

UNDERGRADUATE TECHNICAL PAPER

Configuration and Implementation of an OpenSSL 3 Provider

Architecture, a Reproducible Digest Prototype, and Experimental Evaluation

Expanded bachelor's-degree-level study
12 September 2026

Target environment: OpenSSL 3.5.5 on Linux x86_64
Includes source code, configuration, test evidence, and a downloaded reference collection.

Document version 2.0 • English
Accompanied by reproducible source code and reference snapshots.

Abstract

OpenSSL 3 separates high-level cryptographic interfaces from algorithm implementations through its provider architecture. This separation enables software modules and hardware integrations to supply algorithms without requiring applications to call implementation-specific entry points. However, successful integration depends on more than compiling a shared library: the provider must expose correctly typed callbacks, advertise algorithms and properties, manage operation state, and participate in a configuration and selection policy.

This paper investigates how an application discovers, selects, and executes a provider implementation. A reproducible prototype, named `edu`, supplies an EDU-SHA256 digest through the provider interface. It delegates the actual SHA-256 computation to the default provider in a separate library context. This deliberate restriction makes the work an investigation of integration mechanics rather than the design of a new cryptographic primitive. A configuration file and an application-level loading path are implemented and tested on OpenSSL 3.5.5. Eight functional checks cover reference digest comparison, streaming and context duplication, provider identity, mandatory property failure, configuration activation, and coexistence with the default provider. All eight checks pass. The expanded investigation adds 37 integration cases and 3,362 direct callback assertions, all of which also pass. Further chapters analyze deployment, policy boundaries, and proposed performance evaluation.

The results show that provider availability, algorithm advertisement, and property-based selection are distinct conditions that should be verified independently. They also illustrate why a working provider is not automatically suitable for production or for a validated cryptographic boundary. The principal contribution is an inspectable teaching artifact connecting the documented architecture to executable code and explicit experimental evidence.

Keywords: OpenSSL 3, provider, EVP, configuration, dynamic loading, SHA-256, property query, cryptographic software engineering.

Study boundaries

This paper concerns OpenSSL providers, not BoringSSL. The earlier BoringSSL workspace is not a dependency of the experiment. It does not claim a novel hash algorithm, FIPS validation, a security audit of the prototype, or measured performance improvement.

Contents

1. Introduction	5
2. Background and Related Work	7
3. Configuring and Selecting a Provider	9
4. Prototype Design and Implementation	11
5. Experimental Method and Results	14
6. Security, Policy, and Engineering Limitations	17
7. Findings from the Baseline Experiment	19
8. Expanded Research Design	20
9. The Provider ABI as an Engineering Contract	24
10. Configuration Experiments and Failure Diagnosis	28
11. Memory Ownership and Digest State	32
12. Extended Testing Methodology	36
13. Extended Results and Interpretation	40
14. Performance Evaluation: A Proposed Method	44
15. Packaging, Deployment, and Operational Review	48
16. Policy Boundaries and Cryptographic Assurance	52
17. Extending the Design Beyond Digests	56
18. Reproducibility and Artifact Organization	59
19. Discussion and Threats to Validity	63
20. Final Conclusions	67
References	69
Appendix A. Complete provider source	70
Appendix B. Client and build scripts	74
Appendix C. Test harness source	76
Appendix D. Experimental records	81
Appendix E. Review and reproduction guide	88

Abbreviations

Term	Meaning
ABI / API	Application binary / programming interface
EVP	OpenSSL's high-level cryptographic interface family
DSO	Dynamically loaded shared object
HSM	Hardware security module
FIPS	Federal Information Processing Standards
SHA-256	256-bit Secure Hash Algorithm

1. Introduction

1.1 Motivation and problem statement

Cryptographic applications require a stable way to request operations while implementations evolve. A program may need a software digest today and a hardware-backed signature tomorrow. Embedding implementation choices in every application call makes this evolution expensive. OpenSSL 3 addresses part of this problem by allowing implementations to be supplied by providers and obtained through high-level interfaces. Providers replace important extension roles previously associated with ENGINE, although migration also involves changes to object handling and application APIs. [1,12]

The practical problem is that a provider can be present on disk without being loaded, loaded without implementing the desired operation, or advertising an algorithm that does not satisfy an application's selection properties. An administrator may see a provider in a listing and incorrectly conclude that a particular digest is being executed by it. A developer may implement a mathematical operation correctly but violate the callback's buffer or ownership contract. These are separate classes of failure and require separate observations.

1.2 Research questions and objectives

The study asks three questions. RQ1: How do configuration, provider loading, and algorithm fetching combine to determine which implementation executes? RQ2: Which interfaces and lifecycle responsibilities are needed for a minimal usable digest provider? RQ3: What evidence can a small functional experiment establish, and what remains outside its scope?

The objectives are to explain the architecture at undergraduate level, implement a readable module, compare configuration-driven and programmatic loading, and provide reproducible positive and negative tests. The experiment is intentionally narrow: a digest has simpler state than a private-key operation, yet still exposes discovery, dispatch,

allocation, streaming, finalization, duplication, and cleanup. These concepts form a foundation for later study of signature, key-management, and HSM providers.

1.3 Method and contribution

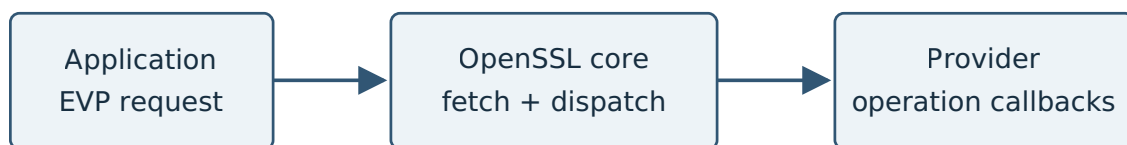
The method combines a version-pinned documentation review with design-and-build experimentation. The implementation is compiled against the installed OpenSSL 3.5.5 headers and library. Expected output is checked against Python's SHA-256 interface and a known published example. Logs, program source, and reference snapshots accompany the paper. The contribution is not algorithmic novelty: it is a traceable connection between architectural concepts and a running provider.

There are two important methodological choices. First, the provider advertises a distinct name, `EDU-SHA256`, so its selection can be distinguished from ordinary `SHA256`. Second, it uses a separate internal library context and an explicit default-provider property for its backend. This prevents the wrapper from fetching itself. It also creates an intentional policy limitation, examined in Section 6: the backend does not inherit an application's cryptographic policy automatically.

2. Background and Related Work

2.1 Providers, the core, and EVP

A provider is a collection of implementations exposed to the OpenSSL core through structured interfaces. An application generally requests an EVP operation rather than invoking the provider's implementation functions directly. Provider initialization establishes a provider context and a dispatch table. When the core requests a particular operation class, the provider can return an algorithm array describing the implementations it offers. A dynamically loaded module exports `SSL_provider_init` as its entry point. [1,2]



Configuration and properties influence availability and selection.

Figure 1. Conceptual request path. Original illustration for this study; the diagram simplifies the interfaces described in [1,2].

The indirection is explicit. An `SSL_DISPATCH` associates function identifiers with callback pointers; identifiers are defined by the public core-dispatch interface. An `SSL_ALGORITHM` connects an algorithm name and property definition to a dispatch table. The operation identifier, such as `SSL_OP_DIGEST`, determines what kind of algorithm the core is asking for. The identifiers are not application commands: they are part of the provider ABI. [9]

2.2 Library contexts and selection

An `SSL_LIB_CTX` separates relevant OpenSSL state, including provider use and configuration, from other contexts in the process. A program may use the default context or create its own. Isolation is useful when a

library should not silently change the host application's provider selection. It is not a separate operating-system process and does not sandbox a provider module. [8]

Fetching resolves a requested algorithm under the applicable context and property constraints. Property definitions describe an implementation; property queries express requirements or preferences. For this prototype, `provider=edu` is a mandatory equality constraint. A query such as `provider=missing` should fail when no matching implementation is available. Properties guide selection; they do not themselves prove that an implementation is trustworthy. [7]

2.3 Standards and security assessment literature

SHA-256 transforms a message into a 256-bit digest, using 512-bit message blocks. The Secure Hash Standard specifies the algorithm and its processing rules. It does not specify the OpenSSL provider lifecycle. The two specifications therefore address different correctness obligations: the mathematical transformation and the software interface that exposes it. [13]

The 2024 Trail of Bits assessment of OpenSSL, organized through OSTIF, is relevant because it treats configuration and provider infrastructure as security-sensitive components. Its discussion of provider configuration illustrates how a seemingly simple activation option can be misinterpreted. The report is historical evidence about the assessed code, not evidence that every finding applies unchanged to OpenSSL 3.5.5. This study uses the 3.5.5 documentation for the meaning of configuration fields and treats the audit as motivation for explicit negative testing. [14]

3. Configuring and Selecting a Provider

3.1 A controlled configuration file

The experiment uses a project-local configuration rather than editing the system configuration. The build script generates an absolute module pathname, which removes ambiguity about the file being loaded. The `OPENSSL_CONF` environment variable points a test process at this file. Configuration diagnostics are enabled so that configuration errors are surfaced instead of being silently overlooked. Provider sections explicitly activate both the teaching provider and the default provider. [4]

```
config_diagnostics = 1
openssl_conf = initialization
[initialization]
providers = providers
[providers]
default = default_section
edu = edu_section
[default_section]
activate = 1
[edu_section]
module = /absolute/path/to/build/edu.so
activate = 1
```

The explicit default-provider entry is a deliberate coexistence decision. An application should not assume that the default provider will remain implicitly available after another provider is explicitly activated. The study tests a conventional `sha256` request as well as `EDU-SHA256` to confirm that both intended paths remain usable. The internal default provider used by the prototype belongs to a different context and is not a substitute for making the default provider available in the application's context.

3.2 Command-line and programmatic loading

A command-line experiment can bypass the configuration file and identify the module search directory directly:

```
OPENSSL_CONF=/dev/null openssl dgst \
  -provider-path ./build -provider edu \
  -propquery 'provider=edu' -EDU-SHA256 -binary
```

The C client instead creates a library context, sets its provider search path, loads `edu`, and calls `EVP_MD_fetch`. It checks the provider attached to the fetched method before processing input. This is stronger evidence of selection than checking only whether the provider is available. The provider-loading API returns handles whose lifetime must be coordinated with fetched methods and operation contexts. [5,6]

```
OSSL_PROVIDER_set_default_search_path(libctx, module_dir);
provider = OSSL_PROVIDER_load(libctx, "edu");
md = EVP_MD_fetch(libctx, "EDU-SHA256", "provider=edu");
```

3.3 Diagnostic sequence

Question	Observation	Typical issue
Was the intended configuration used?	Inspect process environment and explicit file path.	A process reads a different configuration.
Did the module load?	Inspect <code>openssl list -providers -verbose</code> .	Wrong path, missing symbol, incompatible library.
Was the algorithm offered?	Inspect advertised digest names.	Wrong operation ID or missing algorithm entry.
Did selection match?	Fetch with mandatory properties; inspect provider identity.	Name or property mismatch.
Did execution succeed?	Check return values, length, and reference output.	Callback or state-management defect.

This sequence localizes errors without conflating availability with execution. Configuration should also be reviewed as executable-loading policy: the selected module runs inside the application. The experiment grants no trust to an arbitrary shared object merely because it can be named in a configuration file.

4. Prototype Design and Implementation

4.1 Design rationale

The prototype is a provider bridge, not a standalone implementation of SHA-256. It exposes a new digest name through the provider ABI but delegates computation to a fetched default-provider SHA-256 method. A bridge is appropriate for teaching because the reader can concentrate on module integration while relying on an existing backend for the primitive. It also makes the limitation visible: agreement with the backend is expected, and the experiment cannot establish an independent implementation's cryptographic correctness.

Two context types divide responsibility. The provider context owns a private library context, the backend provider handle, and the fetched SHA-256 method. Each digest context owns an `EVP_MD_CTX` and refers to its provider context. Initialization creates these provider-wide resources once. Digest operations allocate their own mutable state. Cleanup releases operation contexts before provider-owned resources. Shared state is limited, but no concurrency stress test is claimed.

Resource	Owner	Release action
Private library context	Provider instance	<code>OSSL_LIB_CTX_free</code>
Default-provider handle	Provider instance	<code>OSSL_PROVIDER_unload</code>
Fetched SHA-256 method	Provider instance	<code>EVP_MD_free</code>
Inner digest state	Digest operation	<code>EVP_MD_CTX_free</code>

4.2 Initialization and discovery

The exported entry point allocates provider state, creates the private library context, loads its default provider, and fetches SHA256 with `provider=default`. It returns success only after these dependencies exist. On failure, the same teardown function unwinds partially created

resources. The returned provider dispatch table exposes teardown, query-operation, and metadata functions. The incoming core callback table is unused in this deliberately small prototype. [2]

The query function returns the algorithm table for digest requests and NULL for other operation classes. Its tables have static storage duration, so the code sets the query's no-cache output to zero. The advertised property is `provider=edu`. The provider name supplied to the loader and the algorithm's property string are intentionally consistent.

```
static const OSSL_ALGORITHM algorithms[] = {
    { "EDU-SHA256", "provider=edu", digest_dispatch,
      "Educational SHA-256 bridge" },
    { NULL, NULL, NULL, NULL }
};
```

4.3 Digest lifecycle and parameters

The digest implementation supplies allocation, initialization, update, finalization, release, duplication, and parameter callbacks. `newctx` creates operation state; `init` initializes the backend digest; repeated updates supply message fragments; finalization emits the digest. Duplication uses `EVP_MD_CTX_copy_ex` so that a partially processed message can branch into independent continuations. These callbacks implement the operation contract described by the digest-provider interface. [3]

Before finalization, the wrapper checks the output-length pointer and requires room for 32 bytes. It sets the reported length to zero before attempting the backend call. The baseline assessed this guard by inspection. The expanded callback tests in Chapters 11–13 exercise capacities from zero through 33 and test recovery after rejection. This separates the original inspection evidence from the later execution evidence.

The algorithm reports a digest size of 32 bytes, a block size of 64 bytes, and a false extendable-output flag. `OSSL_PARAM` provides typed name/value exchange; callers may request a subset of available parameters. The implementation checks whether each requested parameter exists and

uses the corresponding setter, propagating setter failure. Unknown unrequested fields are not invented, and optional operation parameters are not advertised. [10]

4.4 Building and maintaining the module

The provider and client are compiled as C11 with optimization and warnings promoted to errors. The provider is a position-independent shared object linked against the host OpenSSL library. The client is an ordinary executable. The prototype therefore depends on this host's OpenSSL installation; it is not a claim of compatibility with every OpenSSL release or operating system.

```
cc -std=c11 -O2 -fPIC -Wall -Wextra -Werror \  
-shared edu_provider.c -o build/edu.so \  
$(pkg-config --cflags --libs openssl)
```

The full implementation is included in Appendix A and in the accompanying source directory. Provider code should remain reviewable as software that executes with the application's privileges. Keeping the example small does not eliminate the need for deployment controls, robust errors, and broader testing in a production project.

5. Experimental Method and Results

5.1 Environment and procedure

The experiments were executed on 12 September 2026 using OpenSSL 3.5.5, reported by both the executable and development package interface on the Linux x86_64 host. The exact build and platform output is saved in `results/environment.txt`. The shell build script creates the provider and client, generates the configuration file, and executes the Python test harness. The harness captures exit codes and output rather than judging success from a provider listing alone.

For explicit command-line loading, the harness sets `OPENSSL_CONF=/dev/null` to remove system configuration from that test path. For configuration-driven tests, it supplies the generated file. This controlled distinction permits the two loading mechanisms to be evaluated separately. Test messages include zero-length input, a short familiar string, all possible byte values, and a large repeated message.

5.2 Test matrix

ID	Test and acceptance criterion	Observed
T1	Empty message matches SHA-256 reference.	PASS
T2	abc matches SHA-256 reference.	PASS
T3	256-byte binary input matches reference.	PASS
T4	1,000,000 bytes of a match reference.	PASS
T5	Provider identity is edu; streaming and duplicated contexts produce the expected digest.	PASS
T6	Mandatory provider=missing query fails.	PASS
T7	Configuration activates edu and permits the educational digest.	PASS

T8	Ordinary SHA-256 remains usable with explicit default-provider activation.	PASS
----	--	------

All eight checks completed successfully. For `abc`, the observed hexadecimal digest was:

```
ba7816bf8f01cfea414140de5dae2223
b00361a396177a9cb410ff61f20015ad
```

The line break is presentational only; the value is a single 64-character hexadecimal string. In T5, the client updates one context with `a`, duplicates it, and supplies `bc` to both copies. Equality with the expected digest tests state copying and streaming across the outer provider boundary. T6 deliberately expects a nonzero command exit code: failure is the correct result when the mandatory property cannot be satisfied.

5.3 Interpretation and validity

The tests support a narrow conclusion: the module can be loaded and selected through the tested interfaces, and its implemented digest path behaves correctly for the selected cases. They do not prove security for all possible messages, allocation failures, loader failures, or concurrent interleavings. The large input is a functional test of message handling, not a throughput benchmark. No latency or memory-use improvement is claimed.

Reference comparison also requires qualification. Python's hash implementation may itself use OpenSSL on the host, and the provider intentionally delegates to OpenSSL's default implementation. Consequently, agreement is strong evidence about integration but weak evidence of independent algorithm correctness. The short known-answer example adds a stable external expectation, while the NIST specification explains the algorithm being requested. [13] A stronger cryptographic implementation study would require independently sourced vectors, broader coverage, and an independently implemented backend.

Reproducibility is supported by the saved source, compiler command, reference version tags, configuration generator, and machine-readable results. External validity is limited to the tested platform and version. Porting to Windows would require a different shared-library build/export setup; substituting another OpenSSL version would require compiling and rerunning the same tests rather than assuming identical behavior.

6. Security, Policy, and Engineering Limitations

6.1 Configuration is part of the trust boundary

A provider module executes within the requesting process. An attacker who can replace a module or redirect configuration to an untrusted module may gain a much broader capability than choosing an incorrect digest. Configuration files, module directories, search paths, and deployment permissions should therefore be controlled together. A property string is selection metadata, not an attestation mechanism. The audit literature reinforces the need to consider parser behavior and human interpretation as part of provider security. [14]

The generated absolute path is useful in this experiment because it makes the selected module obvious. In production it should be combined with an intentional installation layout and update process. The environment variables used by the tests should not become an uncontrolled production policy channel. Listing active providers is useful operational evidence, but it does not reveal every backend action inside a custom provider.

6.2 The bridge and policy inheritance

The private context solves a recursion problem but creates a policy boundary. The outer application selects `EDU-SHA256`; the provider then performs a second, explicit fetch of default-provider SHA-256 in its own context. That backend selection does not automatically respect the application's default properties. A caller imposing a restricted policy cannot infer compliance merely from the outer operation succeeding.

The implementation does not advertise `fips=yes`. This is intentional. FIPS-related use involves appropriate module installation, configuration, validated versions and operating conditions, and correct algorithm selection. Naming a module, adding a property, or successfully computing a standard digest is not sufficient evidence of

validation. OpenSSL's FIPS guidance describes a separate configuration and operational workflow. [11] This prototype neither performs that workflow nor makes a compliance claim.

6.3 Work required for production

Production engineering would require richer diagnostics, well-defined failure behavior, thread and lifetime stress tests, memory-safety tooling, packaging/version compatibility checks, and a documented update mechanism. Allocation-failure injection should examine every partial-initialization path. Direct callback tests should exercise undersized output buffers and malformed parameter requests. A security review should examine both the module and the conditions under which it is loaded.

An HSM extension would add problems deliberately excluded here: key discovery, authorization, sessions, login state, device errors, and coordination between key management and signature operations. A TLS deployment would require additional operations and integration tests beyond a digest implementation. This staged approach is educationally useful: the small provider first teaches dispatch and ownership, then a larger project can add domain-specific responsibilities.

6.4 Ethical reporting of experimental software

A teaching prototype should state exactly what it implements and tests. In this work, “provider implementation” means the provider-facing integration and digest adaptation layer. “SHA-256 implementation” would imply a stronger claim that the project does not satisfy. Likewise, a successful eight-test run is a functional result, not a substitute for security certification. Keeping these distinctions explicit prevents a demonstration from acquiring unsupported production claims when copied into another project.

7. Findings from the Baseline Experiment

This chapter records the baseline conclusions. Chapters 8–20 extend the investigation, implementing selected boundary tests proposed here and reporting their results separately.

This study connected provider configuration, discovery, selection, and execution in a reproducible OpenSSL 3.5.5 example. RQ1 is answered by separating the configuration/loading path from algorithm fetching: a provider must be available, advertise the desired operation, and satisfy the caller’s property requirements. RQ2 is answered by the implemented initialization, query, metadata, digest lifecycle, parameter, and teardown interfaces. RQ3 is answered by the test matrix and its limits: the experiments establish selected integration behavior, not comprehensive cryptographic security.

The prototype passed eight functional checks through command-line, configuration-driven, and programmatic use. Its most useful result is methodological: verifying the provider attached to the fetched method and including an expected failure provides stronger evidence than observing a successful digest alone. The bridge design also makes the distinction between outer selection policy and internal backend policy concrete.

Future work should add fault injection, sanitizers, concurrency testing, and direct callback boundary tests. A second implementation could replace the default-provider backend with an independently tested primitive or a hardware service, while preserving the same external test structure. Extending the experiment to key management and signatures would then reveal additional lifecycle and policy requirements. These steps would transform the present undergraduate integration study into a broader investigation of deployable cryptographic modules.

8. Expanded Research Design

The baseline experiment answers whether the teaching provider can be made to work through several ordinary interfaces. A longer engineering study must ask a more demanding question: what exactly does “works” mean, and how should evidence be organized so that another reader can evaluate the claim? This chapter develops a requirements model and an evidence model for the extended investigation. The additional work does not turn the bridge into a new cryptographic primitive. It increases the precision with which its integration behavior is described.

8.1 Unit of analysis

The unit of analysis is the combination of the module, the host OpenSSL library, the configuration used by a particular process, and the caller’s selection request. Examining the shared object alone would miss important interactions. For example, a valid module can remain unavailable because the application reads a different configuration file. Conversely, a correct digest can be returned by another provider even when the intended module was never selected. The experiment therefore treats observable application behavior as an outcome of several cooperating components.

This choice also determines the meaning of reproducibility. Copying the C source is necessary but insufficient. A reproducing investigator needs the compiler invocation, the development headers, the runtime version, the module path, and the environment used for the test. The saved artifacts supply these details or provide commands that regenerate them. Where the host still contributes a dependency, such as the system compiler, the paper reports that dependency rather than presenting the directory as a complete operating-system image.

8.2 Functional and nonfunctional requirements

Functional requirements describe responses that can be observed directly. The provider should load, advertise its digest, produce the expected bytes, accept streaming input, and reject an unsatisfied

selection constraint. Nonfunctional requirements concern qualities such as maintainability, diagnosability, compatibility, and confidence in resource management. They often require more than a single successful test. A readable ownership table is evidence of design discipline, but it is not equivalent to a memory-safety proof.

ID	Requirement	Evidence used
FR1	Expose EDU-SHA256 through the provider interface.	Programmatic fetch and algorithm-table inspection.
FR2	Return 32 correct output bytes for tested messages.	Reference comparisons and direct finalization checks.
FR3	Support streaming and operation-state duplication.	C client that branches a partially updated context.
FR4	Respect outer mandatory property matching.	Positive and negative fetch requests.
FR5	Load from an explicit configuration file.	Controlled process environment and activation matrix.
NFR1	Make ownership reviewable.	Design tables and cleanup-path inspection.
NFR2	Make experiments reproducible.	Scripts, source snapshots, and machine-readable results.
NFR3	Avoid unsupported assurance claims.	Explicit separation of observations and limitations.

The matrix deliberately does not label NFR1 “proved.” The direct tests exercise many normal allocations and releases, but they do not force every allocator to fail at every call. Similarly, NFR2 is supported on the recorded host; it is not evidence that every operating system can compile the module unchanged. This distinction is useful in undergraduate engineering because it prevents test completion from being mistaken for complete requirement verification.

8.3 Competing explanations

A digest match has several possible explanations. The module may have executed correctly; another provider may have executed instead; the input may have been altered in the harness; or the reference may share the same defect as the implementation. The design reduces some of these ambiguities. A distinct algorithm name and an explicit provider property reduce accidental selection. Binary input and output avoid newline and hexadecimal-format assumptions. The client checks the provider attached to the fetched method. None of these steps makes the backend independent of OpenSSL, so correlated algorithm errors remain a limitation.

A failed command is similarly ambiguous. Failure could be the desired property rejection or an unrelated loading problem. The negative property case is run beside a successful case with the same executable, module path, and message. Only the property changes. The difference supports the intended explanation, although a richer error-classification harness could make the inference stronger. The paper reports the measured exit code rather than claiming to have exhaustively classified every OpenSSL error-stack entry.

8.4 Evidence levels

Four evidence levels are distinguished throughout the extended discussion. Documentary evidence describes the public interface as specified in a version-pinned manual. Inspection evidence follows the prototype's control flow and resource ownership. Execution evidence records what a concrete test actually did. Proposed evidence describes tests or operational reviews that would be appropriate in future work but were not performed. These levels should not be silently substituted for one another.

For example, the baseline paper inspected the finalization guard but did not invoke it with an undersized buffer. The extension adds execution evidence for capacities from zero through 33 bytes. This is a real increase in coverage, not a wording change. Yet the same extension still

does not provide allocator-failure injection. The appropriate conclusion is specific: the tested buffer guards behaved as intended; exhaustive failure-path assurance remains open.

8.5 Scope of the extended study

The extension retains the original provider code so that additional evidence can be attributed to a fixed implementation. It adds a white-box callback harness and a broader integration harness. The white-box harness directly invokes the prototype's entry point with null core arguments because this implementation does not consume them. That is a controlled property of this prototype, not a general technique guaranteed to work for arbitrary providers. The integration harness continues to use ordinary OpenSSL command-line requests.

The resulting study therefore has two complementary perspectives. One observes the module as an application would use it. The other deliberately reaches the provider-facing interface to test conditions that an ordinary EVP call may filter or normalize before they reach the module. Agreement between these perspectives is useful, but neither eliminates the need to understand the specific layer at which an assertion is made.

9. The Provider ABI as an Engineering Contract

A provider implementation is a binary extension whose correctness depends on agreement between independently compiled components. The public headers describe this agreement with types, identifiers, structures, and callback signatures. Reading the interface as a contract is more productive than treating it as a collection of symbols that merely need to compile. Every callback has inputs, outputs, ownership implications, and assumptions about when it is invoked.

9.1 From exported symbol to operation

The first boundary is the module entry point. The loader must find the exported name and call it using the expected signature. The entry point then returns a provider context and a provider dispatch table. A later query requests an operation class and obtains an algorithm table. An algorithm entry identifies a further dispatch table for that operation. These are separate transitions. A defect in an early transition can prevent a later callback from being reached at all.

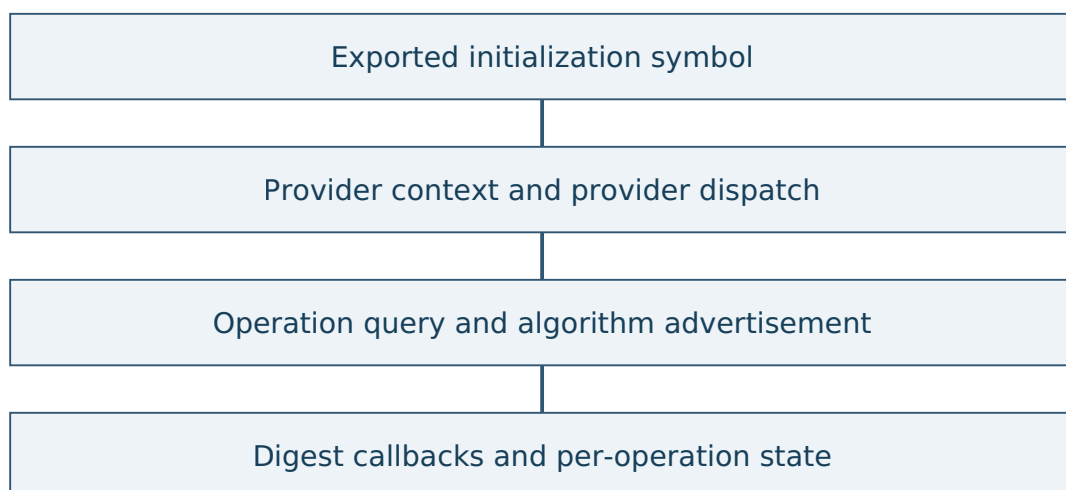


Figure 2. Four reviewable transitions in the teaching module. Each transition can fail independently.

The practical implication is that diagnostic output should correspond to the transition under investigation. A missing entry point is a loader/interface issue. An unsupported operation is an advertisement issue. A property mismatch is a selection issue. A wrong digest length is an execution or metadata issue. Treating all of them as “OpenSSL cannot find my algorithm” obscures the relevant evidence and encourages accidental fixes.

9.2 Dispatch-table structure

The source uses static arrays terminated by a zero identifier and a null function pointer. Static lifetime is important because the core may retain the returned description beyond the query call. Returning a pointer to an automatic local array would produce a lifetime defect even if the array appeared correct during initialization. In the prototype, no dynamic algorithm advertisement is needed, so static data also reduces the number of allocation and synchronization decisions.

The casts to a generic function-pointer type can look unusual to a reader new to this API. They are part of the dispatch representation; the operation-specific function type determines how the pointer is called. Review should therefore check the identifier and the actual implementation signature together. A correctly spelled callback name is not enough if it is paired with the wrong function identifier. The public core-dispatch declarations provide the reference for that comparison. [9]

9.3 Supported and unsupported operations

The provider advertises only a digest operation. It should not return its digest table when asked for a cipher, signature, or other operation class. The callback harness makes this distinction observable by requesting a cipher and expecting a null result. It then requests a digest and checks the algorithm name and property string. These checks test the provider’s own advertisement logic without relying on how the command-line tool formats an algorithm listing.

An unsupported operation is not necessarily an error in a small provider. Providers may have a limited scope. The engineering defect would be to advertise support that the module cannot fulfill, or to let unrelated callbacks be interpreted as the requested operation. The study therefore values narrow, accurate advertisement over a broad list of untested capabilities.

9.4 Algorithm names and identity

The name EDU-SHA256 communicates a teaching-specific operation identity while retaining a recognizable relationship to SHA-256. It prevents the example from silently competing with the standard name in an unqualified application request. That design choice makes experiments easier to interpret, although a real replacement provider may intentionally advertise standard names. Such a replacement would need a more careful property and deployment policy because a familiar algorithm name would no longer identify one implementation by itself.

The property definition and the provider handle are also distinct concepts. The prototype makes them consistent, but the paper does not assume that a property string is cryptographically bound to the module's identity. The programmatic client checks the fetched method's provider object. That observation is useful for diagnosing selection within the OpenSSL process. It is not remote attestation and does not establish the integrity of the module file.

9.5 Parameter descriptors as schema

A gettable-parameter callback acts like a small schema: it describes the names and types the implementation understands. The corresponding get-parameters callback performs the exchange. Correctness involves both the value and its representation. A digest size of 32 stored using an incompatible type is not a valid response merely because a debugger can find the number 32 somewhere in memory. Typed setters help keep the implementation aligned with the requested representation. [10]

The extension tests this principle by asking for the size with a UTF-8-string parameter rather than an integer size parameter. The prototype's setter rejects the incompatible request. It also receives an unknown parameter name and leaves the caller's storage unchanged. That latter test verifies the behavior of this implementation for an unrecognized request; it should not be generalized into a rule that every arbitrary malformed parameter list must be accepted.

9.6 Binary compatibility and deployment

Compilation against public headers is an important first step, but it is not a complete compatibility claim. The host loader, the module's dependencies, and the runtime OpenSSL library still have to agree. The build script records an ordinary Linux shared-object workflow and links against the installed library. It does not attempt to construct a portable binary for unrelated distributions or architectures.

A deployable provider should specify its supported OpenSSL versions, target platforms, build settings, and dependency assumptions. It should also test installation layouts in which the module and its dependencies are found through the intended paths. This paper avoids guessing about compatibility that has not been exercised. The strongest binary claim supported here is that the module built and ran on the recorded OpenSSL 3.5.5 host.

10. Configuration Experiments and Failure Diagnosis

Configuration is often presented as a short preliminary step before the interesting implementation work. In a provider-based application it is part of the executable selection mechanism and deserves its own experimental treatment. This chapter examines explicit activation, coexistence, mandatory properties, and missing modules. The aim is to distinguish a configuration that is syntactically readable from a configuration that actually produces the intended operation.

10.1 Process-local configuration

The test harness starts a separate process for each command-line experiment. Each process receives an explicitly selected `OPENSSL_CONF` value. This prevents previous tests from leaving provider state in the same process and makes the configuration choice visible in the harness. It also means the tests do not evaluate dynamic reconfiguration of an already-running application. That would be a different experiment with different lifetime and synchronization questions.

The generated configuration contains an absolute module pathname. This trades portability of the file itself for clarity during execution. The build script regenerates the pathname after the project is moved, so the reproducing user is not expected to edit a hard-coded path manually. Keeping generation and execution together reduces the chance that the paper's example file and the actual test file diverge.

10.2 Activation matrix

The extended harness varies only the activation value in the edu provider section. The default-provider section remains active. Four cases are evaluated: 1, true, 0, and false. The first two permit the educational digest; the latter two do not. These results agree with the

pinned OpenSSL 3.5.5 configuration description. They should not be used to rewrite the historical findings of the 2024 audit, which discussed a different assessed state of the software. [4,14]

edu activation value	Expected educational digest	Observed outcome
1	Available	Success
true	Available	Success
0	Unavailable	Nonzero command exit
false	Unavailable	Nonzero command exit

Each case uses a complete configuration rather than assuming that changing a fragment is sufficient. The harness uses a final-occurrence replacement so that it changes the edu activation line rather than the default activation line. This small implementation detail matters because an incorrect test transformation could otherwise produce a misleading interpretation of default-provider behavior.

10.3 Missing-module experiment

The missing-module case changes the generated pathname from edu.so to a deliberately nonexistent file. The requested algorithm remains EDU-SHA256. The command fails as expected. This demonstrates that the tested configuration does not somehow supply the educational algorithm after its intended module path has been invalidated. It does not test every possible loader failure, such as a module with the wrong architecture or a module whose dependent shared library is absent.

Operational troubleshooting should avoid changing several variables at once. If an administrator simultaneously changes the module directory, algorithm name, and property query, a later success does not reveal which change fixed the problem. A controlled diagnostic sequence retains a successful baseline and alters one relevant condition. The experiments in this chapter follow that principle wherever practical.

10.4 Mandatory property experiments

The harness compares three queries for the same educational algorithm: `provider=edu`, `provider=missing`, and `fips=yes`. The first succeeds. The latter two fail because the advertised implementation does not satisfy those requirements. The `fips=yes` result is especially useful pedagogically: the module does not claim that property, and the test does not attempt to add it merely to make selection succeed.

Mandatory properties constrain selection rather than transforming the selected implementation. Asking for a property that no implementation provides cannot manufacture the corresponding security assurance. Conversely, an implementation that advertises a property is making metadata available to the selection mechanism; a complete assurance argument must still establish why the advertisement is justified. The example keeps this distinction explicit because property names can otherwise sound stronger than the evidence supporting them. [7,11]

10.5 Default-provider coexistence

The baseline coexistence test verifies ordinary SHA-256 while the `edu` provider is explicitly configured. That test concerns the application's provider set. It is separate from the default provider loaded inside `edu`'s private context. A successful internal backend fetch does not prove that an application request for ordinary SHA-256 will succeed in the application context. Confusing those contexts would hide a genuine configuration problem.

For larger applications, coexistence requirements should be written down as an algorithm inventory. A program may need digest, random-generation, decoding, key-management, and signature operations. Demonstrating one digest does not establish that the rest of the inventory remains available. The present study uses one coexistence check because its application scope is intentionally small; it proposes a broader inventory as a deployment requirement rather than pretending to have tested one.

10.6 Error interpretation and observability

The test records retain exit codes and selected textual output. They do not parse every error stack into a formal taxonomy. That decision keeps the tests robust against incidental wording changes, but it also limits diagnosis. A future harness could classify errors by library and reason codes while preserving the complete original output for review. It should avoid declaring a specific root cause from a generic nonzero status alone.

Provider metadata is another form of observability. The prototype reports a name, a version, and a status. Those values help a developer recognize the module that was loaded. They do not demonstrate that a particular operation was executed, so the client also checks the fetched method's provider. Good operational evidence combines these observations rather than expecting one listing command to answer every question.

10.7 Configuration review procedure

A configuration review can proceed in four stages. First identify the process and the exact file it reads. Next identify the provider entries and module paths. Then identify the algorithm requests and property constraints actually used by the application. Finally execute representative success and failure cases in a controlled environment. This procedure can be documented before implementation, making later configuration changes reviewable against a stable expectation.

The review should also identify who may modify the files and directories involved. Because a provider is native code, accidental or unauthorized module replacement is not merely a configuration inconvenience. The paper does not implement an operating-system access-control policy, but it explains why that policy belongs in the deployment design. Configuration correctness and file integrity are related responsibilities, not interchangeable ones.

11. Memory Ownership and Digest State

Provider development combines cryptographic computation with ordinary systems-programming obligations. The mathematical output can be correct while the surrounding code leaks memory, retains invalid pointers, or releases resources in the wrong order. The teaching provider is small enough that its complete ownership graph can be examined. This chapter explains that graph and connects it to the direct callback experiments.

11.1 Provider-wide resources

The provider context contains three owned resources: a library context, a default-provider handle within it, and a fetched SHA-256 method. Initialization establishes them in that order. A later resource depends on earlier resources being available. Teardown reverses the important dependency direction: it releases the fetched method, unloads the backend handle, and then frees the library context. The allocation containing these fields is released last.

A single teardown path is also used for partial initialization failure. This reduces duplicated cleanup code, but it creates a review obligation: each cleanup operation must tolerate the states that can reach it. The prototype initializes the provider structure to zero so that fields not yet established are null. The successful experiments exercise the ordinary path; they do not systematically force every intermediate allocation or load to fail.

11.2 Per-operation resources

Every digest operation gets a separate inner `EVP_MD_CTX`. Its mutable message state is therefore not stored directly in the provider-wide structure. The digest context also holds a reference to the provider structure in the ordinary C sense of a pointer, not an independently

counted application-level ownership claim. Correct lifetime therefore depends on the surrounding provider and method lifecycle as well as on the local allocation code.

The client follows a conservative release order: digest contexts are freed first, then the fetched method, then the provider handle, and finally the application library context. This explicit order makes the example easy to review. It should not be converted into an unsupported claim that every other sequence is safe or unsafe under every OpenSSL reference-counting condition. Such claims require the exact API contract and targeted tests. [5,6,8]

11.3 Digest state transitions

Conceptual state	Transition	Responsibility
Allocated	newctx creates inner state.	Return a valid context or failure.
Initialized	init selects the cached SHA-256 method.	Establish a fresh operation.
Updating	update accepts a message fragment.	Preserve previous input and process new input.
Finalized	final returns 32 digest bytes.	Respect output capacity and reported length.
Duplicated branch	dupctx copies an intermediate operation.	Own independent mutable inner state.
Released	freectx frees the inner state and wrapper.	End ownership without reusing the pointer.

The table is a conceptual model for valid use, not a complete specification of every possible invalid sequence. The prototype does not maintain an additional explicit enum for these states. It delegates much of the underlying state behavior to EVP. The tests therefore avoid asserting undocumented behavior for arbitrary sequences such as updating a released pointer or finalizing uninitialized memory. Those are not meaningful production use cases to support by accident.

11.4 Finalization and capacity

The finalization callback receives an output pointer, a capacity, and a place to report the number of bytes written. The prototype requires at least 32 bytes. It reports zero before rejecting an insufficient buffer and does not call the backend when the guard fails. This arrangement has two testable consequences: caller storage should remain untouched during the rejected attempt, and the operation should remain available for a later correctly sized finalization.

The callback harness initializes a 64-byte output array with a sentinel value and varies the advertised capacity from zero to 33. For capacities below 32, it checks failure, a zero reported length, and unchanged sentinel bytes. It then retries with sufficient capacity and compares the result with the reference digest. For capacities 32 and 33, it checks success directly. Bytes beyond the 32-byte digest remain sentinel values in each successful case.

These observations provide concrete evidence about the boundary guard. They do not establish that arbitrary invalid pointers are safe to pass. A capacity value is a promise made by the caller about accessible storage; testing ordinary allocated arrays does not simulate malicious pointer values. The harness deliberately separates legitimate boundary conditions from undefined memory accesses that would not constitute a reasonable interface guarantee.

11.5 Duplication and independence

The baseline duplication test branches after processing one byte. Both branches then receive the same suffix and produce the same expected result. This proves that the implemented duplication path is reachable and preserves the tested prefix. It is a modest test: because both suffixes are equal, a stronger future case should feed different suffixes and verify two different expected digests. The paper retains that limitation rather than upgrading the claim beyond the implemented test.

A naive duplication that merely copied a pointer to the same inner `EVP_MD_CTX` would make both wrappers refer to shared mutable state and risk double release. The prototype instead allocates a new wrapper and uses `EVP_MD_CTX_copy_ex`. The function's return value is checked, and a failed copy releases the newly allocated wrapper. This is an example of how a small amount of deliberate ownership code prevents a large class of confusing integration defects.

11.6 Nulls, parameters, and cleanup

The direct tests include a null output pointer and a null output-length pointer in finalization. The implementation rejects them before producing a digest. It also tests reinitialization of an operation after these rejected calls. Parameter tests check a wrong type and an unknown name. Together these cases examine several boundaries that successful hashing alone would not exercise.

Memory-safety tooling remains an important next step. A sanitizer build could detect certain invalid accesses during the same tests. Leak analysis could inspect repeated load and unload cycles. Allocation-failure injection could examine rarely reached cleanup paths. None of these tools was run as part of the present results, so the evidence is described as boundary and ownership testing rather than as a complete memory-safety assessment.

12. Extended Testing Methodology

The extended test plan separates three layers: application-visible integration, provider-facing callbacks, and documentary comparison. This separation makes it easier to identify what each result means. Integration tests demonstrate that a normal OpenSSL consumer can use the module. Callback tests reach interface boundaries that may not be directly controllable from a high-level consumer. Documentation establishes the intended meanings of the interfaces and version-specific configuration options.

12.1 Deterministic input generation

The new integration harness generates each byte using a simple arithmetic pattern, reducing the risk that all tested messages contain only a single repeated value. For an input of length n , byte i is computed from $(17i + 31)$ modulo 256. This is not a source of cryptographic randomness and is not presented as one. It is a deterministic way to exercise a variety of byte values while making every test message reproducible without storing separate binary fixtures.

The selected lengths cover ordinary small messages, output-size landmarks, SHA-256 block and padding boundaries, and larger application sizes. The set is 0, 1, 2, 3, 31, 32, 33, 55, 56, 57, 63, 64, 65, 119, 120, 121, 127, 128, 129, 255, 256, 257, 4095, 4096, 4097, 65535, 65536, and 65537. These values are not a proof of exhaustive coverage. They are a deliberate partition of boundary conditions that is more informative than choosing several arbitrary lengths.

12.2 Why block boundaries matter

SHA-256 operates on 64-byte message blocks, but padding and encoded message length also consume space in the final block. Values around 55 and 56 bytes, and corresponding positions in a later block, are

therefore useful reference-test boundaries. The provider itself does not implement that padding; its backend does. Testing these values mainly establishes that the bridge passes lengths and bytes correctly and does not accidentally truncate at a boundary. The underlying algorithmic interpretation comes from the hash standard. [13]

The sizes around 4096 and 65536 bytes serve a different purpose. They resemble common application buffer landmarks, without claiming any special behavior of this host's I/O implementation. A successful command-line comparison at those lengths shows that the tested path handles the full input. It does not demonstrate a particular buffering strategy inside the executable or establish that a single update callback received the entire message.

12.3 Reference computation

For each generated input, Python computes an expected digest through `hashlib.sha256`. The command-line operation is asked for binary output, and the harness compares bytes rather than a human-readable label. It checks the process exit code as well as the digest. This avoids accepting an empty output or a formatted error message as though it were a digest result.

The reference is a practical integration oracle, but it is not necessarily algorithmically independent. Python may use OpenSSL internally, and the provider bridge certainly does. The study therefore distinguishes a reference computation from an independent cryptographic implementation. Published known-answer values and a separate backend would strengthen an algorithm-validation argument, but the current work is principally about interface behavior.

12.4 White-box callback testing

The callback harness loads `edu.so` with the platform dynamic loader, obtains the initialization symbol, and retrieves the returned dispatch tables. It uses the public typed extraction helpers to obtain callback pointers. Because this specific module ignores the incoming core

handle and callbacks, the harness can supply nulls for those arguments. A provider that depends on core allocation, error, or configuration callbacks would require a more complete test environment.

This is an important limitation of the method, not a hidden shortcut. The harness is white-box testing for a known implementation. Its purpose is to evaluate the digest guard, metadata, unsupported-operation response, and reinitialization behavior of that implementation. The ordinary integration tests still establish that the same module works when OpenSSL itself supplies the surrounding lifecycle.

12.5 Counting tests and assertions

The extension contains 37 integration cases: 28 message-length cases, three property cases, four activation cases, one missing-module case, and one parallel-process case. The callback program reports 3,362 assertions. These numbers measure different things. An assertion can check one sentinel byte, while an integration case may involve a complete process, load, fetch, and digest. Adding the numbers together would create a misleading impression of thousands of independent end-to-end tests.

The large assertion count mainly reflects byte-by-byte guard checks across the output-capacity loop. It is reported for reproducibility, not as a security score. A smaller number of carefully chosen failure cases can be more informative than many repeated assertions about the same behavior. The test design is therefore explained alongside the counts.

12.6 Parallel process test

The harness launches sixteen independent digest processes using four worker threads in the Python controller. Each process has its own address space and provider instance. All results match their reference values. This demonstrates that the installed module and process-level invocation can be used concurrently without conflict in the tested scenario.

It does not test multiple threads sharing a provider context inside one application. It also does not test concurrent use of one digest context, which would be a separate state-sharing question. The machine-readable result explicitly records that `shared_provider_context` is false. This annotation prevents a later reader from describing the result as an in-process thread-safety test.

12.7 Negative tests and stopping rules

Each test stops on an unexpected result. Negative tests define success as the expected rejection rather than as a zero process exit. The harness does not silently continue and report a partial pass. A complete successful run writes a JSON result file and a final summary. The saved build log supplies the execution record used by the paper.

The test suite is intentionally deterministic and bounded. It is not a fuzzing campaign and has no probabilistic coverage claim. Future fuzzing should target a clearly defined input surface, such as parameter combinations or state transitions, and report its duration, corpus, instrumentation, and discovered failures separately. Reusing the word “tested” without those distinctions would hide important differences in assurance.

13. Extended Results and Interpretation

The expanded experiments were run against the unchanged educational provider on the same OpenSSL 3.5.5 host as the baseline. Compilation completed with warnings treated as errors. The callback program reported 3,362 passing assertions, and the integration harness reported 37 passing cases. The original eight baseline checks also passed as part of the extended build script. This chapter interprets those observations without treating the counts as independent measures of cryptographic strength.

13.1 Result summary by category

Category	Cases or coverage	Result
Baseline integration	8 checks	All passed
Deterministic message lengths	28 lengths, 0 through 65,537 bytes	All matched reference
Mandatory properties	3 requests	One selected; two rejected as expected
Activation values	4 configurations	Two activated; two remained unavailable
Missing module	1 invalid path	Request failed as expected
Independent processes	16 executions grouped in 1 case	All matched reference
Direct callbacks	3,362 assertions, including capacities 0–33	All passed

The results support the expected behavior of the wrapper’s selection and data path. They also provide a concrete example of evidence triangulation: algorithm output is tested through the command-line

tool, while output-buffer handling is tested directly through the callback. A failure at either layer would require a different debugging approach even if both eventually affected the same digest request.

13.2 Interpretation of the capacity sweep

For insufficient capacities, the wrapper returned failure, set the output length to zero, and preserved all sentinel bytes. A subsequent sufficiently sized finalization succeeded. For capacities at least 32, the digest was written and bytes after the output remained unchanged. These observations are consistent with the source-level guard occurring before backend finalization.

The sweep also establishes that the interface is measured in bytes rather than in the length of a hexadecimal representation. A SHA-256 digest requires 32 binary output bytes, while its usual hexadecimal representation occupies 64 characters. Confusing these units is a realistic application error. The callback checks binary capacity; formatting belongs to a different layer.

13.3 Interpretation of parameter tests

The parameter tests returned the expected size, block size, and extendable-output flag. A string-typed size request was rejected. An unknown named parameter left its associated integer unchanged. These results demonstrate that the prototype is using typed setters rather than blindly writing through every caller-provided pointer.

They do not establish behavior for every malformed `OSSL_PARAM` array. The arrays in the harness are properly terminated and use valid allocated storage. Missing terminators, inaccessible pointers, and arbitrary corrupted structures are outside this experiment. This boundary is worth stating because successful type rejection can otherwise be exaggerated into a general parser-hardening claim.

13.4 Version-specific configuration observations

The activation experiment agrees with the selected documentation version: true-like values activate the provider and false-like values do not. The distinction is especially useful when reading older discussions of provider configuration. The correct response to an historical finding is to identify the assessed version and retest the current version, not to assume either that the finding remains unchanged or that it is irrelevant.

The results do not identify which change introduced any historical behavior difference. Establishing that would require source-history analysis, release comparison, and possibly reproducing an older build. The paper therefore limits its statement to observed OpenSSL 3.5.5 behavior and the wording of the corresponding snapshot. This is a more defensible conclusion than inferring a complete maintenance history from one experiment.

13.5 Selection policy and the backend

The `fips=yes` request fails at outer selection because the educational algorithm does not advertise that property. This confirms an important boundary in the experiment: the module does not become available under that query simply because it delegates to a familiar digest algorithm. The test does not inspect a validated module and cannot establish compliance.

The successful `provider=edu` request remains compatible with the separate internal `default-provider` fetch. That nested selection is intentional but demonstrates why the outer provider identity is not a complete explanation of the cryptographic execution chain. An operational report that records only “`provider=edu`” would omit the backend used by the bridge. A production bridge should document or expose its backend policy in an appropriately controlled way.

13.6 What remained unchanged

The extended tests did not require a change to the provider's implementation. This is useful evidence that the baseline code already handled the newly exercised boundary conditions. It is not evidence that no undiscovered defects remain. The test plan was derived partly from the source, so it naturally emphasizes behaviors the source makes visible. An independent reviewer might identify different risks or failure modes.

No performance claim is added. No security validation, device integration, or in-process multithreaded stress run is reported. The paper's assurance remains deliberately scoped to the observed functional and boundary behavior. Subsequent chapters discuss how those remaining concerns could be investigated without relabeling a proposal as an experimental result.

14. Performance Evaluation: A Proposed Method

The functional results do not measure throughput or latency. Nevertheless, performance is a natural concern for an implementation that introduces another layer around a digest backend. A bachelor's-level engineering paper should explain how such a concern could be investigated without manufacturing measurements. This chapter therefore presents a proposed experiment and analytical model. No numerical performance result is claimed.

14.1 Costs that should be separated

A first invocation can include shared-object loading, provider initialization, backend loading, method fetching, context allocation, digest initialization, message processing, and finalization. A repeated invocation with a previously fetched method may include only some of these costs. Reporting one elapsed time without stating which costs are included would make comparisons difficult to interpret.

For the bridge, a useful conceptual decomposition is total time equals setup time plus per-operation overhead plus backend processing time. This is an accounting model rather than a prediction with measured coefficients. It helps formulate experiments: compare cold loading with warm operation, vary message size, and keep the intended backend constant. For tiny messages, fixed overhead may dominate; for large messages, backend processing may dominate. Those are hypotheses to test, not observations from the current run.

14.2 Baselines and comparable work

The most relevant baseline would fetch default-provider SHA-256 directly in the same application and process messages through the same high-level EVP pattern. Comparing the bridge with an unrelated

command-line utility would confound provider overhead with process startup, input handling, and formatting. A fair benchmark should use binary buffers already in memory and equivalent update schedules.

A second comparison could retain the bridge but vary how long provider and method objects are reused. This would estimate the cost of repeatedly establishing resources rather than the cost of the digest callbacks themselves. The benchmark must not optimize one path by caching resources while repeatedly rebuilding the other unless that difference is the explicit subject of the experiment.

Experiment	Fixed conditions	Varied condition
Cold-start cost	Message, executable, backend	Fresh process versus warmed process
Method-fetch overhead	Provider already loaded	Fetch once versus fetch each operation
Message scaling	Cached method and context policy	Input size
Update fragmentation	Total message size	Number and size of update calls
Parallel throughput	Independent operation contexts	Worker count

14.3 Measurement procedure

A proposed measurement should use an appropriate monotonic clock, run enough iterations to exceed timer granularity, and repeat the experiment to characterize variability. It should record the operating system, CPU model, compiler options, OpenSSL build, message sizes, and iteration counts. Warm-up policy should be stated explicitly, because a benchmark that excludes initialization cannot be interpreted as application startup latency.

The experiment should also validate output outside the timed inner loop or at a controlled sampling frequency. Otherwise an optimization error could make a fast but incorrect path look attractive. The compiler must

not be allowed to remove the computation as unused work. Retaining and checking a digest is a straightforward way to keep the requested operation meaningful.

14.4 Statistical interpretation

Repeated observations should be summarized with measures that expose variability, not only the most favorable run. Median and spread can be useful for noisy timing data; confidence intervals require an explicit sampling and independence model. The paper would need to explain why repeated measurements are sufficiently comparable rather than assuming that a large iteration count automatically removes bias.

Potential confounders include CPU frequency scaling, other processes, thermal state, memory allocation, and cache effects. A completely idle laboratory machine may be difficult to obtain, but uncontrolled conditions should still be reported. A claim such as “the provider adds five percent overhead” would be incomplete without the workload, reuse policy, and uncertainty that produced the number.

14.5 Memory and resource measurements

Runtime memory investigation should distinguish provider-wide allocations from operation allocations. The bridge deliberately caches a backend method once per provider instance and allocates an `EVP_MD_CTX` per digest context. A workload with many concurrent operation contexts would exercise a different resource pattern from one that repeatedly reuses a single context. Measuring only process memory after startup could miss that distinction.

Leak checking and peak-memory measurement are also different tasks. A stable peak does not prove that every resource is released correctly, and a leak-free short run does not describe the peak resource demand of a large concurrent workload. A complete investigation should state which property each tool is being used to assess.

14.6 Reporting a negative or inconclusive result

If the measured difference falls within run-to-run variability, the appropriate conclusion may be that the experiment could not resolve the overhead under the chosen conditions. It would be misleading to force such a result into a claim of equal performance. Likewise, a performance regression may be acceptable for a provider that enables a required hardware or policy feature. The evaluation must connect measurements to the application's requirements.

The present prototype was not optimized for benchmark leadership. Its purpose is to make the provider boundary explicit. This chapter supplies a reproducible direction for future work while maintaining the distinction between a planned experiment and a completed one.

15. Packaging, Deployment, and Operational Review

A provider that works in a developer's build directory is not yet a deployed service component. Deployment introduces file layout, dependency resolution, permissions, upgrade coordination, and rollback. This chapter develops a reviewable deployment model for a future production module while identifying which parts are present in the teaching artifact.

15.1 Separating build and installation

The prototype remains in a local build directory. Its configuration is generated for that directory, and no system configuration is changed. This makes the experiment easy to repeat and easy to remove. A production package would need an intentional installation prefix, an agreed module directory, and a method for locating dependent libraries. The chosen layout should be documented rather than inferred from whichever directory happened to exist during compilation.

Build artifacts and runtime inputs should also be separated conceptually. Source code and compiler logs support reproducibility; the runtime needs the module and its dependencies, together with configuration and any operational data. Shipping every build output into a privileged module directory makes review harder and increases accidental exposure of files that have no runtime purpose.

15.2 Configuration ownership

An organization should identify which component owns the provider configuration. If an application ships its own configuration, that file must be coordinated with the application's invocation environment. If a system administrator supplies a shared configuration, applications need a documented compatibility expectation. The study avoids taking a position for all deployments because the correct ownership model depends on the environment.

The important engineering property is consistency: the person or process changing provider policy should know which applications will read the changed file. Editing a file that one shell command uses may have no effect on a service that inherits a different environment. Conversely, modifying a global file may affect applications that were not part of the change request.

15.3 Integrity and access control

The module directory should not be writable by identities that are not trusted to change application code. The same reasoning applies to a configuration file that can redirect loading to another directory. A checksum can detect accidental changes when compared with a trusted manifest, but the manifest itself must have a trustworthy source. Merely placing a checksum beside an untrusted module does not authenticate either file.

The paper includes SHA-256 manifests for artifact integrity and version tracking. They are not a software-signing system. A production distribution might integrate signed packages, controlled repositories, and organizational release approvals. Those mechanisms belong to the deployment context and are not implemented by the educational provider.

15.4 Upgrade compatibility

An upgrade can change the module, the host OpenSSL library, the configuration, or all three. The test matrix should be rerun for the intended combination. If an operation succeeds after an upgrade, the selection identity should still be checked; otherwise a fallback or a different implementation may conceal a compatibility problem. Negative property and missing-module tests can be useful regression guards because they verify that policy failures remain visible.

Keeping the prior artifact and configuration available supports rollback, but rollback also needs a decision rule. A production release plan should state what failures trigger rollback and how the previous state

will be restored. This paper does not execute a service rollout, so it presents that requirement as a design recommendation rather than as completed operational evidence.

Change	Minimum review question	Suggested regression evidence
New module binary	Does it advertise the same intended operations?	Loading, metadata, selection, and functional tests.
New OpenSSL runtime	Does the tested ABI and configuration remain valid?	Rebuild and repeat the full local suite.
New configuration	Which application requests change selection?	Positive and mandatory-negative property cases.
New installation path	Are the module and dependencies resolved correctly?	Clean invocation from the intended service environment.

15.5 Logging and observability

Useful provider diagnostics identify the stage of failure without exposing sensitive application material. This digest example has no private keys, but future providers may process secrets. A log that indiscriminately records parameter contents or input buffers can become a security problem even when it helps debugging. The diagnostic design should decide which identifiers and status information are safe to record.

The teaching client prints the selected provider identity and a known digest result. The harness stores configuration-case outcomes and reference hashes for deterministic public inputs. These records are appropriate for the experiment. They should not be copied unchanged into a production logging policy for arbitrary customer data.

15.6 Supportability and handover

A deployable provider needs documentation that can be used by someone other than its original author. At minimum, that documentation should describe the supported algorithms, configuration options, required properties, installation paths, dependency versions, expected diagnostics, and test procedure. An ownership diagram is useful when maintainers extend the code because it explains why a resource is released at a particular stage.

The accompanying artifact is organized with this handover goal in mind: source, tests, results, and references are stored separately. The PDF provides the design argument, while executable scripts provide the procedure. Neither format replaces the other. A prose-only paper would be difficult to reproduce, and a source-only archive would leave important assumptions implicit.

15.7 Limits of the deployment discussion

No production service, package-signing infrastructure, or operating-system hardening profile was installed as part of this study. The local module has not undergone an independent security review. The deployment discussion should therefore be read as an engineering framework for a subsequent project, not as a statement that the present artifact meets those production requirements.

16. Policy Boundaries and Cryptographic Assurance

Provider selection can make cryptographic policy more explicit, but it can also create an illusion that policy is enforced solely by naming a provider. The educational bridge exposes this issue clearly: an outer fetch selects `edu`, while a second fetch inside the module selects the default provider. Understanding both decisions is necessary to explain which code performs the operation.

16.1 Outer and inner selection

The application's library context contains its own provider availability and selection state. The bridge creates a different context for its backend. The backend request specifies `provider=default`, so it does not recursively resolve to `EDU-SHA256` and does not depend on an unqualified choice in the outer context. This is a deliberate way to make the example predictable.

Predictability is not the same as policy inheritance. If the application intended to restrict all cryptographic computation to a particular class of implementations, the bridge's separate context could violate that intention unless the backend policy were designed accordingly. The example does not claim to preserve arbitrary outer policy. Its source and paper make the internal choice visible.

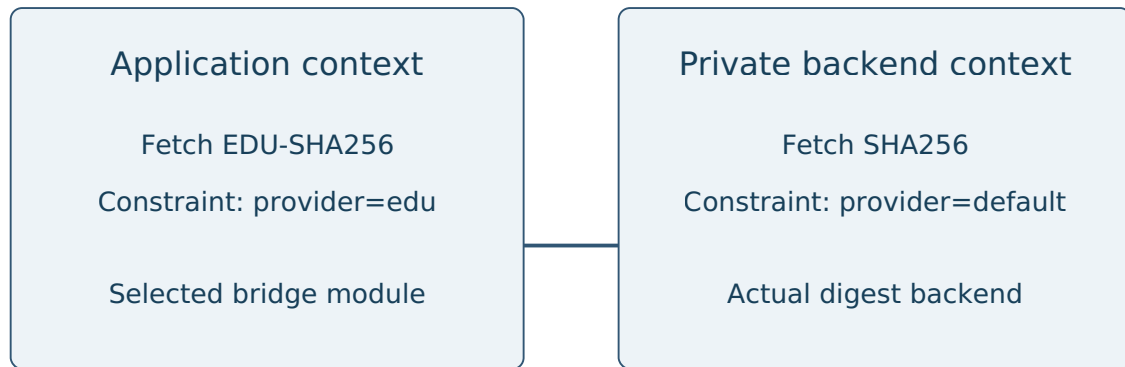


Figure 3. The prototype has two explicit selection domains. The connection is application code, not automatic inheritance of policy.

16.2 Properties are claims used for selection

A property query tells the fetch mechanism which advertised implementations are acceptable. It does not verify the truth of every advertisement. A production system therefore needs a basis for trusting the module that supplies the property definition. The loader and property mechanism solve routing problems; software provenance, validation, and operational controls solve different assurance problems.

The `fips=yes` negative case makes the distinction tangible. The prototype does not advertise the property, so the request fails. Adding the property string to the algorithm table would change selection behavior but would not perform a validation process. The paper intentionally does not make that change. [7,11]

16.3 Standard algorithm versus validated module

SHA-256 is a standardized algorithm. A module can compute its mathematical output correctly without being part of an approved or validated deployment. The algorithm specification describes the transformation; module assurance involves additional requirements and a defined operating context. Confusing these levels can lead an implementer to interpret a successful known-answer test as a compliance certificate.

The bridge is especially unsuitable for such an inference because it delegates to a backend and supplies only limited integration evidence. Even if a backend had a relevant validation status, a complete argument would have to examine the selected version, configuration, boundaries, and operating conditions. The study does not perform that analysis and does not label the prototype as validated.

16.4 Data and error channels

The provider receives message data and can influence errors observed by the application. In the current example the input messages are deterministic test data, but a real digest request may still involve sensitive information. A provider implementation should not assume that a digest operation is harmless to log simply because it has no private-key argument. Input contents, timing, and contextual metadata may matter to the application's confidentiality requirements.

The prototype's diagnostic support is minimal: it propagates failures and uses ordinary error printing in the client. It does not implement a provider-specific error catalogue through core callbacks. That omission is acceptable for a teaching bridge but is a clear limitation for maintainers who need reliable operational diagnosis without excessive logging.

16.5 Trust assumptions in the laboratory

The experiment trusts the local compiler, the installed OpenSSL library, the operating system, and the downloaded documentation sources for their respective roles. It also trusts the module file being tested not to change between compilation and execution. These assumptions are normal for a small laboratory exercise, but they should be visible because they bound what the results can establish.

The manifests preserve file identity for later inspection. They help detect accidental changes in the reference collection and project artifacts. They do not prove that the host was uncompromised or that

the compiler produced a faithful binary. A reproducibility package increases inspectability; it does not remove every trust assumption in the software supply chain.

16.6 A policy-oriented acceptance argument

A future deployment could structure acceptance around explicit claims. One claim might state that a particular application request selects a named implementation. A second might state that the module's backend follows an approved policy. A third might state that the deployed files match a reviewed release. Each claim requires its own evidence. The teaching project directly supports only selected parts of the first claim.

This decomposition is useful because it gives reviewers concrete questions to ask. Which property query was used? Which provider was attached to the fetched method? Does the module perform a second fetch? Which files and configuration were deployed? What independent evaluation supports the module's assurances? Answers to these questions are more informative than a broad statement that "OpenSSL providers are enabled."

17. Extending the Design Beyond Digests

A digest provider is a useful first implementation because it has message state but no persistent private-key object. The same architectural ideas can support more complex operations, but the complexity does not grow only by adding another function to the dispatch table. New operation families introduce new data lifetimes, policy decisions, and failure modes. This chapter describes a possible extension path without claiming that these operations have been implemented.

17.1 From a bridge to an independent backend

The smallest conceptual extension is to replace the default-provider SHA-256 backend with another implementation while retaining the outer algorithm interface. The existing integration tests would still be useful: they would verify loading, selection, message handling, and output. However, the assurance meaning of the result would change. With an independent backend, vector comparison could provide stronger algorithm-level evidence than it does for the present wrapper.

Such an extension should not begin by writing cryptographic compression code casually. The project would need a clear source and review history for the primitive, relevant test vectors, and an evaluation of implementation risks. The teaching bridge avoids that undertaking deliberately. It demonstrates the integration boundary so that an independently justified backend could later be placed behind it.

17.2 Message authentication

A message-authentication operation introduces a key and therefore additional responsibilities that a public digest example does not exercise. Key material needs a defined owner and lifetime. Initialization parameters may become meaningful. Error and logging policies must

account for secrets. The implementation must also distinguish an algorithm's public parameters from sensitive state used during a specific operation.

The current ownership tables provide a starting point but would need to be extended rather than copied mechanically. A reviewer should ask where key bytes originate, whether they are duplicated, when copies are cleared, and which component is authorized to use them. Successful hashing does not establish answers to any of those questions.

17.3 Signatures and key management

A signature-capable provider typically needs a coherent relationship between the representation of keys and the operations that use them. Applications may import, generate, load, or reference keys through different paths. An implementation that supports a signature callback but cannot provide the associated key-management behavior may not satisfy a real application's workflow. OpenSSL's provider and migration documentation points to the wider set of operations involved in provider-based key handling. [1,12]

A sensible undergraduate follow-on project would choose one narrowly defined signature workflow and draw its object lifecycle before implementing it. For example, it could specify that a key is loaded from one controlled source and used for one supported signature operation. The test plan should then include incorrect key types, unsupported parameters, failure to load the key, and expected verification outcomes. This is a proposed project, not an implemented feature of edu.

17.4 Hardware-backed operations

A hardware-backed provider adds a boundary outside the process. Device sessions, authentication, communication errors, and retry behavior become part of the operation. A failure may occur because the cryptographic request is invalid or because the device is unavailable. Treating both as one generic error can make applications difficult to operate and may lead to unsafe retries.

The digest bridge's backend context is entirely local. It does not model device state or establish any HSM integration. Nevertheless, the separation between outer operation and backend selection is a useful teaching analogy. It encourages the implementer to expose a stable application interface while documenting the additional assumptions and policies of the backend.

17.5 TLS integration

A working digest provider is not automatically a TLS provider. A TLS application may require a coordinated set of digest, cipher, key-exchange, signature, random-generation, and key-handling capabilities. It may also impose protocol constraints that are not visible in a standalone digest command. An extension project should begin with the actual application's operation inventory rather than assuming that successful digest fetching is enough.

Integration tests should exercise the intended application path, including expected failures. A successful isolated operation does not show that certificate processing, negotiated algorithms, or provider selection across the connection use the desired implementation. The present study does not start a TLS connection and makes no claim about TLS interoperability.

17.6 Avoiding accidental scope expansion

Every advertised capability expands the review burden. Unsupported operations should remain unsupported until implementation and tests justify them. A staged project can first establish loading and dispatch, then add one operation with boundary tests, and finally integrate its target application. The current artifact addresses the first two stages for a digest bridge; application-specific integration and deployment remain future work.

18. Reproducibility and Artifact Organization

A technical paper is stronger when another investigator can inspect the exact experiment rather than reconstructing it from abbreviated listings. The expanded artifact therefore includes the provider source, the original client, the callback harness, the integration harnesses, configuration generation, result records, and reference snapshots. This chapter explains how these pieces support reproduction and where host dependencies remain.

18.1 Source and result separation

The implementation directory contains executable source and scripts. The results directory contains observed outputs. The references directory contains downloaded documentary material and provenance. This organization helps distinguish a test definition from the record produced by running it. A result file should not be treated as a replacement for the script that generated it.

The paper's rendered HTML and PDF are also kept alongside their generating sources. This makes presentation changes reviewable and allows the table of contents and page numbers to be regenerated consistently. The original shorter edition is archived separately so that the expanded edition does not erase the earlier deliverable.

18.2 Reproduction commands

```
cd room4/papers/psc
./implementation/extended-build.sh
cat results/extended-tests.json
cat results/extended-build.log
```

The command compiles the provider and client, runs the eight baseline checks, compiles and executes the direct callback test, and runs the 37-case integration extension. It requires a Linux C toolchain, pkg-config,

OpenSSL development files, Python 3, and the platform dynamic-loader interface used by the callback test. It does not install the module into a system directory.

The generated configuration uses the current absolute build path. Rebuilding after a move is therefore part of the reproduction procedure. Copying an old configuration file without regenerating it could point at the wrong module, undermining the interpretation of a successful command. The script is designed to remove that ambiguity.

18.3 Recording the environment

The saved OpenSSL version output records the runtime and build information reported by the installed executable. The compilation uses pkg-config to obtain headers and libraries. A reproducing investigator should check that those development files correspond to the intended runtime rather than assuming that every tool named openssl on the machine uses the same installation.

For broader comparative work, additional environment fields should be recorded: compiler version, operating-system release, architecture, relevant environment variables, and module checksum. The expanded artifact includes environment observations and integrity manifests, but it does not claim to recreate an entire virtual machine. A clean-machine reproduction remains a distinct and valuable exercise.

18.4 Reference snapshots

The twelve OpenSSL manual sources are pinned to the openssl-3.5.5 tag rather than downloaded only from a moving documentation page. Plain-text conversions are included for offline reading. The two PDF references are the NIST hash standard and the Trail of Bits assessment. The manifest records source URLs, retrieval date, and file hashes. This helps a reader distinguish the exact material used in the study from later revisions.

A snapshot is especially important for configuration semantics. The paper explicitly compares a historical assessment with a later documented version without assuming that their observations describe the same release. Preserving the reference version makes that qualification inspectable rather than relying on a vague statement that “the documentation says so.”

18.5 Reproducibility versus repeatability

Repeating the scripts on the same host is a narrower achievement than reproducing the result independently on another host. The current results demonstrate the former and provide material for the latter. An independent reproduction could expose assumptions about library search paths, compiler options, platform loading behavior, or packaging that were invisible on the original machine.

For this reason the paper does not infer broad portability from one successful build. It provides a method for checking portability: rebuild, run the same positive and negative tests, inspect provider identity, and compare the result records. Differences should be analyzed rather than dismissed as environmental noise.

18.6 Maintaining result integrity

Result files should be regenerated after implementation changes and identified with the implementation they describe. Editing a test expectation merely to make a failure disappear would damage the evidential value of the study. When behavior changes intentionally, the requirement and rationale should be updated together with the test.

The saved JSON records contain expected conditions and observed success status for the integration cases. The callback log reports its completed assertion count. These records are useful, but they remain outputs of local software. They are not externally certified measurements. The paper’s claims are bounded accordingly.

18.7 Suggested independent review

An independent reviewer can begin by reading the provider source before examining the claimed outcomes. The reviewer can trace initialization and teardown, identify how backend selection is constrained, and check the guard in finalization. Running the baseline suite then establishes the ordinary integration path. Running the callback harness provides more targeted evidence about the identified boundaries.

A useful final exercise is to propose an untested failure case and explain why the current suite would or would not detect it. This turns reproduction into critical evaluation rather than mechanical execution. Examples include allocation failure, different suffixes after context duplication, and concurrent threads sharing provider-wide resources. The paper explicitly leaves these questions available for further work.

19. Discussion and Threats to Validity

The extended study provides more evidence than the baseline while preserving the same central limitation: edu is a bridge to an existing default-provider digest. This chapter evaluates the strength of the findings, alternative explanations, and the educational value of the design. It also identifies where the expansion has closed a gap and where it has merely described future work more carefully.

19.1 Internal validity

Internal validity concerns whether the observations support the intended explanation. Distinct algorithm naming, mandatory property matching, and inspection of the fetched provider reduce uncertainty about which outer implementation is selected. Controlled configuration files reduce uncertainty about activation. Deterministic binary inputs reduce uncertainty caused by shell quoting or text conversion.

Some uncertainty remains. The command-line test uses process exit status as part of its acceptance criterion and does not classify every error code. A negative case that fails for an unrelated reason could potentially be misinterpreted if its positive counterpart were not also checked. Running related positive and negative cases under otherwise matching conditions mitigates this problem but does not make the harness a complete diagnostic system.

19.2 Construct validity

Construct validity asks whether the measurements correspond to the concepts being discussed. “Provider correctness” is too broad to be measured by one digest comparison. The paper decomposes it into loading, advertisement, selection, execution, state handling, and specific boundary behavior. This decomposition improves the relationship between a claim and the observation used to support it.

The distinction between assertion count and integration-case count is another construct-validity issue. The 3,362 assertions include repeated sentinel checks. Treating them as thousands of independent cryptographic tests would misrepresent what was measured. The paper instead reports the capacity range and exact behaviors exercised, leaving the count as a reproducibility detail.

19.3 External validity

The implementation was built and tested on one Linux x86_64 OpenSSL 3.5.5 environment. Other platforms may use different module suffixes, export conventions, compiler behavior, or dependency layouts. Other OpenSSL versions may change available interfaces or configuration semantics. The paper therefore does not extrapolate a successful local run into universal support.

The operation scope is also limited. A digest has simpler state than a private-key provider or a hardware-backed operation. The ownership and dispatch lessons transfer conceptually, but the experimental results do not establish correctness for those more complex domains. An extension would need new requirements and tests rather than only a renamed algorithm table.

19.4 Correlated reference implementations

The strongest limitation on algorithm-level conclusions is backend correlation. The provider delegates to OpenSSL's default SHA-256, and Python may use the same library family. A shared algorithm defect could therefore survive comparison. The short known-answer case helps anchor expected output, but the study remains an integration experiment rather than an independent SHA-256 validation.

This limitation does not make the integration evidence meaningless. Incorrect lengths, wrong byte handling, selection mistakes, and callback-state errors can still produce mismatches even when the backend is shared. The appropriate interpretation is layered: the tests exercise the bridge and its interfaces while relying on the backend for the cryptographic primitive.

19.5 White-box bias

The direct callback tests were designed with knowledge of the source. They target visible guards, metadata setters, and supported operations. This makes them precise, but it can bias the test plan toward behaviors the implementer already anticipated. Independent test design or fuzzing might reveal conditions not suggested by the current control flow.

The paper addresses this partly by documenting untested areas explicitly. It does not claim allocation-failure coverage, in-process thread safety, adversarial pointer robustness, or exhaustive malformed-parameter handling. Naming those gaps is not a substitute for testing them, but it prevents the results from being presented as broader assurance than they provide.

19.6 Educational value

The bridge offers a manageable route into a difficult interface. A student can see the exported entry point, understand the two dispatch levels, trace ownership, and observe the difference between a provider being loaded and an algorithm being fetched. The implementation is small enough to include in full, which allows the paper to make precise claims about actual code rather than abstract fragments.

The broader lesson is that cryptographic software engineering includes many obligations outside the primitive itself. Configuration, binary interfaces, typed parameters, resource lifetimes, and evidence quality can determine whether an application uses cryptography as intended. The study makes these obligations concrete without presenting a new primitive as an undergraduate exercise.

19.7 What the extension adds

The extension adds measured evidence for output-capacity guards, guard recovery, parameter-type rejection, unsupported-operation behavior, reinitialization, activation values, additional message boundaries, and concurrent independent processes. These are

substantive additions to the original eight-test prototype. It also adds design frameworks for deployment, performance evaluation, and policy review.

The latter frameworks remain proposals. They make the paper more useful as a basis for future work, but they are not included in the list of completed experiments. Maintaining that separation is essential to the credibility of a longer paper: additional pages should add explanation and inspectable evidence, not unsupported claims.

20. Final Conclusions

The expanded investigation shows how a small OpenSSL provider can be understood as a complete engineering artifact rather than as a single cryptographic function. The provider must be loaded through the intended path, advertise the right operation, satisfy selection constraints, implement the callback contract, manage resources, and return observable results. The teaching bridge provides a compact example in which each of these responsibilities can be inspected.

The experimental contribution consists of eight baseline checks, 37 extended integration cases, and a direct callback program that completed 3,362 assertions. All passed on the recorded OpenSSL 3.5.5 host. The extension materially strengthens the evidence for boundary handling and configuration behavior while leaving the provider implementation unchanged. It does not establish a novel hash implementation, a validated cryptographic module, production readiness, or universal binary compatibility.

The research questions can now be answered at two levels. Architecturally, configuration and loading establish availability, while fetching and properties establish selection. At the implementation level, dispatch tables, typed metadata, operation contexts, and orderly cleanup form the required integration structure. At the evidential level, different observations support different claims: a listing establishes availability, a provider identity check helps establish selection, and a reference comparison helps establish tested execution behavior.

The most important caution concerns policy boundaries. The bridge's private context makes backend selection explicit and avoids recursion, but it does not automatically inherit application policy. A provider name or property advertisement cannot replace an assurance argument about the backend and deployment. This insight applies beyond the teaching example and should guide any future hardware, key-management, or signature extension.

Future work should prioritize independently designed tests, different-suffix context-duplication cases, allocator-failure injection, memory-safety instrumentation, and in-process concurrency. A separate performance study should follow the proposed measurement method and publish its workload and uncertainty. A production project would additionally require packaging, integrity controls, version support, and operational review.

The result is a reproducible undergraduate study with a clear scope: it explains and tests provider integration while keeping cryptographic and operational claims proportional to the evidence. Its source and reference collection are intended to support further investigation, not to hide assumptions behind a successful demonstration.

References

OpenSSL manual snapshots are pinned to the openssl-3.5.5 source tag. All references were retrieved on 12 September 2026. Original downloads and readable conversions are included in the reference directory.

1. OpenSSL Project. *provider*. openssl-3.5.5. [Online source](#). Local file: 01-provider.pod.
2. OpenSSL Project. *provider-base*. openssl-3.5.5. [Online source](#). Local file: 02-provider-base.pod.
3. OpenSSL Project. *provider-digest*. openssl-3.5.5. [Online source](#). Local file: 03-provider-digest.pod.
4. OpenSSL Project. *config*. openssl-3.5.5. [Online source](#). Local file: 04-config.pod.
5. OpenSSL Project. *OSSL_PROVIDER*. openssl-3.5.5. [Online source](#). Local file: 05-OSSL_PROVIDER.pod.
6. OpenSSL Project. *EVP_DigestInit / EVP_MD_fetch*. openssl-3.5.5. [Online source](#). Local file: 06-EVP_DigestInit.pod.
7. OpenSSL Project. *property*. openssl-3.5.5. [Online source](#). Local file: 07-property.pod.
8. OpenSSL Project. *OSSL_LIB_CTX*. openssl-3.5.5. [Online source](#). Local file: 08-OSSL_LIB_CTX.pod.
9. OpenSSL Project. *openssl-core_dispatch.h*. openssl-3.5.5. [Online source](#). Local file: 09-openssl-core_dispatch.h.pod.
10. OpenSSL Project. *OSSL_PARAM*. openssl-3.5.5. [Online source](#). Local file: 10-OSSL_PARAM.pod.
11. OpenSSL Project. *fips_module*. openssl-3.5.5. [Online source](#). Local file: 11-fips_module.pod.
12. OpenSSL Project. *ossl-guide-migration*. openssl-3.5.5. [Online source](#). Local file: 12-ossl-guide-migration.pod.
13. NIST. *Secure Hash Standard (FIPS PUB 180-4)*. August 2015. [Online source](#). Local file: 13-NIST-FIPS-180-4.pdf.
14. Max Ammann, Fredrik Dahlgren, Spencer Michaels, and Jim Miller, Trail of Bits. *OpenSSL Security Assessment*. 18 April 2024. [Online source](#). Local file: 14-OpenSSL-security-audit-2024.pdf.

Appendix A. Complete Provider Source

The following source is the unchanged experimental provider used in both the baseline and expanded tests. Its backend is default-provider SHA-256 in a private context.

A.1 Provider implementation

Complete file: `implementation/edu_provider.c`. Line numbers are provided for review and are not part of the source file.

```

1  /* Educational provider bridge. Not an independent SHA-256 or FIPS
module. */
2  #include <openssl/core.h>
3  #include <openssl/core_dispatch.h>
4  #include <openssl/core_names.h>
5  #include <openssl/params.h>
6  #include <openssl/evp.h>
7  #include <openssl/provider.h>
8  #include <openssl/crypto.h>
9
10 typedef struct { OSSL_LIB_CTX *libctx; OSSL_PROVIDER *backend; EVP_MD
*sha; } PROVIDER;
11 typedef struct { PROVIDER *provider; EVP_MD_CTX *inner; } DIGEST;
12
13 static void teardown(void *v) {
14     PROVIDER *p = v;
15     if (p == NULL) return;
16     EVP_MD_free(p->sha);
17     OSSL_PROVIDER_unload(p->backend);
18     OSSL_LIB_CTX_free(p->libctx);
19     OPENSSL_free(p);
20 }
21 static void *newctx(void *v) {
22     DIGEST *d = OPENSSL_zalloc(sizeof(*d));
23     if (d == NULL) return NULL;
24     d->provider = v;
25     d->inner = EVP_MD_CTX_new();
26     if (d->inner == NULL) { OPENSSL_free(d); return NULL; }
27     return d;
28 }
29 static void freectx(void *v) {
30     DIGEST *d = v;
31     if (d != NULL) { EVP_MD_CTX_free(d->inner); OPENSSL_free(d); }

```

```

32 }
33 static int init(void *v, const OSSL_PARAM params[]) {
34     DIGEST *d = v;
35     (void)params; /* No optional operation parameters are advertised. */
36     return EVP_DigestInit_ex2(d->inner, d->provider->sha, NULL);
37 }
38 static int update(void *v, const unsigned char *in, size_t n) {
39     return EVP_DigestUpdate(((DIGEST *)v)->inner, in, n);
40 }
41 static int final(void *v, unsigned char *out, size_t *outl, size_t
capacity) {
42     unsigned int n = 0;
43     if (outl == NULL) return 0;
44     *outl = 0;
45     if (out == NULL || capacity < 32) return 0;
46     if (!EVP_DigestFinal_ex(((DIGEST *)v)->inner, out, &n)) return 0;
47     *outl = n;
48     return 1;
49 }
50 static void *dupctx(void *v) {
51     DIGEST *old = v, *copy = newctx(old->provider);
52     if (copy != NULL && !EVP_MD_CTX_copy_ex(copy->inner, old->inner)) {
53         freectx(copy); return NULL;
54     }
55     return copy;
56 }
57 static const OSSL_PARAM *digest_gettable(void *provctx) {
58     static const OSSL_PARAM params[] = {
59         OSSL_PARAM_size_t(OSSL_DIGEST_PARAM_SIZE, NULL),
60         OSSL_PARAM_size_t(OSSL_DIGEST_PARAM_BLOCK_SIZE, NULL),
61         OSSL_PARAM_int(OSSL_DIGEST_PARAM_XOF, NULL),
62         OSSL_PARAM_END
63     };
64     (void)provctx; return params;
65 }
66 static int digest_params(OSSL_PARAM params[]) {
67     OSSL_PARAM *p;
68     if ((p=OSSL_PARAM_locate(params,OSSL_DIGEST_PARAM_SIZE)) != NULL &&
69         !OSSL_PARAM_set_size_t(p,32)) return 0;
70     if ((p=OSSL_PARAM_locate(params,OSSL_DIGEST_PARAM_BLOCK_SIZE)) !=
NULL &&
71         !OSSL_PARAM_set_size_t(p,64)) return 0;
72     if ((p=OSSL_PARAM_locate(params,OSSL_DIGEST_PARAM_XOF)) != NULL &&
73         !OSSL_PARAM_set_int(p,0)) return 0;
74     return 1;
75 }
76 static const OSSL_DISPATCH digest_dispatch[] = {
77     { OSSL_FUNC_DIGEST_NEWCTX, (void (*)(void))newctx },
78     { OSSL_FUNC_DIGEST_FREECTX, (void (*)(void))freectx },
79     { OSSL_FUNC_DIGEST_DUPCTX, (void (*)(void))dupctx },
80     { OSSL_FUNC_DIGEST_INIT, (void (*)(void))init },
81     { OSSL_FUNC_DIGEST_UPDATE, (void (*)(void))update },

```

```

82     { OSSL_FUNC_DIGEST_FINAL, (void (*)(void))final },
83     { OSSL_FUNC_DIGEST_GETTABLE_PARAMS, (void (*)(void))digest_gettable },
84     { OSSL_FUNC_DIGEST_GET_PARAMS, (void (*)(void))digest_params },
85     { 0, NULL }
86 };
87 static const OSSL_ALGORITHM algorithms[] = {
88     { "EDU-SHA256", "provider=edu", digest_dispatch, "Educational
SHA-256 bridge" },
89     { NULL, NULL, NULL, NULL }
90 };
91 static const OSSL_ALGORITHM *query(void *p, int operation, int
*no_cache) {
92     (void)p; *no_cache = 0;
93     return operation == OSSL_OP_DIGEST ? algorithms : NULL;
94 }
95 static const OSSL_PARAM *provider_gettable(void *p) {
96     static const OSSL_PARAM params[] = {
97         OSSL_PARAM_utf8_ptr(OSSL_PROV_PARAM_NAME, NULL, 0),
98         OSSL_PARAM_utf8_ptr(OSSL_PROV_PARAM_VERSION, NULL, 0),
99         OSSL_PARAM_int(OSSL_PROV_PARAM_STATUS, NULL), OSSL_PARAM_END
100     };
101     (void)p; return params;
102 }
103 static int provider_params(void *v, OSSL_PARAM params[]) {
104     OSSL_PARAM *p; (void)v;
105     if ((p=OSSL_PARAM_locate(params,OSSL_PROV_PARAM_NAME)) != NULL &&
106         !OSSL_PARAM_set_utf8_ptr(p,"Educational provider bridge"))
return 0;
107     if ((p=OSSL_PARAM_locate(params,OSSL_PROV_PARAM_VERSION)) != NULL &&
108         !OSSL_PARAM_set_utf8_ptr(p,"1.0")) return 0;
109     if ((p=OSSL_PARAM_locate(params,OSSL_PROV_PARAM_STATUS)) != NULL &&
110         !OSSL_PARAM_set_int(p,1)) return 0;
111     return 1;
112 }
113 static const OSSL_DISPATCH provider_dispatch[] = {
114     { OSSL_FUNC_PROVIDER_TEARDOWN, (void (*)(void))teardown },
115     { OSSL_FUNC_PROVIDER_QUERY_OPERATION, (void (*)(void))query },
116     { OSSL_FUNC_PROVIDER_GETTABLE_PARAMS, (void (*)(void))provider_gettable },
117     { OSSL_FUNC_PROVIDER_GET_PARAMS, (void (*)(void))provider_params },
118     { 0, NULL }
119 };
120 int OSSL_provider_init(const OSSL_CORE_HANDLE *handle, const
OSSL_DISPATCH *in,
121                       const OSSL_DISPATCH **out, void **provctx) {
122     PROVIDER *p = OPENSSL_zalloc(sizeof(*p));
123     (void)handle; (void)in;
124     if (p == NULL) return 0;
125     p->libctx = OSSL_LIB_CTX_new();
126     if (p->libctx != NULL) p->backend = OSSL_PROVIDER_load(p-
>libctx,"default");

```

```
127     if (p->backend != NULL) p->sha = EVP_MD_fetch(p-  
>libctx,"SHA256","provider=default");  
128     if (p->sha == NULL) { teardown(p); return 0; }  
129     *provctx = p; *out = provider_dispatch; return 1;  
130 }
```

Appendix B. Client and Build Scripts

B.1 Programmatic loading and duplication client

Complete file: `implementation/client.c`. Line numbers are provided for review and are not part of the source file.

```

1  #include <openssl/evp.h>
2  #include <openssl/provider.h>
3  #include <openssl/err.h>
4  #include <stdio.h>
5  #include <string.h>
6  int main(int argc, char **argv) {
7      int ok = 0; unsigned char a[EVP_MAX_MD_SIZE], b[EVP_MAX_MD_SIZE];
8      unsigned int na=0, nb=0;
9      OSSL_LIB_CTX *libctx = OSSL_LIB_CTX_new();
10     OSSL_PROVIDER *provider = NULL; EVP_MD *md = NULL;
11     EVP_MD_CTX *x = EVP_MD_CTX_new(), *y = EVP_MD_CTX_new();
12     if (argc != 2 || !libctx || !x || !y) goto end;
13     if (!OSSL_PROVIDER_set_default_search_path(libctx,argv[1])) goto end;
14     provider = OSSL_PROVIDER_load(libctx,"edu");
15     if (!provider) goto end;
16     md = EVP_MD_fetch(libctx,"EDU-SHA256","provider=edu");
17     if (!md ||
18     strcmp(OSSL_PROVIDER_get0_name(EVP_MD_get0_provider(md)),"edu")) goto end;
19     if (!EVP_DigestInit_ex2(x,md,NULL) || !EVP_DigestUpdate(x,"a",1) ||
20     !EVP_MD_CTX_copy_ex(y,x) || !EVP_DigestUpdate(x,"bc",2) ||
21     !EVP_DigestUpdate(y,"bc",2) || !EVP_DigestFinal_ex(x,a,&na) ||
22     !EVP_DigestFinal_ex(y,b,&nb) || na != 32 || nb != na ||
23     memcmp(a,b,na)) goto end;
24     printf("provider=edu; streaming and context duplication agree\n");
25     for(unsigned int i=0;i<na;i++) printf("%02x",a[i]);
26     puts(""); ok=1;
27 end:
28     if (!ok) ERR_print_errors_fp(stderr);
29     EVP_MD_CTX_free(x); EVP_MD_CTX_free(y); EVP_MD_free(md);
30     OSSL_PROVIDER_unload(provider); OSSL_LIB_CTX_free(libctx);
31     return ok ? 0 : 1;
32 }
```

B.2 Baseline build and configuration generation

Complete file: `implementation/build.sh`. Line numbers are provided for review and are not part of the source file.

```

1  #!/usr/bin/env bash
2  set -euo pipefail
3  cd -- "${dirname -- "${BASH_SOURCE[0]}"}"
4  mkdir -p build
5  cc -std=c11 -O2 -fPIC -Wall -Wextra -Werror -shared edu_provider.c -o
build/edu.so $(pkg-config --cflags --libs openssl)
6  cc -std=c11 -O2 -Wall -Wextra -Werror client.c -o build/client $(pkg-
config --cflags --libs openssl)
7  cat > build/openssl-edu.cnf <<CONF
8  config_diagnostics = 1
9  openssl_conf = initialization
10 [initialization]
11 providers = providers
12 [providers]
13 default = default_section
14 edu = edu_section
15 [default_section]
16 activate = 1
17 [edu_section]
18 module = $PWD/build/edu.so
19 activate = 1
20 CONF
21 python3 test.py

```

B.3 Expanded experiment entry point

Complete file: `implementation/extended-build.sh`. Line numbers are provided for review and are not part of the source file.

```

1  #!/usr/bin/env bash
2  set -euo pipefail
3  cd -- "${dirname -- "${BASH_SOURCE[0]}"}"
4  ./build.sh
5  cc -std=c11 -O2 -Wall -Wextra -Werror callback_test.c -o build/
callback_test $(pkg-config --cflags --libs openssl) -ldl
6  OPENSSL_CONF=/dev/null ./build/callback_test "$PWD/build/edu.so"
7  python3 extended_test.py

```

Appendix C. Test Harness Source

The harnesses separate application-level requests from direct callback observations. The latter technique relies on this specific provider ignoring the core arguments supplied during initialization.

C.1 Baseline integration harness

Complete file: `implementation/test.py`. Line numbers are provided for review and are not part of the source file.

```

1  import subprocess,os,hashlib,json,pathlib
2  w=pathlib.Path(__file__).resolve().parent
3  base=os.environ.copy();base['OPENSSL_CONF']=''/dev/null'
4  results=[]
5  def run(args,data=b'',env=base):return
subprocess.run(args,input=data,stdout=subprocess.PIPE,stderr=subprocess.PIPE,e
nv=env)
6  args=['openssl','dgst','-provider-path',str(w/'build'),'-'
provider','edu','-propquery','provider=edu','-EDU-SHA256','-binary']
7  for label,data in [('',b''),('abc',b'abc'),
('binary',bytes(range(256))),('large',b'a'*1000000)]:
8  r=run(args,data);assert r.returncode==0 and
r.stdout==hashlib.sha256(data).digest(),r.stderr
9
results.append(dict(test=label,result='PASS',bytes=len(data),sha256=r.stdout.h
ex()))
10 r=run([str(w/'build/client'),str(w/'build')]);assert r.returncode==0 and
hashlib.sha256(b'abc').hexdigest().encode() in r.stdout
11 results.append(dict(test='streaming and duplicate context; provider
identity',result='PASS',output=r.stdout.decode()))
12 r=run([a if a!='provider=edu' else 'provider=missing' for a in
args],b'abc');assert r.returncode!=0
13 results.append(dict(test='unsatisfied mandatory property
fails',result='PASS'))
14 env=base.copy();env['OPENSSL_CONF']=str(w/'build/openssl-edu.cnf')
15 r=run(['openssl','dgst','-EDU-SHA256','-binary'],b'abc',env);assert
r.returncode==0 and r.stdout==hashlib.sha256(b'abc').digest()
16 results.append(dict(test='configuration loads edu
provider',result='PASS'))
17 r=run(['openssl','dgst','-sha256','-binary'],b'abc',env);assert
r.returncode==0 and r.stdout==hashlib.sha256(b'abc').digest()
18 results.append(dict(test='explicit default provider
coexists',result='PASS'))
19 r=run(['openssl','list','-providers','-verbose'],env=env);assert
r.returncode==0

```

```

20 (w.parent/'results/providers.txt').write_bytes(r.stdout)
21 (w.parent/'results/tests.json').write_text(json.dumps(results,indent=2))
22 (w.parent/'results/
environment.txt').write_bytes(run(['openssl','version','-a']).stdout)
23 print(json.dumps(results,indent=2))

```

C.2 Direct callback harness

Complete file: `implementation/callback_test.c`. Line numbers are provided for review and are not part of the source file.

```

1  /* White-box ABI checks for this specific educational provider. */
2  #include <openssl/core.h>
3  #include <openssl/core_dispatch.h>
4  #include <openssl/core_names.h>
5  #include <openssl/params.h>
6  #include <openssl/evp.h>
7  #include <dlfcn.h>
8  #include <stdio.h>
9  #include <string.h>
10 #include <stdlib.h>
11 static unsigned checks;
12 #define CHECK(x) do { if (!(x)) { fprintf(stderr,"FAIL line %d:
%s\n",__LINE__,#x); exit(1); } checks++; } while(0)
13 static const OSSL_DISPATCH *entry(const OSSL_DISPATCH *d, int id) {
14     while (d->function_id && d->function_id != id) d++;
15     CHECK(d->function_id == id); return d;
16 }
17 int main(int argc, char **argv) {
18     CHECK(argc == 2);
19     void *module = dlopen(argv[1], RTLD_NOW | RTLD_LOCAL);
20     CHECK(module != NULL);
21     OSSL_provider_init_fn *start = (OSSL_provider_init_fn
*)dlsym(module,"OSSL_provider_init");
22     CHECK(start != NULL);
23     const OSSL_DISPATCH *dispatch = NULL;
24     void *provider = NULL;
25     /* Safe here only because the prototype ignores core handle/
callbacks. */
26     CHECK(start(NULL,NULL,&dispatch,&provider));
27     OSSL_FUNC_provider_query_operation_fn *query =
OSSL_FUNC_provider_query_operation(
28         entry(dispatch, OSSL_FUNC_PROVIDER_QUERY_OPERATION));
29     OSSL_FUNC_provider_teardown_fn *teardown =
OSSL_FUNC_provider_teardown(
30         entry(dispatch, OSSL_FUNC_PROVIDER_TEARDOWN));
31     int no_cache = -1;
32     CHECK(query(provider,OSSL_OP_CIPHER,&no_cache) == NULL);
33     const OSSL_ALGORITHM *alg = query(provider,OSSL_OP_DIGEST,&no_cache);

```

```

34     CHECK(alg && no_cache == 0 && strcmp(alg->algorithm_names,"EDU-
SHA256")==0);
35     CHECK(strcmp(alg->property_definition,"provider=edu")==0);
36     const OSSL_DISPATCH *d = alg->implementation;
37     OSSL_FUNC_digest_newctx_fn *create =
OSSL_FUNC_digest_newctx(entry(d,OSSL_FUNC_DIGEST_NEWCTX));
38     OSSL_FUNC_digest_freectx_fn *release =
OSSL_FUNC_digest_freectx(entry(d,OSSL_FUNC_DIGEST_FREECTX));
39     OSSL_FUNC_digest_init_fn *init =
OSSL_FUNC_digest_init(entry(d,OSSL_FUNC_DIGEST_INIT));
40     OSSL_FUNC_digest_update_fn *update =
OSSL_FUNC_digest_update(entry(d,OSSL_FUNC_DIGEST_UPDATE));
41     OSSL_FUNC_digest_final_fn *final =
OSSL_FUNC_digest_final(entry(d,OSSL_FUNC_DIGEST_FINAL));
42     OSSL_FUNC_digest_get_params_fn *params =
OSSL_FUNC_digest_get_params(entry(d,OSSL_FUNC_DIGEST_GET_PARAMS));
43     size_t size=0, block=0; int xof=-1;
44     OSSL_PARAM
requested[]={OSSL_PARAM_size_t(OSSL_DIGEST_PARAM_SIZE,&size),
45             OSSL_PARAM_size_t(OSSL_DIGEST_PARAM_BLOCK_SIZE,&block),
46             OSSL_PARAM_int(OSSL_DIGEST_PARAM_XOF,&xof),OSSL_PARAM_END};
47     CHECK(params(requested) && size==32 && block==64 && xof==0);
48     char wrong_type[8]="wrong";
49     OSSL_PARAM
wrong[]={OSSL_PARAM_utf8_string(OSSL_DIGEST_PARAM_SIZE,wrong_type,sizeof
wrong_type),OSSL_PARAM_END};
50     CHECK(params(wrong)==0);
51     int untouched=99;
52     OSSL_PARAM
unknown[]={OSSL_PARAM_int("edu.unknown",&untouched),OSSL_PARAM_END};
53     CHECK(params(unknown)==1 && untouched==99);
54     for(size_t capacity=0;capacity<=33;capacity++) {
55         void *ctx=create(provider); CHECK(ctx!=NULL && init(ctx,NULL));
56         CHECK(update(ctx,(const unsigned char *)"abc",3));
57         unsigned char out[64]; memset(out,0xa5,sizeof out); size_t
written=777;
58         int result=final(ctx,out,&written,capacity);
59         if(capacity<32) {
60             CHECK(result==0 && written==0);
61             for(size_t k=0;k<sizeof out;k++) CHECK(out[k]==0xa5);
62             /* A failed size guard must not consume the state. */
63             CHECK(final(ctx,out,&written,sizeof out)==1 && written==32);
64         } else CHECK(result==1 && written==32);
65         for(size_t k=32;k<sizeof out;k++) CHECK(out[k]==0xa5);
66         unsigned char reference[32]; unsigned int n=0;
67         CHECK(EVP_Digest("abc",3,reference,&n,EVP_sha256(),NULL) &&
n==32);
68         CHECK(memcmp(out,reference,32)==0);
69         release(ctx);
70     }
71     void *ctx=create(provider); CHECK(ctx && init(ctx,NULL));
72     unsigned char out[32]; size_t n=3;

```

```

73     CHECK(final(ctx,NULL,&n,32)==0 && n==0);
74     CHECK(final(ctx,out,NULL,32)==0);
75     CHECK(init(ctx,NULL) && update(ctx,(const unsigned char *)"abc",3));
76     CHECK(final(ctx,out,&n,32)==1 && n==32);
77     release(ctx); teardown(provider); CHECK(dlclose(module)==0);
78     printf("PASS: %u assertions; capacities 0..33; guard recovery;
metadata types; unsupported operation; reinitialization\n",checks);
79 }

```

C.3 Extended integration harness

Complete file: `implementation/extended_test.py`. Line numbers are provided for review and are not part of the source file.

```

1  """Deterministic integration cases; no timing or security certification
claims."""
2  from pathlib import Path
3  import subprocess,os,hashlib,json,concurrent.futures
4  w=Path(__file__).resolve().parent
5  base=os.environ.copy();base['OPENSSL_CONF']=' /dev/null'
6  command=['openssl','dgst','-provider-path',str(w/'build'),' -
provider','edu','-propquery','provider=edu','-EDU-SHA256','-binary']
7  records=[]
8  def run(args,data=b'',env=base):
9      return
subprocess.run(args,input=data,stdout=subprocess.PIPE,stderr=subprocess.PIPE,e
nv=env)
10 def record(name,ok,detail):
11     assert ok,(name,detail)
12     records.append({'name':name,'result':'PASS','detail':detail})
13     # SHA-256 block/padding boundaries and application-level sizes.
14
lengths=[0,1,2,3,31,32,33,55,56,57,63,64,65,119,120,121,127,128,129,255,256,25
7,4095,4096,4097,65535,65536,65537]
15 for n in lengths:
16     data=bytes((i*17+31)%256 for i in range(n));r=run(command,data)
17     expected=hashlib.sha256(data).digest()
18     record(f'length-{n}',r.returncode==0 and r.stdout==expected,
{'bytes':n,'sha256':expected.hex()})
19 for prop,success in [ ('provider=edu',True), ('provider=missing',False),
('fips=yes',False)]:
20     c=command.copy();c[c.index('-propquery')+1]=prop;r=run(c,b'abc')
21     record(f'property-{prop}',(r.returncode==0)==success,
{'expected_success':success,'exit_code':r.returncode})
22 for flag,success in [ ('1',True), ('true',True), ('0',False),
('false',False)]:
23     original=(w/'build/openssl-edu.cnf').read_text()
24     # Replace only the last activation setting: the edu section.
25     prefix,sep,suffix=original.rpartition('activate = 1')

```

```

26  cfg=w/'build'/f'activation-{flag}.cnf';cfg.write_text(prefix+'activate
= '+flag+suffix)
27  env=base.copy();env['OPENSSL_CONF']=str(cfg)
28  r=run(['openssl','dgst','-EDU-SHA256','-binary'],b'abc',env)
29  record(f'activation-{flag}',(r.returncode==0)==success,
{'expected_success':success,'exit_code':r.returncode})
30  bad=w/'build/missing-module.cnf';bad.write_text((w/'build/openssl-
edu.cnf').read_text().replace('/edu.so','/does-not-exist.so'))
31  env=base.copy();env['OPENSSL_CONF']=str(bad)
32  r=run(['openssl','dgst','-EDU-SHA256','-binary'],b'abc',env)
33  record('missing-module',r.returncode!=0,{'exit_code':r.returncode})
34  def worker(i):
35      data=(f'independent-process-{i}'.encode())*100
36      r=run(command,data)
37      return r.returncode==0 and r.stdout==hashlib.sha256(data).digest()
38  with concurrent.futures.ThreadPoolExecutor(max_workers=4) as pool:
39      answers=list(pool.map(worker,range(16)))
40      record('parallel-processes',all(answers),{'processes':16,'workers':
4,'shared_provider_context':False})
41  (w.parent/'results/extended-
tests.json').write_text(json.dumps(records,indent=2))
42  print(f'PASS: {len(records)} extended integration cases')

```

Appendix D. Experimental Records

D.1 Recorded OpenSSL environment

```

OpenSSL 3.5.5 27 Jan 2026 (Library: OpenSSL 3.5.5 27 Jan 2026)
built on: Wed Jul 15 00:00:00 2026 UTC
platform: linux-x86_64
options: bn(64,64)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -O3 -O2 -flto=auto -
ffat-lto-objects -fexceptions -g -grecord-gcc-switches -pipe -Wall -
Werror=format-security -Wp,-D_FORTIFY_SOURCE=2 -Wp,-D_GLIBCXX_ASSERTIONS -
specs=/usr/lib/rpm/redhat/redhat-hardened-cc1 -fstack-protector-strong -
specs=/usr/lib/rpm/redhat/redhat-annobin-cc1 -m64 -march=x86-64-v2 -
mtune=generic -fasynchronous-unwind-tables -fstack-clash-protection -fcf-
protection -Wa,--noexecstack -Wa,--generate-missing-build-notes=yes -specs=/
usr/lib/rpm/redhat/redhat-hardened-ld -specs=/usr/lib/rpm/redhat/redhat-
annobin-cc1 -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -
DOPENSSL_BUILDING_OPENSSL -DZLIB -DNDEBUG -D_GNU_SOURCE -DPURIFY -
DDEV_RANDOM="/dev/urandom\" -DOPENSSL_PEDANTIC_ZEROIZATION -
DREDHAT_FIPS_VENDOR="\"Red Hat Enterprise Linux OpenSSL FIPS Provider\" -
DREDHAT_FIPS_VERSION="\"3.5.5-5fb82caa2911ea82\" -DSYSTEM_CIPHERS_FILE="/
etc/crypto-policies/back-ends/opensslcnf.config"
OPENSSLDIR: "/etc/pki/tls"
ENGINESDIR: "/usr/lib64/engines-3"
MODULESDIR: "/usr/lib64/openssl-modules"
Seeding source: os-specific
CPUINFO: OPENSSL_ia32cap=0x7ef8320b078bffff:
0x0040069c219c97a9:0x0000000000000010:0x0000000000000000:0x0000000000000000

```

D.2 Provider listing

```

Providers:
  default
    name: OpenSSL Default Provider
    version: 3.5.5
    status: active
    build info: 3.5.5
    gettable provider parameters:
      name: pointer to a UTF8 encoded string (arbitrary size)
      version: pointer to a UTF8 encoded string (arbitrary size)
      buildinfo: pointer to a UTF8 encoded string (arbitrary size)
      status: integer (arbitrary size)
  edu
    name: Educational provider bridge
    version: 1.0

```

```

status: active
gettable provider parameters:
  name: pointer to a UTF8 encoded string (arbitrary size)
  version: pointer to a UTF8 encoded string (arbitrary size)
  status: integer (max 4 bytes large)

```

D.3 Baseline checks

Test	Observed
empty	PASS
abc	PASS
binary	PASS
large	PASS
streaming and duplicate context; provider identity	PASS
unsatisfied mandatory property fails	PASS
configuration loads edu provider	PASS
explicit default provider coexists	PASS

D.4 Extended integration cases

Digest values are hexadecimal encodings of 32-byte outputs. The length cases use the deterministic pattern defined in `extended_test.py`. This is an execution record, not a performance table.

length-0 — PASS

Input bytes: 0. Observed digest matched the reference:

```
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
```

length-1 — PASS

Input bytes: 1. Observed digest matched the reference:

```
ffe679bb831c95b67dc17819c63c5090d221aac6f4c7bf530f594ab43d21fa1e
```

length-2 — PASS

Input bytes: 2. Observed digest matched the reference:

```
110bc268fbce35b0d4321f1840b7d3e667f4e4409d287b70d7ed047f04266915
```

length-3 — PASS

Input bytes: 3. Observed digest matched the reference:

```
281b1449df500b9d64253ed8fd1c13a291b5c4afe850094e12ac1acfdb7422e7
```

length-31 — PASS

Input bytes: 31. Observed digest matched the reference:

```
04997df863936d7e406f369c4e66a609f6729848cafc0238261fa079f51eae18
```

length-32 — PASS

Input bytes: 32. Observed digest matched the reference:

```
a402b15f7ad564cc6c78dc2c5549a1ac481b9bd598ed32706fd8870e04c9daf1
```

length-33 — PASS

Input bytes: 33. Observed digest matched the reference:

```
69f676dd611b6e6fc4558543bd8620fb4bd61ea027d26c2d583062cbd3594765
```

length-55 — PASS

Input bytes: 55. Observed digest matched the reference:

```
44cb7046a6225ff9ff5ed5bae3e3274c68d6d77b53bd512be64b0ff623047b54
```

length-56 — PASS

Input bytes: 56. Observed digest matched the reference:

```
87535826792c4242e9b8ec88665347cc48167f7ac41162e45e9bc75e364c6057
```

length-57 — PASS

Input bytes: 57. Observed digest matched the reference:

```
e29fea04b11a2a3e4ad1549321f4d0030a02a6753208e36defaf5069a838937c
```

length-63 — PASS

Input bytes: 63. Observed digest matched the reference:

```
ec158dffda7701d514de0d84ad9f189f1c1c68e524f0e9e302d13e78481a3edb
```

length-64 — PASS

Input bytes: 64. Observed digest matched the reference:

```
bb7d30cc3725cb327f99f56256cf4fe8e7d69155cbad113369e58ffe5dcddc85
```

length-65 — PASS

Input bytes: 65. Observed digest matched the reference:

```
0f30b745a6589a725d1959d54dd231353ff286cdb5e8f41c803aa7bfae0ab676
```

length-119 — PASS

Input bytes: 119. Observed digest matched the reference:

```
51d8dc11ed3ac1e3058fa13ae625c26b5efcd20eb16fb4129c3905ae5007d660
```

length-120 — PASS

Input bytes: 120. Observed digest matched the reference:

```
f5cb5a553f03ba04ea7195ab98a1b97d784d0ae9b7ea2b0ae62c564a9277a328
```

length-121 — PASS

Input bytes: 121. Observed digest matched the reference:

```
b08439081b274faa89a3724896b0075b9f4897b07ce845eeb825b83e0ab36590
```

length-127 — PASS

Input bytes: 127. Observed digest matched the reference:

```
d5de78d3ae27de67e93064e3a89fafbcb2c7f4fae7727fe1269b6b200c61e55a
```

length-128 — PASS

Input bytes: 128. Observed digest matched the reference:

```
1da9f1860f36da462b272e4f3f849871f3ba2bd9239feb68c8a5a7a47deaf4e1
```

length-129 — PASS

Input bytes: 129. Observed digest matched the reference:

```
14cdf8079a998856c2394e961dd0504f0d854399168642d8d075f731cd5805f9
```

length-255 — PASS

Input bytes: 255. Observed digest matched the reference:

```
79eec89509cbadc8cfa9a843d764b44bc3f6afc69fc6aa32e7623e09e85eb683
```

length-256 — PASS

Input bytes: 256. Observed digest matched the reference:

```
14b2d5dc35dd606a5a798167d93b638a31f50bac5f7d4ee744d94f703a3b46b1
```

length-257 — PASS

Input bytes: 257. Observed digest matched the reference:

```
4910f325052e17944a0e09fe802658bbc3d635f830835a7993144e9302fcd178
```

length-4095 — PASS

Input bytes: 4095. Observed digest matched the reference:

```
b9ae8333b1877555e104aa7e804cb1efe89035e9d3819bc5e240d06319c3be8c
```

length-4096 — PASS

Input bytes: 4096. Observed digest matched the reference:

```
daf79852461993f687e8e126356e4acb2d8c328b8ab40fa1b016e31e08bd2669
```

length-4097 — PASS

Input bytes: 4097. Observed digest matched the reference:

```
7aecdfc4d236df5b5e429b2d8a3f5cfd93cc25cca2bc16de8e5392bff015f3ca
```

length-65535 — PASS

Input bytes: 65535. Observed digest matched the reference:

```
37cc08c28efb1bdf4243ef740a6947e708495a7ae7f068a3de8cf28add1b7872
```

length-65536 — PASS

Input bytes: 65536. Observed digest matched the reference:

```
ea7fa9ff800454270faaa555764710d0491fcb9e0886dd3bfbcb60725f247682
```

length-65537 — PASS

Input bytes: 65537. Observed digest matched the reference:

```
c7759c77e124ae66339f572aadab0a4d840a6cab823c7b34e51786068a8c362d
```

property-provider=edu — PASS

```
{
  "expected_success": true,
  "exit_code": 0
}
```

property-provider=missing — PASS

```
{
  "expected_success": false,
  "exit_code": 1
}
```

property-fips=yes — PASS

```
{
  "expected_success": false,
  "exit_code": 1
}
```

activation-1 — PASS

```
{
  "expected_success": true,
  "exit_code": 0
}
```

activation-true — PASS

```
{
  "expected_success": true,
  "exit_code": 0
}
```

activation-0 — PASS

```
{  
  "expected_success": false,  
  "exit_code": 1  
}
```

activation-false — PASS

```
{  
  "expected_success": false,  
  "exit_code": 1  
}
```

missing-module — PASS

```
{  
  "exit_code": 1  
}
```

parallel-processes — PASS

```
{  
  "processes": 16,  
  "workers": 4,  
  "shared_provider_context": false  
}
```

D.5 Callback summary

```
PASS: 3362 assertions; capacities 0..33; guard recovery; metadata types;  
unsupported operation; reinitialization  
PASS: 37 extended integration cases
```

The assertion count includes bitwise sentinel comparisons and must not be interpreted as the number of independent integration scenarios.

Appendix E. Review and Reproduction Guide

E.1 Preparation

Verify the source and reference files against the supplied manifest before changing the experiment. Record the OpenSSL executable version and the development package selected by pkg-config. If several OpenSSL installations exist on the machine, identify the one used by both compilation and execution. Build in the local project directory; do not install the teaching module into a system provider directory merely to reproduce the tests.

E.2 Baseline reproduction

```
./implementation/build.sh  
cat results/tests.json  
cat results/providers.txt
```

There should be eight successful baseline checks. Confirm that the programmatic client reports provider=edu. Compare the abc digest with the known value in the paper. Confirm that the expected property failure is recorded as a passing negative test rather than being ignored.

E.3 Expanded reproduction

```
./implementation/extended-build.sh  
cat results/extended-tests.json  
tail -n 2 results/extended-build.log
```

The saved log in this edition contains the observed run. When reproducing, redirect output to a new file if the original record must be preserved. The callback test should complete its assertions, and the integration harness should report 37 cases. If a run fails, preserve the compiler output, command, environment, and error text before changing the implementation.

E.4 Questions for source review

Review area	Question
Initialization	Are outputs returned only after required provider resources exist?
Partial failure	Can every allocated resource be released on the path where a later step fails?
Advertisement	Does each algorithm table remain alive for as long as the core may use it?
Dispatch	Do function identifiers match callback signatures?
Input handling	Are byte lengths preserved across wrapper and backend calls?
Output handling	Are capacity, written length, and backend finalization ordered correctly?
Duplication	Does a new operation own independent mutable state?
Policy	Is the backend selection explicit and consistent with the intended assurance?
Cleanup	Are operation contexts released before the resources they depend on?

E.5 Suggested additional experiments

Feed different suffixes to duplicated contexts and compare against two expected values. Introduce controlled allocator failures and record whether cleanup remains correct. Build with memory instrumentation and rerun the same deterministic tests. Add an in-process multithreaded client with independent digest contexts. These are suggested extensions, not completed experiments in this edition.

For a performance project, implement both direct-default and bridge paths in the same executable. Separate provider loading, method fetching, and warm digest processing. Report message sizes, update

fragmentation, iteration counts, uncertainty, and output validation. Avoid using the large-message functional case as a performance measurement: no timing was recorded for that purpose.

E.6 Artifact interpretation

The PDF explains the reasoning; the source defines the implementation; the scripts define the procedure; the result files record observed execution. If these disagree after an edit, regenerate the affected artifacts and update the claims. An archived result should never be presented as evidence for changed code without rerunning the relevant tests.

E.7 Completed and proposed work

Completed in this edition	Still proposed or outside scope
Dynamic provider and application client.	Independent SHA-256 implementation.
Eight baseline and 37 extended integration cases.	Exhaustive state-space exploration.
Direct callback capacity and parameter checks.	Allocation-failure campaign and sanitizers.
Sixteen independent parallel processes.	In-process thread-safety stress tests.
Version-pinned reference collection.	Independent clean-machine reproduction.
Functional results on OpenSSL 3.5.5.	FIPS validation or production security approval.

This distinction is the final acceptance criterion for the paper itself: every experimental claim should correspond to an executable procedure and a recorded result, while proposals should remain visibly labeled as proposals.