NØNOS Bootloader Architecture

UEFI Secure Boot • Ed25519 Signatures • Groth16 ZK Proofs • 185 Rust Files

nonos-boot/src/

loader/
ELF64 Parser
Segment Loading
Relocation (PIE)
Memory Allocation
42 files

crypto/
Ed25519 Verify
BLAKE3 Hashing
Production Keys
KeyStore (16 max)
4 files

zk/
Groth16 Verify
BLS12-381 Pairing
Circuit Registry
Boot Attestation
14 files

entropy/
RDRAND/RDSEED
TSC Jitter
UEFI RNG Protocol
RTC Entropy
6 files

security/
Secure Boot Check
TPM PCR Extend
Platform Security
Crypto Self-Test
7 files

network/
PXE Boot
TFTP Download
HTTP Fetch
DHCP/Static IP
15 files

handoff/
BootHandoffV1
Memory Map Copy
ExitBootServices
Kernel Jump
6 files

verify/
Capsule Validation
Signature Check
Hash Verification
Integrity Chain
4 files

boot/
Logo Display
Phase Banners
Status Output
9 files

config/
NVRAM Settings
Boot Policies
Configuration
7 files

hardware/
CPU Detection
ACPI Tables
Memory Info
7 files

capsule/
Capsule Format
ZK Metadata
Kernel Extract
3 files

multiboot/
NVRAM Prefs
Boot Manager
Entry Selection
4 files

secure_loader/
Secure Loading
Measurements
Validation
14 files

crypto_visual/
BLAKE3 Display
Ed25519 Visual
ZK Status
13 files

testing/
Test Framework
Memory Tests
Crypto Tests
12 files

Boot Flow

UEFI Entry → Security Init → Load Kernel → Verify Sig → ZK Verify → Collect Entropy → ExitBootSvc → Jump to Kernel

```
185 Rust Files | 20 Subsystems | 3 Ed25519 Keys
3 ZK Circuits | 7 Domain Strings | 440 NONOS Refs
```

```
Magic: 0x4E4F4E4F ("NONO")
Handoff: BootHandoffV1 + 32KB Stack
```

```
Target: NØNOS Kernel
extern "C" fn(BootHandoffV1*)
```

Core Subsyster Support Module
UEFI/Kernel Interface

UEFI Firmware

Boot Services | Runtime Svc | GOP | File System | RNG | PXE Base Code | Secure Boot | TPM | Memory Map | ACPI | SMBIOS

NØNOS Boot Flow

UEFI Entry → Security Init → Kernel Load → Verify → Handoff → Jump

Phase 1: UEFI Initialization

1.1 EFI Entry Point
efi_main(Handle, SystemTable)
`main.rs:42`

1.2 Boot Services Access
Allocate memory, protocols
`BootServices API`

1.3 Display Logo
NØNOS ASCII art banner
`boot/logo.rs`

1.4 Security Init
Check Secure Boot, TPM
`security/init.rs`

1.5 Init Crypto
Load production keys
`crypto/keys.rs`

1.6 Crypto Self-Test
BLAKE3, Ed25519 verify
`security/crypto.rs`

Phase 2: Kernel Loading

2.1 Load Capsule
ESP or Network boot
`loader/file/file.rs`

2.2 Validate ELF
Magic, headers, segments
`loader/validate/`

2.3 Allocate Memory
AllocatePages for segments
`loader/memory/`

2.4 Copy Segments
LOAD segments to memory
`loader/segment/`

2.5 Relocations (PIE)
Fix R_X86_64_RELATIVE
`loader/reloc/reloc.rs`

2.6 Zero BSS
Clear uninitialized data
`write_bytes(0)`

Phase 3: Cryptographic Verification

3.1 Ed25519 Verify
Check 64-byte signature
`crypto/verify.rs`

3.2 ZK Proof Check
Parse ZK block if present
`zk/boot_attest.rs`

3.3 Groth16 Verify
BLS12-381 pairing check
`zk/zkverify.rs`

3.4 Compute Hash
BLAKE3 kernel hash
`SHA-256 for TPM`

3.5 TPM Extend
PCR measurement
`security/tpm.rs`

3.6 Build Results
CryptoHandoff struct
`handoff/types.rs`

Phase 4: Kernel Handoff

4.1 Collect Entropy
RDRAND + TSC + RTC
`entropy/collector.rs`

→ **4.2 Alloc Handoff**
BootHandoffV1 page
`+ 32KB stack`

→ **4.3 Collect Info**
FB, ACPI, SMBIOS
`handoff/config.rs`

→ **4.4 Populate**
Fill BootHandoffV1
`magic=0x4E4F4E4F`

→ **4.5 ExitBootServices**
Get final memory map
`No more UEFI calls!`

→ **4.6 Copy MemMap**
UEFI→BootHandoff
`MemoryMapEntry[]`

→ **4.7 JUMP!**
ASM transfer
`jmp kernel`

```
mov rdi, handoff_ptr ; arg0
mov rsp, stack_top ; new stack
jmp kernel_entry ; never returns
```

BootHandoffV1 Contents:
magic | version | flags | entry | fb | mmap | acpi
smbios | timing | meas | rng | zk | cmdline

Kernel Entry Signature:
`extern "C" fn(handoff: *const BootHandoffV1) -> !`
First instruction of NØNOS kernel

Boot Time Budget:
Phase 1: ~50ms | Phase 2: ~100ms
Phase 3: ~200ms | Phase 4: ~10ms

Handoff Flags:
`WX | NXE | SMEP | SMAP`
`SECURE_BOOT | ZK_ATTESTED`

ZK Boot Attestation

Groth16 ZK-SNARK Verification • BLS12-381 Pairing • Zero-Knowledge Boot Proof

ZK Proof Block Format

Magic	[4 bytes]	[0x4E, 0xC3, 0x5A, 0x50] "NoZP"
Version	[4 bytes]	1 (little-endian u32)
Program Hash	[32 bytes]	BLAKE3 circuit identifier
Capsule Commitment	[32 bytes]	BLAKE3 binding hash
Public Inputs Len	[4 bytes]	n * 32 (field elements)
Proof Blob Len	[4 bytes]	192 (Groth16 compressed)
Public Inputs	[n bytes]	Fr field elements 32-byte aligned
Groth16 Proof	[192 bytes]	A (G1) + B (G2) + C (G1) 48 + 96 + 48 = 192 bytes compressed

Verification Pipeline

1. **Parse ZK Block** — Locate magic, extract fields — `parse_zk_proof()`
2. **Validate Sizes** — proof=192, inputs%32==0 — `MAX_INPUT_SIZE=256KB`
3. **Compute Binding** — BLAKE3 domain-separated — `select_binding()`
4. **Verify Commitment** — constant-time compare — `ct_eq32(&local, &capsule)`
5. **Lookup VK** — Find verifying key by hash — `registry::lookup()`
6. **Deserialize VK** — arkworks CanonicalDeserialize — `VerifyingKey<Bls12_381>`
7. **Deserialize Proof** — A ∈ G1, B ∈ G2, C ∈ G1 — `Proof { a, b, c }`
8. **Parse Inputs** — Convert to Fr elements — `Fr::from_be_bytes_mod_order`
9. **Prepare VK** — Precompute pairings — `prepare_verifying_key()`
10. **Groth16 Pairing Check** — $e(A, B) = e(\alpha, \beta) \cdot e(\sum a_i \cdot L_i, \gamma) \cdot e(C, \delta)$ — `Groth16::verify_proof(&pvk, &proof, &inputs)` — BLS12-381 optimal ate pairing

Circuit Registry

boot-authority Kernel boot authorization `BootAuthority | Attestation`	**update-authority** System update authorization `UpdateAuthority`
recovery-key Recovery operation auth `RecoveryKey`	**Verification Key** 568+ bytes (BLS12-381) Compressed canonical form

`ENTRIES: &[(&[u8; 32], &[u8])] = program_hash → vk_bytes`

Domain Separation Strings

Program Hash:	`"NONOS:ZK:PROGRAM:v1"`
Commitment:	`"NONOS:CAPSULE:COMMITMENT:v1"`
Circuit Key:	`"NONOS:CIRCUIT_KEY:v1"`

All use `BLAKE3::new_derive_key()` for domain separation

Verification Result

ZkVerifyResult::Valid Proof verified successfully	**ZkVerifyResult::Invalid** Proof or commitment failed
ZkVerifyResult::Unsupported Unknown program hash	**ZkVerifyResult::Error** Deserialization or internal

Passed to Kernel (BootHandoffV1.zk)

ZkAttestation Struct

verified:	u8 (1 = valid)
flags:	u8 (reserved)
program_hash:	[u8; 32]
capsule_commitment:	[u8; 32]

BootAttestationResult

zk_verified:	bool
program_hash:	[u8; 32]
capsule_commitment:	[u8; 32]
detail:	ZkVerifyResult

Security Properties

✓ Zero-Knowledge	Proof reveals nothing about witness
✓ Soundness	Cannot forge proof without witness
✓ Binding	Commitment ties proof to kernel
✓ Non-malleable	Proof cannot be modified
✓ Constant-time	Side-channel resistant comparison
✓ Zeroization	Proof wiped after verification

NØNOS Build Tools

nonos-boot/tools/ • Key Generation → Kernel Signing → ZK Proof Embedding

1 Key Generation

Create Ed25519 signing keypairs

keygen/

```
nonos-keygen --count 4 --out-dir keys/
--signers signers.json --threshold 2
--allow-write-secrets
```

Output Files:

keys/signer1.key	32-byte Ed25519 seed
keys/signer1.pub.hex	Public key (hex)
keys/signers.json	Threshold config
keys/generation_log.json	Audit trail

Fingerprints: SHA-256, BLAKE3
Atomic writes, zeroization

.key

2 Kernel Signing

Sign ELF with Ed25519

sign-kernel/

```
nonos-sign-kernel
--key keys/signer1.key
--input kernel.elf --output kernel.signed
```

Output Format:

[kernel_data]	Original ELF binary
[signature]	64-byte Ed25519 sig

Signed over: kernel_data
Prints: public key, BLAKE3 hash
Rust array format for embedding

.signed

3 ZK Proof Embedding

Append Groth16 proof block

embed-zk-proof/

```
embed-zk-proof
--input kernel.signed --proof proof.bin
--program-id "boot-attest-v1" --output final
```

Output Format:

[kernel_data]	Original ELF
[Ed25519 sig]	64 bytes
[ZK Block]	80 + inputs + 192

Magic: 0x4E 0xC3 0x5A 0x50 "NøZP"
Program hash: BLAKE3 derived
Commitment: binds proof to kernel

Final Capsule

Ready for NØNOS Boot

Kernel ELF Binary
~2-10 MB typical

Ed25519 Signature [64 bytes]

ZK Proof Block
Magic + Version [8 bytes]
Program Hash [32 bytes]
Commitment [32 bytes]
Lengths [8 bytes]
Public Inputs [n bytes]
Groth16 Proof [192 bytes]

Supporting Tools

zk-embed/

Derive PROGRAM_HASH from circuit ID
Emit VK bytes as Rust constants

nonos-attestation-circuit/

R1CS circuit definition
Prove boot authorization

zk-ceremony/

Trusted setup scripts
Powers of tau, phase 2

generate_keys.rs

Create proving/verifying keys
BLS12-381 Groth16 params

Bootloader Integration

Embedding in src/crypto/keys.rs

```
pub const NONOS_SIGNING_KEY: &[u8; 32] = &[
0x57, 0x9a, 0xdc, 0xf9, ...
];
// src/zk/registry/keys.rs
pub const VK_BOOT_AUTHORITY_BLS12_381_GROTH16: &[u8] = ...
pub const PROGRAM_HASH_BOOT_AUTHORITY: [u8; 32] = ...
```

Build Commands

Full build pipeline:

```
# 1. Generate keys (one-time ceremony)
cd tools/keygen && cargo run -- --count 3
# 2. Build and sign kernel
cargo build --release --target x86_64-unknown-uefi
cd tools/sign-kernel && cargo run -- ...
# 3. Embed ZK proof (if available)
cd tools/embed-zk-proof && cargo run -- ...
```

NONOS Bootloader: Cryptographic Subsystem

Ed25519 Signatures + BLAKE3 Hashing + Domain Separation

KEYSTORE

src/crypto/keys.rs

KeyStore Structure

```
keys: [[u8; 32]; 16] // MAX_KEYS
count: usize

static KEYSTORE: Mutex<KeyStore>
static INIT_DONE: AtomicBool
```

Production Keys (Ed25519)

Key 1 (Primary):
579adc...9b7ffb (NONOS_SIGNING_KEY)

Key 2 (Backup):
3d4017...f4660c

Key 3 (Recovery):
fc51cd...908025

Embedded at compile time

BLAKE3 HASHING

Domain Separation Strings

KeyID Derivation:
"NONOS:KEYID:ED25519:v1"
derive_keyid(pubkey) -> [u8; 32]

Entropy Accumulation:
"NONOS:BOOT:ENTROPY:ACCUM"
collect_boot_entropy() mixing

Entropy Output:
"NONOS:BOOT:ENTROPY:OUTPUT"
Final entropy extraction (XOF)

ZK Program Hash:
"NONOS:ZK:PROGRAM:v1"
Circuit VK normalization

ZK Commitment:
"NONOS:CAPSULE:COMMITMENT:v1"

SIGNATURE VERIFICATION

src/crypto/verify.rs

verify_signature_bytes()

1. Check sig.len() == 64 (SIG_LEN)
2. Check is_initialized()
3. Parse Ed25519 Signature
4. For each key in KEYSTORE:
   - VerifyingKey::from_bytes()
   - pk.verify(data, &sig)
5. Return KeyId on success

verify_signature_full()

1. Extract sig from blob[meta.offset]
2. Extract payload from blob
3. Reject all-zero signatures
4. Call verify_signature_bytes()

Uses ed25519-dalek crate

CRYPTO SELF-TEST

perform_crypto_self_test()

BLAKE3 Test:
hash("NONOS-crypto-selftest") x2
Verify h1 == h2 (determinism)

Ed25519 Test:
VerifyingKey::from_bytes(NONOS_KEY)

VERIFY ERRORS

VerifyError enum:

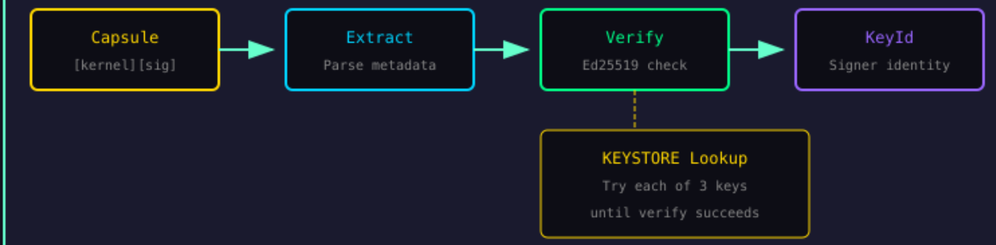
- Bounds - offset/length overflow
- MalformedSignature - bad format
- InvalidSignature - verify failed
- KeyNotFound - no matching key
- NotInitialized - keystore empty

CAPSULE METADATA

CapsuleMetadata struct:

```
offset_sig: usize // Signature location
len_sig: usize // 64 bytes (SIG_LEN)
offset_payload: usize // Kernel data start
len_payload: usize // Kernel size
signer_keyid: Option<KeyId>
payload_hash: [u8; 32] // BLAKE3 hash
header_version: u32, header_timestamp: u64
```

SIGNATURE VERIFICATION FLOW



CRYPTO STATISTICS

Algorithms:

- Ed25519 - 256-bit elliptic curve signatures
- BLAKE3 - Fast cryptographic hash (XOF)
- SHA-256 - TPM PCR measurements

Key Properties:

- Public Key: 32 bytes (PK_LEN)
- Signature: 64 bytes (SIG_LEN)
- KeyId: 32 bytes (BLAKE3 derived)

Cryptographic Foundation: Zero external network calls | All keys embedded at compile time | Domain-separated hashing prevents cross-protocol attacks

Dependencies: ed25519-dalek, blake3, sha2 | Crates verified against known hashes

NONOS Bootloader: Entropy Collection

Multi-Source Hardware Entropy for Kernel RNG Seed

EFI_RNG PROTOCOL

#[cfg(feature = "efi-rng")]

```
collect_efi_rng()
1. bs.locate_protocol::<Rng>()
2. rng.get_rng(None, &mut buf[64])
3. h.update(&buf)
4. scrub(&mut buf) // Zero after use
```

UEFI firmware hardware RNG

RDRAND / RDSEED

Intel/AMD Hardware RNG Instructions

```
collect_hw_rng_bytes()
HW_RNG_ITERATIONS = 32
For each iteration:
  - Try RDSEED first (true entropy)
  - Fallback to RDRAND (PRNG)
  - Accumulate into 64-byte buffer
```

CPU instruction-level entropy

TSC JITTER

Timing Variations from CPU

```
collect_tsc_jitter()
TSC_JITTER_ROUNDS = 256
For each round:
  - t1 = rdtsc_serialized()
  - bs.stall(23 + (round*7 % 17))
  - t2 = rdtsc_serialized()
  - delta = t2 - t1 (with rotation)
```

Unpredictable timing variations

RTC ENTROPY

Real-Time Clock Data

```
collect_rtc_entropy()
st.runtime_services().get_time()
16-byte buffer contains:
  - year (4B) + month + day
  - hour + minute + second
  - nanosecond (4B) + timezone
  - daylight savings flag
```

Nanosecond precision adds entropy

BLAKE3 ENTROPY MIXING

Domain-Separated Accumulation and Output Extraction

Accumulation Phase

"NONOS:BOOT:ENTROPY:ACCUM"

```
blake3::Hasher::new_derive_key(DS_ENTROPY_ACCUM)
```

Output Phase (XOF)

"NONOS:BOOT:ENTROPY:OUTPUT"

```
finalize_xof().fill(&mut out[64]) // 64 bytes
```

WEAK ENTROPY RESCUE

Fallback when entropy quality is low

```
rescue_weak_entropy()
1. rescue = rdtsc_serialized()
2. BLAKE3(current_out || rescue) -> out[0..32]
3. XOF(hash || rescue) -> out[32..64]
```

KERNEL HANDOFF

Seed delivered to kernel RNG

```
seed_entropy()
collected = collect_boot_entropy(bs)
info.entropy64.copy_from_slice(&collected)
scrub(&mut collected) // Zero local copy
```

RTC TIMESTAMP

8-byte timestamp for misc use

```
get_rtc_timestamp() -> [u8; 8]
year[2] + month + day
hour + minute + second
milliseconds (from nanosecond)
```

CONSTANTS

```
HW_RNG_ITERATIONS = 32
TSC_JITTER_ROUNDS = 256
Output Size = 64 bytes
scrub() zeros sensitive data
```

ENTROPY FLOW SUMMARY

EFI_RNG (64B) + RDRAND/RDSEED (64B) + TSC Jitter (256 rounds) + RTC (16B)

-> BLAKE3 Accumulator -> Domain-Separated XOF -> 64-byte seed -> BootHandoffV1.rng.seed32 -> Kernel CSPRNG

NONOS Bootloader: Security Subsystem

TPM Measured Boot + UEFI Secure Boot + Self-Tests

TPM 2.0 MEASURED BOOT

src/security/tpm.rs

PCR Indices

pcr::BOOTLOADER = 8

pcr::KERNEL = 9

pcr::CAPSULE = 14

Standard TPM 2.0 PCRs

TCG2 Protocol

GUID: 607f766c-7455-42be-930b-e4d76db2720f

EV_POST_CODE = 0x00000001

HashLogExtendEvent offset: 0x18

extend_pcr_measurement()

1. Validate data not empty, pcr_index <= 23
2. SHA-256 hash of measurement data
3. locate_tcg2_protocol(bs)
4. extend_pcr_via_tcg2(tcg2, pcr, digest)

-> HashLogExtendEvent(flags=0x10, digest, event)

5. Return true on success, false if no TPM

UEFI SECURE BOOT CHECKS

src/security/check.rs

Platform Checks

check_secure_boot()
UEFI SecureBoot variable

check_platform_key()
PK enrolled status

check_signature_db()
db/dbx databases

Hardware Checks

check_hardware_rng()
RDRAND/RDSEED available

check_measured_boot()
TPM2 availability

All checks return bool

SecurityContext

```
pub struct SecurityContext {
    secure_boot_enabled: bool,
    measured_boot_available: bool,
    hardware_rng_available: bool,
    platform_key_enrolled: bool,
}
```

CRYPTOGRAPHIC SELF-TESTS

src/security/crypto.rs

blake3_selftest()

test_data = b"NONOS-BLAKE3-SELFTEST"

hash1 = blake3::hash(test_data)

hash2 = blake3::hash(test_data)

return hash1 == hash2 // Determinism check

ed25519_selftest()

VerifyingKey::from_bytes(NONOS_SIGNING_KEY)

return result.is_ok() // Key parseable

run_all_selftests()

blake3_selftest() && ed25519_selftest()

Called during boot initialization

MEASUREMENT FLOW

measure_boot_components()

PCR8: extend(bootloader_hash) // Self measurement

PCR9: extend(kernel_hash) // Kernel binary

PCR14: extend(capsule_hash) // Full capsule

Returns true only if ALL succeed

SIGNATURE VERIFICATION

verify_kernel_signature_advanced()

1. Extract signature from capsule
2. Hash kernel payload with BLAKE3
3. Ed25519 verify against keystore

Returns (valid: bool, keyid: Option<KeyId>)

SECURITY POSTURE

assess_security_posture()

FULL: SecureBoot + TPM + HW-RNG + PK

PARTIAL: Some but not all features

MINIMAL: Basic signature verification only

NONE: No security features available

Security Defense Layers

Layer 1: UEFI Secure Boot (firmware) | Layer 2: Ed25519 Signature (bootloader) | Layer 3: TPM Measurements (hardware) | Layer 4: ZK Attestation (cryptographic)

NONOS Bootloader: Network Boot

PXE/TFTP + HTTP Boot + DHCP Configuration

MODULE STRUCTURE

src/network/mod.rs

Core Modules:

- dhcp_core // DHCP protocol
- tftp // TFTP client
- http // HTTP client
- pxe // PXE protocol

Support Modules:

- retry // Retry logic
- signature // Verify downloads
- static_ip // Manual config
- diagnostics // Network debug

PXE / TFTP BOOT

Traditional Network Boot

PXE Functions:

```
is_pxe_available() -> bool
get_tftp_server_from_dhcp() -> Ipv4
get_boot_filename_from_dhcp() -> &str
```

TFTP Download:

```
tftp_download(server, filename, buf)
-> TftpDownload { bytes_read, status }
```

Kernel Fetch:

```
fetch_kernel_via_pxe(bs, buf)
-> Result<usize, NetworkError>
```

HTTP BOOT

Modern Network Boot

URL Parsing:

```
parse_url(url) -> (host, port, path)
```

HTTP Response:

```
HttpResponse {
    status_code: u16,
    content_length: usize,
    body: &[u8]
}
```

Kernel Fetch:

```
fetch_kernel_via_http(bs, url, buf)
```

DHCP CONFIGURATION

IP Address Assignment

DhcpConfig:

```
ip_address: Ipv4Addr
subnet_mask: Ipv4Addr
gateway: Ipv4Addr
dns_server: Ipv4Addr
```

Functions:

```
configure_dhcp_via_pxe()
get_dhcp_assigned_ip()
is_dhcp_configured() -> bool
```

Static IP Alternative:

```
set_static_ip(StaticIpConfig)
```

RETRY CONFIGURATION

RetryConfig:

- DEFAULT_MAX_RETRIES = 3
- DEFAULT_INITIAL_DELAY_US = 100_000
- DEFAULT_MAX_DELAY_US = 5_000_000
- NET_TIMEOUT_SECS = 30

Exponential backoff with jitter

DOWNLOAD VERIFICATION

verify_downloaded_kernel():

1. Compute BLAKE3 hash of kernel
2. Fetch .sig file from server
3. Ed25519 verify(kernel, sig)
4. Return verified kernel bytes

Same verification as local boot

NETWORK BOOT CONTEXT

NetworkBootContext:

```
boot_option: NetworkBootOption
config: NetworkConfig
kernel_url: Option<String>
signature_url: Option<String>
timeout_secs: u32
```

NetworkBootOption:

- PxeTftp // Traditional PXE
- HttpBoot // UEFI HTTP Boot
- DhcpOnly // Config only
- Disabled // Local boot

NETWORK BOOT FLOW

MAX_KERNEL_SIZE = 64 MB | NET_MAX_RETRIES = 3 | NET_TIMEOUT_SECS = 30 | Supports UEFI HTTP Boot & Legacy PXE

NONOS Bootloader: Kernel Handoff

BootHandoffV1 Structure + Assembly Transfer Protocol

BootHandoffV1 Structure

src/handoff/types.rs – #[repr(C)]

Header (16 bytes)

```
magic: u32 = 0x4E4F4E4F // "NONO"
version: u16 = 1
size: u16 = size_of::<Self>()
```

Flags (u64) + Entry Point

```
WX (1<<0) | NXE (1<<1) | SMEP (1<<2) | SMAP (1<<3)
UMIP (1<<4) | IDMAP_PRESERVED (1<<5) | FB_AVAILABLE (1<<6)
ACPI_AVAILABLE (1<<7) | TPM_MEASURED (1<<8)
SECURE_BOOT (1<<9) | ZK_ATTESTED (1<<10)
entry_point: u64 – Kernel entry address
```

FramebufferInfo

```
ptr: u64 // FB base
size: u64 // Total bytes
width: u32 // Pixels
height: u32 // Pixels
stride: u32 // Bytes/row
pixel_format: u32
```

MemoryMap

```
ptr: u64 // Entries base
entry_size: u32
entry_count: u32
desc_version: u32

MemoryMapEntry:
    type,phys,virt,pages,attr
```

Firmware Tables

```
acpi.rsdp: u64 // RSDP address
smbios.entry: u64 // SMBIOS entry
```

Timing

```
tsc_hz: u64 // TSC freq
unix_epoch_ms: u64
```

Modules + Cmdline

```
modules: { ptr: u64, count: u32 } // Module array
cmdline_ptr: u64 // Kernel command line
```

BootHandoffV1::is_valid()

magic == 0x4E4F4E4F && version == 1 && size == sizeof(Self)

SECURITY HANDOFF

Measurements

```
kernel_sha256: [u8; 32] // Kernel hash
kernel_sig_ok: u8 // Signature valid
secure_boot: u8 // UEFI SecureBoot
zk_attestation_ok: u8 // ZK proof valid
```

ZkAttestation

```
verified: u8 // ZK verified flag
flags: u8 // Attestation flags
program_hash: [u8; 32] // Circuit identity
capsule_commitment: [u8; 32]
```

RNG SEED

```
rng.seed32: [u8; 32] // Boot entropy
```

From multi-source entropy collection

CryptoHandoff

Internal transfer structure

```
signature_valid: bool
secure_boot: bool
kernel_hash: [u8; 32]
zk_attested: bool
zk_program_hash, zk_capsule_commitment
```

ASSEMBLY KERNEL ENTRY

src/handoff/exit.rs – x86_64 Long Mode Transfer

jump_to_kernel()

```
// Calling convention: SystemV AMD64
mov rdi, {handoff} // Arg 1: handoff ptr
mov rsp, {stack} // New stack pointer
xor rbp, rbp // Clear frame pointer
jmp {entry} // Jump to kernel

// Never returns (-> !)
// Kernel: extern "C" fn(u64) -> !
```

Kernel Entry Signature

```
pub type KernelEntry =
    extern "C" fn(u64) -> !
```

Kernel receives:

- RDI = BootHandoffV1 pointer
- RSP = Fresh 16KB stack
- RBP = 0 (no frame chain)

Long mode, paging enabled, no interrupts

HANDOFF_MAGIC = 0x4E4F4E4F ("NONO") | HANDOFF_VERSION = 1 | Full hardware state passed to kernel

NONOS Bootloader: ELF Loader Subsystem

ET_EXEC + ET_DYN Loading with Relocation Support

LOADER MODULES

src/loader/

Core:

- core/load.rs // Main loader
- core/alloc.rs // Page alloc

Validation:

- validate/mod.rs // ELF checks
- validate/program.rs

Support:

- reloc.rs, image.rs, errors.rs

ELF VALIDATION

validate_elf() -> ValidationResult

Header Checks:

- Magic: 0x7F 'E' 'L' 'F'
- Class: ELFCLASS64
- Machine: EM_X86_64
- Type: ET_EXEC or ET_DYN

Segment Validation:

- PT_LOAD segments only
- Bounds checking, no overlaps

LOAD STRATEGIES

ET_EXEC (Static)

load_exec_kernel()

AllocateType::Address(base) - Fixed address

No relocations needed

ET_DYN (PIE)

load_dyn_kernel()

AllocateType::AnyPages - Relocatable

process_elf_relocations() applied

MEMORY ALLOCATION

UEFI Boot Services

Constants:

- PAGE_SIZE = 4096
- MAX_ALLOCS = 64

Functions:

- record_alloc(addr, pages)
- free_all(bs, allocations)

MemoryType::LOADER_DATA

KERNEL LOAD FLOW

1. Capsule Load — load_validated_capsule() — Verify sig, extract payload or pass raw ELF

→ **2. Validate ELF** — validate_elf(payload) — Check header, segments Calculate min/max addr

→ **3. Allocate Pages** — bs.allocate_pages() — ET_EXEC: fixed addr ET_DYN: any pages

→ **4. Copy Segments** — copy_nonoverlapping() — Load PT_LOAD segments Zero BSS regions

→ **5. Relocate** — process_elf_relocations() — ET_DYN only: apply R_X86_64_RELATIVE

→ **6. Return Image** — KernelImage { ... } — address, size, entry metadata, allocations

Segment Copy (for each PT_LOAD)

```
copy_nonoverlapping(payload[p_offset], dst, p_filesz)
write_bytes(dst + p_filesz, 0, p_memsz - p_filesz) // BSS
```

Relocation Processing

```
load_bias = base_phys - min_addr
R_X86_64_RELATIVE: *ptr += load_bias
```

Entry Point Validation

```
entry >= base && entry < base + total_bytes
Err(LoaderError::EntryNotInRange) on failure
```

KernelImage Structure

```
pub struct KernelImage {
    address: usize, // Load address
    size: usize, // Total size
    entry_point: usize, // Entry address
    metadata: CapsuleMetadata, allocations, alloc_count
}
```

ValidationResult

```
pub struct ValidationResult {
    elf: Elf64, // Parsed ELF
    is_exec: bool, // ET_EXEC or ET_DYN
    min_addr, max_addr: u64, // Address range
    loads: [LoadSegment; 16], load_count: usize
}
```

LoaderError

InvalidElfMagic, InvalidElfClass

InvalidElfMachine, InvalidElfType

TooManyLoadSegments, SegmentOverlap

AllocationFailed, EntryNotInRange

Supports 64-bit ELF | ET_EXEC (fixed) + ET_DYN (PIE) | UEFI page allocation | R_X86_64_RELATIVE relocations

NØNOS Kernel Architecture

x86_64 Platform Abstraction Layer • ZeroState Privacy OS

NØNOS Kernel Subsystems

process/
Process Control Block
Context Switching
Uses: GDT, IDT, Syscall

memory/
Page Tables (4-Level)
Physical/Virtual Alloc
Uses: CPU, ACPI

sched/
Task Scheduling
Timer Interrupts
Uses: Time, IDT, APIC

interrupts/
IRQ Management
Handler Dispatch
Uses: IDT, APIC

syscall/
System Call Interface
User/Kernel Transition
Uses: arch::syscall

drivers/
Device Drivers
Hardware Abstraction
Uses: PCI, Port, DMA

fs/
Filesystem (VFS)
RAM-only Mode
Uses: Memory, Storage

security/
Capabilities
Audit, Hardening
ZeroState Security

crypto/
Ed25519, BLAKE3
ML-KEM, ML-DSA
Post-Quantum Ready

smp/
Multi-Processor
AP Startup, IPI
Uses: CPU, APIC, GDT

modules/
Signed Capsules
Sandbox, Registry
ZeroState Modules

ipc/
Inter-Process Comm
Shared Memory, Signals
Uses: Memory, Syscall

net/
Network Stack (smoltcp)
TCP/IP, Sockets
Uses: PCI, IRQ, DMA

ui/
User Interface
TUI, Graphics
Uses: VGA, Input

94 Modules | 15+ Subsystems | 256 IRQ Vectors | 6 IST Stacks | 256 Max CPUs | 4

Core Path Submodule Kernel Layer

arch/x86_64/

cpu/
CPUID Detection
Feature Flags
MSR Access
TSC Operations
Per-CPU Data

gdt/
Segments
TSS Management
IST Stacks
Selectors
SYSCALL Setup

idt/
256 Vectors
Exception Handlers
IRQ Handlers
IST Assignment
Handler Registry

interrupt/
Local APIC
x2APIC Support
I/O APIC
Legacy 8259A PIC
IPI / MSI-X

time/
TSC (Primary)
HPET
PIT (8254)
RTC (CMOS)
Unified API

syscall/
SYSCALL/SYSRET
MSR Config
Handler Dispatch
Security Policies
Statistics

acpi/
RSDP/XSDT
MADT (CPUs)
FADT (Power)
HPET/MCFG
NUMA Tables

I/O & Boot
serial/ (16550A UART)
vga/ (Text Mode)
keyboard/ (PS/2, USB)
pci/ (Config Space)
boot/ | multiboot/ | uefi/

Hardware Layer

CPU x86_64 | Memory | APIC/IOAPIC | PCI Bus | Timers | Serial/VGA | Keyboard | Storage | Network | ACPI/UEFI

NØNOS Boot Sequence

x86_64 Kernel Initialization Flow

PHASE 0
BOOTLOADER

1. **Load Kernel ELF**
   GRUB loads to memory
2. **Multiboot2 Parse**
   Boot info structure
3. **Enter Long Mode**
   Transition to 64-bit

PHASE 1
EARLY INIT

4. **Serial Init**
   COM1 @ 115200 baud
5. **CPU Detection**
   CPUID, SSE, AVX
6. **GDT + IDT**
   Segments, 256 vectors

PHASE 2
PLATFORM

7. **ACPI Parse**
   MADT, FADT, HPET
8. **APIC Setup**
   Local + I/O APIC
9. **Syscall MSRs**
   STAR, LSTAR

PHASE 3
SUBSYSTEMS

10. **Memory**
    Heap, page tables
11. **Scheduler**
    Process management
12. **Enable IRQs**
    Start scheduler

CPU Mode Transitions

Real Mode
16-bit, 1MB limit

→

Protected Mode
32-bit, GDT required

→

Long Mode
64-bit, paging on

→

KERNEL ACTIVE
All subsystems ready

BOOT TIME
~850ms

SYSTEM READY - NØNOS Kernel v0.8.0

CPU Subsystem

arch/x86_64/cpu/ - Detection, Features & Management

CPU MODULE STRUCTURE

cpuid.rs
CPUID instruction wrapper

features.rs
100+ CPU feature flags

msr.rs
Model-Specific Registers

tsc.rs
Time Stamp Counter

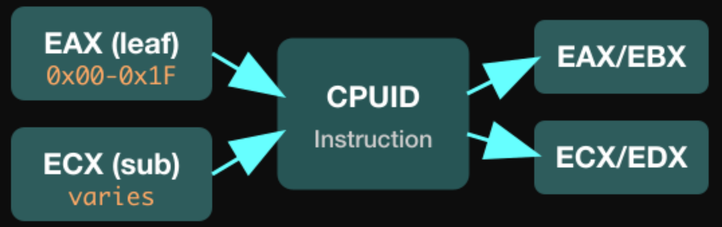
topology.rs
Core/Thread detection

per_cpu.rs
Per-CPU data (256 max)

KEY MSR REGISTERS

0xC0000080	IA32_EFER	Extended features
0xC0000081	IA32_STAR	SYSCALL segments
0xC0000082	IA32_LSTAR	SYSCALL entry
0xC0000100	IA32_FS/GS	Segment bases

CPUID INSTRUCTION FLOW



CPU TOPOLOGY

PACKAGE 0

Core 0
Thread 0, 1

Core 1
Thread 0, 1

CACHE HIERARCHY

L1
32 KB
~4 cycles

L2
256 KB
~12 cycles

L3
8-16 MB
~40 cycles

CPU FEATURES (100+ FLAGS)

SIMD
SSE, SSE2, SSE3
AVX, AVX2, AVX-512
FMA, BMI1, BMI2
+20 more

SECURITY
AES-NI, SHA
RDRAND, RDSEED
SMEP, SMAP, CET
+15 more

TIME STAMP COUNTER

```
cpu::rdtsc() // Fast read (~20 cycles)
cpu::rdtscp() // Serialized + CPU ID
```

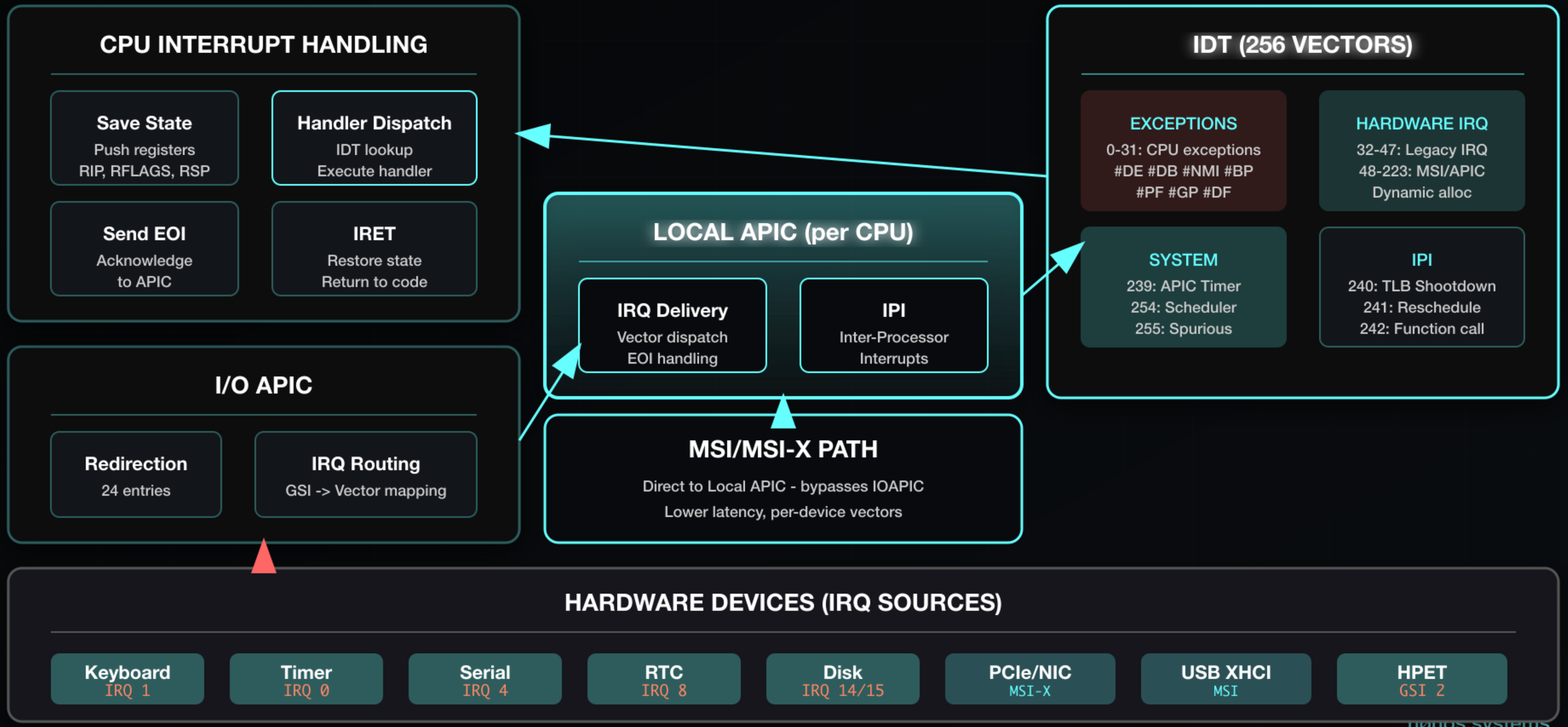
Invariant TSC: constant rate across P/C states

PER-CPU DATA

```
cpu_id: u32 // APIC ID
features: CpuFeatures // Cached
kernel_stack: u64 // via GS segment
```

Interrupt Architecture

arch/x86_64/interrupt/ - APIC, IOAPIC & IDT



Timer Subsystem

arch/x86_64/time/ - Multi-Source Timekeeping

CLOCK SOURCE HIERARCHY (Priority Order)

TSC | HPET | APIC Timer | PIT 8254 | RTC

TSC (Time Stamp Counter)

PRIMARY - Highest Resolution

```
cpu::rdtsc() // ~20 cycles
cpu::rdtscp() // Serialized + CPU ID
```

Resolution	Invariant TSC
~1 ns	Constant rate

HPET (High Precision)

SECONDARY - For Calibration

ACPI HPET table detection
Memory-mapped registers

Resolution	Frequency
~100 ns	14.3 MHz+

APIC Timer

PER-CPU - Scheduling

Local APIC timer interrupt
One-shot or periodic mode

Use Case	Mode
Preemption	TSC-deadline

UNIFIED TIME API

```
time::now() -> Instant
time::uptime() -> Duration
time::sleep(Duration)
time::delay_us(u64)
```

```
time::tsc_frequency() -> u64
time::calibrate_tsc()
time::monotonic_ns() -> u64
```

FALLBACK SOURCES

PIT 8254
Legacy timer
~840 ns
1.193 MHz

RTC (CMOS)
Wall clock time
1 second
Battery-backed

TSC Calib
PIT used to calibrate TSC frequency

SYSCALL / SYSRET Flow

arch/x86_64/syscall/ - User/Kernel Transition

USER SPACE (Ring 3)

CPL = 3

User Application

```
// libc wrapper
write(fd, buf, n) {
  syscall(SYS_write,
    fd, buf, n);
```

Register Setup

```
RAX = syscall #
RDI = arg1
RSI = arg2
RDX = arg3
R10/R8/R9 = 4-6
```

SYSCALL Instruction

KERNEL SPACE (Ring 0)

CPL = 0

SYSCALL Entry

```
RCX = user RIP (return addr)
R11 = user RFLAGS
```

Handler Dispatch

```
syscall_table[RAX](args)
// ~300 system calls
```

SYSRET

```
RAX = return value
Restore RIP, RFLAGS
```

MSR CONFIGURATION

IA32_STAR
Kernel/User CS/SS selectors

IA32_LSTAR
SYSCALL entry point (64-bit)

IA32_SFMASK
RFLAGS mask (clear IF, DF)

IA32_EFER
SCE bit enables SYSCALL

SYSCALL TABLE

File I/O read, write, open	**Process** fork, exec, exit	**Memory** mmap, brk, mprotect
Network socket, send, recv	**IPC** pipe, shmem, signal	**NØNOS** vault, crypto, zeroize

PERFORMANCE

SYSCALL/SYSRET: ~100-200 cycles | No stack switch needed | Faster than INT 0x80

Crypto Subsystem

Post-Quantum Ready Cryptographic Primitives

HASH FUNCTIONS (hash/)

- **BLAKE3** — Primary
- **SHA-512** — 512-bit
- **SHA3** — Keccak
- **SHA-256** — unified/
- **HMAC** — MAC
- **HKDF** — KDF
- **SHA-1** — Legacy only
- **Keccak256 / SHAKE128/256** — Ethereum compatible

SYMMETRIC (symmetric/)

- **ChaCha20-Poly1305** — NØNOS Primary AEAD
- **AES-256-GCM** — AEAD
- **AES-128/256** — Block cipher
- **Poly1305** — MAC

core/aead.rs - Unified AEAD traits

ASYMMETRIC (asymmetric/)

- **Ed25519** — Signatures
- **X25519** — Key exchange
- **P-256** — NIST curve
- **secp256k1** — Bitcoin/ETH
- **RSA** — Legacy
- **Curve25519** — DH base

core/traits.rs - Kem, Sig, Ed25519Sig traits

POST-QUANTUM (pqc/)

NIST PQC Standards Ready

- **ML-KEM (Kyber)** — kyber.rs - KEM
- **ML-DSA (Dilithium)** — dilithium.rs - Sig
- **SPHINCS+** — sphincs/
- **McEliece** — mceliece/
- **NTRU** — ntru/

quantum.rs - Hybrid PQ utilities

ZERO-KNOWLEDGE (zk/)

- **Groth16** — zkSNARK
- **Halo2** — Recursive
- **nonos_zk** — Attestation

zk_kernel/ - Native ZK Proofs

Pedersen | Schnorr | Sigma | Range | Equality | Membership | PLONK

syscall_zk_verify, syscall_zk_commit, syscall_zk_prove_*

APPLICATION (application/)

- **vault.rs** — Secure key storage
- **nonos_signing.rs** — Module signatures
- **ethereum/** — keccak256, ecrecover
- **certification/** — X.509 parsing

CORE API (core/)

- **api.rs** — init_crypto_subsystem generate_keypair, sign, verify
- **aead.rs** — Aead trait, aead_wrap/unwrap
- **syscall.rs** — syscall_blake3_hash, ...
- **traits.rs** — Kem, Sig, KyberKem, ...

UTILITIES (util/)

- **bigint/** — Arbitrary precision math
- **entropy.rs** — RDRAND/RDSEED
- **constant_time/** — Side-channel safe
- **rng/** — get_random_bytes

hmac.rs | misc.rs

SECURITY PROPERTIES

- **ZeroState** — Keys RAM-only, Zeroize on drop
- **Side-Channel** — Constant-time ops, No timing leaks
- **Quantum Ready** — ML-KEM + ML-DSA, Hybrid mode

Memory safety via Rust | No unsafe crypto | NIST compliance | Test vectors validated

NØNOS DEFAULT ALGORITHM SELECTION

- **Symmetric AEAD** — ChaCha20-Poly1305, Primary for all encryption
- **Signatures** — Ed25519 + ML-DSA (hybrid), Module signing, auth
- **Key Exchange** — X25519 + ML-KEM (hybrid), Forward secrecy
- **Hash** — BLAKE3 (fast) / SHA-512, Integrity, KDF input

Feature flags: mlkem512/768/1024, mldsa2/3/5, zk-halo2, zk-groth16

Driver Subsystem

drivers/ - 188 files, 23 modules - Hardware Abstract

Common Pattern

```
static CONTROLLER: spin::Once<T> = Once::new();
Single initialization, lock-free access after init
```

DRIVER PUBLIC API (mod.rs exports)

Initialization
- init_all_drivers() init_nvme()
- init_ahci() init_xhci() init_tpm()

Controller Access
- get_nvme_controller()
- get_ahci_controller() get_xhci_*

Network
- wifi_connect() wifi_scan()
- get_virtio_net_device()

TPM / Security
- measure_module() read_pcr()
- tpm_get_random_bytes()

Display / Audio
- init_gpu() with_gpu_driver()
- get_audio_controller()

Stats / Info
- get_hardware_stats()
- get_all_devices() DeviceInfo

STORAGE DRIVERS

NVMe
- NvmeController
- Admin/IO queues
- PRP/SGL lists
- Namespace mgmt

AHCI
- AhciController
- FIS-based cmds
- Command slots
- Secure erase

Controller Submodules
- controller/core.rs init.rs io.rs
- controller/admin.rs commands.rs
- dma.rs queue.rs security.rs stats.rs
- types.rs constants.rs error.rs

NETWORK DRIVERS

VirtIO-Net
- VirtQueue
- modern_regs
- smoltcp
- PacketBuf

E1000
- RX/TX rings
- EEPROM
- PHY ctrl
- checksum

Intel WiFi
- firmware
- scan/conn
- tx/rx DMA
- PCIe regs

Network Stack Integration
- VirtioNetInterface VirtioSmolBridge
- interface.rs validation.rs rate_limiter
- RTL8139 (legacy) network/stack.rs
- EtherType parsing MAC address mgmt

USB / AUDIO / DISPLAY

xHCI
- TRB rings
- Event/Cmd
- DeviceCtx
- USB 3.x

HD Audio
- BDL DMA
- Codec verb
- Streams
- PCM fmt

GPU
- VBE modes
- Framebuf
- Backbuf
- 32bpp

Controller Details
- CommandRing EventRing TransferRing
- SlotContext EpContext InputContext
- VGA text Console framebuf Keyboard
- AudioFormat StreamState DisplayMode

SECURITY / TPM

TPM 2.0
- extend_pcr_sha256()
- measure_component()
- verify_boot_chain()
- PCR 0-23 banks

security/
- validate_dma_buffer()
- validate_mmio_region()
- validate_lba_range()
- RateLimiter

Security Primitives
- safe_mmio_read32/write32
- is_config_write_allowed()
- DMA bounds LBA partition PRP validation
- Boot measurement log Quote generation

PCI/PCIe BUS LAYER

PciManager
- scan_and_collect()
- find_device_by_class()
- find_device_by_id()

ConfigSpace
- read32/write32
- BAR decoding
- 0xCF8/0xCFC ports

MSI / MSI-X
- configure_msi()
- configure_msix()
- Vector allocation

Capabilities
- enumerate_capabilities()
- get_pcie_info()
- PM, AER, SR-IOV

SecurityPolicy
- validate_config_write()
- approve_bus_master()
- Allow/Block lists

DmaEngine
- DmaDescriptor
- scatter-gather
- 64-bit addressing

PciStats
- config reads
- config writes
- violations

HARDWARE LAYER

NVMe SSD	SATA HDD	Intel NIC	VirtIO NIC	Intel WiFi	USB 3.x	HD Audio	GPU	TPM 2.0	PS/2	RTL8139
PCIe 4.0 x4	AHCI 6Gbps	E1000/E1000E	QEMU/KVM	AX200/AX210	xHCI	Intel HDA	Bochs VBE	MMIO/LPC	8042 KBC	Legacy NIC

Memory Subsystem

memory/ - 21 modules - Physical, Virtual, DMA, Security

VIRTUAL ADDRESS SPACE

USER SPACE (Ring 3)

0x0000_0000_0000_0000
0x0000_7FFF_FFFF_FFFF
128 TiB canonical low

Non-Canonical Hole

KERNEL SPACE (Ring 0)

KTEXT / KDATA
0xFFFF_FFFF_8000_0000
32+32 MiB code/data

DIRECTMAP
0xFFFF_FFFF_B000_0000
256 MiB phys map

KHEAP
0xFFFF_FF00_0000_0000
256 MiB heap

KVM (Kernel VM)
0xFFFF_FF10_0000_0000
512 MiB alloc

MMIO
0xFFFF_FF30_0000_0000
512 MiB device I/O

VMAP
0xFFFF_FF50_0000_0000
256 MiB vmalloc

DMA
0xFFFF_FF60_0000_0000
256 MiB DMA buf

FIXMAP
0xFFFF_FFA0_*
64 GiB

PERCPU
0xFFFF_FFC0_*
16 MiB/CPU

KPTI TRAMPOLINE
0xFFFF_FFFF_FFFE_0000
User/Kernel switch

4-LEVEL PAGING

PML4 (512 ent) → PDPT (512 ent) → PD (512 ent) → PT (512)

Page Sizes
4 KiB (PT) 2 MiB (PD) 1 GiB (PDPT)
PAGE_SIZE=4096 HUGE_PAGE_2M/1G

CR3
PML4 phys addr

Self-Ref
Slot 510

VIRTUAL MEMORY

virt/
map_page()
unmap_page()
translate_addr()
VmFlags

virtual_memory/
AddressSpace
VmArea
VmProtection
VmType

paging/
PageMapping
PagePermissions
tlb.rs
flush_tlb()

mmu/
init_mmu()
current_cr3()
invalidate_page()
ProtectionFlags

SECURITY / HARDENING

secure_memory/
SecurityLevel
RegionType
allocate_memory()
Capsules

safety/
validate_access()
verify_stack_integrity()
Corruption detect

hardening/
Guard pages
Stack canaries
W^X enforce
verify_kernel_*

kaslr/
validate()
verify_slide_integrity()
Randomization

UNIFIED MEMORY API

MemoryType / MemoryProtection
Anonymous | KernelCode | KernelData
UserCode | UserData | UserHeap | UserStack
Device | SecureCapsule | Shared

Core Functions
init_unified_vm()
map_memory(va, size, prot, type)
unmap_memory(va, size)
translate_virtual(va) -> PhysAddr

Address Conversion
phys_to_virt(phys) -> VirtAddr
virt_to_phys(virt) -> Option<PhysAddr>

Security
allocate_secure_region(size, owner, level)
validate_access(pid, va, write)

TLB / Page Fault
flush_tlb_range(start, size)
handle_unified_page_fault(addr, code)

PHYSICAL MEMORY

phys/
allocator.rs
bitmap.rs
Frame alloc
Zone stats

frame_alloc/
PhysFrame
AllocFlags
allocate_frame()
deallocate()

buddy_alloc/
BuddyBlock
AllocatedBlock
Power-of-2
Coalescing

page_allocator/
alloc_pages()
free_pages()
Contiguous allocation

DMA / MMIO / HEAP

dma/
DmaRegion
DmaPool
DmaConstraints
StreamingMapping

mmio/
mmio_r8/16/32/64
mmio_w8/16/32/64
MmioRegion
MmioFlags

heap/
SecureHeapAlloc
HeapStats
verify_heap_integrity()

layout/
validate_layout()
PAGE_SIZE=4096
HUGE_PAGE_2M
MAX_CPUS=64

ADDITIONAL MODULES

boot_memory/
Early alloc
Pre-heap init

region/
Memory regions
Range tracking

page_info/
Page metadata
Ref counting

proof/
Memory proofs
Verification

Statistics: HeapStats, AllocStats, VmStats
get_memory_system_stats() unified API

INITIALIZATION SEQUENCE

init_all_memory_subsystems():
1. layout::validate_layout() → 2. phys::init() → 3. frame_alloc::init()
4. heap::init() → 5. allocator::init() → 6. safety::init() → 7. kaslr::validate()

verify_all_memory_integrity()
heap_integrity + stack_integrity + kaslr_integrity + kernel_data + page_tables

KPTI (Spectre/Meltdown)

KPTI_TRAMPOLINE = 0xFFFF_FFFF_FFFE_0000
PCID_KERNEL = 0x0001 PCID_USER = 0x0002
Separate kernel/user page tables

ELF Subsystem

elf/ - 21 modules - Executable and Linkable Format Parser & Loader

ELF FILE STRUCTURE

ElfHeader (64 bytes)

```
0x7F 'E' 'L' 'F'
e_type: ET_EXEC/DYN/REL
e_machine: EM_X86_64
e_entry: entry point
e_phoff, e_shoff
```

ProgramHeader[]

PT_LOAD	Loadable segment
PT_DYNAMIC	Dynamic linking
PT_INTERP	Interpreter path
PT_TLS	Thread-local
PT_GNU_RELRO	Read-only reloc
PT_GNU_STACK	Stack perms

SectionHeader[]

SHT_PROGBITS	.text/.data
SHT_SYMTAB	Symbol table
SHT_STRTAB	String table
SHT_RELA	Relocations
SHT_DYNAMIC	Dynamic info

DynamicEntry[]

DT_NEEDED	Dependencies
DT_SYMTAB	Symbol table
DT_STRTAB	String table
DT_RELA	Relocations
DT_INIT/FINI	Init/Fini funcs

Symbol (24 bytes)

```
st_name, st_info
st_value, st_size
STB_GLOBAL/LOCAL/WEAK
```

RelaEntry (24 bytes)

```
r_offset, r_info
r_addend
type | sym << 32
```

CORE MODULES

types/

```
ElfHeader
ProgramHeader
SectionHeader
Symbol, RelaEntry
```

loader/

```
ElfLoader
ElfImage
LoadedSegment
DynamicInfo
```

reloc/

```
process_relocations()
RelocationContext
apply.rs, utils.rs
x86_64 relocs
```

symbol/

```
SymbolLookup
SymbolResolver
ResolvedSymbol
Name resolution
```

HASH / GOT / LINKING

hash/

```
gnu_hash()
sysv_hash()
DualHashLookup
```

got/

```
GlobalOffsetTable
GotEntry
PLT/GOT relocs
```

dynlink/

```
DynLinkInfo
Dynamic linking
info parsing
```

libmgr/

```
LibraryManager
LinkMap
```

interpreter/

```
InterpreterInfo
ld-linux.so
```

aslr/

```
AslrManager
Randomization
```

PROCESS / STACK / TLS

process/

```
ProcessBuilder
ProcessImage
create_process()
```

stack/

```
StackLayout
StackConfig
setup_user_stack
```

auxv/

```
AuxvBuilder
AuxEntry
AT_PHDR, AT_ENTRY
```

tls/

```
TlsInfo, TCB_SIZE
```

init/ fini/

```
InitArrayRunner
```

embedded/

```
EmbeddedLibrary
```

x86_64 RELOCATION TYPES

R_X86_64_64	S + A (absolute 64-bit)
R_X86_64_PC32	S + A - P (PC-relative)
R_X86_64_GLOB_DAT	S (GOT entry)
R_X86_64_JUMP_SLOT	S (PLT entry)
R_X86_64_RELATIVE	B + A (PIE/DSO)
R_X86_64_COPY	Copy symbol
R_X86_64_PLT32	L + A - P
R_X86_64_GOTPCREL	G + GOT + A - P

TLS Relocations:

R_X86_64_DTPMOD64	TLS module ID
R_X86_64_DTPOFF64	TLS offset
R_X86_64_TPOFF64	TP offset

ELF TYPES & MACHINES

e_type:

ET_EXEC (2)	Executable
ET_DYN (3)	Shared/PIE
ET_REL (1)	Relocatable

e_machine:

```
EM_X86_64 (62)
```

ELF LOADING FLOW

1. Validate — validate_elf()
2. Parse — ElfLoader::load()
3. Map — PT_LOAD segments
4. Relocate — process_relocs()
5. Setup Stack — setup_user_stack
6. Execute — jump to e_entry

User Stack Layout (setup by auxv/stack)

```
| argc | argv[] | NULL | envp[] | NULL | auxv[] | AT_NULL |
```

ADDITIONAL

minimal/

```
validate_elf
```

errors/

```
ElfError
```

cache/

```
ImageCache
```

constants

```
ELF_MAGIC
```

PUBLIC API (mod.rs)

```
load_elf_executable()
init_elf_loader()
get_elf_loader()
validate_elf()
validate_elf_detailed()
create_process()
create_process_with_args()
process_relocations()
setup_user_stack()
```

Re-exports:

```
ElfHeader, ProgramHeader
Symbol, RelaEntry
ElfImage, LoadedSegment
```