

UNDERGRADUATE TECHNICAL PAPER

Configuration and Implementation of an OpenSSL 3 Provider

Architecture, a Reproducible Digest Prototype, and Experimental Evaluation



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 1.0 • 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 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 4

2. Background and Related Work 5

3. Configuring and Selecting a Provider..... 6

4. Prototype Design and Implementation 8

5. Experimental Method and Results..... 10

6. Security, Policy, and Engineering Limitations 12

7. Conclusion and Future Work 13

References 14

Appendix A. Provider source 15

Appendix B. Client and reproduction 17

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

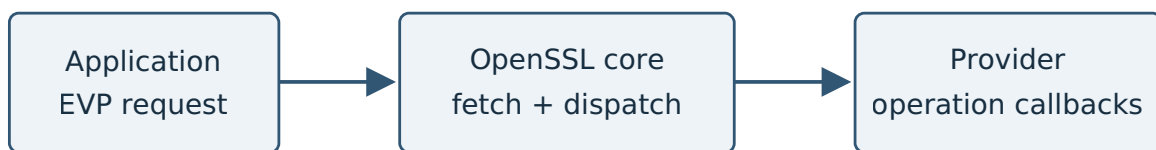
The method combines a version-pinned documentation review with design-and-build experimentation. The implementation is compiled against the installed OpenSSL 3.5.5 headers and library. Expected output is checked against Python's SHA-256 interface and a known published example. Logs, program source, and 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 `OSL_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 `OSL_DISPATCH` associates function identifiers with callback pointers; identifiers are defined by the public core-dispatch interface. An `OSL_ALGORITHM` connects an algorithm name and property definition to a dispatch table. The operation identifier, such as `OSL_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	EVP_MD_CTX_free
--------------------	------------------	-----------------

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. This guard is visible in the source, but direct undersized-buffer injection is not part of the automated tests. The distinction matters: code inspection and observed execution are different forms of 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. Conclusion and Future Work

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.

References

OpenSSL manual snapshots are pinned to the `openssl-3.5.5` source tag. All references were retrieved on 12 September 2026. The accompanying `references/` directory contains the original downloads, readable manual conversions, provenance, and checksums.

1. OpenSSL Project. *provider*. `openssl-3.5.5`. [Online source](#). Local copy: `01-provider.pod`.
2. OpenSSL Project. *provider-base*. `openssl-3.5.5`. [Online source](#). Local copy: `02-provider-base.pod`.
3. OpenSSL Project. *provider-digest*. `openssl-3.5.5`. [Online source](#). Local copy: `03-provider-digest.pod`.
4. OpenSSL Project. *config*. `openssl-3.5.5`. [Online source](#). Local copy: `04-config.pod`.
5. OpenSSL Project. *OSSL_PROVIDER*. `openssl-3.5.5`. [Online source](#). Local copy: `05-OSSL_PROVIDER.pod`.
6. OpenSSL Project. *EVP_DigestInit / EVP_MD_fetch*. `openssl-3.5.5`. [Online source](#). Local copy: `06-EVP_DigestInit.pod`.
7. OpenSSL Project. *property*. `openssl-3.5.5`. [Online source](#). Local copy: `07-property.pod`.
8. OpenSSL Project. *OSSL_LIB_CTX*. `openssl-3.5.5`. [Online source](#). Local copy: `08-OSSL_LIB_CTX.pod`.
9. OpenSSL Project. *openssl-core_dispatch.h*. `openssl-3.5.5`. [Online source](#). Local copy: `09-openssl-core_dispatch.h.pod`.
10. OpenSSL Project. *OSSL_PARAM*. `openssl-3.5.5`. [Online source](#). Local copy: `10-OSSL_PARAM.pod`.
11. OpenSSL Project. *fips_module*. `openssl-3.5.5`. [Online source](#). Local copy: `11-fips_module.pod`.
12. OpenSSL Project. *ossl-guide-migration*. `openssl-3.5.5`. [Online source](#). Local copy: `12-openssl-guide-migration.pod`.
13. NIST. *Secure Hash Standard (FIPS PUB 180-4)*. August 2015. [Online source](#). Local copy: `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 copy: `14-OpenSSL-security-audit-2024.pdf`.

Appendix A. Provider Source

Complete experimental module, `implementation/edu_provider.c`. The code is original to this study. The backend is the default provider in a private context; this is not an independent SHA-256 implementation.

```
/* Educational provider bridge. Not an independent SHA-256 or FIPS module. */
#include <openssl/core.h>
#include <openssl/core_dispatch.h>
#include <openssl/core_names.h>
#include <openssl/params.h>
#include <openssl/evp.h>
#include <openssl/provider.h>
#include <openssl/crypto.h>

typedef struct { OSSL_LIB_CTX *libctx; OSSL_PROVIDER *backend; EVP_MD *sha; } PROVIDER;
typedef struct { PROVIDER *provider; EVP_MD_CTX *inner; } DIGEST;

static void teardown(void *v) {
    PROVIDER *p = v;
    if (p == NULL) return;
    EVP_MD_free(p->sha);
    OSSL_PROVIDER_unload(p->backend);
    OSSL_LIB_CTX_free(p->libctx);
    OPENSSL_free(p);
}

static void *newctx(void *v) {
    DIGEST *d = OPENSSL_zalloc(sizeof(*d));
    if (d == NULL) return NULL;
    d->provider = v;
    d->inner = EVP_MD_CTX_new();
    if (d->inner == NULL) { OPENSSL_free(d); return NULL; }
    return d;
}

static void freectx(void *v) {
    DIGEST *d = v;
    if (d != NULL) { EVP_MD_CTX_free(d->inner); OPENSSL_free(d); }
}

static int init(void *v, const OSSL_PARAM params[]) {
    DIGEST *d = v;
    (void)params; /* No optional operation parameters are advertised. */
    return EVP_DigestInit_ex2(d->inner, d->provider->sha, NULL);
}

static int update(void *v, const unsigned char *in, size_t n) {
    return EVP_DigestUpdate(((DIGEST *)v)->inner, in, n);
}

static int final(void *v, unsigned char *out, size_t *outl, size_t capacity) {
    unsigned int n = 0;
    if (outl == NULL) return 0;
    *outl = 0;
    if (out == NULL || capacity < 32) return 0;
    if (!EVP_DigestFinal_ex(((DIGEST *)v)->inner, out, &n)) return 0;
    *outl = n;
    return 1;
}

static void *dupctx(void *v) {
    DIGEST *old = v, *copy = newctx(old->provider);
    if (copy != NULL && !EVP_MD_CTX_copy_ex(copy->inner, old->inner)) {
        freectx(copy); return NULL;
    }
    return copy;
}

static const OSSL_PARAM *digest_gettable(void *provctx) {
    static const OSSL_PARAM params[] = {
```

```

        OSSL_PARAM_size_t(OSSL_DIGEST_PARAM_SIZE, NULL),
        OSSL_PARAM_size_t(OSSL_DIGEST_PARAM_BLOCK_SIZE, NULL),
        OSSL_PARAM_int(OSSL_DIGEST_PARAM_XOF, NULL),
        OSSL_PARAM_END
    };
    (void)provctx; return params;
}
static int digest_params(OSSL_PARAM params[]) {
    OSSL_PARAM *p;
    if ((p=OSSL_PARAM_locate(params,OSSL_DIGEST_PARAM_SIZE)) != NULL &&
        !OSSL_PARAM_set_size_t(p,32)) return 0;
    if ((p=OSSL_PARAM_locate(params,OSSL_DIGEST_PARAM_BLOCK_SIZE)) != NULL &&
        !OSSL_PARAM_set_size_t(p,64)) return 0;
    if ((p=OSSL_PARAM_locate(params,OSSL_DIGEST_PARAM_XOF)) != NULL &&
        !OSSL_PARAM_set_int(p,0)) return 0;
    return 1;
}
static const OSSL_DISPATCH digest_dispatch[] = {
    { OSSL_FUNC_DIGEST_NEWCTX, (void (*)(void))newctx },
    { OSSL_FUNC_DIGEST_FRECTX, (void (*)(void))freectx },
    { OSSL_FUNC_DIGEST_DUPCTX, (void (*)(void))dupctx },
    { OSSL_FUNC_DIGEST_INIT, (void (*)(void))init },
    { OSSL_FUNC_DIGEST_UPDATE, (void (*)(void))update },
    { OSSL_FUNC_DIGEST_FINAL, (void (*)(void))final },
    { OSSL_FUNC_DIGEST_GETTABLE_PARAMS, (void (*)(void))digest_gettable },
    { OSSL_FUNC_DIGEST_GET_PARAMS, (void (*)(void))digest_params },
    { 0, NULL }
};
static const OSSL_ALGORITHM algorithms[] = {
    { "EDU-SHA256", "provider=edu", digest_dispatch, "Educational SHA-256 bridge" },
    { NULL, NULL, NULL, NULL }
};
static const OSSL_ALGORITHM *query(void *p, int operation, int *no_cache) {
    (void)p; *no_cache = 0;
    return operation == OSSL_OP_DIGEST ? algorithms : NULL;
}
static const OSSL_PARAM *provider_gettable(void *p) {
    static const OSSL_PARAM params[] = {
        OSSL_PARAM_utf8_ptr(OSSL_PROV_PARAM_NAME, NULL, 0),
        OSSL_PARAM_utf8_ptr(OSSL_PROV_PARAM_VERSION, NULL, 0),
        OSSL_PARAM_int(OSSL_PROV_PARAM_STATUS, NULL), OSSL_PARAM_END
    };
    (void)p; return params;
}
static int provider_params(void *v, OSSL_PARAM params[]) {
    OSSL_PARAM *p; (void)v;
    if ((p=OSSL_PARAM_locate(params,OSSL_PROV_PARAM_NAME)) != NULL &&
        !OSSL_PARAM_set_utf8_ptr(p,"Educational provider bridge")) return 0;
    if ((p=OSSL_PARAM_locate(params,OSSL_PROV_PARAM_VERSION)) != NULL &&
        !OSSL_PARAM_set_utf8_ptr(p,"1.0")) return 0;
    if ((p=OSSL_PARAM_locate(params,OSSL_PROV_PARAM_STATUS)) != NULL &&
        !OSSL_PARAM_set_int(p,1)) return 0;
    return 1;
}
static const OSSL_DISPATCH provider_dispatch[] = {
    { OSSL_FUNC_PROVIDER_TEARDOWN, (void (*)(void))teardown },
    { OSSL_FUNC_PROVIDER_QUERY_OPERATION, (void (*)(void))query },
    { OSSL_FUNC_PROVIDER_GETTABLE_PARAMS, (void (*)(void))provider_gettable },
    { OSSL_FUNC_PROVIDER_GET_PARAMS, (void (*)(void))provider_params },
    { 0, NULL }
};
int OSSL_provider_init(const OSSL_CORE_HANDLE *handle, const OSSL_DISPATCH *in,
                      const OSSL_DISPATCH **out, void **provctx) {
    PROVIDER *p = OPENSSL_zalloc(sizeof(*p));
    (void)handle; (void)in;
    if (p == NULL) return 0;
    p->libctx = OSSL_LIB_CTX_new();
    if (p->libctx != NULL) p->backend = OSSL_PROVIDER_load(p->libctx,"default");
    if (p->backend != NULL) p->sha = EVP_MD_fetch(p->libctx,"SHA256","provider=default");
    if (p->sha == NULL) { teardown(p); return 0; }
    *provctx = p; *out = provider_dispatch; return 1;
}

```

Appendix B. Client and Reproduction

B.1 Programmatic client

```
#include <openssl/evp.h>
#include <openssl/provider.h>
#include <openssl/err.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
    int ok = 0; unsigned char a[EVP_MAX_MD_SIZE], b[EVP_MAX_MD_SIZE];
    unsigned int na=0, nb=0;
    OSSL_LIB_CTX *libctx = OSSL_LIB_CTX_new();
    OSSL_PROVIDER *provider = NULL; EVP_MD *md = NULL;
    EVP_MD_CTX *x = EVP_MD_CTX_new(), *y = EVP_MD_CTX_new();
    if (argc != 2 || !libctx || !x || !y) goto end;
    if (!OSSL_PROVIDER_set_default_search_path(libctx,argv[1])) goto end;
    provider = OSSL_PROVIDER_load(libctx,"edu");
    if (!provider) goto end;
    md = EVP_MD_fetch(libctx,"EDU-SHA256","provider=edu");
    if (!md || strcmp(OSSL_PROVIDER_get0_name(EVP_MD_get0_provider(md)),"edu")) goto end;
    if (!EVP_DigestInit_ex2(x,md,NULL) || !EVP_DigestUpdate(x,"a",1) ||
        !EVP_MD_CTX_copy_ex(y,x) || !EVP_DigestUpdate(x,"bc",2) ||
        !EVP_DigestUpdate(y,"bc",2) || !EVP_DigestFinal_ex(x,a,&na) ||
        !EVP_DigestFinal_ex(y,b,&nb) || na != 32 || nb != na || memcmp(a,b,na)) goto end;
    printf("provider=edu; streaming and context duplication agree\n");
    for(unsigned int i=0;i<na;i++) printf("%02x",a[i]);
    puts(""); ok=1;
end:
    if (!ok) ERR_print_errors_fp(stderr);
    EVP_MD_CTX_free(x); EVP_MD_CTX_free(y); EVP_MD_free(md);
    OSSL_PROVIDER_unload(provider); OSSL_LIB_CTX_free(libctx);
    return ok ? 0 : 1;
}
```

B.2 Reproduction procedure

On a Linux host with OpenSSL 3 development headers, a C compiler, pkg-config, and Python 3, run the following from the paper directory:

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

The build script generates a configuration file with the current absolute module path, compiles both programs, and runs the eight checks. Rebuild after relocating the directory. It modifies only local project files. Tests explicitly control OPENSSL_CONF and do not edit the system configuration.

B.3 Artifact map

Artifact	Purpose
----------	---------

OpenSSL_Provider_Configuration_and_Implementation.pdf	Rendered paper
paper.html / paper_body.html	Rendered and editable document source
render_paper.py	Rebuild PDF with WeasyPrint 62.3
implementation/	Provider, C client, build script, Python tests
results/	Observed environment, build log, test results, provider listing
references/	14 downloaded reference works and manifest

No performance numbers, device tests, fuzzing campaigns, or compliance results are implied by these artifacts.