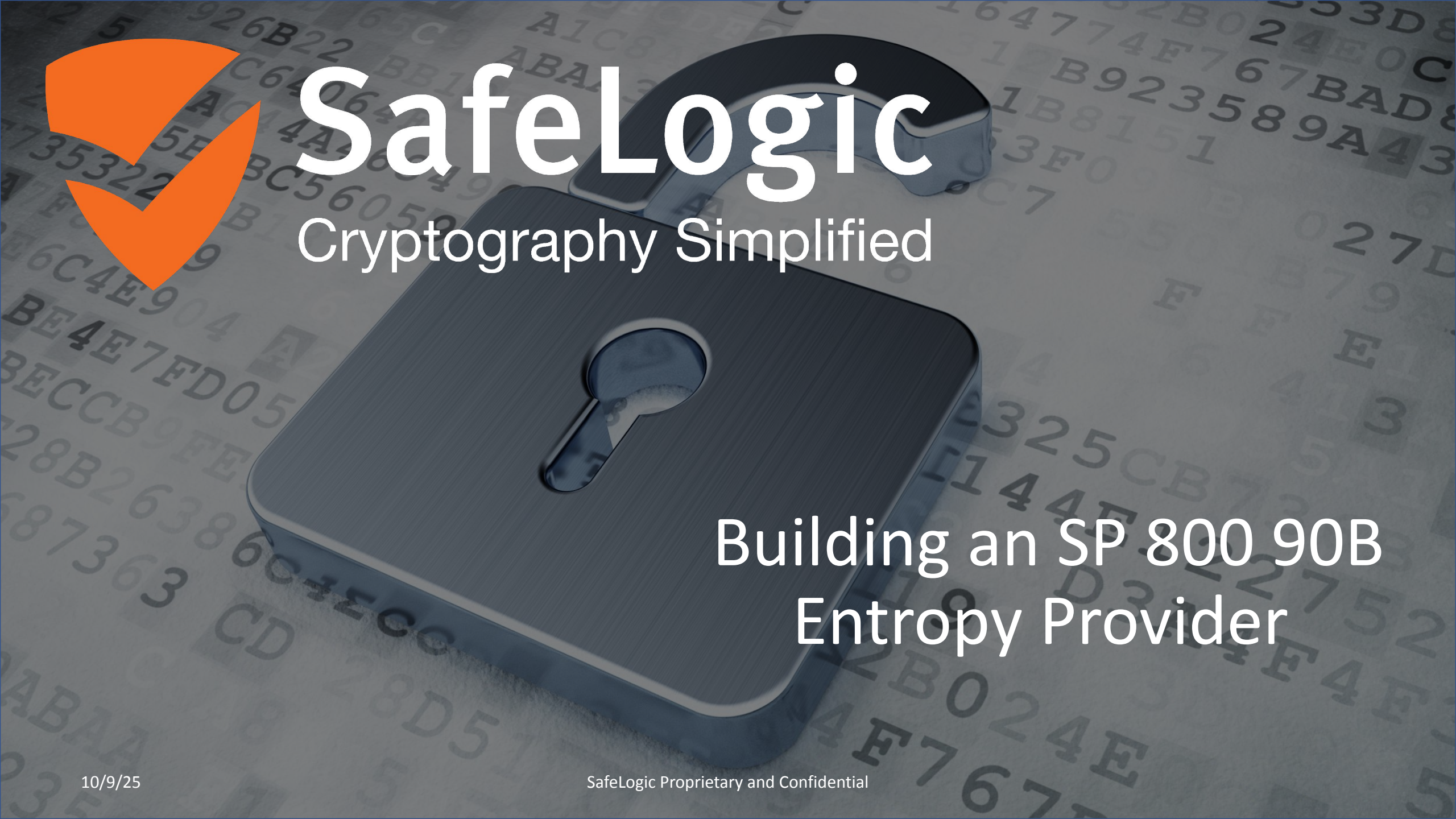


SafeLogic

Cryptography Simplified

Building an SP 800 90B Entropy Provider

SafeLogic

Adam Gorak

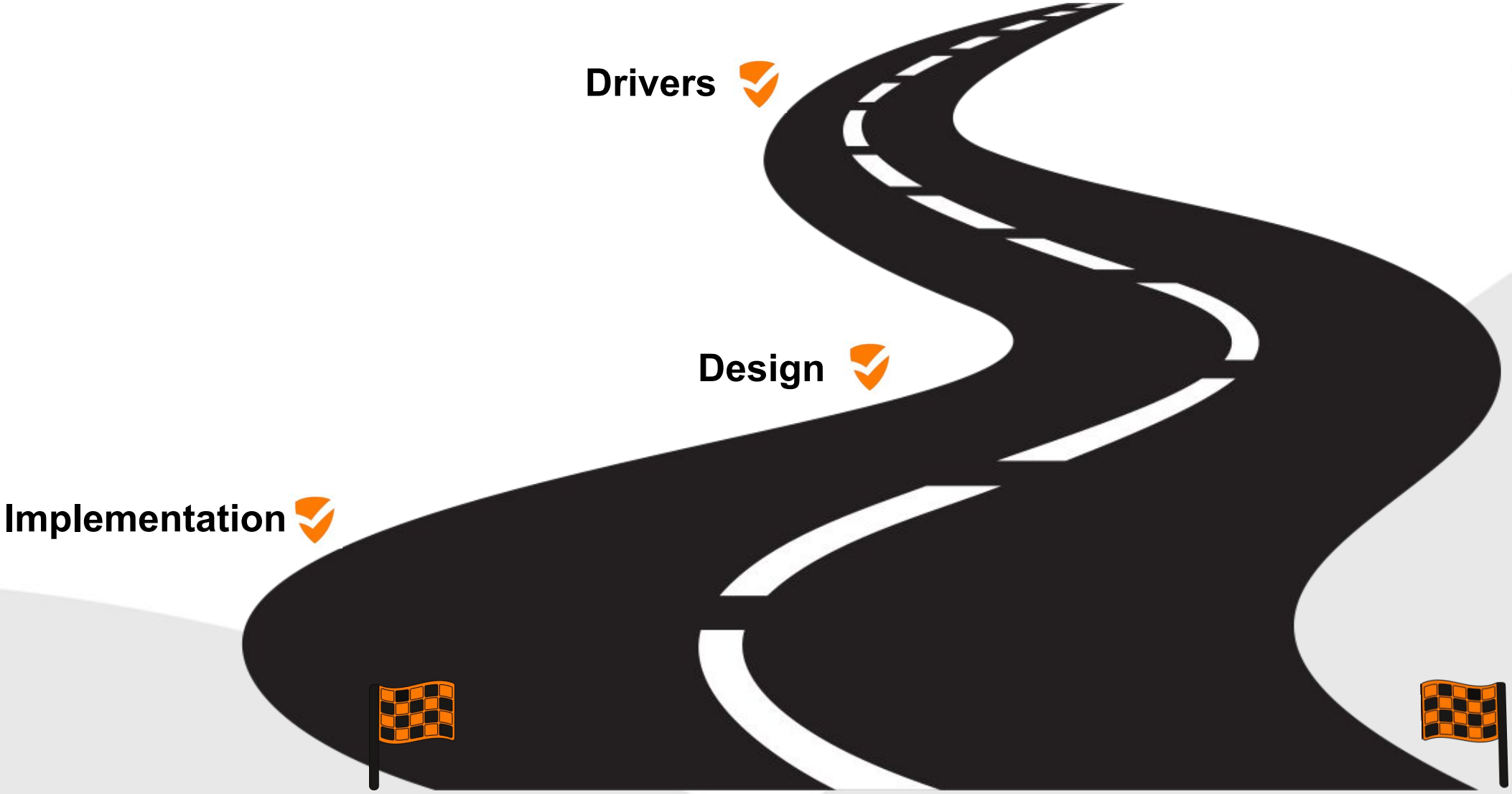
Senior Cryptographic Implementation Engineer

Jake Maynard

Director of Engineering

Todays Roadmap

SafeLogic



What are the drivers for building an entropy solution?

Patent History

Patent number: 9548862
Type: Grant
Filed: Nov 17, 2014
Date of Patent: Jan 17, 2017
Assignee: Safelogic, Inc. (Palo Alto, CA)

This is not our first time!

Managing entropy in computing devices for cryptographic key generation

Nov 17, 2014

This disclosure describes cryptographic secure implementation of a Pseudo Random Number Generator (PRNG) architecture based on existing Fortuna algorithm, but providing improvements thereupon for gathering and measuring entropy. The improvement includes a unique step of initial seeding that is not covered by Fortuna. The solution should be adapted to a variety of computing and communicating devices, including mobile devices.

Skip to: Description · Claims · References Cited · Patent History · Patent History

Description

TECHNICAL FIELD

This patent application is related to providing cryptographic solutions in general. In particular, this disclosure describes utilization of entropy for secure cryptographic key generation.

Can we get rid of this?

Certificate #5040

Details

Module Name	CryptoComply 140-3 FIPS Provider
Standard	FIPS 140-3
Status	Active
Sunset Date	8/26/2029
Overall Level	1
Caveat	No assurance of the minimum strength of generated SSPs (e.g., keys) and random strings. No assurance of minimum security of SSPs (e.g., keys, bit strings) that are externally loaded, or of SSPs established with externally loaded SSPs
Security Level Exceptions	• Physical security: N/A • Non-invasive security: N/A
Module Type	Software
Embodiment	Multi-Chip Stand Alone
Description	SafeLogic's CryptoComply 140-3 FIPS Provider is designed to provide FIPS 140-3 validated cryptographic functionality and is available for licensing.
Product URL	http://www.safelogic.com/cryptocomply/

NIAP / Common Criteria Requirements

118	Labgram #118/Valgram #137 - Entropy Source Validation Certificates	04/12/2024	NIAP Staff

RESOURCES - LABGRAMS

LABGRAM #118/VALGRAM #137 - ENTROPY SOURCE VALIDATION CERTIFICATES

Validators and CCTLs,

In accordance with NIAP Policy 5, Entropy Assessment Reports (EARs) must include a NIST Entropy Source Validation (ESV) certificate. The ESV certificate(s) must also be included in the Check-out package, along with any other NIST CAVP/CMVP certificate, as specified in Labgram #102/Valgram #122. The transition to ESV certificates as evidence for the min-entropy estimate will occur as follows:

- Effective immediately, vendors and CCTLs may submit EARs and check-out packages that refer to an ESV certificate.
- During CY24, vendors and CCTLs may submit an EAR and check-out package without an ESV certificate to allow time for manufacturers to obtain ESV certificates for their hardware noise sources as well as to accommodate any vendor-proprietary noise sources.
- As of 1 January 2025, for any product not yet in-evaluation, all EARs and check-out packages must include an ESV certificate. Assumptions of entropy associated with third-party claims will no longer be allowed.

Section 9 – Sensitive security parameter management

9.3.A Entropy Caveats

Applicable Levels:	All
Original Publishing Date:	September 21, 2020
Effective Date:	September 21, 2020
Last Modified Date:	July 26, 2024
Relevant Assertions:	AS09.08

Question/Problem

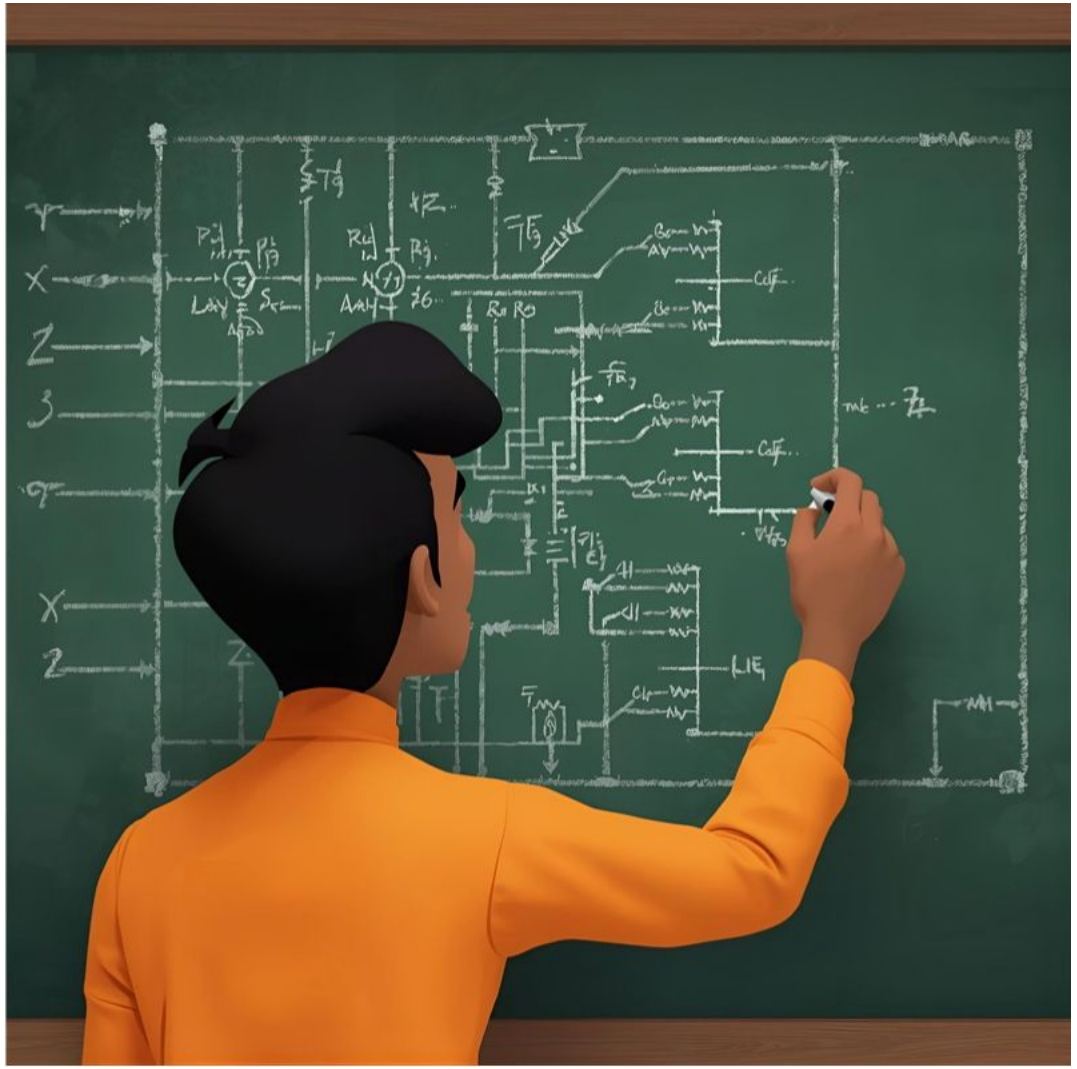
When is it necessary for the module to provide the evidence of the amount of generated entropy?

How to handle the case when the amount of generated entropy is sufficient to meet the minimum SSP strength requirement (112 bit) but not necessarily sufficient to account for a comparable strength of the generated SSPs?

certificate number(s) shall be present on the module certificate and in the Security Policy.

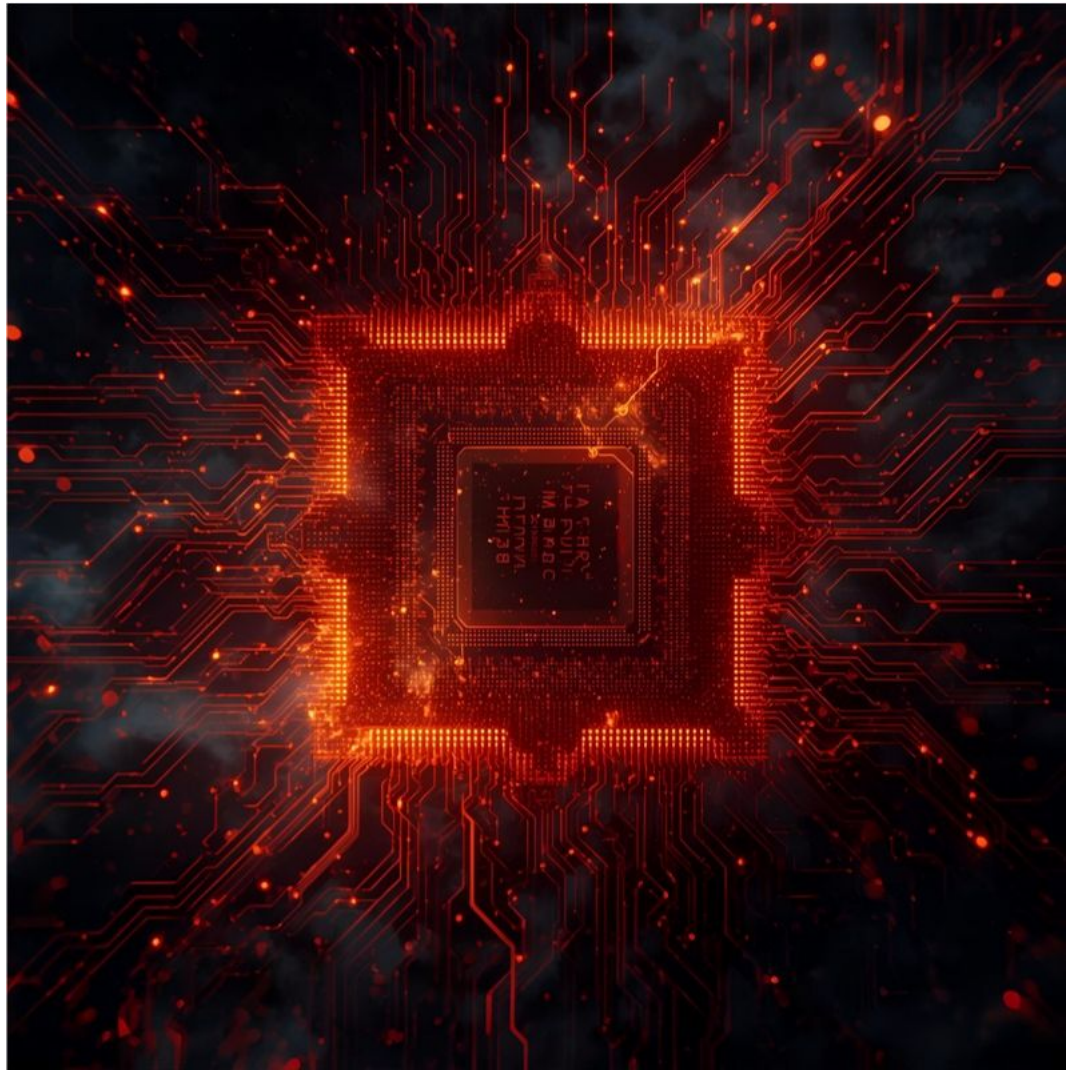
12. Until **January 1, 2026**, new module submissions to the CMVP can meet an earlier version of Resolution 2(b) of this IG, located at the following URL: https://csrc.nist.gov/CSRC/media/Projects/cryptographic-module-validation-program/documents/IG%209.3.A%20Resolution%202b%5BMarch%2026%202024%5D.pdf. This is a 'soft' transition in that modules that meet this earlier version of Resolution 2(b) will not be moved to the Historical list on the transition date.

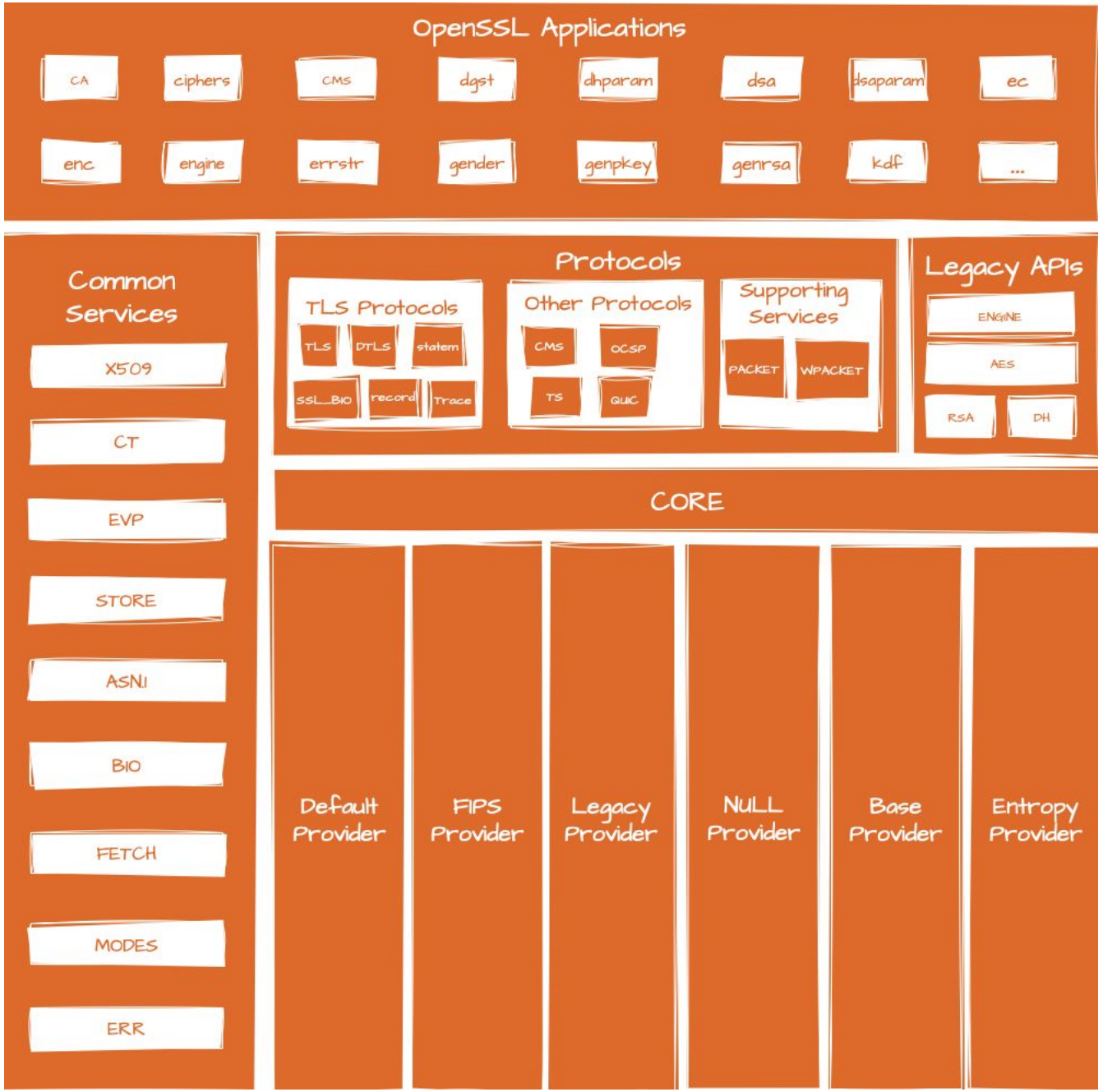
Design Thinking



Requirements

- Produces Full Entropy
- Meet customers where they are in their OpenSSL Journey
- Portable to many Operating Environments
- Meet Performance Requirements
- Easy to Integrate with our FIPS modules and yours!





Why not Build an OpenSSL 3.x Compatible Entropy Provider ?

Closer Look at Scenario 1(b) from I.G. 9.3.A?

1. The module is either generating the entropy itself or it is making a call to request the entropy from a well-defined source via the entropy source's GetEntropy() interface.

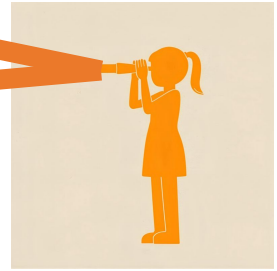
   Examples include:

   (b) A software, firmware, or hybrid module that contains an approved DRBG, that is seeded exclusively from one or more known entropy sources, located within the TOEPP (from IG 9.5.A). For instance, a software library on a Linux platform making a call to a **SP 800-90B** entropy source within the module's TOEPP for seeding its DRBG.

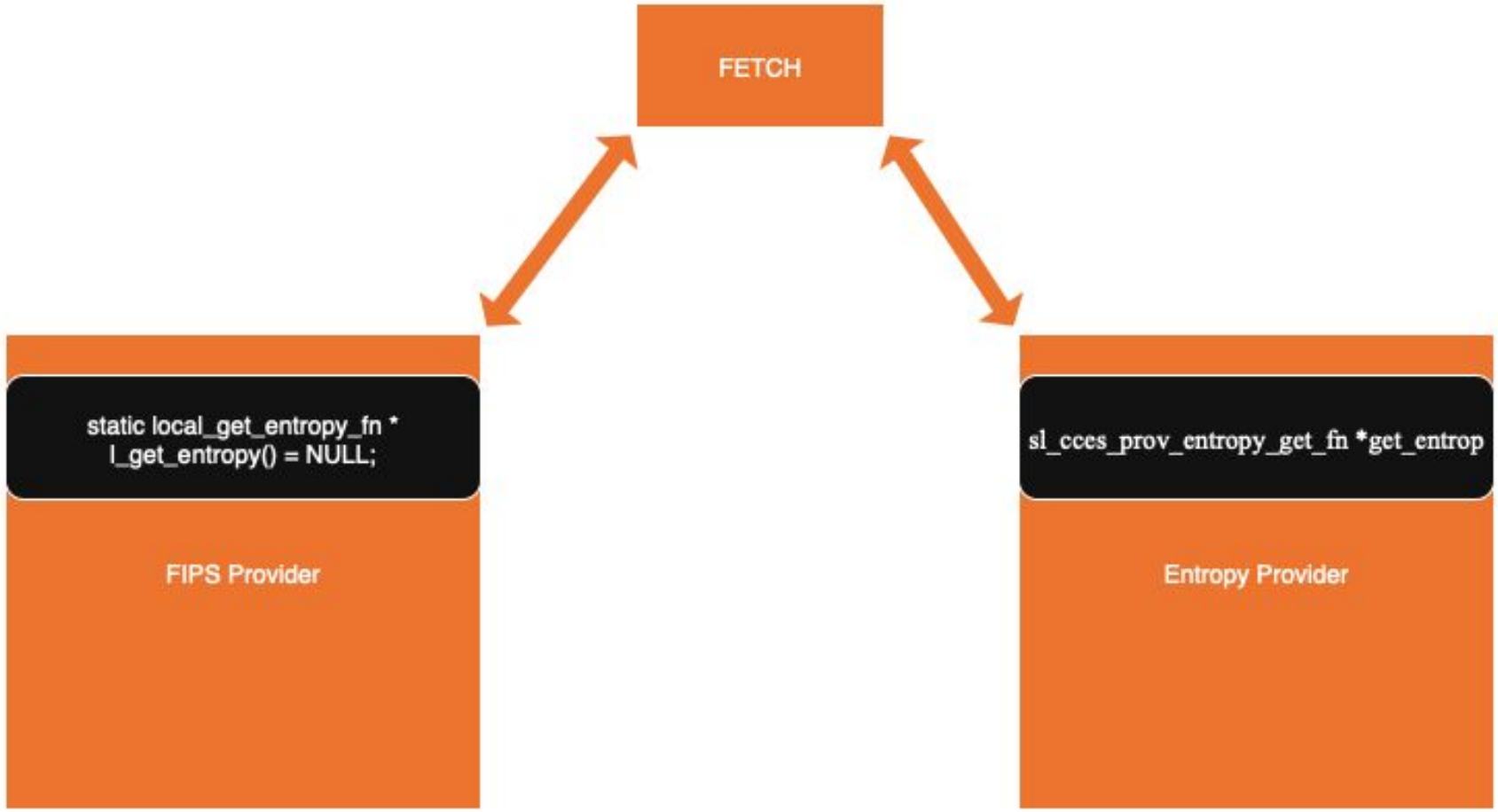
   What is required: (i) the testing lab **shall** corroborate the entropy strength estimate of the sources as provided by the vendor, (ii) the Security Policy **shall** state the minimum number of bits of entropy requested per each GET function call.

   If the amount of entropy used to generate the module's SSPs employed in an approved mode is less than 112 bits, then this module cannot be validated.

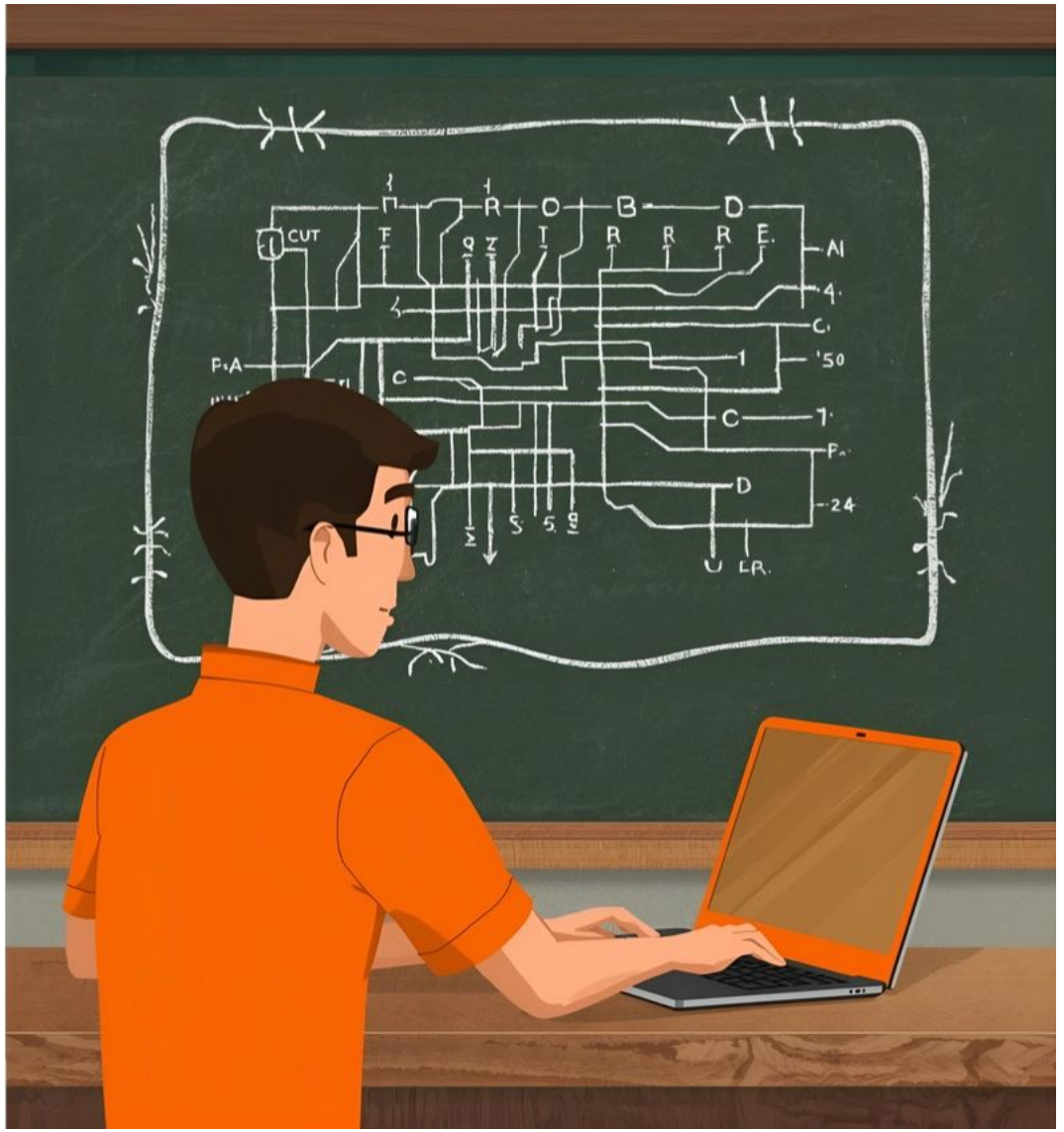
   If the amount of entropy used to generate the module's SSPs is at least 112 bits while the module generates SSPs with a comparable cryptographic strength greater than the amount of available entropy, the following caveat **shall** be included in the module's certificate: *The module generates SSPs (e.g., keys) whose strengths are modified by available entropy*.



Fetching to the Rescue



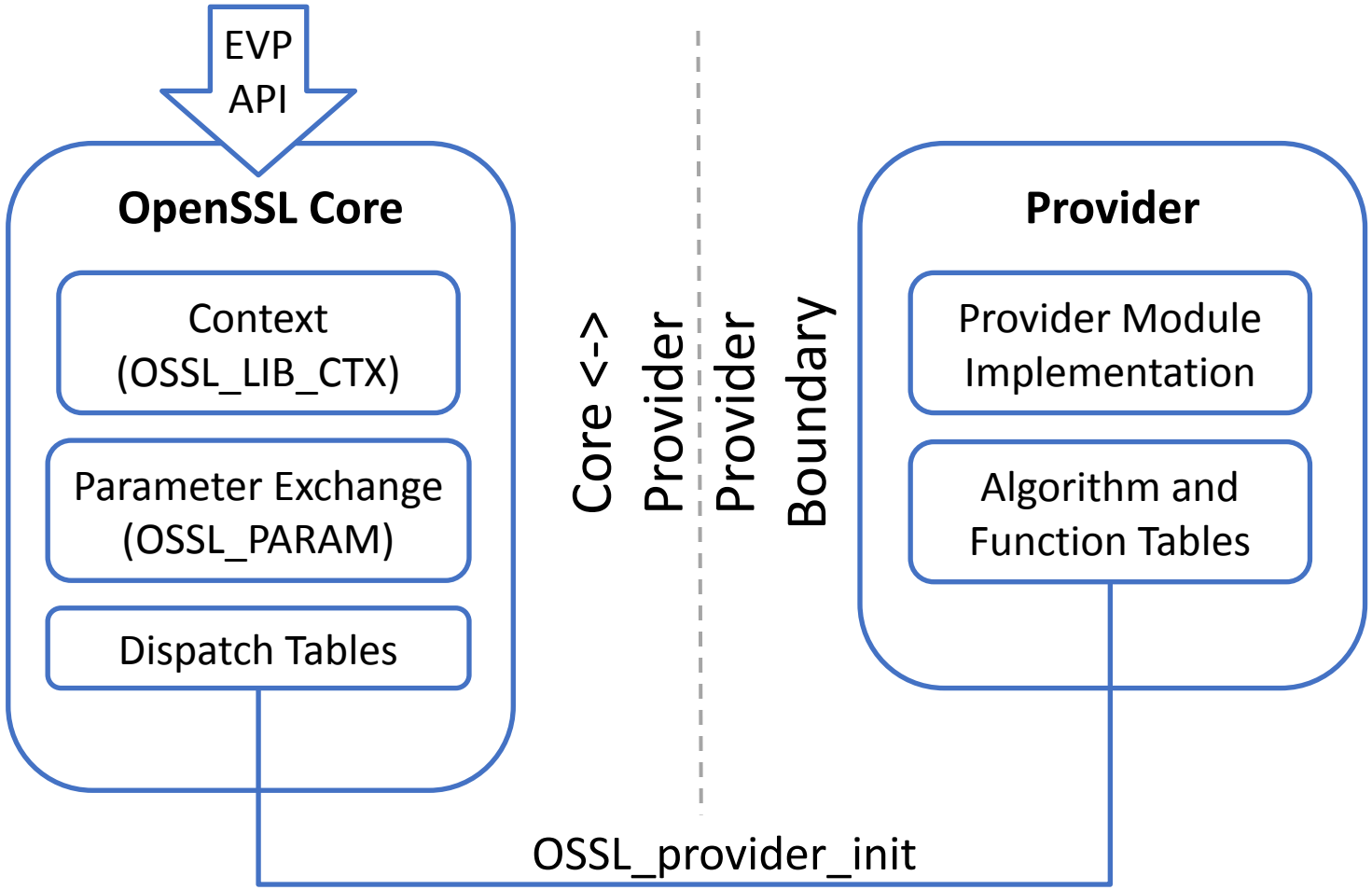
Implementation



Provider Framework – Overview

- What is it?
  - Introduced in **OpenSSL 3.0**
  - A **plug-in style API** for cryptographic implementations
  - Replaces the old "ENGINE" framework
- Key Features
  - **Providers** supply cryptographic algorithms (cipher, digest, key management, etc.)
  - Supports **built-in** (default, legacy, base) and **custom** providers (FIPS, 3rd party providers,)
  - Fine-grained control over algorithm selection and properties
- Benefits
  - Modular & extensible design
  - Enables FIPS 140-3 compliance through the FIPS provider
  - Simplifies integration of hardware accelerators or custom crypto
  - Better separation between the OpenSSL core and crypto implementations

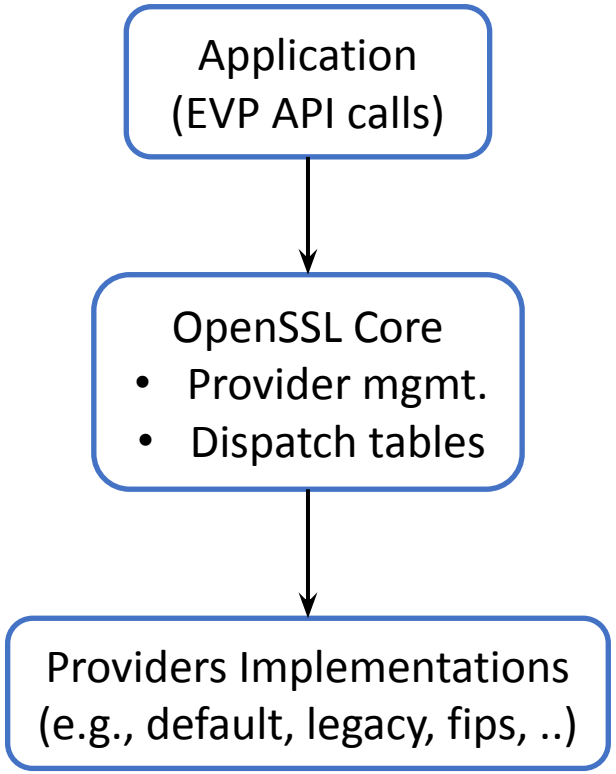
Provider Framework - Architecture



Provider Framework – Core Concepts

SafeLogic

- Providers: Shared modules implementing cryptographic algorithms.
- Dispatch Tables: Function pointers mapping algorithm operations.
- Contexts (OSSL_LIB_CTX): Hold state, configurations, and loaded providers.
- Parameter Exchange (OSSL_PARAM): Flexible structure for algorithm settings.
- Core ↔ Provider Boundary: OpenSSL core loads providers dynamically via OSSL_provider_init.
- EVP API → uses Core → dispatches to algorithms in Providers.
- Providers return function tables for supported operations (cipher, digest, keymgmt, etc.).



Basic Steps to Build and Use Provider

SafeLogic

- Create a provider module (shared lib) and export OSSL_provider_init()
- Place the module in the OpenSSL modules directory
- Enabled the provider in configuration
- Use it.

Example Simple Entropy Provider

Source Code

SafeLogic

```c
int OSSL_provider_init(const OSSL_CORE_HANDLE *handle,
                       const OSSL_DISPATCH *in,
                       const OSSL_DISPATCH **out,
                       void **provctx) {
  *out = my_entropy_dispatch_table;
  return 1;
}
```

```c
static const OSSL_DISPATCH my_entropy_dispatch_table[] = {
    {OSSL_FUNC_PROVIDER_QUERY_OPERATION,
     (void (*)(void))my_entropy_query_operation},
    {OSSL_FUNC_PROVIDER_GETTABLE_PARAMS,
     (void (*)(void))my_entropy_gettable_params},
    {OSSL_FUNC_PROVIDER_GET_PARAMS, (void (*)(void))my_entropy_get_params},
    {0, NULL}};
```

```c
static const OSSL_ALGORITHM *
my_entropy_query_operation(void *provctx, int operation_id, int *no_cache) {
  *no_cache = 0;
  switch (operation_id) {
  case OSSL_OP_RAND:
    return my_entropy_query_operation_rand_array;
  default:
    return NULL;
  }
}
```

```c
static const OSSL_ALGORITHM my_entropy_query_operation_rand_array[] = {
    {"myentropy", "provider=myentropy", my_entropy_rand_dispatch_table, NULL},
    {NULL, NULL, NULL, NULL}};
```

```c
static const OSSL_DISPATCH my_entropy_rand_dispatch_table[] = {
    {OSSL_FUNC_RAND_NEWCTX, (void (*)(void))my_entropy_newctx},
    {OSSL_FUNC_RAND_FREECTX, (void (*)(void))my_entropy_freectx},
    {OSSL_FUNC_RAND_INSTANTIATE, (void (*)(void))my_entropy_instantiate},
    {OSSL_FUNC_RAND_UNINSTANTIATE, (void (*)(void))my_entropy_uninstantiate},
    {OSSL_FUNC_RAND_GENERATE, (void (*)(void))my_entropy_generate},
    {OSSL_FUNC_RAND_GETTABLE_CTX_PARAMS,
     (void (*)(void))my_entropy_gettable_ctx_params},
    {OSSL_FUNC_RAND_GET_CTX_PARAMS, (void (*)(void))my_entropy_get_ctx_params},
    {OSSL_FUNC_RAND_GET_SEED, (void (*)(void))my_entropy_get_seed},
    {OSSL_FUNC_RAND_CLEAR_SEED, (void (*)(void))my_entropy_clear_seed},
    {0, NULL}};
```

Provider Entry Point

- Definition:

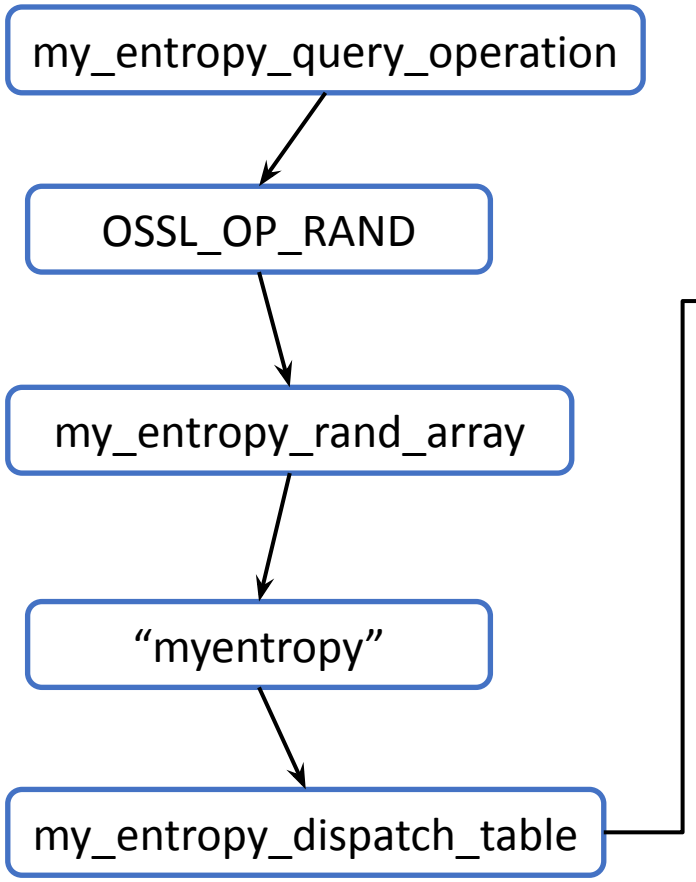
```
typedef int (OSSL_provider_init_fn)(const OSSL_CORE_HANDLE *handle,
                                    const OSSL_DISPATCH *in,
                                    const OSSL_DISPATCH **out,
                                    void **provctx);
```

- This function is expected to be present in any dynamically loadable provider module. If this function doesn't exist in a module, that module is not an OpenSSL provider module. The Core expects this function symbol to be named: OSSL_provider_init
- handle: pointer to opaque type OSSL_CORE_HANDLE. This can be used together with some functions passed via “in” to query (core) data.
- in: the array of functions that the Core passes to the provider.
- out: the array of base functions that the provider passes back to the Core.
- provctx: a provider side context object, optionally created if the provider needs it.
- Returns 1 on success, 0 otherwise.

Provider Dispatch Tables

- Typical main dispatch table functions:
  - OSSL_FUNC_PROVIDER_TEARDOWN
  - OSSL_FUNC_PROVIDER_GETTABLE_PARAMS
  - OSSL_FUNC_PROVIDER_GET_PARAMS
  - **OSSL_FUNC_PROVIDER_QUERY_OPERATION**
  - Etc.
- The query operation function returns further information about operations supported by the provider, i. e.:
  - OSSL_OP_DIGEST
  - OSSL_OP_CIPHER
  - OSSL_OP_RAND
  - Etc.
- All operations supported by the Core can be found in openssl/core_dispatch.h

OSSL Provider as Entropy Source




```
OSSL_FUNC_RAND_NEWCTX                =>  my_entropy_newctx
OSSL_FUNC_RAND_FREECTX               =>  my_entropy_freectx
OSSL_FUNC_RAND_INSTANTIATE           =>  my_entropy_instantiate
OSSL_FUNC_RAND_UNINSTANTIATE         =>  my_entropy_uninstantiate
OSSL_FUNC_RAND_GENERATE              =>  my_entropy_generate
OSSL_FUNC_RAND_RESEED                =>  my_entropy_reseed
OSSL_FUNC_RAND_ENABLE_LOCKING        =>  my_entropy_enable_locking
OSSL_FUNC_RAND_LOCK                  =>  my_entropy_lock
OSSL_FUNC_RAND_UNLOCK                =>  my_entropy_unlock
OSSL_FUNC_RAND_GETTABLE_CTX_PARAMS   =>  my_entropy_gettable_ctx_params
OSSL_FUNC_RAND_GET_CTX_PARAMS        =>  my_entropy_get_ctx_params
OSSL_FUNC_RAND_VERIFY_ZEROIZATION    =>  my_entropy_verify_zeroization
OSSL_FUNC_RAND_GET_SEED              =>  my_entropy_get_seed
OSSL_FUNC_RAND_CLEAR_SEED            =>  my_entropy_clear_seed
```

Function my_entropy_get_seed realizes GetEntropy conceptual interface

Example Simple Entropy Provider

Source Code and Building


```c
static size_t my_entropy_get_seed(void *vseed, unsigned char **pout,
                                  int entropy, size_t min_len, size_t max_len,
                                  ossl_unused int prediction_resistance,
                                  ossl_unused const unsigned char *adin,
                                  ossl_unused size_t adin_len) {
  fprintf(stderr, "-- my_entropy_get_seed\n");
  size_t bytes_needed = entropy >= 0 ? (((size_t)entropy) + 7) / 8 : 0;
  unsigned char *buf = OPENSSL_secure_malloc(bytes_needed);
  // #include <unistd.h>
  getentropy(buf, bytes_needed);
  *pout = buf;
  return bytes_needed;
}

static void my_entropy_clear_seed(void *vseed, unsigned char *out,
                                  size_t outlen) {
  OPENSSL_secure_clear_free(out, outlen);
}
```

```
$ cc -bundle -o lib/ossl-modules/myentropy.dylib \
? my_entropy_prov.c -I include -L lib -l crypto

$ nm -n -U lib/ossl-modules/myentropy.dylib
00000000000039c4 T _OSSL_provider_init (1)
00000000000039f4 t _my_entropy_query_operation (3)
0000000000003a44 t _my_entropy_gettable_params
0000000000003a5c t _my_entropy_get_params
0000000000003b80 t _my_entropy_newctx
0000000000003bb8 t _my_entropy_freectx
0000000000003be8 t _my_entropy_instantiate
0000000000003c10 t _my_entropy_uninstantiate
0000000000003c24 t _my_entropy_generate
0000000000003c54 t _my_entropy_gettable_ctx_params
0000000000003c70 t _my_entropy_get_ctx_params
0000000000003d88 t _my_entropy_get_seed (6)
0000000000003e4c t _my_entropy_clear_seed
0000000000004058 s _my_entropy_dispatch_table (2)
0000000000004098 s _my_entropy_query_operation_rand_array (4)
00000000000040d8 s _my_entropy_rand_dispatch_table (5)
0000000000004178 s _my_entropy_gettable_ctx_params_array
0000000000004218 s _my_entropy_gettable_params_array
```

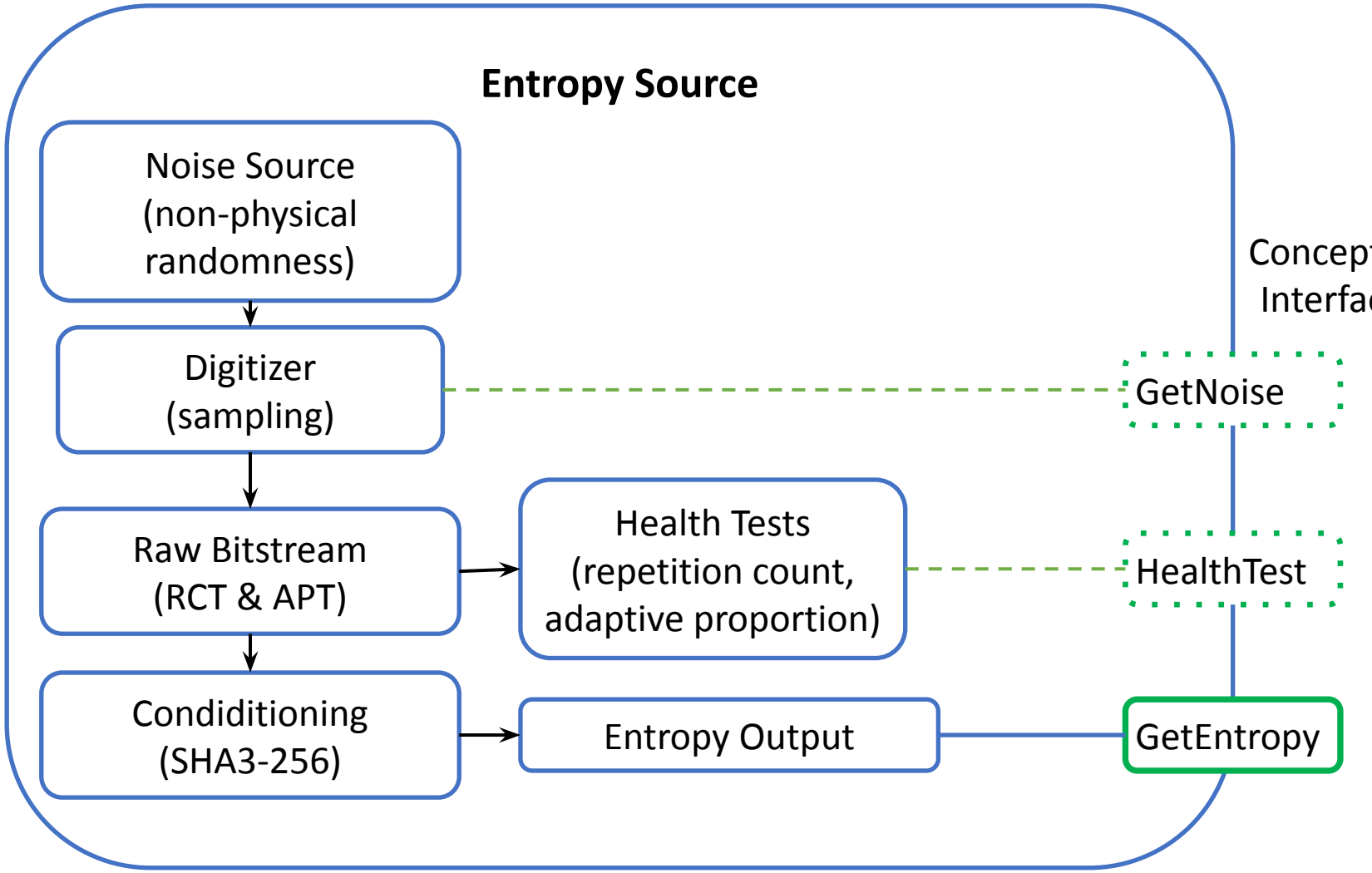
Example Simple Entropy Provider

Configuration and Usage


```
$ cat openssl.cnf
openssl_conf = openssl_init

[openssl_init]
providers     = provider_sect
random        = random_sect

[provider_sect]
default       = default_sect
myentropy     = myentropy_sect

[default_sect]
activate      = 1

[myentropy_sect]
activate      = 1

[random_sect]
seed = myentropy
```

```
$ setenv DYLD_LIBRARY_PATH lib
$ env | grep OPENSSL
OPENSSL_CONF=openssl.cnf
OPENSSL_MODULES=lib/ossl-modules
$ setenv DYLD_LIBRARY_PATH lib
$ bin/openssl genrsa | head -5
-- my_entropy_get_seed
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCh64OIKmqsAz3x
stSP8fvlnyqstoZPX5kmDdFRWSzkmMEeWCtCEcCFo4dQc5Abmwqa9IJGcteS0+7A
bPPA4cRTxsPmhKSQ7SYOXsVdvoLy20KaT3oLGFv7K5+9kQsW9I1YrNisEb/kN05y
LPwHUUzLKaxBytpscSu8xtQPiIQb5ft+UMDjhiKL7AUANHo5QrFigaY1nG1vmWkO
```

SP 800-90B Compliant Entropy Source

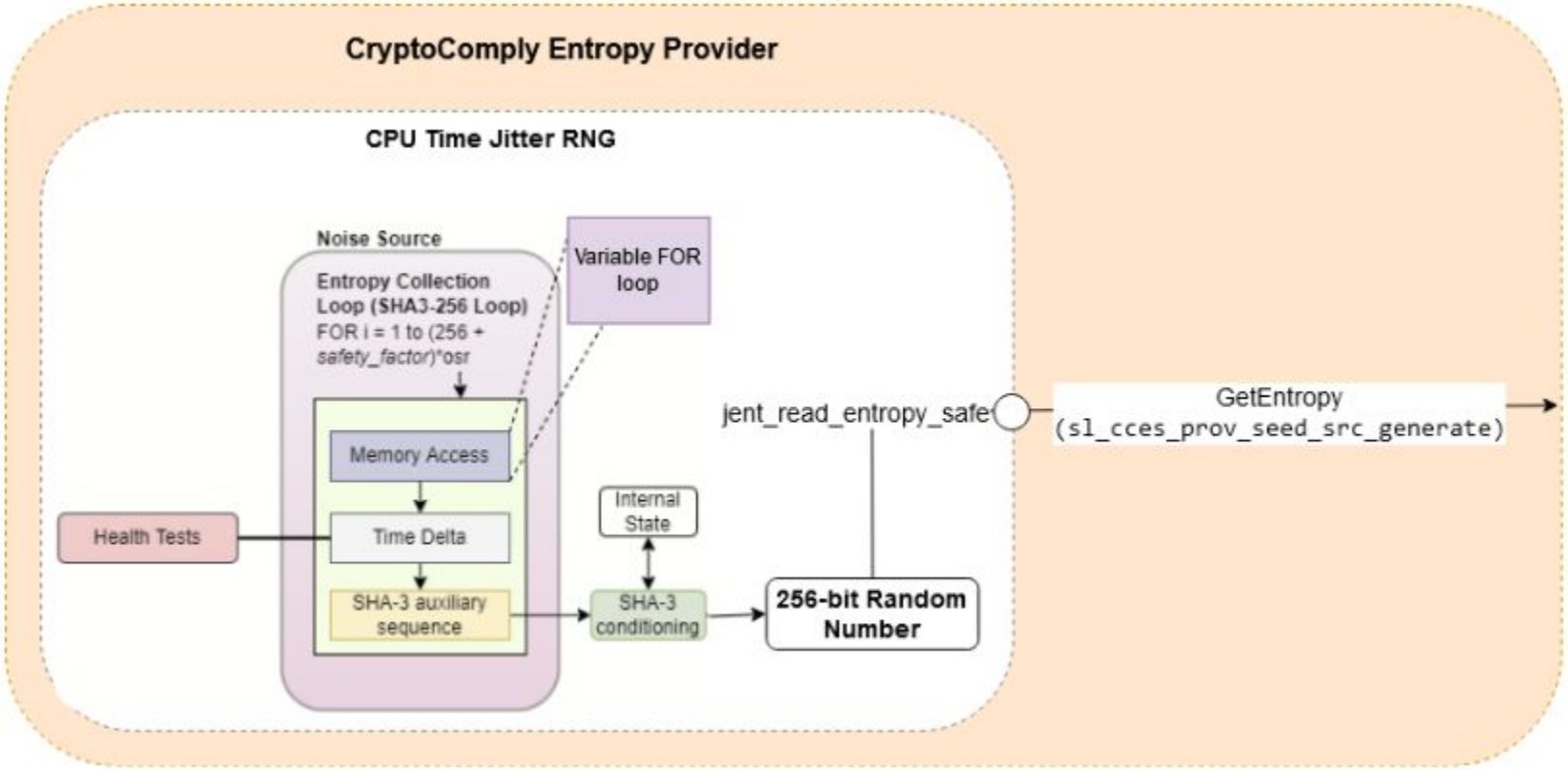


Permitted to be available only in "test mode".

Optional if the execution of the tests can be initiated in another manner.

CryptoComply Entropy Provider - architecture

SafeLogic



CryptoComply Entropy Provider - initialization

- Gets required Core functions.
- Creates provider context.
- Gets provider configuration parameters.
- Performs internal integrity test:
  - The same idea as in FIPS module.
  - Opens and reads entropy source provider module file.
  - Calculates HMAC and verifies its value with the value in configuration file.
- Initializes internal/backend entropy source.
- Sets the main dispatch table (out).

CryptoComply FIPS module integration points

SafeLogic

- Entropy Source provider needs to be registered and available in the Core prior the FIPS provider loading (this check is a part of OSSL_provider_init).
- Entropy provider (context) is used in provider_seeding.c:
  - ossl_prov_get_entropy
  - ossl_prov_cleanup_entropy

CryptoComply Entropy Provider
as default entropy source

SafeLogic

- If the entropy provider name is set as a seed parameter in the random section configuration, the entropy provider is used as a default and global entropy source in OpenSSL:

```
[openssl_init]
providers = provider_sect
random    = random_sect

[provider_sect]
cryptocomply-entropy  = cryptocomply_entropy_sect

[random_sect]
seed = CRYPTOCOMPLY-ENTROPY-SEED-SRC
```

CryptoComply Entropy Provider - example configuration

SafeLogic

```
# openssl.cnf:

openssl_conf = openssl_init

.include cryptocomply-entropy.cnf
.include fipsmodule.cnf

[openssl_init]
providers = provider_sect
random    = random_sect

[provider_sect]
cryptocomply-entropy  = cryptocomply_entropy_sect
default = default_sect
fips = fips_sect

[default_sect]
activate = 1

[random_sect]
seed = CRYPTOCOMPLY-ENTROPY-SEED-SRC
```

```
# cryptocomply-entropy.cnf:

[cryptocomply_entropy_sect]
activate   = 1
module-mac = A8:E6:C8:89:C4:B3:76:8D:A4:E0:40…
# force-internal-timer = 1
```

CryptoComply Entropy Provider - in OpenSSL

SafeLogic

```
$ openssl list -providers
Providers:
  cryptocomply-entropy
    name: CryptoComply Entropy Provider
    version: 1.1.1
    status: active
  default
    name: OpenSSL Default Provider
    version: 3.5.2
    status: active
  fips
    name: 140-3 FIPS Provider
    version: 4.0.0-FIPS 140-3
    status: active
```

```
$ openssl list -random-generators
Provided RNGs and seed sources:
  CRNG-TEST @ fips
  CRYPTOCOMPLY-ENTROPY-SEED-SRC @ cryptocomply-entropy
  CTR-DRBG @ default
  CTR-DRBG @ fips
  HASH-DRBG @ default
  HASH-DRBG @ fips
  HMAC-DRBG @ default
  HMAC-DRBG @ fips
  SEED-SRC @ default
  TEST-RAND @ default
  TEST-RAND @ fips
```

CryptoComply Entropy Provider
- testing requirements

SafeLogic

- The entropy provider is tested for each accredited OE
- The following tests have been performed:
  - Entropy data quality
    - Long data generation run (1000000 samples)
    - Restart data generation (1000 x 1000 samples)
  - ACVP tests of SHA-3 algorithm implementation
- The provider is manually tested by verification of output of the following commands:
  - openssl list -providers
  - openssl list random-generators
- Additionally, the provider is tested by writing, compiling and running simple C program that loads the provider and uses its raw noise generation function.

CryptoComply Entropy Provider
- validation documentation

SafeLogic

- Entropy Assessment Report (EAR)
- Raw datasets for the assessment
- Health-test documentation
- ESV server submission package
- Public Use Document (PUD)
- Data-collection attestations(s)
- OE list & operating bounds
- Conditioning analysis details

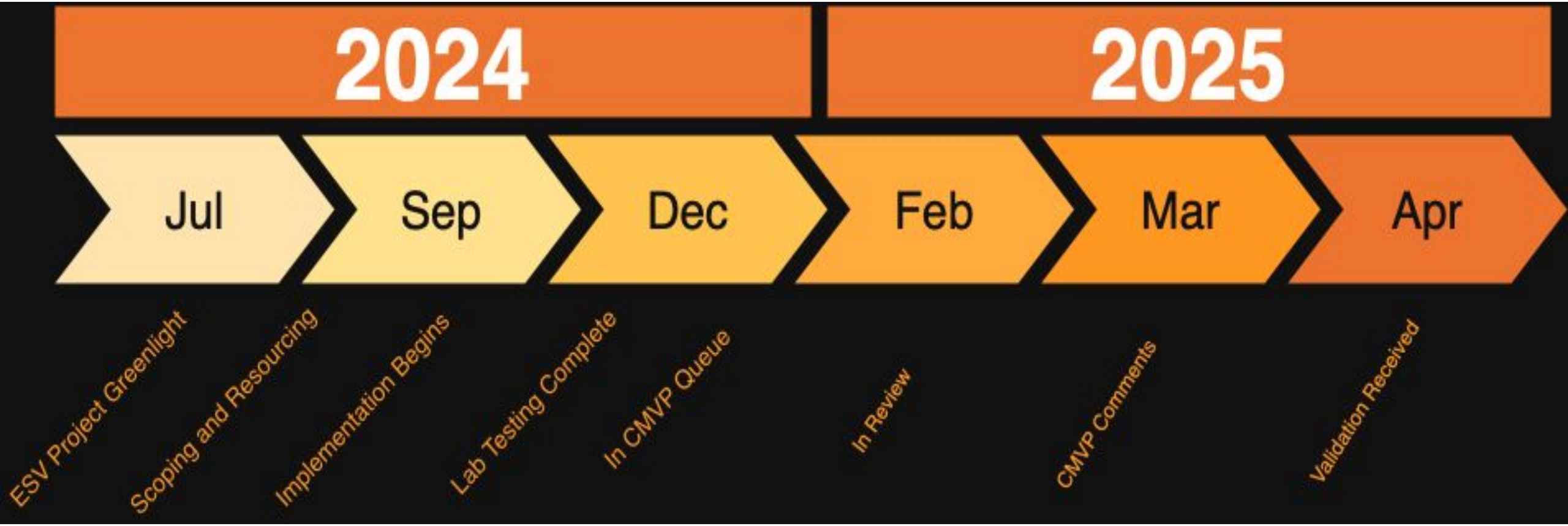
Optionally:

- Module linkage note (when part of a FIPS 140-3 module)

Finishing It Up With NIST's ESV

Project Timeline

SafeLogic



Validation!!!

SafeLogic

Vendor	Implementation	Certificate Number	Validation Date
SafeLogic, Inc.	CryptoComply Entropy Provider	E241	4/18/2025

Entropy Certificate #E241

Details

Implementation Name	CryptoComply Entropy Provider
Standard	SP 800-90B
Description	CPU Jitter v3.6.0 as an OpenSSL-compatible provider
Version	1.1.1
Noise Source Classification	Non-Physical
Reuse Status	Reuse restricted to vendor

	Operating Environments	Vetted Conditioning Component CAVP Certificates
Bits of Entropy per Output: Full entropy. Output Size in Bits: 256	• AlmaLinux 9 running on Intel Xeon E5-4667v4 • Android 13 running on Google Tensor G2 • Debian 11 running on Intel Xeon E5-4667v4 • FreeBSD 13 running on Intel Xeon E5-4667v4 • iOS 16 running on Apple A15 Bionic • iPadOS 17 running on Apple M1 • macOS 13 (Ventura) running on Apple M2 • Oracle Solaris 11.4 running on Intel Xeon E5-4667v4 • Red Hat Enterprise Linux 9 running on Intel Xeon E5-4667v4 • Rocky Linux 9 running on Intel Xeon E5-4667v4 • SUSE Linux Enterprise Server 15 running on Intel Xeon E5-4667v4 • Ubuntu 22.04 running on Intel Xeon E5-4667v4 • Windows 10 running on Intel Xeon E5-4667v4 • Windows 11 running on Intel Xeon E5-4667v4 • Windows Server 2019 running on Intel Xeon E5-4667v4 • Windows Server 2022 running on Intel Xeon E5-4667v4	• A6412 (SHA3-256)

Public Use Document

Vendor

SafeLogic, Inc.
8300 Boone Blvd., Suite 500
Vienna, VA 22182
USA

SafeLogic Inside Sales
sales@safelogic.com
844-436-2797

Related Files

Public Use Document

Validation History

Date	Lab
4/18/2025	Lightship Security Inc.

SafeLogic
Cryptography Simplified

SP 800-90B Non-Proprietary Public Use Document

CryptoComply Entropy Provider
Version: 1.1.1

Document Version: 1.2
Release Date: March 11, 2025

Prepared for:
SafeLogic, Inc.

Prepared by:

Lightship Security
www.lightshipsec.com

SafeLogic, Inc. 1/9

Special Thanks

- Lightship Security
  - Testing Lab
- Stephan Mueller
  @tsec Security

Questions?

SafeLogic

Cryptography Simplified