ERP.1 Audit & Accountability Infrastructure:
Formal Taxonomy, Multi-Plane Cryptographic Logs, and NIST SP 800-53 Attestation

SYNRC Research Group
https://synrc.com

August 3, 2026

Abstract

This document presents the formal specification, architecture, and theoretical foundation of the `synrc/audit` library for high-assurance operating environments based on the ERP.1 three-plane architecture. Built upon standard Erlang/OTP primitives and standard NIST cryptography (ECDSA P-384, SHA-384, HMAC-SHA-512), `synrc/audit` enforces strict privilege segregation, fail-secure append-only event logging, deterministic CRDT Merkle-log merging across multi-DC deployments, and complete compliance with NIST SP 800-53 Rev. 5 Audit and Accountability (AU) controls.

Contents

1 Introduction & Motivation

High-integrity critical systems (defense appliances, state registries, electronic healthcare, infrastructure control systems) demand rigorous mathematical assurance, non-repudiation, and architectural privilege isolation. Traditional POSIX models featuring monolithic `root` processes violate Zero Trust Architecture principles by exposing broad ambient authority.

The ERP.1 architecture reduces the Trusted Computing Base (TCB) to a minimal cryptographic core running on Erlang/OTP on UKI Alpine Linux. The `synrc/audit` library isolates audit generation, aggregation, and signature capabilities across three execution planes, guaranteeing that audit logs remain immutable, cryptographically verifiable, and fail-secure under all operating conditions.

2 ERP.1 Three-Plane Isolation Model

Following the fundamental OS.1 taxonomy defined in `synrc/os/os.txt`, the execution environment is partitioned into three distinct planes:

```
+-------------------------------------------------------------------------------+
| Security Admin BEAM   [S]                   (Trusted Security Services)       |
|  * Highest privilege, minimal TCB (ca, tss, sealed kvs)                       |
|  * Owns long-term ECDSA P-384 keys, HSM/TPM, root & intermediate CAs          |
|  * Never opens network sockets or mounts filesystem volumes                   |
|  * Signs critical audit records & periodic Merkle checkpoints                 |
+-------------------------------------------------------------------------------+
                : sealed, capability-limited length-prefixed channel
+-------------------------------------------------------------------------------+
| System Admin BEAM     [C]                   (Privileged Control Plane)        |
|  * Sole owner of kernel operations, seats, network & capability broker        |
|  * Maintains local append-only hash-chain (SHA-384 + HMAC-SHA-512)            |
|  * Synchronous fail-secure commits (AU-5)                                     |
|  * Forwards logs to SIEM (Wazuh)                                              |
+-------------------------------------------------------------------------------+
                : capability-granted, authenticated bridge
+-------------------------------------------------------------------------------+
| Application BEAM(s)   [A]                   (Untrusted Domain & Sessions)     |
|  * Host user interfaces, session logic, and media pipelines                   |
|  * No access to long-term keys or ambient storage                             |
|  * Generates local events forwarded exclusively via capability bridge         |
+-------------------------------------------------------------------------------+
```

Figure 1: ERP.1 Three-Plane Architecture Isolation Diagram.

2.1 Plane Descriptions

1. **Security Admin BEAM [S]**: Houses `synrc/ca`, `synrc/tss`, and sealed `synrc/kvs`. Long-term private keys never leave this plane. It provides RFC 3161 timestamps and ECDSA signatures for critical events.

2. **System Admin BEAM [C]**: Runs `audit_core`. Handles capability broker passing via `SCM_RIGHTS`, maintains the append-only record chain, applies HMAC-SHA-512 integrity tags, and publishes periodic checkpoints.

3. **Application BEAM [A]**: Runs `audit_client`. Untrusted workloads operating under strict Landlock, seccomp-bpf, and cgroups v2 boundaries.

3 Formal Distributed Audit Model

3.1 Audit Invariants

Audit processing in ERP.1 satisfies four fundamental invariants:

1. **Irreversibility**: Audit entries form an append-only cryptographic hash chain linked via SHA-384 hashes. Past entries cannot be modified or reordered.
2. **Non-Repudiation**: Critical records and batch checkpoints are signed by Security Admin BEAM using ECDSA P-384 keys and sealed with RFC 3161 timestamps.
3. **Isolation**: Application BEAMs cannot read or mutate audit structures of higher planes.
4. **Completeness**: No privileged action (certificate issuance, key unsealing, ABAC policy change, device grant) can execute without a synchronous audit record commit (fail-secure, AU-5).

3.2 Record Model (`#audit_record{}`)

Every audit event follows the mandatory schema specified by NIST AU-3:

```
-record(audit_record, {
    id          :: binary(),                % Monotonic ID (64-bit TS + Rand)
    ts          :: integer(),               % Milliseconds UTC / HLC
    plane       :: security | system | application,
    subject     :: binary(),                % Subject identity
    event       :: atom(),                  % Event type
    resource    :: binary(),                % Resource identifier
    action      :: atom(),                  % Action performed
    outcome     :: success | failure,       % Result
    meta        :: map(),                   % Metadata context
    prev_hash   :: binary(),                % SHA-384 of previous link
    hmac        :: binary(),                % HMAC-SHA-512 (System plane)
    sig         :: binary() | undefined,    % ECDSA P-384 (Security plane)
    tsp         :: binary() | undefined     % RFC 3161 timestamp
}).
```

3.3 Multi-Node CRDT Log Merge

In multi-node / multi-DC environments, each node maintains a local sequential hash chain. Global consensus is achieved using an **Add-only Merkle-CRDT** (G-Set of signed checkpoints):

$$\text{GlobalLog} = \text{G-Set}\left(\{\text{NodeId}, \text{HeadHash}, \text{SignedCheckpoint}, \text{MerkleRoot}\}\right) \tag{1}$$

Merging two node logs L_A and L_B is deterministic and conflict-free:

$$\text{merge}(L_A, L_B) = L_A \cup L_B \tag{2}$$

Linearization sorts records by the tuple $(TS, \text{NodeId}, \text{RecordId})$, producing a deterministic global sequence without altering original node hash chains.

4 NIST Cryptographic Profile

All cryptographic primitives are standard FIPS/NIST compliant:

Purpose	Algorithm	Specification	NIST
Chain	Merkle Hashing	SHA-384 (or SHA-512)	NIST FIPS 180-4
Message Integrity	HMAC-SHA-512	NIST FIPS 198-1	
Digital Signature	ECDSA P-384	NIST FIPS 186-4 (secp384r1)	
Timestamp	RFC 3161	SHA-384 / SHA-512 TimeStampToken	
Key Generation	ECDH P-384	NIST SP 800-56A	

Table 1: NIST Cryptographic Profile for `synrc/audit`.

Control	Control Name	Plane	Implementation in `synrc/audit`
AU-1	Audit Policy and Procedures	[C]	Defined in CMDB metadata.
AU-2	Event Selection	[C],[A]	Configurable event masks per plane.
AU-3	Content of Audit Records	[C]	Mandatory `#audit_record{}` schema.
AU-4	Storage Capacity	[C]	ETS / KVS sizing and segmentation.
AU-5	Response to Processing Failures	[C]	Synchronous fail-secure `gen_server` call.
AU-6	Review, Analysis, Reporting	[C],[S]	Offline verifier (`audit_verify`) & SIEM export.
AU-7	Audit Reduction	[C]	Filtered queries and Merkle proofs.
AU-8	Time Stamps	[C],[S]	UTC ms / HLC + RFC 3161 timestamps.
AU-9	Protection of Audit Info	[C],[S]	Append-only SHA-384 chain + HMAC-SHA-512.
AU-10	Non-Repudiation	[S]	ECDSA P-384 signatures by Security plane.
AU-11	Audit Record Retention	[C]	Archiving tool (`audit_archive`) for cold storage.
AU-12	Audit Generation	[A],[C]	Real-time generation via capability bridge.

Table 2: NIST SP 800-53 Rev. 5 AU Controls Attestation Matrix.

5 NIST SP 800-53 Rev. 5 AU Control Attestation

6 Comprehensive Audit Usage & Code Examples

6.1 1. Starting the Audit Service

To initialize the System Admin BEAM aggregator with ECDSA P-384 Security keys:

```
{PubKey, PrivKey} = audit_crypto:generate_keypair(),
MacKey = audit_crypto:generate_mac_key(),

{ok, Pid} = audit_core:start_link(#{
    node_id     => <<"DC1-Node01">>,
    boot_time   => erlang:system_time(millisecond),
    mac_key     => MacKey,
    sec_keypair => {PubKey, PrivKey}
}).
```

6.2 2. Logging Standard Application Events (AU-12)

From any Application plane process:

```
ok = audit_client:log(
    application,
    <<"user_session_42">>,
    user_login,
    <<"/api/v1/auth">>,
    authenticate,
    success,
    #{<<"ip">> => <<"192.168.1.100">>, <<"seat">> => <<"seat0">>}
).
```

6.3 3. Logging Critical Security Events with ECDSA Signatures (AU-10)

For high-assurance operations requiring non-repudiation:

```
ok = audit_client:log_critical(
    security,
    <<"security_admin">>,
    cert_issuance,
    <<"CA-Root-2026">>,
    sign_certificate,
    success,
    #{<<"subject_dn">> => <<"CN=SystemAdmin">>}
).
```

6.4 4. Generating and Verifying Merkle Checkpoints

To generate a signed checkpoint for active records:

```
Checkpoint = audit_core:get_checkpoint(),
Valid = audit_checkpoint:verify(Checkpoint, PubKey),
io:format("Checkpoint valid: ~p~n", [Valid]).
```

6.5 5. Multi-Node CRDT Log Merging

Merging audit streams from independent nodes:

```
% Node A log set
SetA = audit_merge:add_node_log(audit_merge:new_log_set(), CheckpointA, RecordsA),
% Node B log set
SetB = audit_merge:add_node_log(audit_merge:new_log_set(), CheckpointB, RecordsB),

% Conflict-free merge
MergedSet = audit_merge:merge(SetA, SetB),

% Verify full merged G-Set
true = audit_merge:verify_merged(MergedSet, PubKey),

% Linearize into total order
LinearRecords = audit_merge:linearize(MergedSet).
```

6.6 6. Pure Offline Verification (`audit_verify`)

Perform independent offline verification of a record chain:

```
GenesisHash = audit_chain:genesis_hash(<<"DC1-Node01">>, BootTime),
case audit_verify:verify_chain(Records, GenesisHash, MacKey, PubKey) of
    ok -> io:format("Chain integrity verified.~n");
    {error, {tampered_record, Index, BadRecord}} ->
        io:format("Tampering detected at record ~p!~n", [Index])
end.
```

6.7 7. SIEM & OSCAL Export (`audit_export`)

Exporting log entries for Wazuh / Syslog or compliance evidence:

```
% Export to Syslog / Wazuh CEF line format
CEFString = audit_export:to_siem(Record),

% Export to JSON
JSONData = audit_export:to_json(Records),

% Export to OSCAL Evidence Map
OscalMap = audit_export:to_oscal(Checkpoint, Records).
```

7 Conclusion

The `synrc/audit` library provides a minimal, self-contained, and cryptographically verifiable audit infrastructure for ERP.1 systems. By combining standard NIST algorithms, Erlang/OTP supervision, three-plane capability separation, and CRDT Merkle merging, it fulfills the full spectrum of NIST SP 800-53 Rev. 5 AU requirements without external dependencies.