Lecture Note

Chapter 5:
Hardware Root of Trust

CYENG 225: Microcontroller Essentials for Cyber Applications

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)

Chapter 5 Overview

- **Major Items**
  - Introduces the root of trust.
  - Discusses ideas of measurement and chain of trust. → These ideas are used to demonstrate trusted and authenticated boot, remote attestation, and data sealing.
  - Presents ideas regarding runtime attestation and continuous attestation.
  - Presents ideas for use of PUFs as root of trust.
  - Introduces ideas, and shortcomings, of using authentication for limiting what code can execute in the TCB or TEE.
  - Provides a list of assumptions about the root of trust.

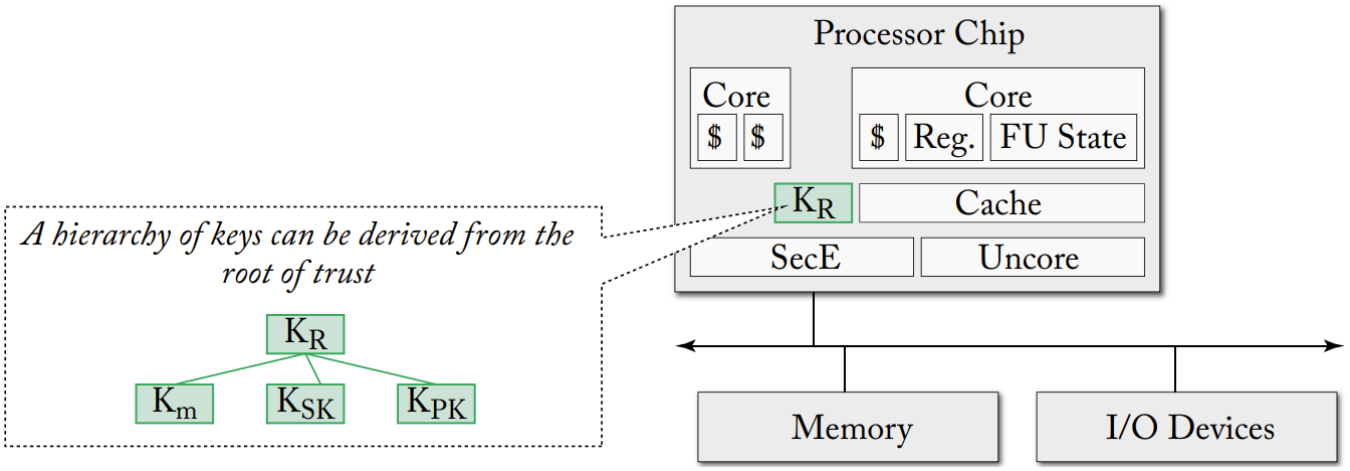
The Root of Trust

- The objective of secure processor designs is to provide **authentication**, **confidentiality**, and **integrity** protections for the TEEs they create.
- Thus, the system that realizes a secure processor architecture has to be trusted.
- The trust in most of these systems derives from **two things**.
- **First**: A root of trust, which is typically a cryptographic key or set of cryptographic keys.
- **Second**: Trust in the manufacturer of the hardware and the hardware itself.
- To form the root of trust, cryptographic keys are needed for authentication, confidentiality, and integrity.
  - **Authentication:** The processor needs to authenticate itself, so a remote party or user can convince oneself that certain data or messages are indeed coming from a particular, trusted system.
    - ✓ Public key cryptography is the choice for authentication: processor has a secret key, and a public key which is known to everybody.
    - ✓ This key can be derived from a processor's secret key.
  - **Confidentiality:** The processor needs encryption keys for confidentiality of data going out of the processor.
    - ✓ For memory protection, ephemeral keys can be used, which are randomly re-generated on each system startup.
    - ✓ For persistent data, the keys need to be generated in a systematic way.
  - **Integrity:** The processor needs keys for integrity checking of data going outside of the processor. These can be derived as well from the secret key.
    - One possible way is to use cryptographic hashes to generate all the keys from the root secret key.
    - Getting access to a derived key, due to one-wayness of hashes, should not leak information about the processor's secret key.

The Root of Trust - Root of Trust and the Processor Key

- Security of the system is derived from a root of trust: A secret (cryptographic key) only accessible to the TCB components.
- **From that secret key**, it is possible to derive all the encryption, signing, and other keys.
- The root of trust key needs to be securely stored in processor chip the TCB.
- Each secure processor requires such a secret key, which should further be unique to each instance of the processor, i.e., each chip.
- One option is to burn it in at the factory by the manufacturer.
- Another option is to use **PUFs** → This requires addition of extra circuits that realize the PUF there is needed to ensure reliable operation of the PUF and **error correction circuits** are needed, and finally extra hardware module to derive cryptographic keys from the PUF output is needed.

A diagram of a secure processor, showing its secret key, KR, which is stored within the processor boundary. The secret key should only be accessible to the components in the TCB, and never leave the boundary of the chip. From the secret key, other keys such as for encryption or signing can be derived.



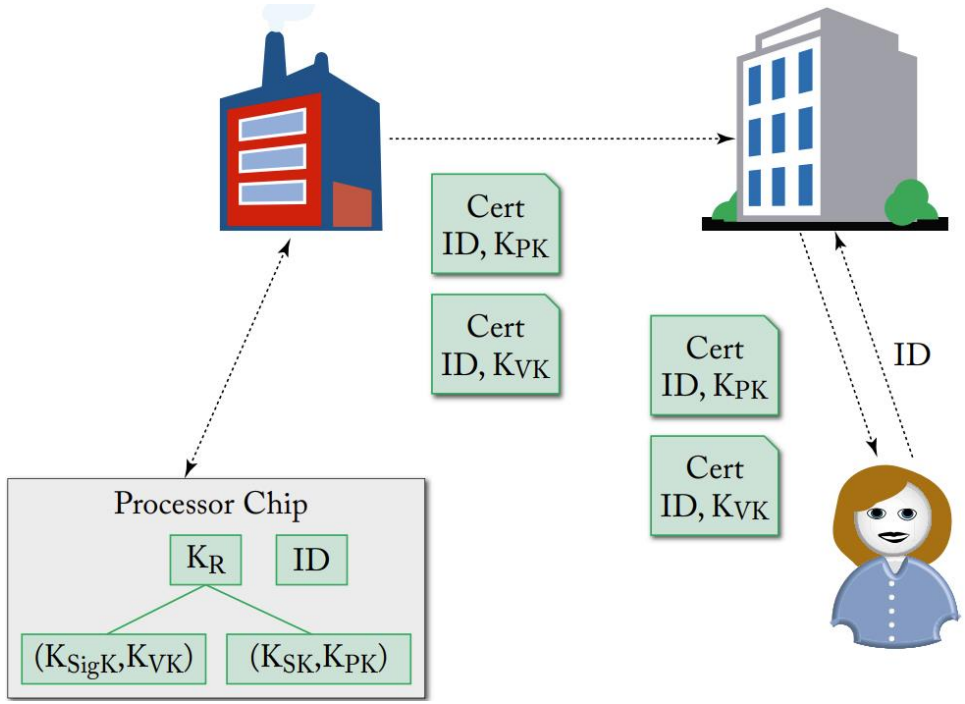
The Root of Trust - PKI and Secure Processors

- **Public Key Infrastructure (PKI)** is needed to distribute the certificates of the public keys of the processors.
- Especially, users need to know which are the keys belonging to the legitimate processors.
- The manufacturer, or a trusted third-party, needs to maintain **the lists of known-good keys** and **the associated processors**.
- Revocation is very difficult, and an open research topic → Once a processor is compromised, its key can be extracted, and thus should be revoked.
- Typically, from the root of trust key, **KR**, are derived keys for confidentiality, integrity, and authentication.
- **A secret key, KSK** and **corresponding public key, KPK**, are derived for encrypting data to be sent to the processor. → The data handled by the TCB, not the TEE as TEE software will have its own keys.
- A **signature generation key**, **KSigK**, and a **corresponding signature verification key**, **KVK**, are also derived. → They are used to sign data generated by the TCB.
- Especially, the TCB's signing key can be used to sign user keys generated in the TEE, so that **the TEE's keys can be tied, via the signature, to the underlying hardware** that the TEE is executing on.
- The manufacturer (if the KR key is burned in) can generate the public and verification keys, and their corresponding certificates.
- The different keys are derived often by use of cryptographic hashes.
- Getting access to a derived key, such as the public encryption key, **due to one-wayness of hashes**, should not leak information about the processor's secret key.
- If the secure processor design **leverages PUFs**, it may require users to run their own key distribution solutions, or use existing, third-party PKI.

The Root of Trust – Access to The Root of Trust

- A physical attack or unscrupulous manufacturer or system integration could steal or leak secret keys.
- A more likely danger, however, is the code that is part of the TCB, and has direct access to the root of trust, which may leak the key unintentionally due to a bug, or due to some attack.
- Usually, BIOS or other low-level code is able to access the secret keys (e.g., to sign digital messages generated by the processor).
- Bugs in the code can lead to exploits that can leak the secret key.

The manufacturer (top-left) initializes the secret key of a processor, and is able to derive different encryption (KSK, KPK) or signing keys (KSigK, KVK). The public (KPK) and verification (KVK) keys, and corresponding certificates, can be generated and given to the certificate authority (top-right). Given some ID of a processor, the users (bottom-right) can obtain the certificates for that processor's keys.



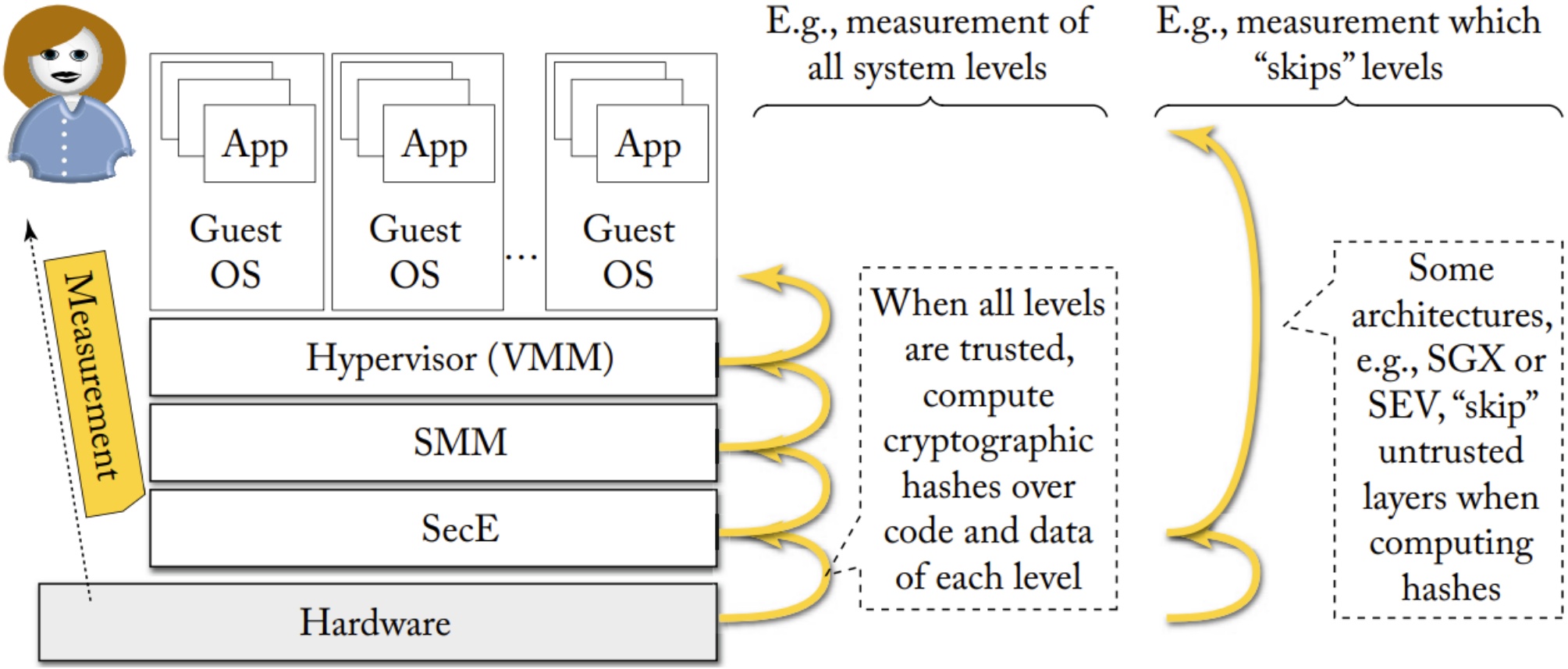
Chain of Trust and Measurements

- When running TEEs on a system, it is crucial to know that **it is correctly configured**, e.g., only decrypt the file system if the correct operating system has been loaded.
- To achieve this, **the trust begins with the secret key and a small piece of code** which is implicitly trusted, such as a small BIOS.
- When a system turns on, this BIOS is the first code to execute.
- This code can then **"measure" next piece of code** that is to be loaded an executed, e.g., the firmware.
- Next, the firmware can "**measure**" the OS that starts up next, etc.
- Here, "**measurement**" is the act of generating a cryptographic hash of the code (and data) after it is loaded and before it is executed.
- The goal is **to prevent someone from modifying this code** (if the code is modified, the cryptographic hash will have a different value than the expected value).
- Eventually, there is **a set of hashes of the different components that booted up one after the other**.

Chain of Trust and Measurements (Cont.)

- Rather than to store each separately, each measurement can be used **to "extend" the prior measurement**.
- To "**extend**" means that **hash value of the prior piece of code is concatenated with the next level of code and data when the hash is generated**.
- **At each level the hash depends on the code and data, and the hash of the prior level**.
- In the end, one hash value depends on all of the prior code.
- This idea has been leveraged in the **Trusted Platform Module (TPM)** from which the term "**extend**" may have originated.
- The TPM is a low-cost co-processor dedicated to security.
- The main objective of the TPM is to provide a root of trust in the platform.
- TPM helps to protect against software attacks by providing attestation and sealed storage mechanisms.

Chain of Trust and Measurements (Cont.)



Left-hand side shows the levels in a typical secure processor architecture. Right-hand side shows two examples of different levels being measured. In the end, the user should receive the measurement, i.e., a cryptographic hash, that reflects the state of the system (the TCB and the TEE).

Trusted and Authenticated Boot

- **Trusted Boot** means that the system will only operate if all the measurements are correct, otherwise the system may not even start up.
- **Authenticated Boot** means that the system does all the measurements, and users can access the measurement values, but the system will still continue to boot up and run even if the measurements are not as expected.
- Currently, there is no standard action to be taken when the measurements are not correct.
- A system can stop executing or secret data can be prevented from being decrypted, as in trusted boot.
- **Failover Modes** → A standby module automatically takes over when the main system fails are not well explored in hardware secure processor architectures.

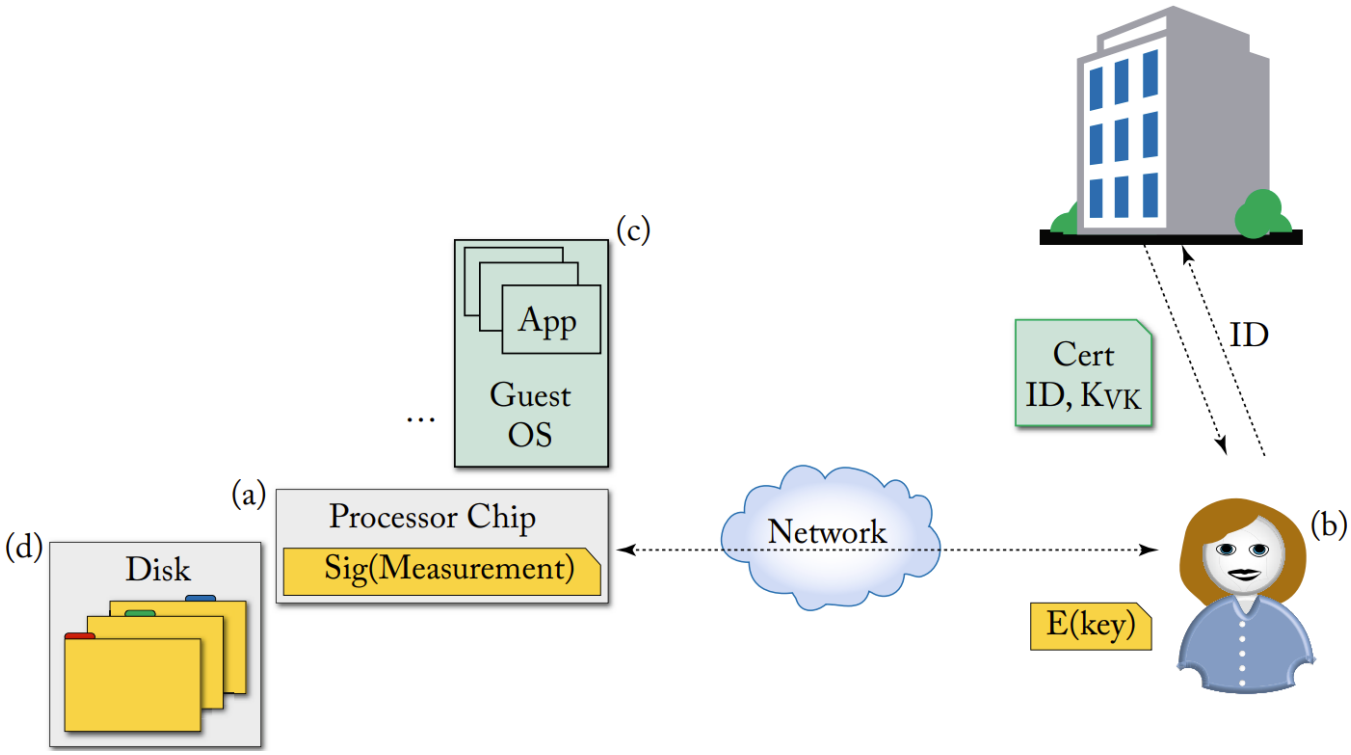
Measurement Validation

- The measurements taken during the trusted boot need **to be validated**.
- The main way to do the validation is **to check the measurements** (i.e., **Test Data**) **against a list of known-good values** (i.e., **Reference Data**).
- By definition of **cryptographic hashes**, even change in one bit of the input **"m"** will cause the resulting hash **"h"** to be totally different, compared to hash of the original input.
- **Problem**: **Any update** such as a new kernel version, or even addition of a new kernel module, or other update will cause all the hashes to change.
- **A Possible Solution** is to **leverage the PKI** used for key distribution, to distribute well-known hashes for variety of system configurations - or users can generate their own databases of known-good values based on software they or their companies use.
- **Once the signature is validated**, the actual measurement value needs to be checked. → Mismatch in the measurement value means that some software in the TCB or TEE is not as expected.

Remote Attestation

- **Remote Attestation** → It combines the root of trust with measurement, especially to attest that a system is running correctly, **the measurements of the software as the system booted up can be signed with the private key of the processor**, and sent to remote parties.
- These remote users can compare the known-good hash values with the received data and satisfy themselves that the system booted up in a correct configuration (or find that there is a problem).
- Because the hashes are signed with the private key of the processor, users are ensured that the measurement could not have been modified or forged after it was generated by the processor.
- It should be remembered that issues of **replay attacks** are critical → One needs to ensure at the protocol level that some correct (but old) measurement is not replied or sent again in future.
- **The processor can make the measurement, of the TEE's code, sign it and send to the user**.
- If he or she verifies the signature and the measurement, then he or she can send sensitive data to the TEE by encrypting it with a new session key.
- **The session key can be encrypted** with the processor's public key, or more likely a key generated by the TEE, and signed by the TCB.

Remote Attestation (Cont.)



a) A processor chip can generate measurements of the TCB (and TEE) and sign them with its signing key. (b) A remote use can obtain certificates from a trusted certificate authority, top-right, and use the certificates to validate the signatures of the measurements. (c) Once the measurement is validated, if it is for TEE, the user can send some sensitive information to the TEE, by sending encrypted information as well as sending the needed decryption key. The decryption key is protected by public key of the TCB or TEE. (d) The user can also send to TEE keys needed for decryption of data stored on remote disk.

Sealing

- **Sealing** → It is the act of encrypting some data with a key derived from a measurement.
- **Unsealing** → It is the act of decrypting data with a key derived form a measurement.
- The idea of sealing and unsealing is **to tie the data to a specific hardware and software configuration**.
- This **prevents malicious uses from accessing data** if they changed the software while the system was turned off (it does not protect against attacks while the system is running nor against time-of check to time-of-use attacks).
- Once the TEE is verified, the user can send back a cryptographic key (encrypted again with the public key of the TCB or public key generated by the TEE).
- That cryptographic key can be used to decrypt files and data stored on the disk.

Time-of-Check to Time-of-Use Attacks

- **The main limitation of any measurement technique** that measures (i.e., generates cryptographic hashes) the code is that it does not say anything about the code after it was measured.
- It only is **based on raw measurements of static code** as it was right before the piece of software executed.
- This leads to well-known **class of Time-of-Check to Time-of-Use (TOCTOU)** problems.
- The system may be correct at time t0, but it may become compromised at time t1, if user asks for the values of measurements at some later time t2, he or she will get the hash value that states that system was okay at t0, but no information about state at t2 is gained.
- One **solution** to this problem is to **perform continuous attestation at system runtime**.

Runtime Attestation and Continuous Monitoring of TCB and TEEs

- Projects such as Copilot have presented ideas about runtime attestation or checking of the state of the system.
- Most common approach is **monitoring of the execution of the system**.
- A co-processor can be added, where main processor executes normally, while **a co-processor performs validation of the execution path**.
- This can be also performed with performance monitoring counters now available with many processors.
- Through observation of **execution patterns**, such as the number of branches taken, branch addresses, frequency of memory accesses, etc., **a pattern of behavior of can be detected**.
- This pattern can be learned and, at runtime, **the performance counter data** can be used to detect whether the program is behaving as expected or it has deviated.
- This can be applied to both the trusted and untrusted components, however, most work focuses on the untrusted components or the software that is being executed.

Runtime Attestation and Continuous Monitoring of TCB and TEEs (Cont.)

- A more structured approach is **analysis of control-flow graphs**.
- **Source code can be analyzed** to understand the control flow graph of software.
- At runtime, **the execution of a program** (i.e., **Test Control Flow**) can be checked to see if it has deviated from **the expected control flow** (i.e., **Reference Control Flow**).
- This can be applied to the system and other software.
- **At the hardware level**, however there is often a state machine that dictates the behavior of the hardware - **how to monitor the correct execution of such state machines** is an open problem.
- **Software can be monitored by hardware**, but it is not clear how hardware, which is the lowest level in the system, can be monitored.
- A possible solution to this problem is to use **hardware odometers**.
- The odometers work focuses on checking how long the hardware has been running (i.e., as an anti-recycling feature), but similar counters or hardware features can check how hardware state machines are running.
- Continuous monitoring can leverage performance counters and look for anomalies.
- This requires knowing the correct and expected behavior of system and the software.
- The attacker can "**hide in the noise**" if they **change** the execution of the software **slightly** and do not affect performance counters significantly. → **Sneaky Attack can be for Hardware and/or Software**.

PUFs and Root of Trust

- **Physical Unclonable Functions (PUFs)** leverage **the unique behavior of a device** due to manufacturing variations (i.e., **Process Variations**) as **a hardware-based fingerprint**.
- A **PUF instance** is extremely difficult to replicate, even by the manufacturer and provides **an unique identifier** for the system.
- **PUF is very much like a secret key**, and PUF response can be used in place of burning-in a key by the manufacturer.
- This can be used as a root of trust to establish whether software is executing on correct platform.
- Once the measurement is done, there is **no real binding between the software and the hardware**.
- Only on correct hardware, the PUF generate correct outputs that will cause software to execute as intended.
- If the software is moved to a different system, **the PUF output** by design will be different on another system, and **the software will no longer execute correctly**.

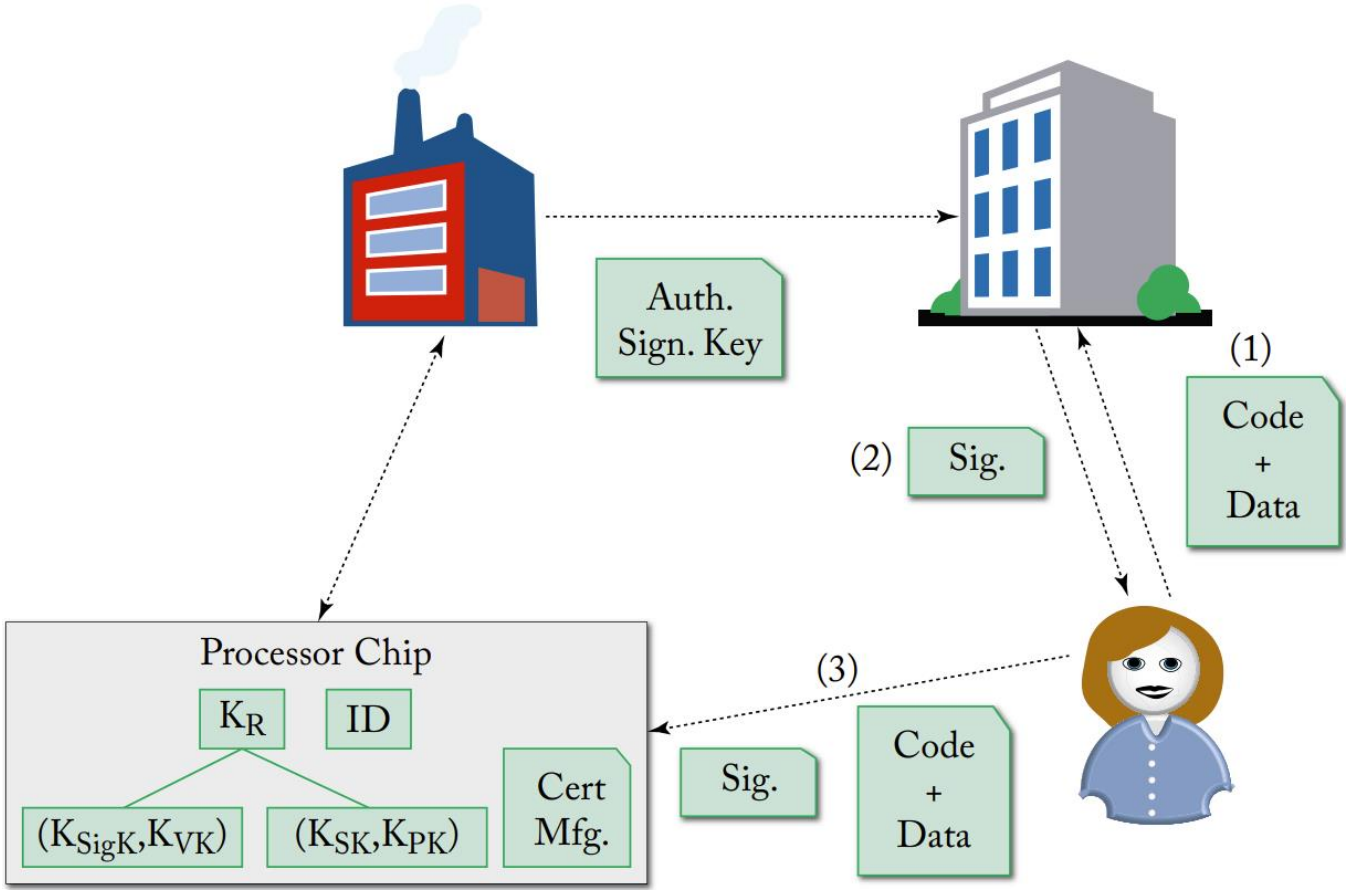
PUFs and Root of Trust (Cont.)

- To ensure that the software is indeed executing on a particular piece of hardware, **hardware-software binding** has been proposed.
- **Hardware-Software Binding** → It is a method to precondition execution path of the software on the output of a PUF.
- The hardware-software binding **can be implemented via an indirection table** in the software, where the **PUF measurements** are used to **decrypt** a table that controls the jump or call addresses used in the source code.
- To enable the hardware-software binding, at system runtime there needs to be **a stream of output from the PUF to be used in the indirection table**.
- A PUF can be read at start up time, and used **as a key to a pseudo-random number generator**.
- Storage of the PUF output may be challenging to secure, and in principle is not different from storing the secret key.
- Hardware-software binding can be made to work with architectures that do not use PUFs, but do have a secret key, unique to each system.
- A run-time accessible PUF can be used, such as the DRAM PUF which can be queried at runtime.
- A run-time PUF reduces the need for storage of secret keys or generation of pseudorandom numbers from a stored key.

Limiting Execution to Only Authorized Code

- **Firmware (TCB) updates or protected software updates** can be authenticated in the processor through **use of signatures** made by a trusted party.
- **Computing Flow**:
  - At manufacturing time, a certificate for the manufacturer's public and verification keys can be inserted into the processor.
  - Users can **send code and data to an authorized party** who has access to the signing key of the manufacturer.
  - The authorized party can inspect the code and data, and **if approved, generate a signature**.
  - The user can then send the code and the signature to the remote processor.
  - The TCB components, with access to the certificate stored in the processor, can **validate the signature**, and, **if correct**, then load and execute the code.

Limiting Execution to Only Authorized Code (Cont.)



The processor chip can be embedded with a certificate for known-good verification keys, such that the processor TCB can authenticate messages, especially code and firmware updates. The manufacturer (top-left) can send signing keys to the trusted third party (top-right). Users can request the trusted third party, or even the manufacturer, to sign their code and data. The code and data can then (bottom) be sent to the processor. The processor can validate the signatures and only load the code if the signature is correct.

Limiting Execution to Only Authorized Code – Lock In and Privacy Concerns

- **Privacy issue** arise from any authentication mechanisms which **limit** what code can run on the TCB, or inside the TEEs.
- If only the manufacturer can update the device's TCB, then they are **dependent upon** for security updates and fixing bugs.
- **Requiring the manufacturer** or another party **to approve code** that can run inside TEE **further limits** what code can be executed in the TEE.
- When a system is running and the TCB or TEE is being authenticated, use of signatures based on keys unique to a processor presents **privacy issues**.
- **If a private key is used directly each time, one can know from which processor the messages are coming and a connection can be made between communication patterns and a specific physical machine**.
- **If a certificate authority is used for key and certificate distribution, it can know exactly when a specific processor is being used by a user who requests certificate about that processor**.
- **Solution**: **Direct Anonymous Attestation (DAA)**, developed for TPM, offers some protections while allowing for remote authentication.

Root of Trust Assumptions

- **Unique of Root of Trust Key Assumption**
  - The unique secret processor key is assumed to be assigned to each processor so any two secure processors can be distinguished.
  - The manufacturer is assumed to never assign two keys to different processors.
  - If using PUFs, then the hardware needs to generate a unique key for each device even in event of errors or measurement noise in the PUF measurement from which the key is derived.
- **Protected Root of Trust Assumption**
  - The root of trust key is assumed to be protected and never disclosed.
  - If keys are burned-in by the manufacturer, the manufacturer is assumed to keep secure database of the keys, and ideally delete the private keys once they have been generated.
  - The manufacturer is entrusted with protecting the keys, and to never disclose those keys.
  - If keys are derived from PUFs, then the enrolling party is trusted not to disclose the keys.
  - The key is assumed to be stored in the processor chip and is part of the TCB and never disclosed.
  - All TCB components (hardware or software) which have access to the key are trusted to never leak it.

Root of Trust Assumptions (Cont.)

- **Fresh Measurement Assumption**
  - Authentication and data sealing **give access** of TEE software (which is presumed to be correctly executing) to data.
  - Measurements are used to unseal data, and **need to be fresh**, meaning that they should be **recently taken to avoid** the **class of Time-of-Check to Time-of-Use (TOCTOU) attacks**.
  - Continuous authentication can be implemented to constantly check the TEE software.
  - A mechanism needs to be in place to **revoke access** to sealed data if measurements change.

Assignment

- **Reading Assignment:**
  - Zferer, J., 2018. **Principles of secure processor architecture design**, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 9048, pp.1-175.
    - ✓"Chapter 5: Hardware Root of Trust", Pages 53-64.

Questions?