_n in the dynamic logic, the result of integrity measurement is stored in an individual BCR_n. Using monotonic counter this architecture can prevent replay attacks [inserire riferimento 8 di TinyTPM ed aggiungere qualche parola a riguardo].Once the

static and the dynamic logic section are measured and configured, the embedded system is ready to accept command and work normally using the TinyTPM functionalities.

In reference to the implementation during self authentication test during TPM boot, a set of commands is supplied to the ICAP interface to read the configuration of the FPGA. The output is grouped in 128 bit blocks and hashed using AES-based hash core. The result is then stored in BCR_0 for later reference

. IP Core protection technique provided by TinyTPM architecture is widely described in the next section.

TinyTPM architecture is secure against an attacker able to monitor che communication between the authentic NVM and the TinyTPM residing n the FPGA because it is not able to gain any knowledge about the keys stored into NVM. Tiny TPM structure is presented before (inserire riferimento nel caso). Feller proposed an update protocol based on SKAP (Session Key Authorization Protocol) communication protocol designed by Chen [17TinyTPM]. This approach relies on public key cryptography to avoid both, TPM impersonaton and weak authorization data attacks. Before starting update process, a secure communication channel is established.

The first step is an attestation of the platform request by the update server, TinyTPM response consist in a digitally signed set of BCR values to authenticate both itself and the system. After attestation, the update can be done. Update Manager sends PRMs(Partial Reconfiguration Module) to embedded system, where the data sent is authenticated and decrypted by the corresponding engines of the TinyTPM. As explained before, TinyTPM structure relies only on AES primitive, so the AES-based hash engine is used for computing hash values of the updated data. The computed hash values are then stored into BCRs of TinyTPM. Remember that an update data is stored in the NVM before loading it into the dynamic logic section of the FPGA.

In reference to the implementation, *TPM_Quote* operation realizes trustworthy re-

porting of the system state and creates a digital signature of the stored BCR values. For this reason, the HMAC for all BCR values is computed sequentially and sent to the update server. Then update server, after a proper checking,sends updated bitstream in blocks of 128 bit size, because of the same blocks size of the utilized HMAC function. The number of the blocks is reported to TinyTPM within the first block. Upon reception the bitstream block is passed to HMAC core and additionally stored into BRA. Along with the last bitstream block the HMAC mode is changed to finalize the computation. Its result is compared to the final block containing the HMAC result computed by the update server. After successfully checking the HMAC, the resulting values is extended into the BCR and the bitstream is configured via the internal configuration access port (ICAP). (Il tutto è anche criptato dall'update server e decriptato nel sistema utente sempre usando la stessa chiave ovviamente eh!)

PUF-based Trusted Execution Enviroment In [**?**] a PUF-based TEE realization is proposed. A key stored in secureROM is the initial root-of-trust, where secureROM is inside SoC. Code included into TEE must be present at manufacturing time. This restrinction limits extension in all applications. The aim of this solution is to allow flexibility and possibility of alteration of the TEE after the manufacturing process.

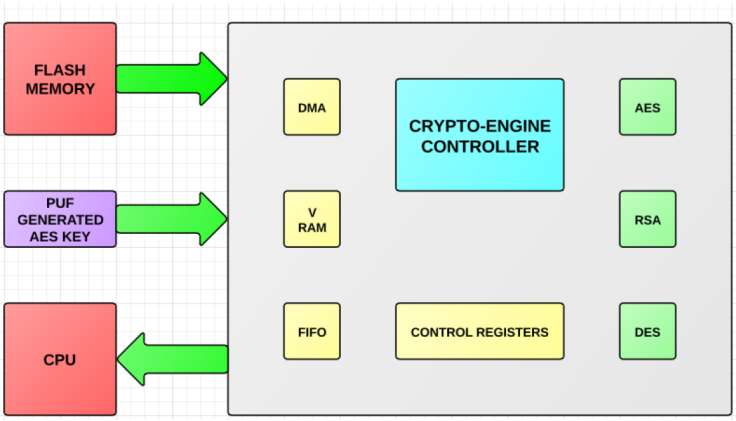


Figura 1.3.3: PUF Supported Crypto-engine

TEE is typically stored in external non-volatile memory, if memory is not properly protected, it is subject to attack. Standard encryption helps to protect the TEE from this attacks. As explained before TTE cannot be customize for a specific implementation after manufacturing time. So in order to address this concern Areno *et al* proposed the incorporation of a PUF inside Crypto-Engine of SoC. PUF will be used to generate secret AES Key giving additional security to TEE components. This approach also provides a means of modifying the TEE without requiring any changes to underlying hardware or software.

Any time an element of the TEE can be added/removed generating a new measurement encrypted using PUF key by crypto-engine. Because all measurements are encrypted by the PUF generated key, whose value is unknown to any entity and as such, any accidental or malicious alteration of this information is currently impossible. Figure 1.3.3 shows modification into SoC architecture.

This architecture is composed of:

- a controller which handles the flow of information through the engine,
- a set of control registers often memory-mapped by CPU,
- several cryptographic processor for specific operation such as AES, RSA or DES.

A small volatile RAM stores key generated by PUF: this key should not be accessible by any external element to blocks of cryptographic calculation.

This implementation can be useful for boot process. During boot time memory stored into secureROM is loaded and executed. Using PUF-based key, the secureROM content can be decrypted. This would provide the location, size and measurement of the TEE on the flash device. So TEE can be loaded into system. After the validation the system can run performing its operations.

TEE expansion with the proposed architecture does not require any additional hardware components. The primary purpose in supporting such addition is based upon

the need of entities such as corporate IT departments to have a presence on mobile devices without emparing the usability of the device. Moreover, most corporate environment requires a device to be completely re-flashed with their own version of the operating system. An application scenario is proposed in [**?**] where a new employer install trusted-application on its device and when he leaves the company TEE component on the device are removed permitting to personal employer device to continue its normal functionalities.
A non-detailed architecture implementation on Xilinx FPGA is described of in [**?**].

ARM TrustZone Technology

ARM®TrustZone®is a system approach that guarantees security for a wide range of clients including server, tablets, wearable devices and enterprise systems. This technology is integrated into Cortex®-A processor family and is extended throughout the system via AMBA®-AXI bus to specific TrustZone System IPBlocks [14].
TrustZone permits to establishing trust in ARM-based platforms. TPMs are designed as fixed-function devices with a predefined and specified feature set while TrustZone represented a much more flexible approach by leveraging the CPU as a freely programmable trusted platform module. To do that, ARM introduced a special CPU mode called *secure mode* in addition to the *regular normal mode*. The notions of a "secure world" and a "normal world" follows previuos definition 1.3. Indeed ARM TrustZone is ARM own TEE implementation.
The TrustZone allows a processor to run application either in secure or non-secure world. It divide the single core in two virtual cores, one for each world. Switching from non-secure world to secure world is made by a Monitor. The possibilieties of a normal world processor to enter the secure world are strongly limitate to guarantee the resilency of TrustZone. Indeed, all the instructions that allow are exception. The assembly *smc (Secure Mode Call)* istruction offer this switching possibility.

[14] http://www.arm.com/products/processors/technologies/trustzone/index.php

Another important hardware change is the addition of NS-bit, a 33rd bit added to each internal address and data element that labels it as belongin to either the secure or non secure world. Figure 1.3.4 shows a general TrustZone architeture scheme.

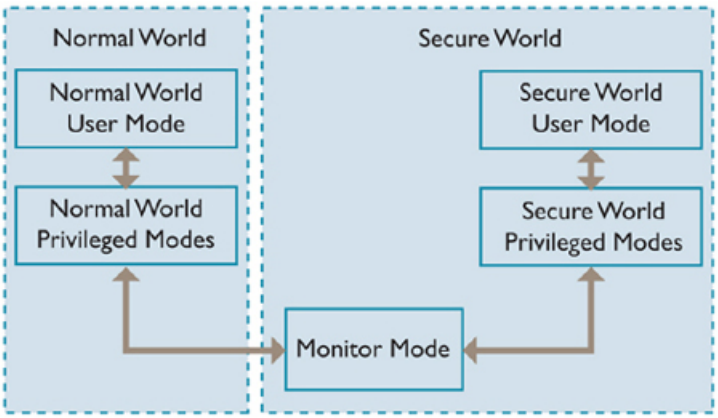


Figura 1.3.4: TrustZone Zone Structucture

Inside a TrustZone-enabled processor there are two virtual MMU: one for every virtual processor. In this way, trusted area memory is logically detached from normal one. Thus there are two different set of andresses translation tables, making them indipendent. The tables uses a NS-field, this field is used by Secure Processor to verify that it belong to secure area. Normal processor ignores this field. However, Secure Processor can access to resources of both the worlds. Instead, caches supports both the worlds.

As presented before, ARM supplies the AMBA3 AXI TrustZone-enabled bus. This components manages the hardware logic to check that normal word cannot access to secret world resources. TrustZone implements three sets of interrupting vectors one for secure world: one for the non secure world and another one for monitor mode.

TrustZone, as other TEE systems, requires a Chain of Trust for its funcionalities. In order to create a Root of Trust, a key called *PUK* is stored into an iROM, a processor-internal read-only memory. Such key is used to verify the integrity of the next booting phase, which will use the key to validate its successor, so the name Root of Trust.

A specific software have to run on TrustZone to exploit its specific hardware architecture. The Secure World has full control of hardware but secure process operating on it cannot directlu interact with non-trusted application. ARM released a standard API for the TrustZone in to simplify use for application developers.

Several trusted systems for mobile devices have been implemented leveraging ARM TrustZone techlogy, like Nokia's Onboard Credentials [**?**], [**?**] , Sierraware0s SierraTEE [15], this is used next.

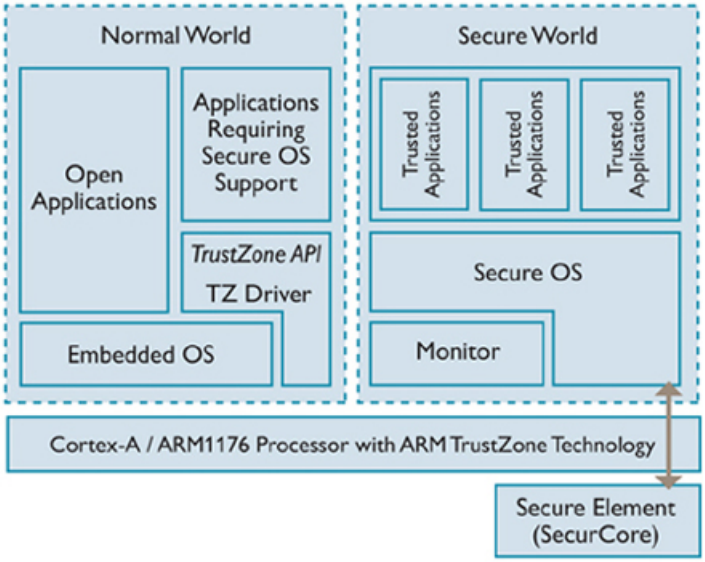


Figura 1.3.5: TrustZone Software Architecture

SRAM PUF-based Root of Trust for ARM TrustZone Zhao *et al* present a method to provide Root of Trust for ARM TrustZone using SRAM PUFs [**?**]. Starting from TEE and TrustZone architecture shown before, [**?**] implements a software entity, the building block. Building block provides the foundation for the root of trust: secure key storage and truly random source. Based on this building block, a root of trust is built to seal or unseal primitives for secure services running in the TEE. TPM is software-implemented.

Zhao proposal is based on this idea: a CPU with TrustZone security only provides an "isolated" execution environment, but not a *trusted* one since it cannot attest to

[15] http://www.sierraware.com

the user or an external verifier that the software running inside an untampered and trustworthy environment.

ARM does not specify explicitly the root of trust for TrustZone, so availability of an unique device key, as PUK key described before, is assumed. This key is accessible only inside the secure world of TrustZone and it is probably stored using Batter-back RAM or eFuse technologies, as already shown in [**?**]. In Zhao work this method is avoided and his own building block provides the foundation of a root of trust like:

- A primary seed extracted by a fuzzy extractor 1.1.3 form on-chip SRAM start-up values. This primary seed can be use to derive a device key,
- A truly random seed extracted from the noise into on-chip SRAM start-up values. This random seed is used to build a RNG for the secure OS.

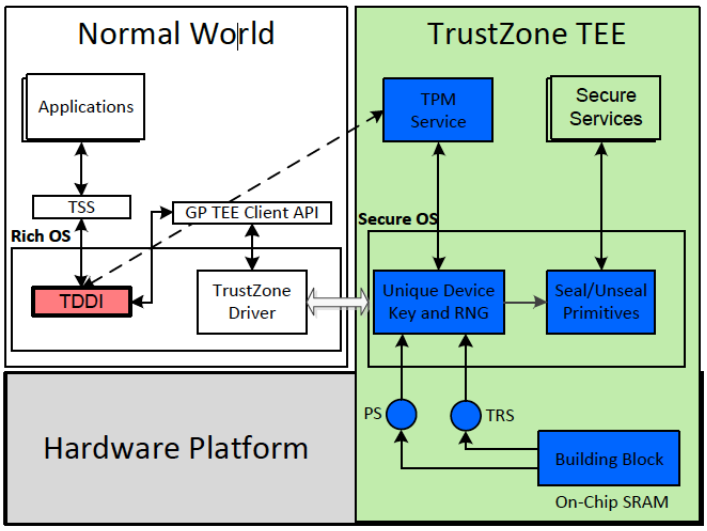


Figura 1.3.6: TrustZone Software Architecture

This block also provides secure boot of the secure OS and secure services running inside the secure world of TrustZone. This feature is mandatory for a TrustZone. The building block is stored into secure on-chip SRAM achieving high security level, indeed this memory can resist against physical attacks on the SoC. The device key is used to provide seal and unseal primitives for secure services, which ensure that only specified secure service and platform can access user data. The seal/unseal

primitives implicitly attest to the user the state of the platform and the secure service. Building Block operations consists in fuzzy extractor *reproduce* functionality, it is described before in 1.1.3. Building Block provides primary seed (PS) and truly random seed (TRS) as shown in figure 1.3.6.

A developer can provide trusted computing functionalities with no need for additional security hardware resources so the build of root of trust is flexible and trustworthy. In order to facilitate the use of the TPM service, a kernel module called TDDI simulating a hardware TPM driver interface is provided in the normal world.

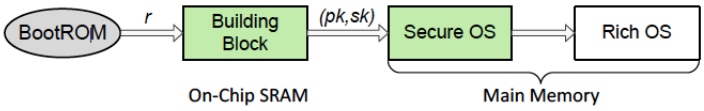


Figura 1.3.7: Building block-based Chain of Trust

The simple seal and unseal primitives bind secure data with both the platform and the particual secure service throught cryptographic encryption or hashing. User data are encrypted with device key and this key is derived from the secure service. Only the device running a legitimate secure service with a right *device key* can obtain the user data. Another situation is the offering of a secure service that permits to store sensitive data.

TPM service, which is realized in software, provides rich trusted computing functions for an OS. These functionalities can be used to bootstrap a trusted mobile OS and further help the normal world run in a trusted state. The TDDI helpt to use TPM services by simulating a TPM hardware driver interface.

In figure 1.3.7 is shown the chain of trust. A code designed for operation into secure world can be stored into insecure non-volatile storage so it is vulnerable to attacks from the normal world. However if a chain of trust is established from BootROM to the mobile OS, it is possible to protect integrity of the code running inside the secure world, such that mobile OS boot becomes trustworthy.

Analizing figure 1.3.7 when a device is powered up, ARM processor into secure mode

performs code into BootROM. This code is immutable and defined at manufacturing time, so it is implicitly trusted. BootROM first operation is to verify the integrity of Building Block, so the image of it is measured and using manufacturer public key is possibile to do it analyzing the signature. So, the BootROM reads the start-up value r of the SRAM on chip, inizialiging it and load the building block in it. If integrity check succeeds, BootROM transfer *r value* to building block and it runs building block in on-chip SRAM.

The role of buidling block is already known: it produces PS and TRS and then derive a symmetric key and unique device pair (pk,sk) from PS. Using this key pair image of Secure OS is verified: if check is successful the building block loads the image into secure memory region, else the system start up is stopped. When secure OS starts up, it initialized the services contained in the image, including TPM. Then it measures the image of normal OS and finally secure OS runs the normal OS.

A significant improvement that PUF-based key can achive into system is the *key update*. [**?**] proposes a key update protocol which the device owner can change his device key regularly. The protocol provides in its steps a communication between the device and the manufacturer, the latter selects another challenge from a set already predisposed during manufacturing.

At the end of protocol device receives new *Helper Data H'* to generate new primary seed, Secure OS encrypted with new *symmetric key k'* and a *certificate* with standard measurement values of the building block and secure OS. A drawback of this protocol is the weakness against downgrading attacks: an adversary can roll back the device key to an old one by copying previous helper data and encrypted blob of secure OS. Authers solve this issue with a secure monocounter.

In [**?**] an implementation is shown using Xilinx Zinq-7000 AP SoC with a 256 KB on-chip SRAM inizialized by BootROM when board is powerred on. SRAM PUF is made using Altera Cyclone II connected via Universal Asynchronous Receiver/Transmitter (UART) build using General Purpose I/O (GPIO). The normal world runs

Linux OS with kernel version 3.8, while the secure world runs Open Virtualization SierraTEE [16]. All the implemented Trusted Computing Base (TCB) is *small* with a size of 3.2K LOC.

1.4 Hardware IP Cores Remote Activation

[16] SierraTEE for ARM TrustZone provides a minimal secure kernel which can be run in parallel with a more fully featured high level OS as Linux, Android, BSD. It also provides driver for the Rich OS to communicate with the secure kernel