四川大学
SICHUAN UNIVERSITY

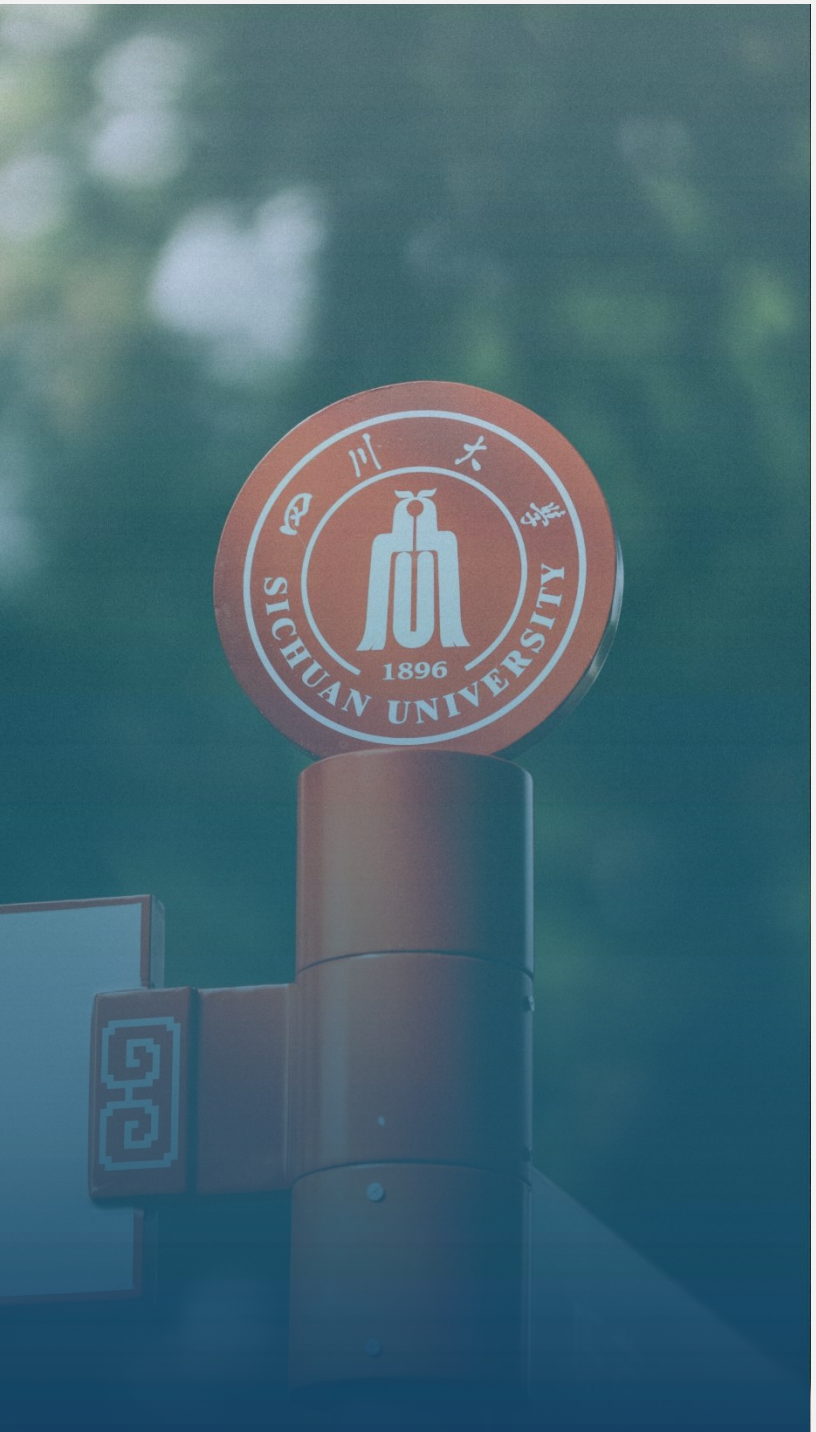
面向 OpenSSL 3 的多进程安全国密Provider

答辩人：

指导老师：

目录

CONTENTS



Part.01

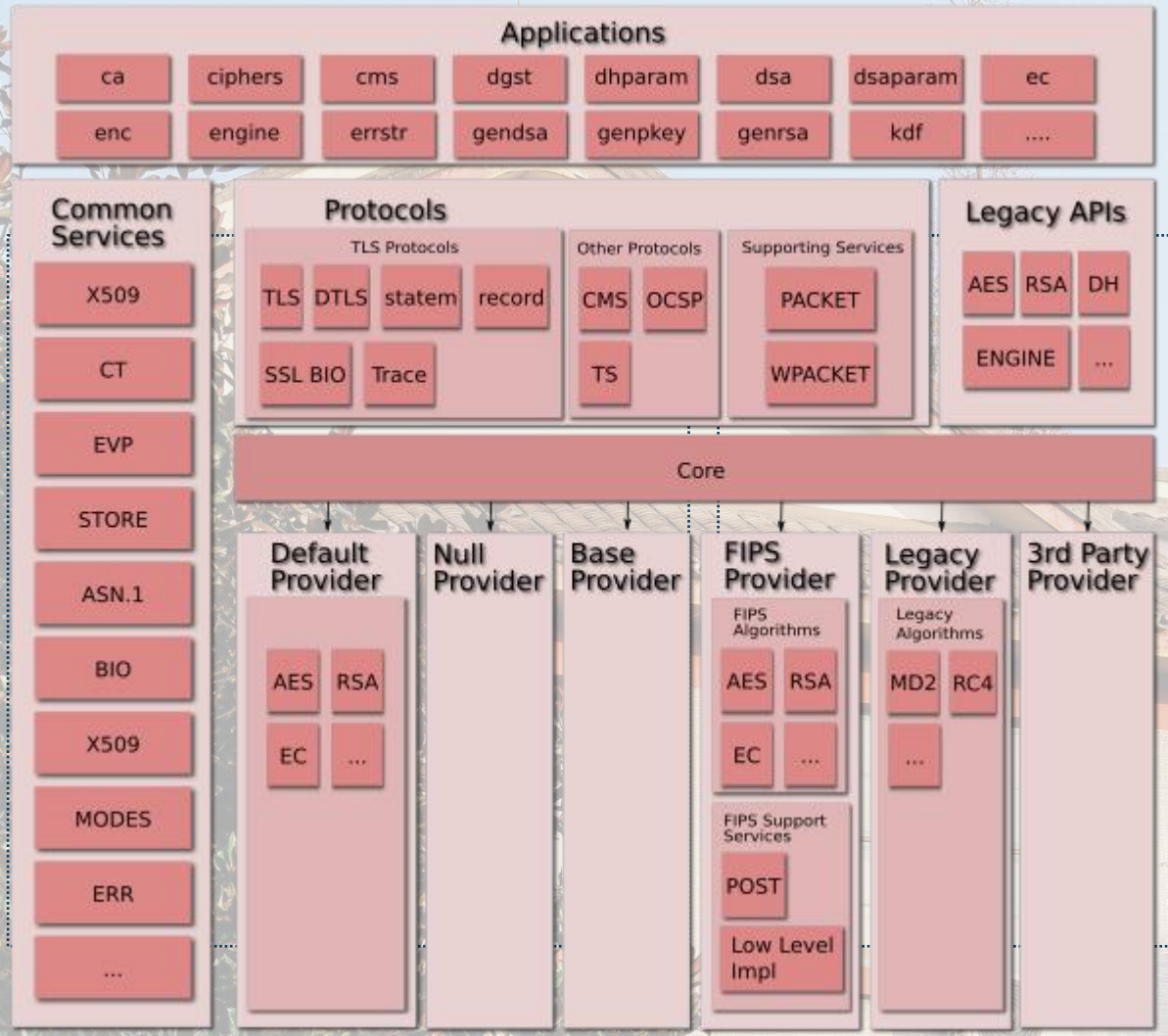
选题背景

选题背景

随着 OpenSSL 3 引入 Provider 架构，密码学实现从过去的 Engine 机制过渡到更现代的“可插拔算法提供者”。企业应用广泛依赖硬件安全模块（HSM）、智能卡或受信任执行环境（TEE）来存储密钥并完成签名、加解密与随机数生成；它们通常通过 PKCS#11 标准接口对接上层。

要在现代 OpenSSL 生态中稳定、可观测地用好这些硬件/中间件，需要一个面向 OpenSSL 3 的 Provider，实现与 PKCS#11 的高质量集成，并在真实生产环境（多进程、多线程、长生命周期）下具备足够的健壮性。

1.1 研究背景


Applications
ca
ciphers
cms
dgst
dhparam
dsa
dsaparam
ec
enc
engine
errstr
gendsa
genpkey
genrsa
kdf
....
Common Services
X509
CT
EVP
STORE
ASN.1
BIO
X509
MODES
ERR
...
Protocols
TLS Protocols
TLS
DTLS
statem
record
SSL BIO
Trace
Other Protocols
CMS
OCSP
TS
Supporting Services
PACKET
WPACKET
Legacy APIs
AES
RSA
DH
ENGINE
...
Core
Default Provider
AES
RSA
EC
...
Null Provider
Base Provider
FIPS Provider
FIPS Algorithms
AES
RSA
EC
...
FIPS Support Services
POST
Low Level Impl
Legacy Provider
Legacy Algorithms
MD2
RC4
...
3rd Party Provider

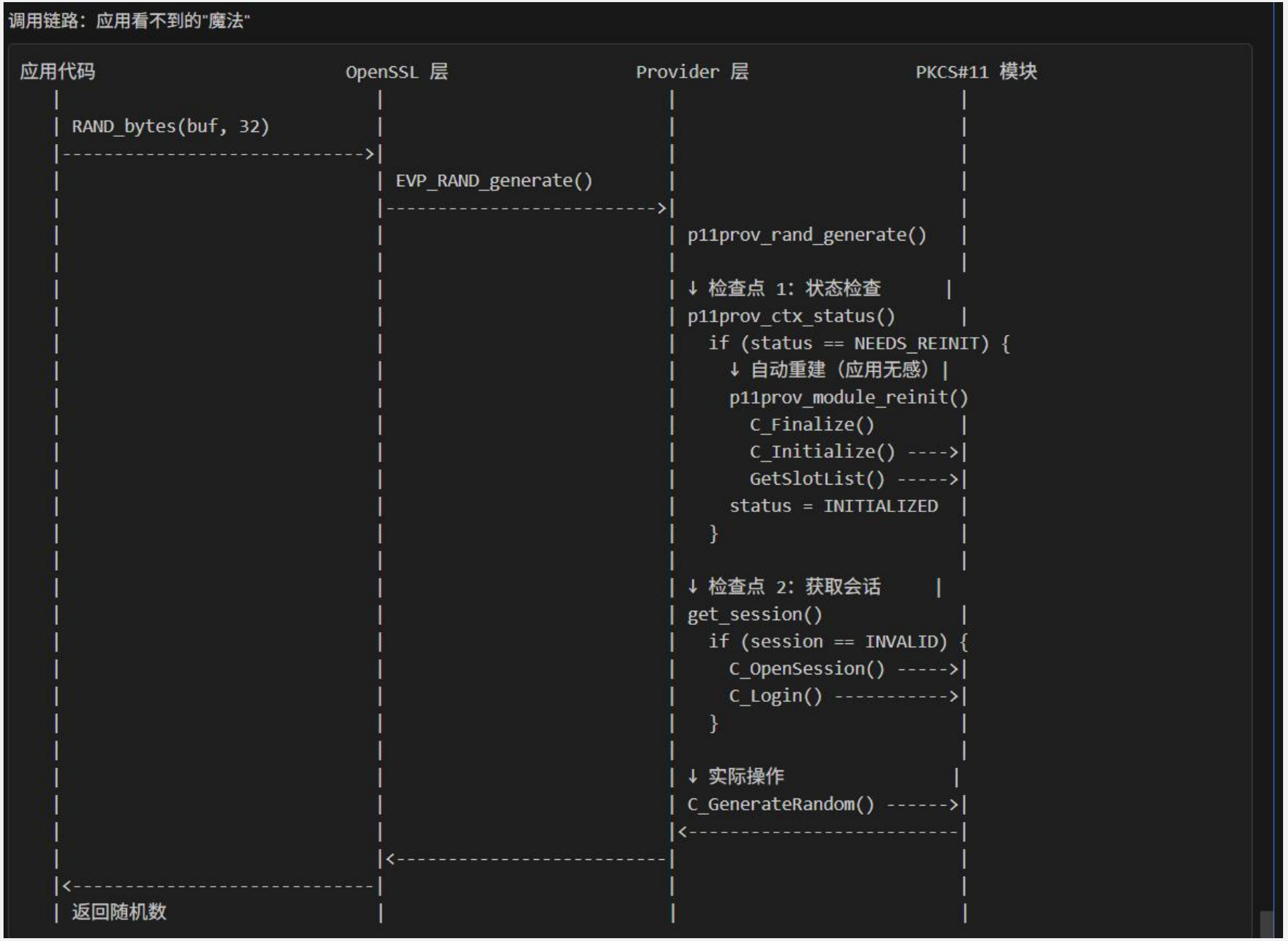
OpenSSL 3 Provider 架构（与 Engine 的差异）

Provider 是 OpenSSL 3 的一等公民，通OSSL_PROVIDER/dispatch 表暴露算法族（如签名、密钥管理、随机数、编解码等）。

应用通过 EVP 层按“名称+属性”获取实现（fetch），支持按 FIPS、硬件能力、性能等维度路由。

与 Engine 相比，Provider 粒度更清晰、隔离性更好、可并存多实现，便于在同一进程内组合多个来源的算法。

调用链路：应用看不到的"魔法"



RUN AND DEBUG

Debug openssl list (provider try)

WATCH

VARIABLES

Locals

ctx = 0x5555556ce8a0

__func__ = [14]

provctx = 0x5555556ce8a0

operation_id = 5

CALL STACK

Paused on breakpoint

helloworld.so!sdfprov_query(void * provctx, int operation_id, int * no_cache) hell...

libcrypto.so.3!ossl_provider_query_operation(const OSSL_PROVIDER * prov, int operation_

libcrypto.so.3!algorithm_do_this(OSSL_PROVIDER * provider, void * cbdata) core_alg...

libcrypto.so.3!ossl_provider_doall_activated(OSSL_LIB_CTX * ctx, int (*)(OSSL_PROVIDER

libcrypto.so.3!ossl_algorithm_do_all(OSSL_LIB_CTX * libctx, int operation_id, OSSL_PROV

libcrypto.so.3!ossl_method_construct(OSSL_LIB_CTX * libctx, int operation_id, OSSL_PROV

libcrypto.so.3!inner_evp_generic_fetch(struct evp_method_data_st * methdata, OSSL_PROVI

libcrypto.so.3!evp_generic_fetch(OSSL_LIB_CTX * libctx, int operation_id, const char *

libcrypto.so.3!EVP_RAND_fetch(OSSL_LIB_CTX * libctx, const char * algorithm, const char

libcrypto.so.3!rand_new_seed(OSSL_LIB_CTX * libctx) rand_lib.c 626:1

libcrypto.so.3!rand_get0_primary(OSSL_LIB_CTX * ctx, RAND_GLOBAL * dgbl) rand_lib.c

libcrypto.so.3!rand_get0_public(OSSL_LIB_CTX * ctx, RAND_GLOBAL * dgbl) rand_lib.c

libcrypto.so.3!RAND_bytes_ex(OSSL_LIB_CTX * ctx, unsigned char * buf, size_t num, unsig

rand_main(int argc, char ** argv) rand.c 202:1

do_cmd(struct lhash_st_FUNCTION * prog, int argc, char ** argv) openssl.c 428:1

main(int argc, char ** argv) openssl.c 309:1

helloworld.c M | openssl.c 5 | provider.c M | simple_rand.c U

helloworld.c > sdfprov_query(void *, int, int *)

```
121  static const OSSL_PARAM *sdfprov_gettable_params(void *provctx)
123      static const OSSL_PARAM params[] = {
126          OSSL_PARAM_utf8_ptr(OSSL_PROV_PARAM_BUILDINFO, NULL, 0),
127          OSSL_PARAM_int(OSSL_PROV_PARAM_STATUS, NULL),
128          OSSL_PARAM_END
129      };
130      return params;
131  }
132
133
134
135  static OSSL_FUNC_provider_query_operation_fn sdfprov_query;
136
137  static const OSSL_ALGORITHM *
138  sdfprov_query(void *provctx,int operation_id, int *no_cache){
139      SDFPROV_CTX *ctx = (SDFPROV_CTX *)provctx;
140      fprintf(stderr, "helloworld: query called operation_id=%d\n", o
141
142          switch (operation_id){
143              case OSSL_OP_DIGEST:
144                  return ctx->op_digest;
145              case OSSL_OP_RAND:
146                  return ctx->op_rand;
147          }
148
149
150          // /* Log algorithms we're returning */
151          // for (const OSSL_ALGORITHM *a = digest_algorithms; a && a
152          //     fprintf(stderr, "helloworld: returning algorithm '%
```

```
Provider-Skeleton$ openssl list -providers
nonsense: Hello, world!
available libcrypto param: openssl-version [type 6, size 0]
available libcrypto param: provider-name [type 6, size 0]
available libcrypto param: module-filename [type 6, size 0]
libcrypto param 'openssl-version' : 3.5.3
Providers:
  default
    name: OpenSSL Default Provider
    version: 3.5.3
    status: active
  helloworld
    name: Hello World Provider
    version: 0.1.0
    status: active
  pkcs11
    name: PKCS#11 Provider
    version: 1.1
    status: active
Provider-Skeleton$
```

```
ASUS$ cd
~$ python
Python 3.10.12 (main, Aug 15 2025, 14:32:43) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
libcrypto param 'openssl-version' : 3.5.3
helloworld: Initialized 1 RAND algorithm(s)
helloworld: Provider initialization complete
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
helloworld: query called operation_id=1
>>> exit()
~$ curl
libcrypto param 'openssl-version' : 3.5.3
helloworld: Initialized 1 RAND algorithm(s)
helloworld: Provider initialization complete
curl: try 'curl --help' or 'curl --manual' for more information
```

```
Try `wget --help' for more options.
~$ ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4642  bytes 23016740 (23.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4642  bytes 23016740 (23.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

~$ ping 8.8.8.8
ping: connect: Network is unreachable
~$
```

多进程/多线程环境的挑战

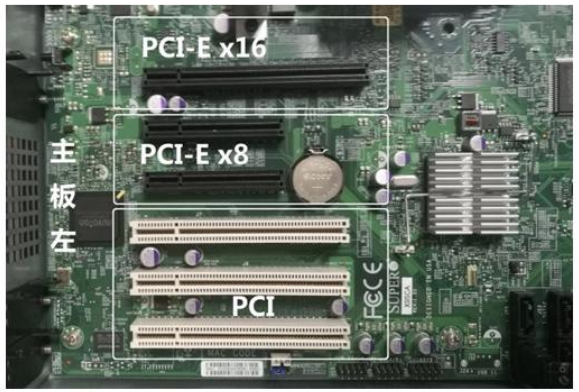
许多 Linux 服务会频繁 fork/exec 或并发处理请求。fork 会复制进程地址空间，但不会“有效复制”线程与锁的内部状态，继承的互斥量/读写锁可能不可用。

第三方模块不保证 fork-safe！

PKCS#11 、SDF 与硬件密码设备

- **硬件密码设备**
  - 指带有防篡改与密钥保护能力的硬件：**HSM（机架/PCIe/网络型）、智能卡/USB Key（UKey）、安全加速卡、TPM 等。**
  - 核心能力：密钥生成与安全存储（不可导出）、签名/验签、加/解密、哈希/KDF、真随机数（TRNG）、权限/审计。

密码卡



PKCS#11 、SDF 与硬件密码设备

- **PKCS#11**
  - 由 RSA/PKCS 系列演进的国际通用“加密令牌接口标准”（Cryptoki）。
  - 抽象模型：Slot（插槽）/Token（令牌）/Session（会话）/Object（对象，密钥/z证书）/Mechanism（算法机制）。
  - **动态库**形式（.so/.dll），统一 API：如 C_Initialize、C_OpenSession、C_Sign 等。
  - 跨厂商/跨平台通用性强，OpenSSL 等中间件普遍支持（OpenSSL 3 可通过 Provider，对应你用的 pkcs11-provider）。
- **SDF（密码设备应用接口规范）**
  - **国内密码行业标准**，标号为 GM/T 0018（ SDF 接口）。
  - 面向国产算法（国密）支持更完备（SM2/SM3/SM4 等），API 直观：SDF_OpenDevice、SDF_OpenSession、SDF_GenerateKeyPair_ECC、SDF_GenerateRandom...
  - 同样以**动态库形式**提供，但数据/对象模型与函数命名风格不同，更贴合国产设备生态。
  - 在国内合规场景（等保/商密）部署广泛，厂商通常同时提供 SDF 与 PKCS#11 两套接口。

Part.02

拟实现项目的功能

模块 1：进程安全基础设施

- **atfork 钩子注册**
  - 实现 `fork_prepare`、`fork_parent`、`fork_child` 三个回调函数
  - 在 Provider 初始化时通过 `pthread_atfork()` 注册
  - 技术要求：回调函数必须是异步信号安全的（AS-safe），不能调用 malloc/printf 等不安全函数（可选降级为 DEBUG 模式允许）
- **fork_prepare（准备阶段）**
  - 获取全局上下文池的读锁（`ctx_pool.rwlock`）
  - 遍历所有已初始化的上下文，对每个上下文：
    - 尝试获取 slots 的写锁，失败则降级为读锁（避免死锁）
  - 验收标准：日志显示"Fork prepare: locked N contexts"

- **fork_parent（父进程恢复）**
  - 释放所有在 prepare 阶段获取的锁
  - 验收标准：父进程在 fork 后能立即继续执行 Provider 操作（签名/加密/随机数）
- **fork_child（子进程重置）**
  - 强制重新初始化所有读写锁（调用 `p11prov_force_rwlock_reinit`）
  - 遍历所有上下文，将状态标记为 `NEEDS_REINIT`
  - 标记 PKCS#11 模块需要重新初始化（`p11prov_module_mark_reinit`）
  - 调用会话池重置：`p11prov_session_pool_fork_reset`
    - 将所有会话句柄设为 `CK_INVALID_HANDLE`
    - 清空 `in_use`、`is_login` 等状态标志
    - 重新初始化每个会话的互斥锁
  - 调用对象池重置：`p11prov_obj_pool_fork_reset`
    - 标记所有缓存的密钥/证书对象为"需重新验证"
  - 验收标准：日志显示"Fork child: reset N contexts, invalidated M sessions"

模块：进程安全

- atfork 钩子（
  - fork_prepare: 锁状态
  - fork_parent: 解锁
  - fork_child: 重置锁 + 标记 NEEDS_REINIT
- 状态机
  - UNINITIALIZED/INITIALIZED/NEEDS_REINIT/IN_ERROR
  - p11prov_ctx_status: 检查并触发重建
- 懒重建★核心★
  - p11prov_module_reinit
  - C_Finalize + C_Initialize
  - 重新枚举 slots/机制

Part.04

项目验收目标

测试： 基本 fork 安全性

测试目标：验证子进程能成功触发重建并执行操作

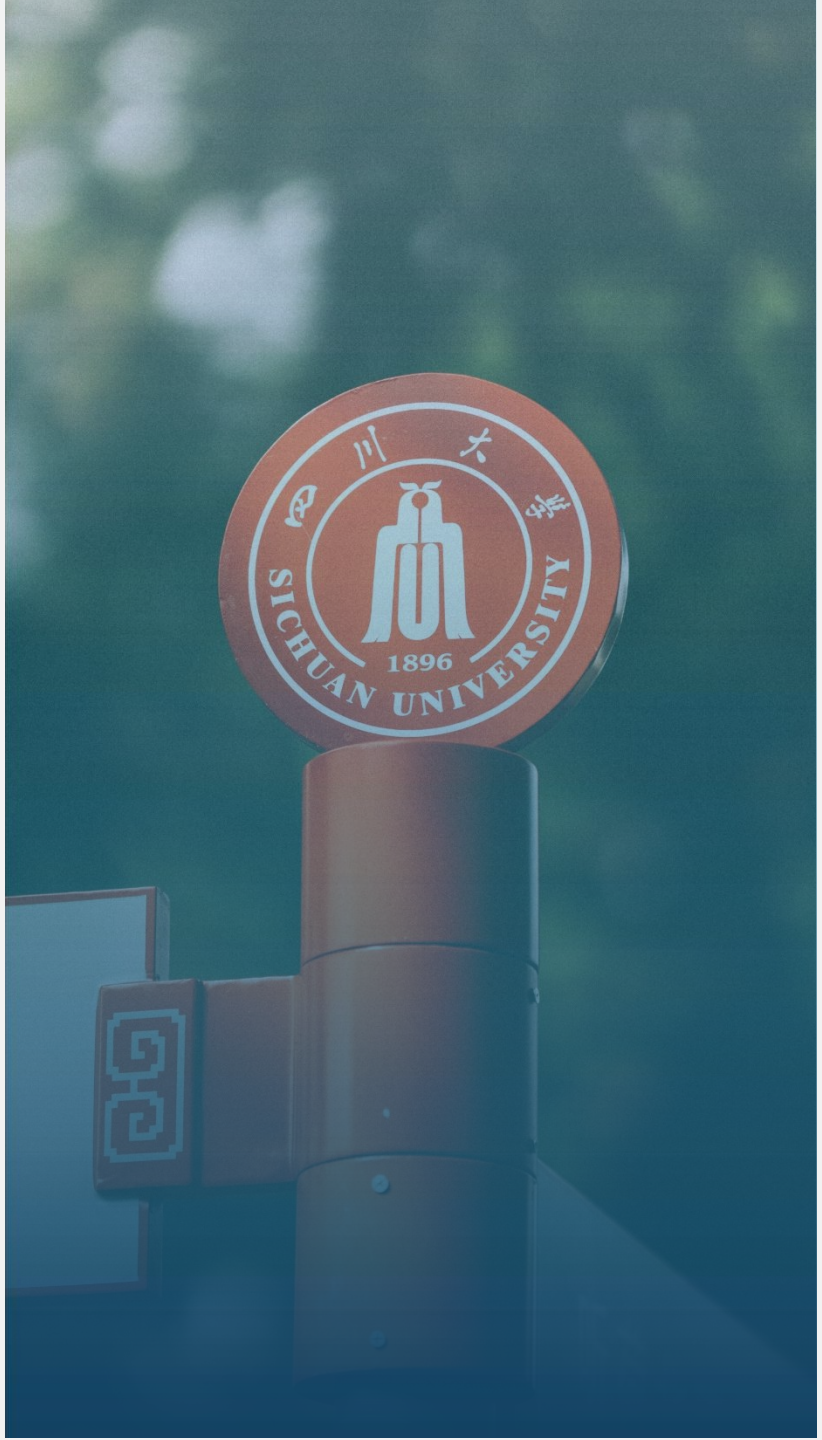
测试代码：

```c
// test_fork_basic.c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <openssl/provider.h>
#include <openssl/evp.h>
#include <openssl/rand.h>

int main() {
    printf("=== Test 1: Basic fork safety ===\n");

    // 1. 父进程加载 Provider
    OSSL_PROVIDER *prov = OSSL_PROVIDER_load(NULL, "pkcs11");
    if (!prov) {
        fprintf(stderr, "Failed to load provider\n");
        return 1;
    }
```

```c
if (pid == 0) {
    // 子进程
    printf("→ Child (PID=%d): starting\n", getpid());

    // 4. 子进程首次操作（应触发重建）
    unsigned char child_rand[32];
    if (RAND_bytes(child_rand, sizeof(child_rand)) != 1) {
        fprintf(stderr, "✗ Child: RAND_bytes failed\n");
        return 1;
    }
    printf("✓ Child: RAND_bytes OK (reinit triggered)\n");

    // 5. 子进程再做一次操作（验证重建后可持续使用）
    if (RAND_bytes(child_rand, sizeof(child_ra
        fprintf(stderr, "✗ Child: second RAND_
        return 1;
    }
    printf("✓ Child: second RAND_bytes OK\n");
} else {
    // 父进程
    // 6. 父进程在 fork 后继续工作（验证不受影响）
    if (RAND_bytes(parent_rand, sizeof(paren
        fprintf(stderr, "✗ Parent: post-fork
        return 1;
    }
    printf("✓ Parent: post-fork RAND_bytes O
```

fork 后必须重建，因为子进程继承的是"父进程的墓碑"（失效的句柄、死锁的锁、断开的连接），只有重建才能"复活"并正常工作。

```
void *dlhandle
void *hDevice;
void *hSession;
void *hKey;

pthread_mutex_t lock
pthread_rwlock_t rwlock
pthread_cond_t cond
```



Part.04

项目分工

分组实现

D

W

懒重建机制开发与实现。测试程序开发与文档说明。

L

Q

Atfork回调函数的实现

共同实现

实现导出OSSL_provider_init()

返回OSSL_DISPATCH函数表

OSSL_FUNC_PROVIDER_QUERY_OPERAION

实现至少一个OpenSSL支持的操作

.....

四川大学
SICHUAN UNIVERSITY

感谢各位老师指正

网络空间安全学院

答辩人：

指导老师：