**VANGUARD – SECURE BOOT**

**Reference:**

[1] <https://riscv.org/2019/01/risc-v-community-releases-opensbi-to-foster-continued-ecosystem-growth/>

[2] <https://www.electronicdesign.com/technologies/embedded-revolution/article/21806085/secure-boot-what-you-need-to-know>

1. **Tổng quan**

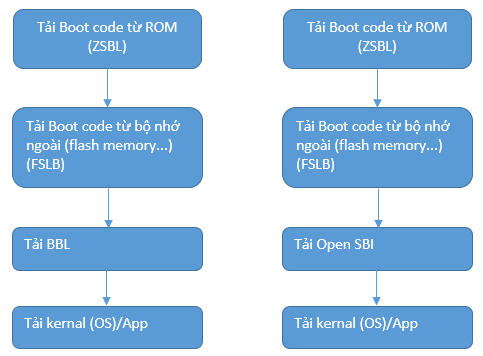
Boot là qua trình phần cứng (SoC/MCU…) tải mã code của phần sụn (firmware), hệ điều hành(OS) hay các ứng dụng (Application) từ bộ nhớ ngoài (flash memory , Ethernet ….) vào bộ nhớ nội (SRAM) hoặc DRAM và thực thi sau khi Power-on reset.

Các đoạn mã code có thể bị chuyển đổi bởi bên thứ 3 (hacker…) nằm kiểm soát hệ thống .

Do đó, việc kiểm tra tính xác thực của các đoạn mã code trước khi tải và thực thi chúng là cực kỳ quan trọng. Việc kiểm tra trước khi tải và thực thi các đoạn mã code chính là quá trình secure boot.

1. **Quy trình boot tham khảo của RISC-V**

Qua tham khảo các tài liệu về quy trình boot, các quy trình dưới đây là phổ biến [1]:



* ZSBL: Zero phase Boot loader.   
  Được lưu trữ trong ROM , là một đoạn mã máy cố định, sẽ được thực thi bởi CPU , có các chức năng sau:
  + Tải mã boot code stage tiếp theo từ thiết bị bộ nhớ ngoài đã được chỉ định.   
    (Có thể hổ trợ nhiều tùy chọn bộ nhớ ngoài-dựa trên giá trị của các boot-mode-pin (phần cứng)
  + Chứa bảng mô tả địa chỉ của các IP trong SoC (discovery page)
* FSBL: First Stage Boot Loader : (T.B.D)
* BBL: Berkeley Boot loader : Một mã nguồn mở hổ trợ quá trình boot:
  + Load kernel
  + Quản lý timer interrupt…
* SBI (OSBI) : open source Supervisor Binary Interface
  + <https://github.com/riscv/opensbi>
  + Chức năng : T.B.D

[Ghi chú]: OSBI được đề nghị sử dụng bởi riscv.org [xxx]

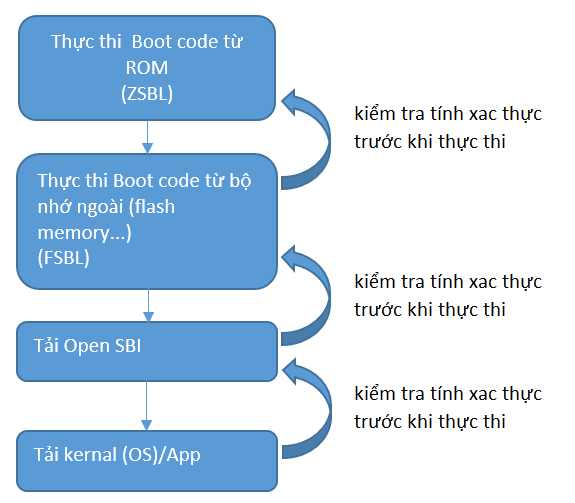
1. **Quy trình secure boot**

Secure boot, về cơ bản, giống như quy trình boot ở trên, nhưng mã code của bước sau sẽ được kiểm tra tính xác thực bởi bước trước đó. Nếu tính xác thữ không được đảm bảo, bước sau sẽ không được thực thi.

ZSBL (Zero phase boot loader) là một đoạn mã code cố định (immutable boot loader).

Đoạn mã code này được nhúng trong “Root-of-Trust device”.

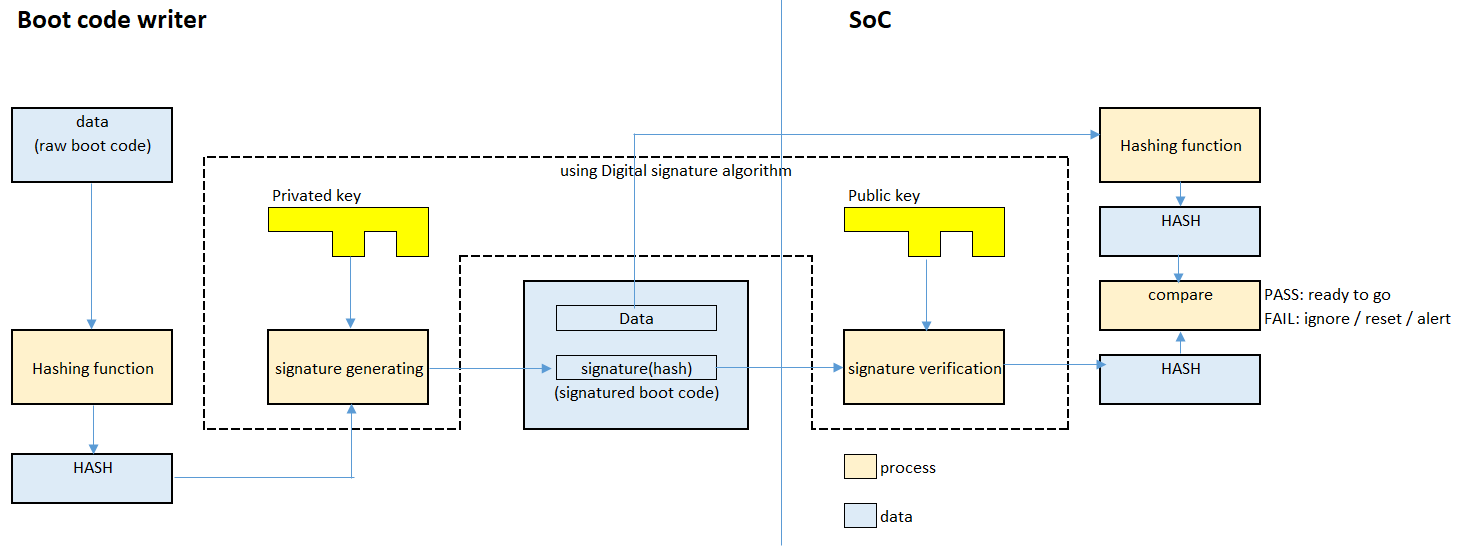
“Root-of-trust-device” là phần cứng lưu trữ mã code của phase ZERO, đông thời kiểm tra tính xác thực của mã code các phase sau, tạo ra một “chain-of-trust” trong quá trình boot.



Việc kiểm tra tính xác thực của các đoạn ma code được thực hiện các thuật toán chữ ký số (Digital signature algorithm). Các thuật toán này còn được gọi là thuật toán mã hóa bất đối xứng (*asymmetric cryptographic algorithm*).

1. **Cơ bản về kiểm tra tính xác thực của boot code bằng thuật toán chữ ký số**

**Dưới đây là quy trình kiểm tra tính xác thực của boot code trong secure boot:**

****

**Về phía người viết Boot code**

* Mã code được băm (hash)
* Mã hash được mã hóa (signature generating) bằng chìa khóa bí mật (private key). Chỉ có người viết code hợp lệ mới có chìa khóa bí mật này. Chìa khóa bí mật tuyệt đối không được rò rĩ ra bên ngoài
* Mã code ban đầu + chữ ký được lưu trữ trong bộ nhớ ngoài, chờ SoC xử lý

**Về phía SoC**

* Dùng khóa công cộng (Public key) để giải mã chữ ký, cho ra một HASH(1).
* Băm mã code sau khi load lên từ bộ nhớ ngoài, tao thành HASH(2)
* So sánh HASH(1) & HASH(2), nếu hai HASH khớp nhau, thì đoạn mã boot được xác thực

Việc sinh ra Public key , private key , mã hóa hash (tạo chữ ký) , giải mã hash được thực hiện bởi một trong các thuật toán Mã Hóa Bất Đối Xứng. Có hai thuật toán mã hóa bất đối xứng thông dụng:

* RSA (Rivest–Shamir–Adleman)
* ECDSA **(**Elliptic Curve Digital Signature Algorithm**)**

Theo đánh giá, ECDSA có độ bảo mật cao hơn, nhưng phức tạp hơn

1. **Thuật Toán RSA**

**Cách tạo public key và private key**

Mấu chốt cơ bản của việc sinh khóa trong RSA là tìm được bộ 3 số tự nhiên e, d và n sao cho:

mned

và một điểm không thể bỏ qua là cần bảo mật cho d sao cho dù biết e, n hay thậm chí cả m cũng không thể tìm ra d được.

cụ thể, khóa của RSA được sinh như sau:

* Chọn 2 số nguyên tố p và q
* Tính n = pq. Sau này, n sẽ được dùng làm modulus trong cả public key và private key.
* Tính một số giả nguyên tố bằng [phi hàm Carmichael](http://math-it.org/Mathematik/Zahlentheorie/Carmichael.html) như sau: λ(n) = BCNN(λ(p), λ(q)) = BCNN(p − 1, q − 1). Giá trị này sẽ được giữ bí mật.
* Chọn một số tự nhiên e trong khoảng (1, λ(n)) sao cho ƯCLN(e, λ(n)) = 1, tức là e và λ(n) nguyên tố cùng nhau.
* Tính toán số d sao cho d ≡ 1/e (mod λ(n)) hay viết dễ hiểu hơn thì de ≡ 1 (mod λ(n)). Số d được gọi là số nghịch đảo modulo của e (theo modulo mod λ(n)).

**Public key sẽ là bộ số (n, e),** và **private key sẽ là bộ số (n, d).** Chúng ta cần giữ private key thật cẩn thận cũng như các số nguyên tố p và q vì từ đó có thể tính toán các khóa rất dễ dàng.

Trong thực hành, chúng ta thường chọn e tương đối nhỏ để việc mã hóa và giải mã nhanh chóng hơn. Giá trị thường được sử dụng là e = 65537. Ngoài ra, chúng ta có thể tính số giả nguyên tố bằng [phi hàm Euler](http://math-it.org/Mathematik/Zahlentheorie/Euler.html) φ(n) = (p − 1)(q − 1) và dùng nó như λ(n). Vì φ(n) là bội của λ(n) nên các số d thỏa mãn de ≡ 1 (mod φ(n)) cũng sẽ thỏa mãn d ≡ 1/e (mod λ(n)). Tuy nhiên, một số tác dụng phụ của việc này là d thường sẽ trở nên lớn hơn mức cần thiết.

**Mã Hóa và giải mã:**

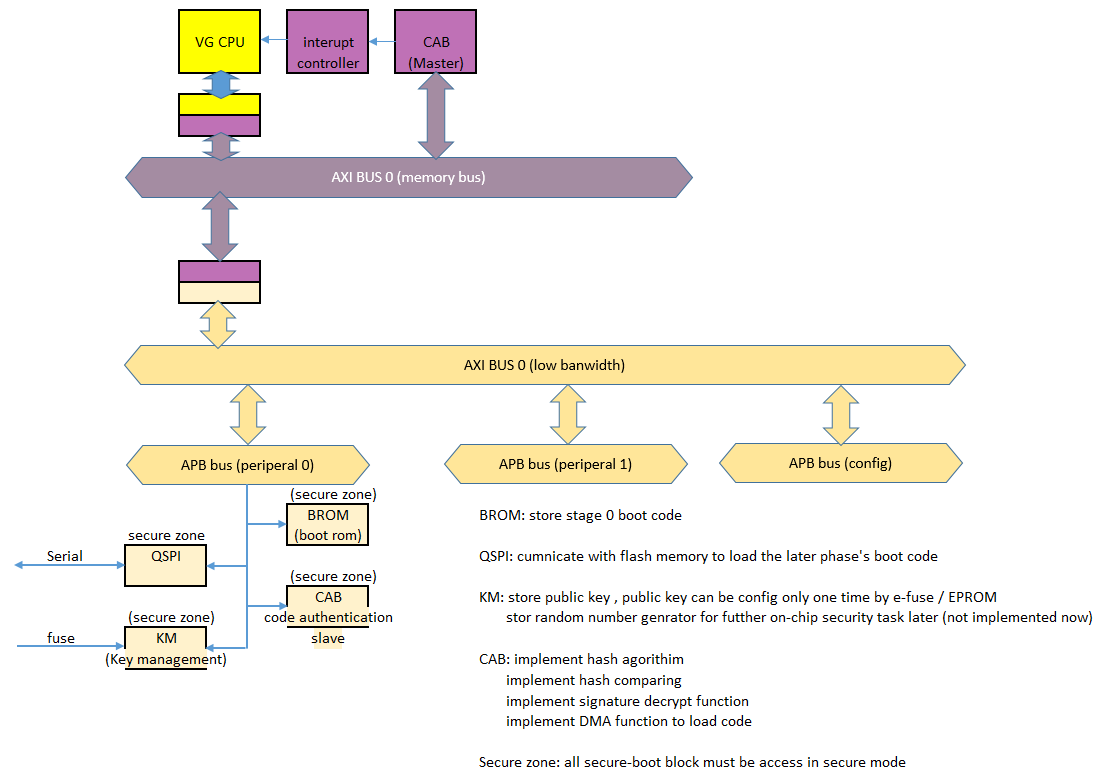
**Mã hóa bằng Private key: T.B.D**

**Giải mã bằng public key: T.B.D**

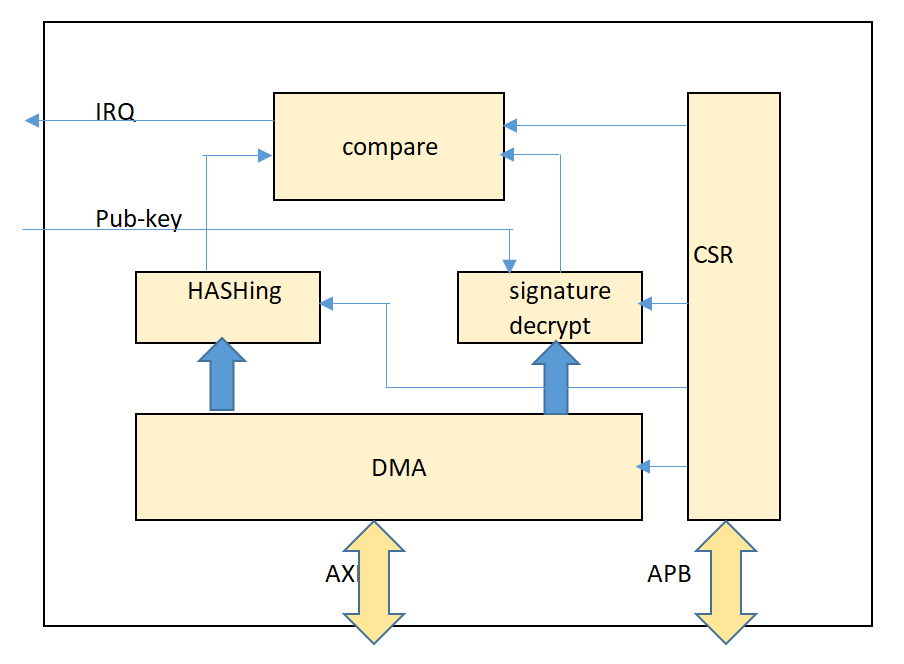
1. **Thuật toán HASH**

**T.B.D**

1. **Hệ thống secure boot của VANGUARD**

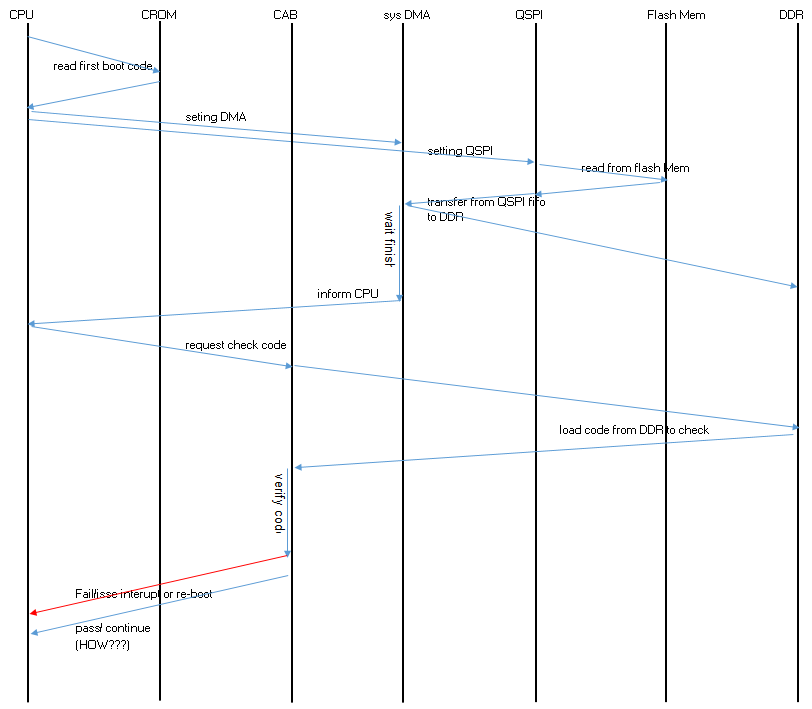
****

**CAB:**

****

1. **Quy trình secure boot của VANGUARD**

Quy trình read boot code stage 1 được định nghĩa trong boot code stage 0 (BROM)

****

**A.I**

* Nghiên cứu sâu RSA và ECDSA (Elliptic Curve Digital Signature Algorithm). Thực hiện design spec cho khối CAB
* Nghiên cứu e-fuse hoặc tính năng tương tự trên FPGA
* Nghiên cứu Open SBI
* Nghiên cứu các mode đặc quyền của RISC-V, và sự tương thích của nó với AMBA BUS
* Define chi tiết chương trình Boot ROM