RoT: The Foundation of Security

Author:
PUFsecurity Corp.
R&D Engineer, Lawrence Liu

Introduction

Security is a topic that will only grow more popular in the future. Its importance is tied to how critical it will be to the success of future products, particularly as the rise of the Internet of Things (IoT) will bring about an exponential growth in the number of "things" connected to each other, and to the Web (see Figure 1, below). IoT devices are distributed, unsupervised, and physically exposed. Attackers can physically tamper with IoT devices which makes software-based security insufficient to protect IoT from fraud, tampering and other integrity and DDoS attacks. Computer hardware and firmware are perceived as more dependable and trustworthy than software, because software is susceptible to design and implementation flaws and not impervious to subversion by malicious code, while it is hard to intercept, tamper or break hardware security.

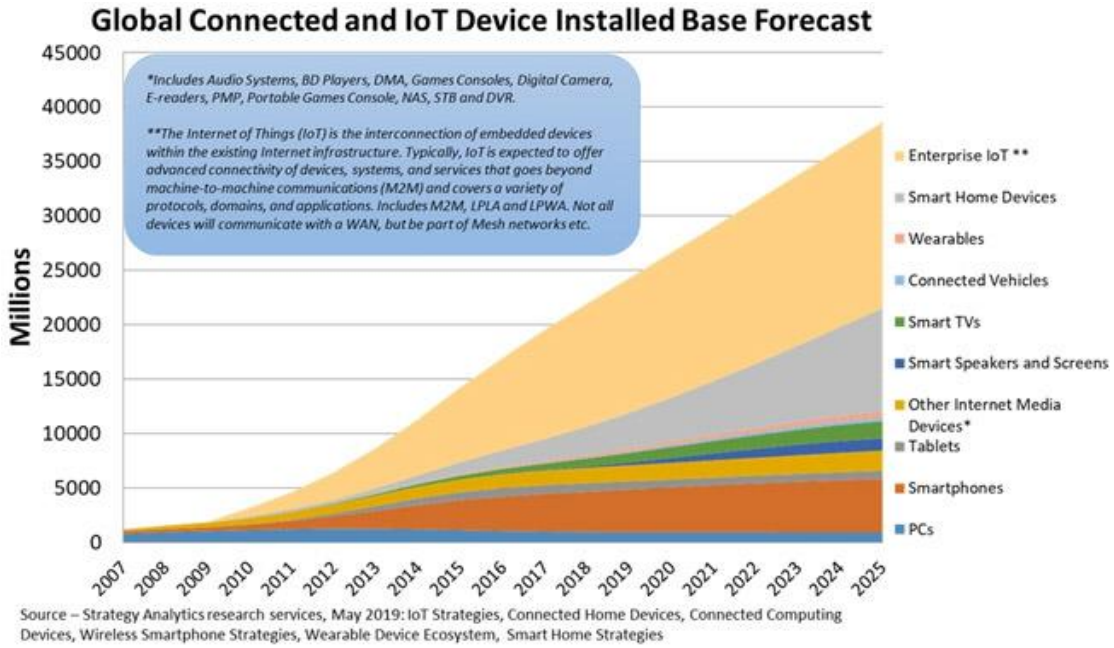

Figure 1: Projected growth of installed IoT devices [7]

A root of trust (RoT) is a set of functions that is always *trusted* by a system's OS such that it's the trust foundation on which all secure operations of a computing system depend. Containing the keys used for digital signing and verification, along with the cryptographic functions to enable the secure boot process, a RoT is an important security asset indeed. Providing trusted execution environment and embedding a RoT in hardware would provide a firm foundation for electronic systems security. The goal of this white paper is to provide an introductory primer to RoTs, ending with some guidance on choosing the right RoT as the trust anchor for a novel hardware based security architecture.

Definitions

The concept of a root of trust has been around for several years now. So as to build on the previous works of defining a RoT, perhaps it is best to start with how a RoT has already been defined.

The Trusted Computing Group, or TCG, is dedicated to the idea of secure computing. They have put forth the TPM (trusted platform module) standard as their solution to ensure computers operate as expected, and hackers have no way to gain access to secured systems. In their TPM standard [4], the TCG actually requires the inclusion of three RoTs, one each for performing the tasks of measurement (RTM), storage (RTS), and reporting (RTR). The TCG defines a RoT as [2]:

"A Root of Trust (RoT) is a component that performs one or more security-specific functions, such as measurement, storage, reporting, verification, and/or update. A RoT is trusted always to behave in the expected manner, because its misbehavior cannot be detected (such as by measurement) by attestation or observation."

Of interest in TCG's definition is the use of the word "component", because it allows a root of trust to be implemented in software, hardware, or firmware.
Similar in concept, but with a more rigidly defined implementation, the United Extensible Firmware Interface (UEFI) forum defines a root of trust as [3]:

"The root of trust is ideally based on a hardware-validated boot process to ensure the system can only be started using code from an immutable source. Since the anchor for the boot process is in hardware it cannot be updated or modified in any way…. When it starts, the root of trust derives its internal keys from supplied device identity inputs and executes self-tests and code validation for itself. If these tests pass, it can move on

to validate the first piece of code in the chain of trust."

Here of note is that a root of trust is defined to be done in hardware, and that its main purpose is to be the first part of a chain of trust. While other standards will leave the implementation of a RoT as open-ended, this white paper will follow UEFI's convention, so that here "RoT" is the same as a hardware root of trust, or HRoT. With "firmware" in the group's title, it is easy to see that having a secure boot process (perhaps using a chain of trust anchored with a RoT – see Figure 2, below) is the primary objective of the UEFI when defining a RoT. Note that the authorization for upstream blocks comes from downstream blocks; for example, in the figure below, the BIOS would need to be authorized by the RoT before execution control can be passed from the RoT to the BIOS.

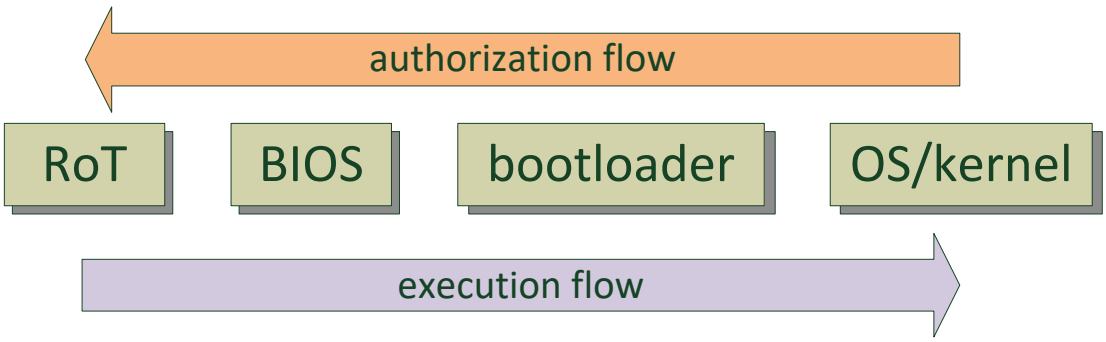

Figure 2: Example of a chain of trust (CoT)

GlobalPlatform's definition of RoT rounds out this collection of already defined standards. GlobalPlatform is another large collaboration of industry members (similar to TCG) to promote a set of standards with the similar goal of trusted and secure computing. It is no surprise that their definition of root of trust does not deviate much from the others so far [1]:

"A computing engine, code, and possibly data, all co-located on the same platform; provides security services (as discussed in Chapter 3). No ancestor entity is able to provide a trustable attestation (in Digest or other form) for the initial code and data state of the Root of Trust. Depending on the implementation, the Root of Trust is either a Bootstrapped or a Non-Bootstrapped Root of Trust."

GlobalPlatform differs from the others by introducing the concept of two types of RoT,

the bootstrapped and non-bootstrapped. Similar in concept to a chain of trust, where one component checks the authenticity of the next component before handing off control, a bootstrapped RoT is composed of several components (also called RoTs, by GlobalPlatform's definition), that execute start up code in a fixed sequence. Here GlobalPlatform introduces the ideas of an initial RoT (iRoT) and an extended RoT (eRoT), which together make up a bootstrapped RoT. But for purposes of simplicity, this paper will focus on just the basic configuration of RoT, also called a "non-bootstrapped RoT" by GP.

What is a RoT?

1) Verification (authentication) record of installed, trusted boot code
2) Execution consistency
3) Provisioning of one or more security service(s)
4) Establishing security from the "root" of a system

Taking the common elements from the previous section's definitions of a root of trust, we can distill the basic elements of a RoT into the above list. The first item is "trust" – that is, the system can depend on the RoT at all times, especially at boot up. Since there is no way to independently verify the authenticity of a RoT at boot up, then it needs to be implicitly trusted right at power/start up of a system. To do so, it is recommended that before installation of a RoT, that it is verified to be valid and authentic, and that a record of that verification is stored on the RoT itself, so that it may be re-verified during runtime, as needed.

During boot up, when starting from a trusted RoT, it is expected that the code/processes built into the RoT executes consistently every time, as specified by the system architect. The easiest way to do so would be to have the code written to an immutable storage element, such as a ROM or OTP, since once written, it cannot be changed. A more sophisticated RoT may allow rewrites of its boot code, but only through authorized agents, which requires a more complex control/access scheme than a basic RoT would have.

A RoT should provide at least one security service. TCG [2] defines these services as falling into one of five categories: measurement, storage, reporting, verification, or update. GlobalPlatform [1] expands upon TCG's list to include nine categories: authentication, confidentiality, identification, integrity, measurement, authorization, reporting, update, and verification. As one can see, depending on how many services

are supported, the label “RoT” actually covers a wide range of devices, from basic parts that only includes a secure storage element, to more complex modules that can be used as a standalone security processor.

The last item on the list refers back to the first word in RoT – “root” – a RoT sits at the very beginning of a system – it is the root, or anchor, upon which the security of the system rests. There is no other element that can verify/authorize/attest to the validity of the RoT – it is element zero of the system. Sitting at such an important position in a system, it is important that designers understand their needs for a RoT, and to find the right one that best fits those needs.

How does a RoT work?

Whereas TCG defines a possible five different security services that a RoT may provide, GlobalPlatform defines a total of nine services. In any case, whether a RoT is called upon to support one, or up to nine services, the number and type of services will provide a good idea of the inner workings of a RoT.

The type of service will dictate what cryptographic engine/module will need to be included in a RoT. GlobalPlatform [1] defines the following list of nine services:

1) Authentication
2) Confidentiality
3) Identification
4) Integrity
5) Measurement
6) Authorization
7) Reporting
8) Update
9) Verification

By comparing this list of services with the five primary functions of cryptography [5], one can begin to see how some of the services can be supported through one cryptographic engine, and how others will require the combined results of a group of modules working together:

1) Confidentiality
2) Authentication

3) Integrity
4) Non-repudiation
5) Key exchange

By breaking “key exchange” into the smaller tasks of key generation / agreement / transport, one can list out the possible cryptographic modules that would need to be included in a RoT, given a list of security services to be supported:

Crypto Functions	**Cryptographic Submodules**
Privacy/Confidentiality	AES, triple-DES, SM4 submodules (symmetric cipher)
Authentication	DSA or ECDSA using asymmetric cipher and hash submodules (digital signature)
Integrity	SHA-2, SHA-3, SM3 submodules (hashing)
Non-repudiation	Through the same algorithms for digital signature
Key Generation	Through KDF using a hash submodule, or a RNG
Key Agreement	ECC-implemented ECDH algorithm (shared secret)
Key Transport	RSA, ECC submodules (asymmetric cipher) or Key Wrapping

Table 1: Cryptographic functions a RoT can support

Once it is seen that one or more cryptographic modules are needed for a RoT, the other required pieces are easily understood. Two of the original four basic requirements of a RoT are always to be trusted, and always to run as expected, indicating the need for a secure storage element, immutable to unauthorized users. Finally, a control logic unit of some sort is needed to have all the parts of a RoT run in harmony, and the final picture of how a RoT works is revealed. A collection of cryptographic modules, secure storage, and a control logic unit to wrap it all together make up a RoT. Depending on the requirements of the endpoints at which the RoT is to be installed, the variations of RoTs are very diverse indeed. It would not be reasonable to expect a “one-size-fits-all” RoT solution to exist.

To organize our thinking about RoTs, they can be thought of as belonging to one of three different categories. The first category (call it type I) consists of those RoTs that only perform the core duty of protecting critical data (boot code or root key) – this data can always be trusted, and no other functions are supported by these types of

RoT (besides secure storage). The second category (call it type II) includes those RoTs that support more functions. The third category (call it type III) is very similar to those in the second, with the difference being in how the control logic is implemented.

Type II RoTs use a dedicated state machine – RoTs with this type of control have their function list fixed during design, with no flexibility to create new functions once the design has been taped out. Type III RoTs use a general-purpose CPU (such as one based on an ARM or RISC-V core) to implement the control logic. This type is more flexible, as changing the control logic is a matter of changing the software that the CPU runs. The tradeoff to using a CPU is the possible area penalty, as a CPU may have more functionality (and therefore logic gates) than is needed to control a RoT. There is a wide variety of CPU cores one can choose from, so users can choose which one is the best fit for their needs in terms of size, performance, and complexity. A custom (fixed SM) solution, on the other hand, will always have exactly what the RoT requires – no more, and no less. As long as users are satisfied that they don't need the flexibility that a CPU can offer, a fixed SM control may be a cheaper solution.

The three different types of RoTs are shown in the following table and figures:

Submodules / Types of RoT	**Type I**	**Type II**	**Type III**
1) Storage Element (e.g., ROM/embedded Flash/OTP)	v	v	v
2) Fixed State Machine control	v	v	
3) CPU-based control			v
4) External Bus Interface (e.g., AHB/AXI/TileLink)	v	v	v
5) Symmetric Cipher Module (e.g., AES/triple DES/SM4)		o	o
6) Asymmetric Cipher Module (e.g., RSA/ECC)		o	o
7) Hash Module (e.g., SHA-2/SHA-3/SM3)		o	o
8) RNG Module (e.g., DBRG/tRNG)		o	o
9) External Memory Interface (e.g., SRAM/DRAM/Flash)		o	o

Table 2: Three types of RoT

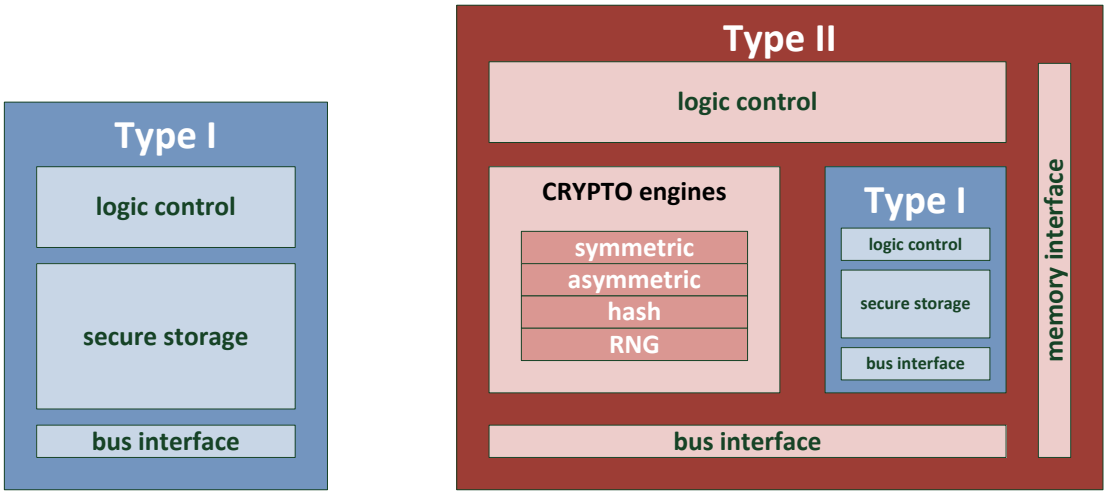

Figure 3: RoT, types I & II

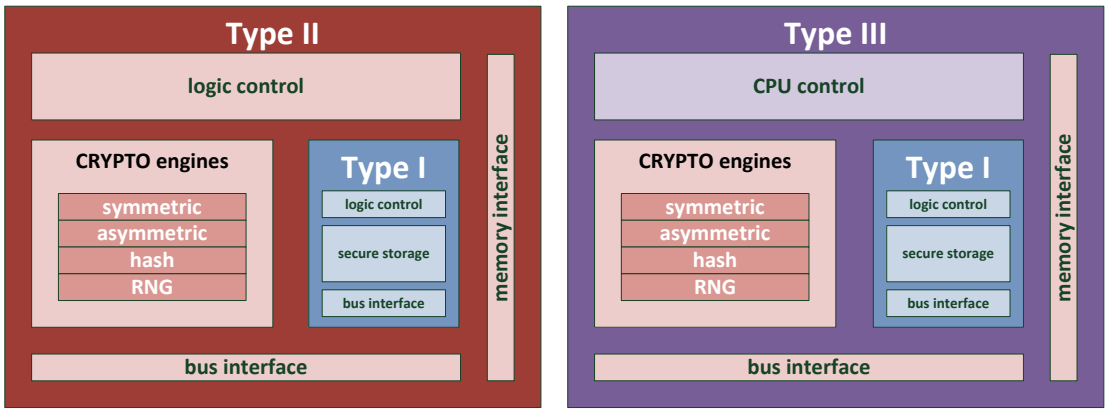

Figure 4: RoT, types II & III

What are the Advanced Requirements of a RoT?

1) Changeable boot code, with restrictions
2) Multiple component RoT
3) Public key cryptography for verification
4) Attack resistance
5) Flexible logic control

To recap, a root of trust is the completely trustworthy boot logic that sits at the “root” of a system. By starting up a system from the RoT, it guarantees the system will start at the same initial state that the designers originally intended for the system. That

sums up the basic functionality expected of a RoT. But since systems can differ widely from each other, based on such factors as usage case, protection profile, cost, and power constraints, some beyond RoTs (as in “beyond the basics” RoTs) could prove to be a better fit. Some examples of such advanced requirements are listed above, at the start of this section.

First is changeable RoT boot code. Over time, the functionality of an installed system may change. By allowing the boot code of a RoT to change in step with the changing requirements, such a RoT will be able to extend the lifetime of a system. Of course, having a mutable RoT will imply the need to add a protection/privilege scheme to ensure that only authorized agents can change the protected boot code of a RoT.

Second is splitting a complex RoT into several component pieces. Just as computer programs are modularized to allow for the compartmentalization of logic, so as to make complex programs less unwieldy, RoTs can also be broken up into smaller components. GlobalPlatform [1] refers to these pieces as iRoT and eRoTs, or initial RoT and extended (or expanded) RoTs. The entire collection of iRoT (always only one per system) and one or more eRoTs is still called a RoT, specifically a bootstrapped RoT, to use GlobalPlatform’s exact term.

Third is the inclusion of public key cryptography. Working together like a mini-CoT (chain of trust), this collection of iRoT and one or more eRoTs will either verify or re-check the verification of the following RoT component, starting from the iRoT. One of the more sophisticated ways of verification involves creating and checking digital signatures, and a public key cryptography engine plays a major role in digital signature algorithms.

Fourth is attack resistance. A basic RoT will work under ideal conditions, but a beyond RoT needs to guarantee that it will continue to work, even under attack. Attacks could be non-invasive, such as side-channel analysis (where the goal is to steal secure data from the RoT, such as the boot code, private keys or unique ID). Attacks could also be more direct, such as physical tampering attacks, one example being fault injection on I/O or power pins in an effort to make the RoT misbehave.

Fifth (and possibly not the last) is a flexible logic control. Flexible logic meaning logic that can be easily changed, which implies either software or firmware-based changes. A general-purpose CPU is the most flexible way to add this to a beyond RoT. Offerings from a commercial provider such as ARM or an open-source one such as RISC-V are just a few of the many available.

How to choose the right RoT?

1) How is the root key created and stored (along with boot code)?
2) Which crypto functions to support?
3) What kind of control logic to employ?
4) How to integrate RoT into design? (bus interface)
5) Attack resistance of the RoT?
6) Has the RoT been certified?

Root Key Generation and Storage

The first consideration is regarding the root key. Since the root key can either act as a unique identifier for the system, or as the basis from which all other keys are derived, it is important that the root key be well protected, as well as very difficult, if not impossible, to guess. To be well protected, the RoT needs to make sure the data path from root key generation to root key storage, as well as the storage location itself, is secure. The data path must be protected from eavesdroppers monitoring the transmission of the root key from its source to destination memory. If the root key is generated outside of the RoT and written to the RoT, then normally this will be done in a "clean room" inside a secured location where security (such as multiple levels of code access doors and human guards) is tight. Alternatively, the RoT could create its own root key through something like a physically unclonable function (PUF), entirely removing the need for a clean room and all the layers of security that entails. Still, this is assuming that the quality of the root key (its measure of entropy) is the same between a hardware secure module (HSM) injected root key, and one created by an on-chip PUF. There are many different qualities of PUF available, so it is up to users to decide the amount of entropy they need for their root key.

Besides protecting the data path of the root key, the root key also needs to be securely stored. Secure storage, in this case, implies that the data is not written as plaintext and that once written, the data should be unable to be modified. Again, the level of protection is up to the user – it could be as simple as an address/data scramble or as complex as a full symmetric cipher to encode the root key, keeping in mind that the cipher key needs to be separate from the root key. To prevent the root key (and the boot code) from being overwritten by a bad agent, it is recommended to use a ROM or OTP-type memory for storage purposes. If it is deemed that the RoT only needs to keep a root key safe, then a PUF would kill two birds with one stone, by acting as a source for the key, as well as its storage element. If the RoT needs to store other data besides the root key, then the user will have to consider the total storage space

required – an external memory (with its corresponding secure interface) will need to be supported by the RoT if the included memory on the RoT isn't large enough. Finally, to prevent the root key from being ascertained through physical examination of the internal memory array, the choice of anti-fuse based OTP memories may give more peace of mind over those based on eFuses.

Supported Cryptographic Functions

The next issue to examine is the included cryptographic sub-modules in the RoT. Each sub-module represents a basic crypto primitive function that it can support – for example, a module for AES, a module for SHA-256, and a module for ECC. Different users will have different needs, so it pays to shop around to find the RoT with the crypto support that best fits one's needs. Obviously, RoTs that do not have the necessary functions should not be considered, but conversely a RoT that is packed with more features than is necessary will just be wasted capability. This is especially apparent if the RoT is to be used in a lightweight application, such as in a battery powered IoT application that is highly sensitive to power/area concerns. Sure, an ECC module that offers full support of all NIST-recommended fifteen curves would be very robust, but if the application doesn't even need public key cryptography, then all that silicon area will be just sitting idle, taking up space.

Control Logic Considerations

Similarly, power consumption and area usage are also important factors when considering what type of control logic to use for a RoT. To run with the lightest footprint, it may be worthwhile to pick the RoT with the minimum function list that meets one's needs, then to pair that with a fixed control logic unit of the lowest complexity that still supports all of the required functions. Assuming that the supported function list won't change over the lifetime of the RoT, then this minimal RoT could be the best choice for lightweight applications. But for a RoT whose role will change over time, requiring new functions in the future, such as digital signature (e.g., ECDSA) or shared secret generation (e.g., ECDH), then using the more flexible CPU control logic would be better suited. Besides the commercial offerings from companies using an ARM-based architecture, one cost-effective alternative would be a CPU based on an open-source design, such as RISC-V. The complexity of the function list will determine the type of control logic that works best for a RoT.

Compatible Interface with the Overall Design

Besides choosing the correct type of storage, function list, and control logic for the RoT,

it is equally important to make sure the RoT has a compatible interface with the overall design. Having a RoT that can be dropped into one's design with little to no modifications would be the easiest solution. Will the RoT be placed in an ARM-based system? It would be good to look for a RoT with native support for the AXI/AHB/APB bus protocol. Perhaps the RoT will be used in an OpenTitan environment? Then TileLink bus compatibility should be on one's RoT shopping list. Or, perhaps the RoT will be put in a signal limited system, so a serial interface is the best choice. If the RoT has been implemented using a peripheral bus, such as APB, then it may be placed in a system as illustrated in the below figure 5.

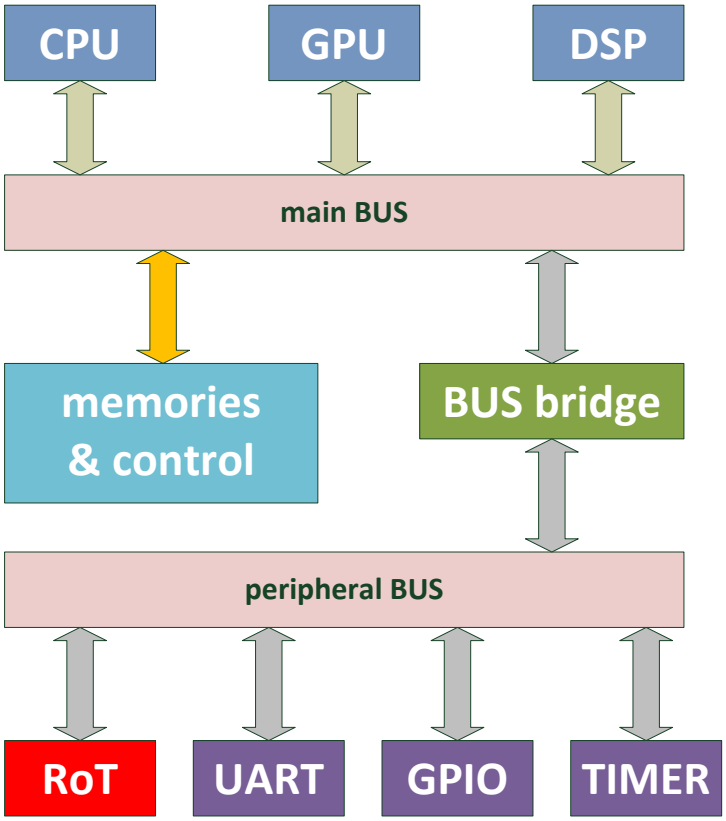

Figure 5: sample placement of RoT in a system

Resistance to Attacks

Having a correct feature list, suitable storage element, and compatible bus interface will help to ensure that the chosen RoT will operate correctly under normal conditions. But what happens when conditions are not normal? That is, what happens if the RoT comes under attack from a hacker? As long as the cryptographic algorithms have been implemented correctly, then besides performing brute force attacks on the

protected data (generally considered too time and resource consuming), a hacker must resort to side channel attacks and direct tampering of the security unit and RoT. To prevent these attacks from succeeding, a secure RoT also needs to have built-in resistance to thwart different types attacks. These attacks could be of either the non-invasive or invasive type. Non-invasive attacks include monitoring the RoT's behavior or emissions to obtain patterns that can be analyzed to grab private keys or other sensitive data. Such observable items include processing time (timing attack), power usage (power analysis), or electromagnetic emission (EM analysis). An invasive attack could take the form of fault injection, or decapping and reverse engineering the RoT. Each of these different attacks will require a different countermeasure, so it goes without saying that users should pay close attention to the list of side-channel attacks that a RoT claims to protect against. Again, it is important to balance the protection offered by these countermeasures with the area penalty that some of them require – for example, many of the power analysis attacks rely on the relationship between the data processed by the crypto sub-module and the power consumption, so to break or obfuscate this relationship, extra logic or power circuits need to be added to the original design – in other words, more gates (hence area) are needed.

Certification: The Way to Narrow Down the Consideration List

After having thought through the above considerations, hopefully one will end up with a short list of suitable RoTs. But is there a way to narrow down the list even further? The answer is certification – certain governing bodies will run products or designs through a battery of testing, then "stamp" it to indicate that it has passed their testing criteria. The closest thing to a worldwide stamp of approval comes from Common Criteria testing, but criticisms of their testing program do exist. One of those is the time it takes to get a product certified, which is why France's ANSSI's Certification Securite de Premier Niveau (CSPN) program has been recently gaining popularity, because of their quick timeline to certification. This is good news for products that are sold in Europe, but if the target market is in the US or Canada, the recognized standard for security testing comes from NIST's Cryptographic Module Validation Program (CMVP), which uses the criteria outlined in the FIPS 140-2 (soon to be 140-3) standard. Finally, if the RoT will be used in a product specifically geared for IoT, then finding one with that has passed Eurosmart's IoT Certification Scheme may be of particular interest. Having a recognized and unbiased third party give their stamp of approval can give users peace of mind and reassure them that they have chosen a quality RoT.

Conclusion

A basic primer on what a root of trust is, starting from its basic definition and moving through the inner workings of a RoT, the author hopes that at this point the reader is familiar with the concept of a RoT and the foundational role it plays in the security of a system. With the diversity of endpoints where a RoT can be installed, the importance of making an informed choice for the best fit RoT cannot be overstressed. The best plan of action would be to methodically consider the various factors, starting with a clear definition of the tasks that the RoT will be called upon to perform. Once the needs have been defined, it is a matter of going "shopping" and picking those security services (and corresponding crypto modules) that can support the required tasks. Finally, by considering the security requirements and the operational environment of the RoT, and thereby performing the risk assessment that the RoT may face, readers can search for those that have the proper level of security certification. From this short list, picking the best fit, in terms of cost/power/performance, should be straightforward enough. However, please be reminded that due to the relative newness of trusted computing (as compared with general computing), a well-qualified, industry-standard list of basic (and advanced) requirements for a trusted, resilient, and verifiable RoT may be slow in coming. Nevertheless, hardware based RoTs, particularly ones based on a PUF, show great promise. They have been described as the "holy grail" of RoT solutions, due to their low cost, low energy, lightweight, unique and tamper proof qualities for the secure authentication of ICs [6]. But nothing is perfect, as PUFs are still susceptible to attacks such as fault injection and side-channel analysis, including model-building. Therefore, to stay ahead of such attacks and future ones, research is still needed to design and implement a trusted, resilient, efficient and effective PUF and PUF-based RoT solutions.

About PUFsecurity

PUFsecurity leverages NeoPUF's physical unclonable technology from our parent company, eMemory, to develop a series of embedded hardware security solutions that combine both digital and analog capabilities.

In the field of IoT security solutions, PUFsecurity provides cost-effective products and leverages over 43 process platforms from eMemory's foundry partners. As a result, we offer a competitive advantage and are confident in promoting our PUF-based security solutions as the best choice for embedded hardware security solutions.

References

[1] “Root of Trust Definitions and Requirements, version 1.1”, GlobalPlatform Technology document GP_REQ_025, June 2018.
https://globalplatform.org/wp-content/uploads/2018/07/GP_RoT_Definitions_and_Requirements_v1.1_PublicRelase-2018-06-28.pdf

[2] “TCG Roots of Trust Specification, revision 0.20”, Trusted Computing Group, July 2018.
https://trustedcomputinggroup.org/wp-content/uploads/TCG_Roots_of_Trust_Specification_v0p20_PUBLIC_REVIEW.pdf

[3] “Establishing the Root of Trust” by Vincent Zimmer and Michael Krau, United Extensible Firmware Interface (UEFI) Forum, August 2016.
https://uefi.org/sites/default/files/resources/UEFI%20RoT%20white%20paper_Final%208%208%2016%20(003).pdf

[4] “Trusted Platform Module Library Part 1: Architecture, revision 1.59” Trusted Computing Group, November 2019.
https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-Library-Family-2.0-Level-00-Revision-1.59_pub.zip

[5] “An Overview of Cryptography” by Gary Kessler, April 2020. https://www.garykessler.net/library/crypto.html

[6] “A Must for AI/IOT: PUF-based Hardware Security” by Charles Hsu, keynote speech for the 30th VLSI Design/CAD Symposium, August 2019.

[7] “IoT Strategies” by Strategy Analytics research services, May 2019.