Controller of Certifying Authorities
Ministry of Electronics & Information Technology
Government of India

IEEE COMPUTER SOCIETY Bangalore Chapter

IAS IEEE INDUSTRY APPLICATIONS SOCIETY Linking Research to Practice Bangalore Chapter

5th INTERNATIONAL CONFERENCE ON

PUBLIC KEY INFRASTRUCTURE AND ITS APPLICATIONS (PKIA 2024)

SEPTEMBER 5-6th, 2024

Experimenting Integration of Custom ECDSA Algorithm in OpenSSL

Geetha Sivanantham, Akila Krishnan
SETS,Chennai

Dr T R Reshmi
Principal Investigator-UBF
Scientist,
SETS,Chennai

C-DAC

IEEE BANGALORE SECTION

Outline

- Introduction
- Literature Review
- Methodology
- OpenSSL Libraries
- OpenSSL Engine
- Service Providers
- Recommendations
- Conclusion
- References

Introduction

- OpenSSL is a widely used cryptographic library essential for establishing secure communication through Transport Layer Security (TLS) protocols.
- It is also a key component in software-based secure key storage solutions, facilitating cryptographic operations.
- There is a growing need to customize OpenSSL's cryptographic modules for two main reasons:
  - i) To mitigate known vulnerabilities in existing OpenSSL implementations.
  - ii) To integrate custom cryptographic solutions into applications like blockchain.
- This article explores the methods for adding custom cryptographic implementations into the OpenSSL library, discussing the advantages and disadvantages of each approach.

Literature Review

Attack	Algorithm	Nature of Attack	Impact	OpenSSL Version
Cold Boot Attack [1]	AES	Attempt to retrieve precomputed AES round keys from memory dumps	Retrieved the distorted 11 round keys of AES 128.	OpenSSL 1.1
Cache Bleed Attack[2]	RSA	Based on the timing conflicts in L1 cache access identifies the exponentiation operations	Retrieves about 60% of 1024 bit key in RSA	OpenSSL 1.0.2f
Singular Curve Point Decompression Attack (SCPD) [4]	ECDSA	Attempt to know SECP curve base point before the scalar multiplication	Fetches private key from one faulty signature	OpenSSL 1.1.1
Fault Injection Attack [5]	ECDSA	Side channel attack	Retrieved the secret key from four ECDSA signatures	OpenSSL 1.1.1

Methodology

- The regular crypto modules will be hosted by libcrypto module under common services which will provide the implementation of all crypto modules.
- The legacy APIs provides a platform for activating engines for common crypto standards like public key crypto standard (PKCS11).
- The third-party provider provides API calls for hosting vendor based crypto algorithms inside OpenSSL like open quantum safe (OQS) interface to OpenSSL which enables crypto operations using post quantum algorithms through OpenSSL calls.
- The addition of custom crypto algorithms into OpenSSL can be accommodated in three ways:
  i) addition of new crypto algorithm into libcrypto module of OpenSSL [6].
  ii) Creating new engine to use the custom crypto implementations [7].
  iii) Defining a service provider library with the set of custom implementations and interfacing with OpenSSL [8].

OpenSSL Libraries

- The addition of custom ECDSA implementation written in C language in to the inbuilt crypto folder of openSSL. The detailed steps of integration are discussed as follows:
    i. The source files of the crypto algorithm written in C language must be added to openSSL/crypto/ec folder.
    ii. EVP interface integration
    iii. Crypto objects OID creation
    iv. Utility Mapping
    v. TLS settings configuration
    vi. OpenSSL configuration settings

```c
#include <openssl/evp.h>

// Define the custom method for your asymmetric algorithm
static EVP_PKEY_METHOD *my_custom_pkey_meth = NULL;

static int my_custom_pkey_sign(EVP_PKEY_CTX *ctx, unsigned char *sig, size_t *siglen,
                               const unsigned char *tbs, size_t tbslen) {
    // Custom signing operation implementation
    return 1; // Return 1 on success
}

static int my_custom_pkey_verify(EVP_PKEY_CTX *ctx, const unsigned char *sig, size_t sigle
                                 const unsigned char *tbs, size_t tbslen) {
    // Custom verification operation implementation
    return 1; // Return 1 on success
}
```

OpenSSL Engine

- The OpenSSL engine is modular framework that enables the integration of various custom cryptographic implementations, hardware security modules, or alternative software algorithms into the OpenSSL library.
- The PKCS11 engine for OpenSSL is a commonly used module engine for integration of secure key storage options with OpenSSL.
- The implementation of custom ECDSA based engine in OpenSSL requires following modifications
    i. Define the Engine Structure
    ii. Implement Engine Methods:
    iii. Integrate with OpenSSL's EVP System
    iv. Compile and Link the Engine:
    v. Test the Engine Integration
    vi. Configure OpenSSL to Use the Engine

```
include <openssl/engine.h>
static const char *engine_oezgan_id = "oezgan";
static const char *engine_oezgan_name = "oezgan engine by Fraunhofer FKIE";

IMPLEMENT_DYNAMIC_CHECK_FN();
IMPLEMENT_DYNAMIC_BIND_FN(bind_helper);

int oezgan_init(ENGINE *e) {
    printf("Oezgan Engine Initializatzion!\n");
    return 786;
}

int bind_helper(ENGINE * e, const char *id)
{
    if (!ENGINE_set_id(e, engine_oezgan_id) ||
            !ENGINE_set_name(e, engine_oezgan_name) ||
            !ENGINE_set_init_function(e,oezgan_init)
    )
        return 0;

    return 1;
}
```

Service Providers to OpenSSL

- OpenSSL3.0 and higher versions hosts the provider API through which custom cryptographic implementations can be implemented.
- . The detailed work flow of service provider library is discussed as follows:
  i. The provider holds a library of cryptographic implementations. The provider.c file is initialized with an identifier, version information and name. The provider capabilities like supported algorithms, data formats, lifecycle of algorithms and error handling will also be coded.
  ii. The custom algorithm must be implemented and attached to calls in the provider.c file. A make file must be written to compile the provider code.
  iii. A test application (testprovider.c) must be written to call the compiled instance of (provider.so) of the provider library inside OpenSSL.
  iv. The OpenSSL command line calls can now be issued to use the provider implementations of cryptographic algorithms.

Recommendations

Consideration of Separate Engines or Service Providers: While using separate engines or service providers with OpenSSL may reduce overhead, it is important to note that the cryptographic algorithms will reside outside the core libcrypto module of OpenSSL.

Potential Security Risks: This external placement of cryptographic modules could introduce security vulnerabilities, making them more susceptible to attacks.

Impact on Applications: Applications such as Hardware Security Modules (HSMs), which interact with OpenSSL through built-in interfaces like PKCS#11, may not be aware of these additional external interfaces due to OpenSSL's abstraction. This lack of awareness could result in compatibility issues or reduced security.

Recommendation for Specific Environments: The engine and service provider approaches are recommended primarily for environments where OpenSSL's cryptographic operations can operate independently without requiring direct application interface integration.

Conclusion and Future Work

- The article highlights the need for custom cryptographic implementations in applications such as permissioned blockchain and Hardware Security Modules (HSMs).
- Embedding a custom cryptographic library directly into OpenSSL is identified as a practical solution for these types of applications.
- OpenSSL version 1 supports the integration of separate cryptographic modules (libcrypto), allowing for significant modifications to the EVP (Envelope) structure for symmetric algorithms and the OSSL structure for asymmetric algorithms.
- This method, while potentially increasing code overhead, enables the addition of cryptographic modules that meet FIPS standards. It also addresses security vulnerabilities present in other approaches and offers improved timing efficiency.
- The choice of integration method and OpenSSL version should be guided by the specific needs of the application, balancing security, performance, and feasibility.

References

1. Mishra, S.P., Attacks & Analysis of OpenSSL AES crypto material from memory dumps.
2. O. Aciicmez and W. Schindler, "A vulnerability in RSA implementations due to instruction cache analysis and its demonstration on OpenSSL" in CT-RSA, San Francisco, CA, US, pp. 256-273, Apr 2008.
3. S. Fan, W. Wang, and Q. Cheng, “Attacking OpenSSL Implementation of ECDSA with a Few Signatures,” Oct. 2016, doi: https://doi.org/10.1145/2976749.2978400.
4. A. Takahashi and Mehdi Tibouchi, “Degenerate Fault Attacks on Elliptic Curve Parameters in OpenSSL,” Jun. 2019, doi: https://doi.org/10.1109/eurosp.2019.00035.
5. “Heartbleed 101 - IEEE Journals & Magazine,” Ieee.org, 2014, doi: https://doi.org/10.1109/MSP.2014.66.
6. J. Lam, S.-G. Lee, H.-J. Lee, and Vincentius Christian Andrianto, “Custom Cryptographic Protocol Implementation Method Based on OpenSSL,” Information Security and Cryptology, vol. 27, no. 3, pp. 459–466, Jan. 2017, doi: https://doi.org/10.13089/jkiisc.2017.27.3.459.
7. https://wiki.OpenSSL.org/index.php/Creating_an_OpenSSL_Engine_to_use_indigenous_ECDH_ECDSA_and_HASH_Algorithms.
8. https://developer.ibm.com/tutorials/awb-quantum-safe-OpenSSL/
9. Jurkiewicz, P. and Niemiec, M. (2016). Implementation of a New Cipher in OpenSSL Environment the Case of INDECT Block Cipher. International Journal of Computer and Communication Engineering, 5(1), pp.41–49. doi:https://doi.org/10.17706/ijcce.2016.5.1.41-49.

THANK YOU