Basic Computer Security Concepts

Chapters 1,2

[1] J. Szefer, “Principles of secure processor architecture design,” *Synth. Lect. Comput. Archit.*, vol. 13, no. 3, pp. 1–173, 2018.

Security Is a Major Issue

- Software complexity and bugs
  - **Larger software** base code increases the vulnerability (because bug density is constant).
  - OS and hypervisors, although isolate somewhat the problem, **suffer for the same problem** (even hypervisors are quite complex today due to performance enhancements)
  - How to isolate trusted from untrusted code **with no full guarantees from OS**?
- Side-channel Attacks
  - **Cloud computing** favors co-residency/multi-tenancy
  - Somehow, infer other “apps” internal data is easier
- Physical Attacks
  - Device electrical **probing** can leak valuable information
  - Cloud infrastructure is vulnerable to these kind of attacks
- Humans

Need For Secure Processor Architecture?

- Software Size
  - OS is ~Millions of code lines (but **hypervisor too**!)
    - https://www.openhub.net/: Linux ~20M, but **Xen ~1M**, **qemu ~1.5M**, **libvirt ~1M**, etc….)
  - ~**20 Bugs per 1000 lines** of code [69]. The likelihood of compromising in-place protection mechanisms (OS or hypervisor) is significant
- Side-Attack Channels
  - **Processor architecture complexity** might introduce bugs (Meltdown) or just open the door (Spectre)
  - For each performance "trick" at least **a time-based side-channel** attacks (BP, prefetchers, caches, etc…). Can't be disabled
- Physical Access
  - Unable to determine is someone is tampering with the system (behind the scenes), for example rogue employees or government compelled companies can probe our server
  - In a VM is trivial but also possible in **Bare Metal hosted system** (e.g. cold attacks)
  - **IoT** are also susceptible of being tampered (without being aware of it) due to remote location
- Add **security features** to the hardware

Trusted Computing Base (**TCB**) (I)

- HW + SW **components** that work **together** to provide some security guarantees to the "Software"
  - "Software" usually privileged software as OS or hypervisor Hardware components are exposed to achieve the desired guarantees (but software optionally can ignore them)
  - **TCB should be as small as possible (both SW and HW sides)**

- TCB design should assume that the everything else is **untrusted**
  - Can be exploited by malicious attacks but can't "**penetrate**" TCB portion

- Any modification or **bugs in the TCB** parts might break the security guarantees
  - **Trustworthiness** is a qualitative designation indicating whether the entity will behave as expected, is free of bugs and vulnerabilities, and is not malicious

- Exhaustive techniques, such as **formal verification**, will be used to guarantee TCB

Trusted Computing Base (TCB) (II)

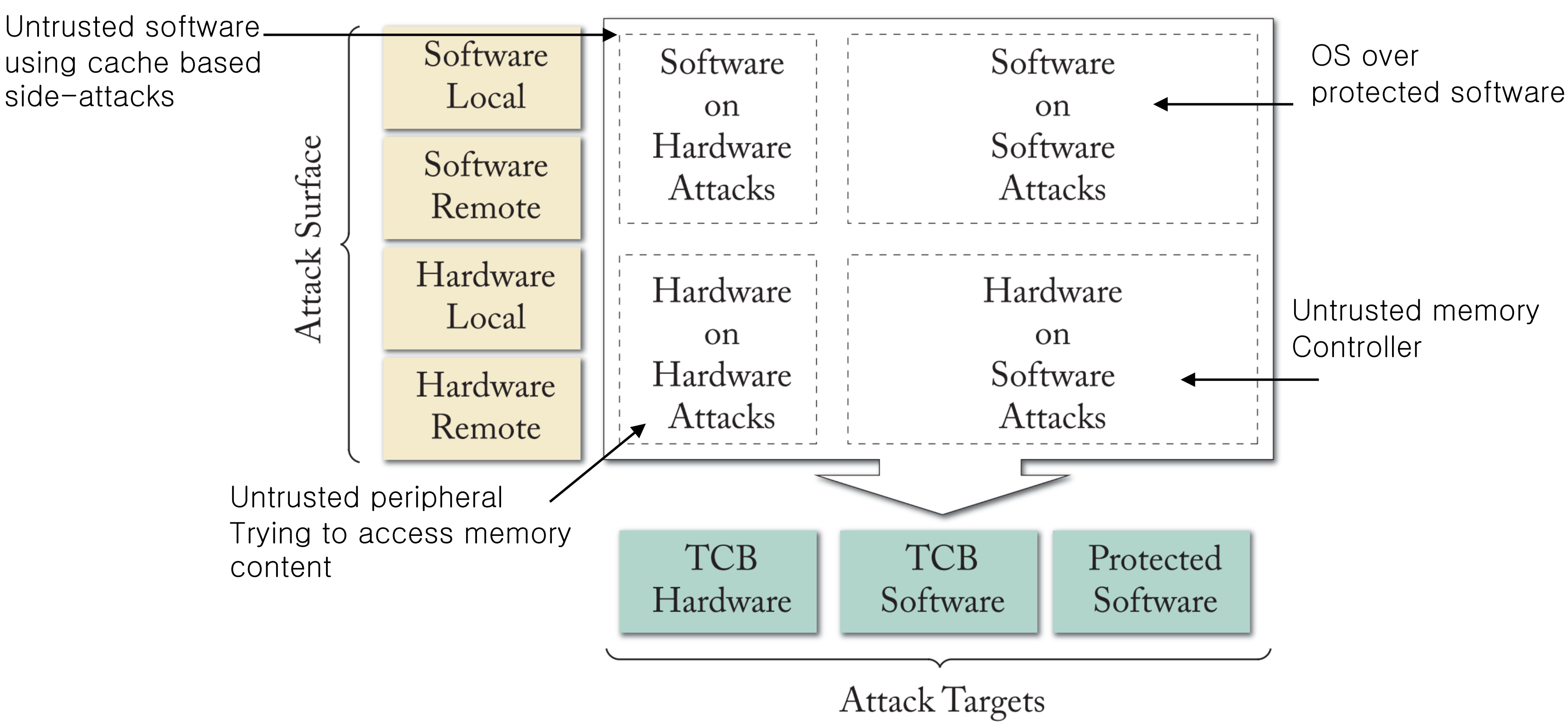
- Architecture designers should (logically) ensure that the **information exchange** between of the components of the system cannot be attacked

- But beyond design, hardware trojan can be added during the manufacturing time
  - Supply chain security
  - Foundry malicious behavior detection mechanisms

- Avoid security via obscurity (apply Kerckhoffs' 2^{nd} of six principles [167])
  - TCB design details **should be public** (e.g., Riscv/Linux openness vs Apple obscurity)
  - The **only secret** are the **cryptographic keys** to guarantee secure information exchange between components

Security Threats to a System: The attack surface

- Combinations of all attack vectors that can be used against a system
  - From non-TCB internal hardware or software
  - From external hardware (e.g., cold attacks on memory, physical probing, …)
  - From external software (e.g., Use network services via buffer overflow…)



Security Threats to a System: Passive and Active Attacks

- In **passive** attacks only observe the behavior of the system to deduce some information about it
  - **Externally** e.g., EM emission, power consumption, etc… (air gaped systems)
  - **Internally** e.g., Access to performance monitoring unit (PMU) to deduce TCB protected information (via side-channel attacks)

- In **active** attacks, the attacker will try to modify system behavior (e.g., Injecting changes in some memory location, usually instructions). Also inject fault in the hardware
  - **Spoofing**: e.g., inject memory commands to read or write protected memory
  - **Splicing**: e.g., mix correct (spoofed before) and malicious memory commands
  - **Replay**: e.g., capture and resent memory commands
  - **Disturbance**: e.g., DDoS or repeated access memory location to propagate changes between DRAM Rows (*Rowhammer*)

Security Threats to a System: Types

- **Man-in-the-middle Attacks**
  - Intercept, copy and re-transmit sensitive information, either **without tampering** with it (passive) or **modifying it** (active).
- **Cover Channels**
  - Is a communication channel that **was not designed** to transfer information
  - Timing, power, thermal, eletro-magnetic emanations, acoustic emanations,…
  - If known at at design time can be prevented (e.g., cache isolation between process avoids timing). Others require physical access to the system under attack
- **Side Channels**
  - Like cover but the sender does not intend to communicate information to the receiver (but elsewhere)
  - The leaked information is a side effect of the implementation and how HW and SW is used
  - Sender and receiver under control of the attacker to build a "cover" attack
- **Attack bandwidth**
  - Cover and side channel mechanisms, in most cases, are stochastic: extraction rate (bandwidth) depends upon its severity

Security Threats to a System: The Threat Model

- Is a **concise specification** of the types of threats that a given secure processor architecture protects against (can't be anything).

- Should specify:
  - TCB: set of trusted hardware and software components
  - Security properties that the TCB aims to guarantee
  - Capabilities of the potential attackers
  - Potential vulnerabilities to address

Threat to Hardware after TCB Design Phase

- Bugs or Vulnerabilities in the TCB
  - By definition TCB assumed free.
  - Software and hardware design security verification is a very active research are
- Hardware trojans and supply chain attacks
  - Globalization: parts of a single system comes from a diversity of IP providers, and manufacturers to the final product
  - Intense work in contra-measures
- Physical Probing and Invasive Attacks
  - What accessibility assumes the design to the system?
    - Mem, decapsulating, ion beams, ….?

Basic Security Concepts

- **Confidentiality** is the prevention of the disclosure of secret or sensitive information to unauthorized users or entities.
  - If attacker find a breach, sensitive information can leak
  - Can be using side/cover or brute force (e.g., password crackers)
- **Integrity** is the prevention of unauthorized modification of protected information without detection
  - Attacks are focused to bypass integrity checks to gain access to the systems e.g., alter bit *s* in certain executable)
- **Availability** is the provision of services and systems to legitimate users when requested or needed
  - Attacks focused on render the system unusable to everybody else (e.g. DDoS attack)
- **Authentication** relates to determining who a user or system is
- **Freshness**: update the authentication requirements across time. Aggregate a nonce/salt in the cypher (e.g., a counter) to prevent replay attacks. Based on a monotonic counter.

Symmetric-Key Cryptography

- Encryption and decryption **uses the same key**
  - The encrypted ciphertext looks random to anyone without the key
- Block Cyphers
  - Algorithms that produce the ciphertext from 1 blocks of information (AES uses 16bytes).
  - Information with multiple blocks should be handled: the same block produce the same results (ECB) or change the results (e.g., according the position of the block in the text)
  - Block Cyphers **only** provide **confidentiality**: need additional mechanism to provide **integrity**: **hashing** (on top of encryption or combined)
- Stream Cyphers
  - Encrypt the text bit by bit with xor pseudo-randomly generated keystreams
  - Faster in hardware but require to **process the whole text** to access to a part
- Algorithmss
  - **Block**: Old algorithms are unsecure (3DES, DES, RC4). Use AES with a strong key (128-256bits)
  - **Stream**: ChaCha

Public-Key (asymmetric) Cryptography

- Key to Encrypt is **public** (pk) != key to Decrypt is **secret** (sk)
  - We only need to interchange pk
  - *sk* cannot be inferred from *pk* (depends on hardness of certain mathematic problems, such as large number factorization)
  - In reverse direction (encrypt with *sk* and decrypt with *pk*), can used for integrity (digital signatures or message authentication codes (MAC))
- Key encapsulation mechanism
  - **Public-key** encryption is **computationally expensive** (if data to transmit is large): use public-key cryptography to interchange a symmetric encryption key
- Post-Quantum Cryptography (**Grover's algorithm**)
  - Algorithms used in current PK, such as RSA, might be computationally vulnerable against brute-force attacks using quantum-computers (theoretically).

Secure Hashing

- One way function (mathematically collision-less and noninvertible) to compute a **fixed length** digest or fingerprint from data with **arbitrary size**.
- Properties
  - **Pre-image resistance:** given *h* impossible to find *m* such as *h=hash(m)*
  - **Second Pre-image resistance:** given *m1* should be impossible to find *m2* such as *hash(m1)==hash(m2)*
  - **Collision Resistance**: should be difficult to find *m3* and *m4* where *m3!=m4* and *hash(m3)==hash(m4)*
- Commonly used for integrity checking
  - Any small change in *m* will completely change *h*
- Just prevent tampering

Common uses of hashes

- In **Message Authentication Codes (MAC).** Only the entity with the correct cryptographic key can generate/check the hash value

- Digital signature using the private key to generate the signatures (and public key to authenticate)

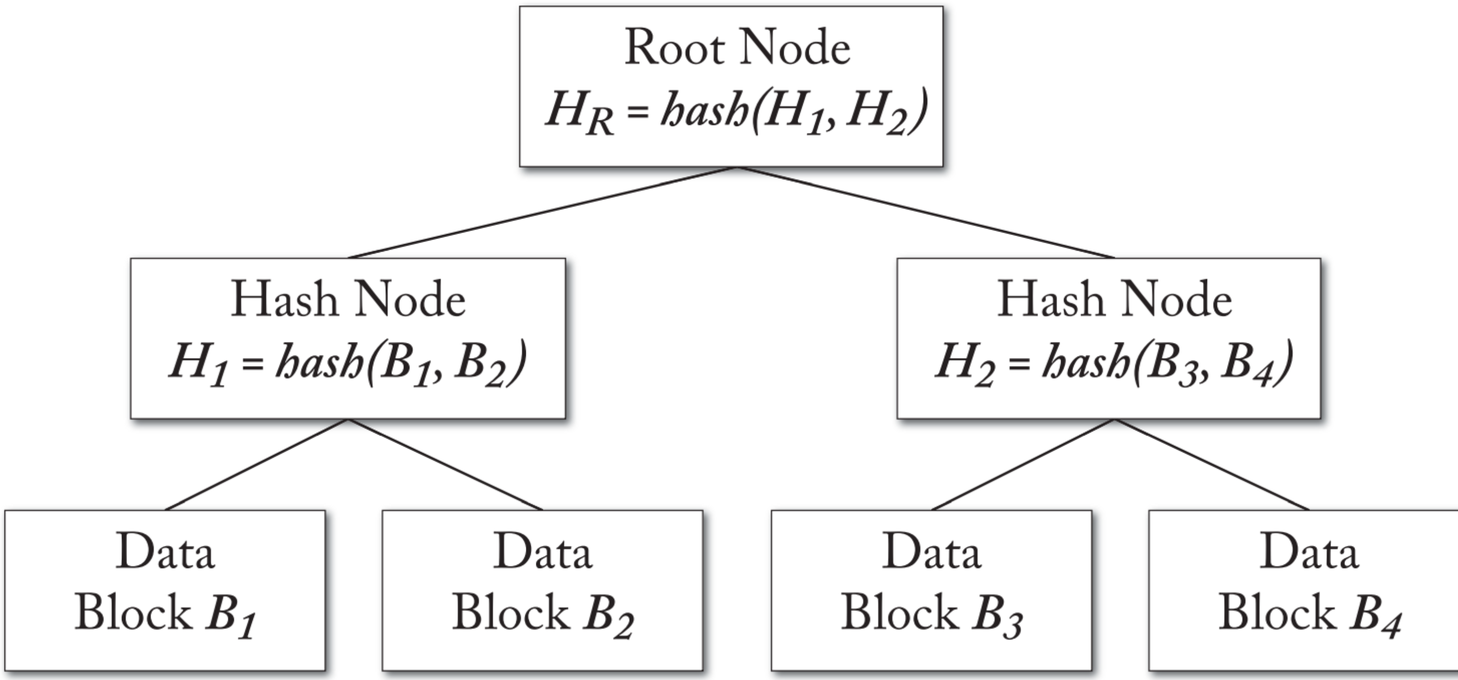
- Key derivation functions (KDF): from a master key derivate other encryption keys

- Hashes trees. In large pools of data speed up the hash computation after altering a part (leaves of a binary tree).

- Most common is SHA-2 or SHA-3 (old SHA1, MD5, MD44. are not secure against current computational power [are brute force susceptible to collisions])
  - SHA-2(256) and SHA-3(512) will be unsecure in the future (if Moore's law don't stop)

Merkle Trees

- Safe hashes are very slow and should run over all the data
- Accelerate hash computation to speed up information updates



Public Key Infrastructure (PKI)

- PKI is a set of policies and protocols for managing, storing, and distributing certificates used in public-key encryption
  - There is a trusted third party (**Certificate Authority**) which can distribute digital certificates that vouch for **correctness of public keys**
  - Certificates for the certificate authorities are usually **pre-distributed** (e.g., browsers come with built-in list of certificates for certificate authorities
- **Digital certificates**
  - Contains some identifying information about a system or a user and their public key.
  - This information is encrypted with a private key of the certificate authority
  - With the public keys of the certificate authorities is used to check authenticity and content of the certificate
  - Have expiration and can be revoked by the CA (CA should keep a CRL available to the clients)
- PKI is used in secure processor in combination with physical unclonable functions (PUF)
  - Depends upon random effects during the manufacturing process
  - Allows Direct Anonymous Attestation (DAA): allows remote authentication of trusted computer (to form a network of TCB) without compromising privacy of the platform

Outline

- Secure Processors Architecture
- Trusted Execution Environments
- Hardware Root of Trust
- Multi-core Protections
- Side-channel Threats and Protections
- Principles of Secure Processor Architecture Design