strongSwan 插件添加指南

Revision: 1.00

Document Title	strongSwan 插件添加指南
Version	1.00
Date	2019-01-23
Status	Release
Document Control ID	

目录

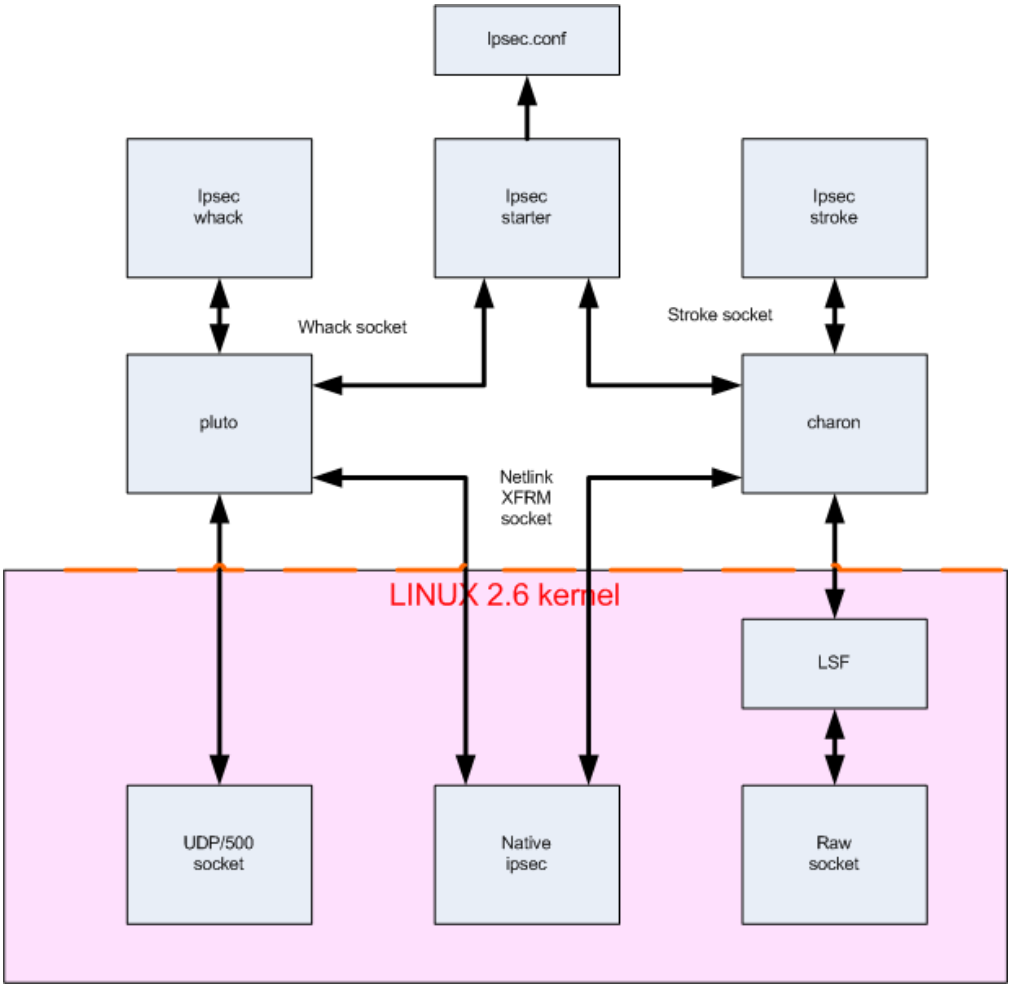
0. Revision history

Revision	Date	Author	Description of change
1.00	2019-01-23	陈皓	第一版

1. strongSwan 简介

strongSwan是一款开源的基于IPSec协议（Internet Protocol Security）的VPN解决方案，可以在Linux、安卓等平台运行，支持众多加密认证算法，市场上使用率很高。strongSwan绝大部分代码由C语言实现，并且使用面向对象的方式实现，有很好的可移植性，可扩展性。

strongSwan整体框架大致如下：



（4.6.3版本的架构图）

strongSwan支持以plugin的方式添加第三方的算法库。在需要使用指定IKE与ESP加密认证方式时，只要有对应的加密认证算法plugin，就可以直接在配置文件中修改关键字实现指定加密算法的使用。

本文档将以国密plugin（GMAlg插件）SM3/SM4为例，为大家介绍如何在strongSwan中添加第三方的plugin。

添加plugin大致需要以下几项工作：

- 在strongSwan配置文件中新增plugin配置
- 在整体架构相应部分新增plugin枚举，mapping
- 编写plugin的Makefile
- 根据相应要添加的plugin功能，实现各算法的接口函数

2. 插件添加说明

2.1. strongSwan 配置文件

2.1.1. 添加新增插件配置文件

- 在conf/Makefile.am 中添加插件的配置文件路径，plugins/gmalg.opt
- 添加插件配置文件，conf/plugins/gmalg.opt

2.1.2. 修改顶级配置文件

在configure.ac配置文件中添加插件的相关配置，新增以下条目：

ARG_ENABL_SET([gmalg], [enables the GMAlg crypto plugin.])

ADD_PLUGIN([gmalg], [s charon scepclient pki scripts manager medsrv attest nm cmd aikgen])

AM_CONDITIONAL(USE_GMALG, test x$gmalg = xtrue)

AC_CONFIG_FILES([

...

src/libstrongswan/plugins/gmalg/Makefile

...

])

2.1.3. 修改 libstrongswan 模块 Makefile

由于plugin属于libstrongswan子模块，需要在src/libstrongswan/Makefile.am添加plugin信息。

if USE_GMALG

SUBDIRS += plugins/gmalg

if MONOLITHIC

libstrongswan_la_LIBADD += plugins/gmalg/libstrongswan-gmalg.la

endif

endif

2.2. strongSwan 代码修改

添加新的plugin时，需要在众多文件中新增枚举类型，mapping关系等，以国密plugin为例，按文件分类需做出以下修改（其中标记*部分为需要参照RFC或国内标准）：

2.2.1. 代码修改详细说明

proposal_substructure.c

此文件主要实现了IPsec proposal子结构组帧时，选取对应加密/哈希算法的标识ID。

*ikev1_encryption_t添加枚举，表示IKE v1协议加密算法ID

IKEV1_ENCR_SM1_CBC = 128,

IKEV1_ENCR_SM4_CBC = 129,

*ikev1_hash_t添加枚举，表示IKE v1协议哈希算法ID

IKEV1_HASH_SM3 = 20,

*ikev1_esp_transid_t添加枚举，表示ESP协议加密算法ID

IKEV1_ESP_ENCR_SM4 = 127,

IKEV1_ESP_ENCR_SM1 = 128,

*ikev1_ah_transid_t添加枚举，表示IKE v1协议AH HMAC算法ID

IKEV1_AH_HMAC_SM3 = 20,

*ikev1_auth_algo_t添加枚举，表示IKE v1协议认证算法ID

IKEV1_AUTH_HMAC_SM3 = 20,

Encryption algorithm mapping--->map_encr添加加解密算法对应关系

{ IKEV1_ENCR_SM4_CBC, ENCR_SM4_CBC},

Integrity algorithm mapping--->map_integ添加完整性验证算法对应关系

{ IKEV1_HASH_SM3, AUTH_HMAC_SM3},

ESP encryption algorithm mapping--->map_esp添加ESP阶段加解密算法对应关系

{ IKEV1_ESP_ENCR_SM4, ENCR_SM4_CBC},

AH authentication algorithm mapping--->map_ah添加IKE阶段AH算法对应关系

{ IKEV1_AH_HMAC_SM3, AUTH_HMAC_SM3},

kernel_netlink_ipsec.c

此文件包含了IPSec通过Netlink与kernel进行交互时的实现。

Algorithms for encryption--->encryption_algs添加字符串类型的加密算法名称，需要与kernel中crypto子模块中对应加解密算法名称保持一致。

{ENCR_SM4_CBC, "cbc(sm4)"},

Algorithms for integrity protection--->integrity_algs添加字符串类型的哈希算法名称，需要与kernel中crypto子模块中对应哈希算法名称保持一致。

{AUTH_HMAC_SM3, "hmac(sm3)"},

keymat_v1.c

此文件定义了一个用于继承其他加密/哈希算法类，管理加解密算法的一个单例对象

在auth_to_prf函数中添加转换关系：

AUTH_HMAC_SM3--->PRF_HMAC_SM3

```
/**
 * Converts integrity algorithm to PRF algorithm
 */
static uint16_t auth_to_prf(uint16_t alg)
{
    switch (alg)
    {
        case AUTH_HMAC_SM3:
            return PRF_HMAC_SM3;
        case AUTH_HMAC_SHA1_96:
            return PRF_HMAC_SHA1;
        case AUTH_HMAC_SHA2_256_128:
            return PRF_HMAC_SHA2_256;
        case AUTH_HMAC_SHA2_384_192:
            return PRF_HMAC_SHA2_384;
        case AUTH_HMAC_SHA2_512_256:
            return PRF_HMAC_SHA2_512;
        case AUTH_HMAC_MD5_96:
            return PRF_HMAC_MD5;
        case AUTH_AES_XCBC_96:
            return PRF_AES128_XCBC;
        default:
            return PRF_UNDEFINED;
    }
```

在auth_to_hash函数中添加转换关系：

AUTH_HMAC_SM3--->HASH_SM3

```
/**
 * Converts integrity algorithm to hash algorithm
 */
static uint16_t auth_to_hash(uint16_t alg)
{
    switch (alg)
    {
        case AUTH_HMAC_SM3:
            return HASH_SM3;
        case AUTH_HMAC_SHA1_96:
            return HASH_SHA1;
        case AUTH_HMAC_SHA2_256_128:
            return HASH_SHA256;
        case AUTH_HMAC_SHA2_384_192:
            return HASH_SHA384;
        case AUTH_HMAC_SHA2_512_256:
            return HASH_SHA512;
        case AUTH_HMAC_MD5_96:
            return HASH_MD5;
        default:
            return HASH_UNKNOWN;
    }
} « end auth_to_hash »
```

keymat.c

此文件用于构造keymat对象，构造时针对不同的算法类型，未在”*proposal_keywords_static.txt*”文件中写名密钥长度的，需要在此处新建时赋予正确的密钥长度值，单位为bit。

keymat_get_keylen_encr函数中添加加密算法与密钥长度（bit）对应关系

{ENCR_SM4_CBC, 128},

keymat_get_keylen_integ函数中添加完整性验证算法密钥长度（bit）对应关系

{AUTH_HMAC_SM3, 256},

oid.txt

根据标准文档，添加各算法的OID（Object Identifier，对象标识符又称为物联网域名）

crypter.c

此文件定义了加密算法宏与字符串名称的转换，OID与ID直接的转换。

添加新增加密算法的枚举对应的字符串名

ENUM_NEXT(encryption_algorithm_names, ENCR_SM4_CBC, ENCR_SM1_ECB, ENCR_CHACHA20_POLY1305,

"SM4_CBC",

"SM1_CBC",

"SM4_ECB",

"SM1_ECB");

encryption_algorithm_from_oid函数中添加对应关系：OID--->算法枚举/密钥长度

```
encryption_algorithm_t encryption_algorithm_from_oid(int oid, size_t *key_size)
{
    encryption_algorithm_t alg;
    size_t alg_key_size;

    switch (oid)
    {
        case OID_SM1_ECB:
            alg = ENCR_SM1_ECB;
            alg_key_size = 16 ;
            break;
        case OID_SM1_CBC:
            alg = ENCR_SM1_CBC;
            alg_key_size = 16 ;
            break;
        case OID_SM4_ECB:
            alg = ENCR_SM4_ECB;
            alg_key_size = 16 ;
            break;
        case OID_SM4_CBC:
            alg = ENCR_SM4_CBC;
            alg_key_size = 16 ;
            break;
```

encryption_algorithm_to_oid函数中添加对应关系：算法枚举/密钥长度--->OID

```
int encryption_algorithm_to_oid(encryption_algorithm_t alg, size_t key_size)
{
    int oid;

    switch(alg)
    {
        case ENCR_SM1_ECB:
            oid = OID_SM1_ECB;
            break;
        case ENCR_SM1_CBC:
            oid = OID_SM1_CBC;
            break;
        case ENCR_SM4_ECB:
            oid = OID_SM4_ECB;
            break;
        case ENCR_SM4_CBC:
            oid = OID_SM4_CBC;
            break;
```

crypter.h

此文件定义了IKE v2中的加密算法类型。

encryption_algorithm_t中添加加密算法枚举

ENCR_SM4_CBC = 127,

ENCR_SM1_CBC = 128,

ENCR_SM4_ECB = 129,

ENCR_SM1_ECB = 130,

hasher.c

此文件定义了哈希算法宏与字符串名称的转换，OID与ID直接的转换。

添加新增HASH算法的枚举对应的字符串名

ENUM_NEXT(hash_algorithm_names, HASH_SM3, HASH_SM3, HASH_IDENTITY,
"HASH_SM3");

以下函数中分别参考其他HASH算法，添加新增HASH的相应实现

hasher_hash_size

```
size_t hasher_hash_size(hash_algorithm_t alg)
{
    switch (alg)
    {
        case HASH_SM3:
            return HASH_SIZE_SM3;
```

hasher_algorithm_from_oid

```
        case OID_SM3:
        case OID_SM2_WITH_SM3:
            return HASH_SM3;
        default:
            return HASH_UNKNOWN;
    } « end switch oid »
} « end hasher_algorithm_from_oid »
```

hasher_algorithm_from_prf

```
hash_algorithm_t hasher_algorithm_from_prf(pseudo_random_function_t alg)
{
    switch (alg)
    {
        case PRF_HMAC_SM3:
            return HASH_SM3;
```

hasher_algorithm_from_integrity

```
hash_algorithm_t hasher_algorithm_from_integrity(integrity_algorithm_t alg,
                                                 size_t *length)
{
    if (length)
    {
        switch (alg)
        {
            case AUTH_HMAC_MD5_96:
            case AUTH_HMAC_SHA1_96:
            case AUTH_HMAC_SHA2_256_96:
             /* not found in GMT, analyse the DUMP data to get the size,
              * maybe it'll change in anytime when there's a new GMT
              */
            case AUTH_HMAC_SM3:
                *length = 12;
                break;
```

hasher_algorithm_to_integrity

```
integrity_algorithm_t hasher_algorithm_to_integrity(hash_algorithm_t alg,
                                                    size_t length)
{
    switch (alg)
    {
        case HASH_SM3:
            return AUTH_HMAC_SM3;
```

hasher_algorithm_for_ikev2

```
bool hasher_algorithm_for_ikev2(hash_algorithm_t alg)
{
    switch (alg)
    {
        case HASH_IDENTITY:
        case HASH_SHA256:
        case HASH_SHA384:
        case HASH_SHA512:
        case HASH_SM3:
            return TRUE;
```

hasher_algorithm_to_oid

```
int hasher_algorithm_to_oid(hash_algorithm_t alg)
{
    int oid;

    switch (alg)
    {
        case HASH_SM3:
            oid = OID_SM3;
            break;
```

hasher.h

此文件定义了IKE v2中的哈希算法类型。

hash_algorithm_t中添加HASH算法枚举

HASH_SM3 = 20,

iv_gen.c

此文件用于返回指定加密算法的随机向量IV产生函数。

iv_gen_create_for_alg函数中添加新增算法对应的IV产生方式

```
iv_gen_t* iv_gen_create_for_alg(encryption_algorithm_t alg)
{
    switch (alg)
    {
        case ENCR_SM1_ECB:
        case ENCR_SM1_CBC:
        case ENCR_SM4_ECB:
        case ENCR_SM4_CBC:
        case ENCR_DES:
        case ENCR_3DES:
        case ENCR_RC5:
        case ENCR_IDEA:
        case ENCR_CAST:
        case ENCR_BLOWFISH:
        case ENCR_3IDEA:
        case ENCR_AES_CBC:
        case ENCR_CAMELLIA_CBC:
        case ENCR_SERPENT_CBC:
        case ENCR_TWOFISH_CBC:
        case ENCR_RC2_CBC:
            return iv_gen_rand_create();
```

prf.c

此文件用于定义HMAC对应的伪随机函数（PRF）

添加新增算法枚举对应的字符串名

ENUM_NEXT(pseudo_random_function_names, PRF_HMAC_SM3, PRF_HMAC_SM3, PRF_AES128_CMAC,

"PRF_HMAC_SM3");

prf.h

此文件用于定义伪随机函数（PRF）ID

pseudo_random_function_t中添加新增类型

PRF_HMAC_SM3 = 20,

proposal.c

此文件定义了proposal类，用于在IPSec工作时构造proposal对象

integ_prf_map中添加HASH与PRF对应关系

{AUTH_HMAC_SM3, PRF_HMAC_SM3},

proposal_keywords_static.txt

此文件定义了所有strongSwan的关键字，用于在配置文件中匹配选取加密/哈希等算法类型。

添加对应算法关键字，用于在配置文件中匹配关键字至对应的算法

sm4cbc, ENCRYPTION_ALGORITHM, ENCR_SM4_CBC, 0

sm3, INTEGRITY_ALGORITHM, AUTH_HMAC_SM3, 0

prfsm3, PSEUDO_RANDOM_FUNCTION, PRF_HMAC_SM3, 0

signer.c

此文件用于定义完整性验证算法枚举与字符串名称匹配。

添加新增算法枚举对应的字符串名

ENUM_NEXT(integrity_algorithm_names, AUTH_HMAC_SM3, AUTH_HMAC_SM3, AUTH_HMAC_SHA2_512_256,

"HMAC_SM3");

signer.h

此文件用于定义完整性验证算法枚举类型。

integrity_algorithm_t中添加新增类型

AUTH_HMAC_SM3 = 20,

hmac.c

此文件定义了hamc类。

hmac_create添加HASH_SM3对应的指定block size

```
static mac_t *hmac_create(hash_algorithm_t hash_algorithm)
{
    private_mac_t *this;

    INIT(this,
        .public = {
            .get_mac = _get_mac,
            .get_mac_size = _get_mac_size,
            .set_key = _set_key,
            .destroy = _destroy,
        },
    );

    /* set b, according to hasher */
    switch (hash_algorithm)
    {
        case HASH_SM3:
        case HASH_SHA1:
        case HASH_MD5:
        case HASH_SHA256:
            this->b = 64;
            break;
```

2.2.2. strongSwan 代码修改文件列表

strongSwan原生代码中需要修改的文件列表如下：

proposal_substructure.c

kernel_netlink_ipsec.c

keymat_v1.c

keymat.c

oid.txt

crypter.c
crypter.h
hasher.c
hasher.h
iv_gen.c
prf.c
prf.h
proposal.c
proposal_keywords_static.txt
signer.c
signer.h
hmac.c

2.3. 新增插件实现

2.3.1. 新增插件 Makefile

可以参考其他插件的Makefile进行新增

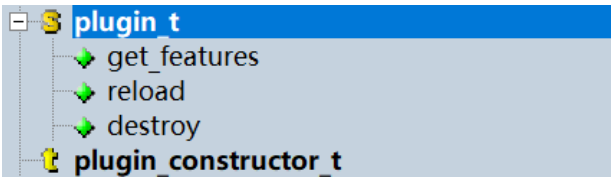
```
AM_CPPFLAGS = \
    -I$(top_srcdir)/src/libstrongswan \
    -I$(top_srcdir)/src/libstrongswan/plugins/gmalg/gmalg/include \
    -DGMALG_INTERIOR=${gmalg_interior}

AM_CFLAGS = \
    $(PLUGIN_CFLAGS)

if MONOLITHIC
noinst_LTLIBRARIES = libstrongswan-gmalg.la
else
plugin_LTLIBRARIES = libstrongswan-gmalg.la
endif

libstrongswan_gmalg_la_SOURCES = \
    gmalg_plugin.h gmalg_plugin.c \
    gmalg_crypter.h gmalg_crypter.c \
    gmalg_hasher.h gmalg_hasher.c \
    gmalg_rng.h gmalg_rng.c \
    gmalg_ec_diffie_hellman.h gmalg_ec_diffie_hellman.c\
    gmalg_ec_private_key.h gmalg_ec_private_key.c \
    gmalg_ec_public_key.h gmalg_ec_public_key.c \
    gmalg_ec_util.h gmalg_ec_util.c \
    gmalg/gmalg.c \
    gmalg/sm3.c \
    gmalg/sm4.c \
    gmalg/sm2.c \
    gmalg/gmalg_ecc.c \
    gmalg/gmalg_random.c \
    gmalg/gmalg_debug.c

libstrongswan_gmalg_la_LDFLAGS = -module -avoid-version
libstrongswan_gmalg_la_LIBADD  = $(GMALG_LIB)
```

2.3.2. 实现 strongSwan 定义的 plugin 接口

strongSwan 广泛使用了宏 METHOD，用于定义一个函数，此宏定义位于 object.h

```
/**
 * Method declaration/definition macro, providing private and public interface.
 *
 * Defines a method name with this as first parameter and a return value ret,
 * and an alias for this method with a _ prefix, having the this argument
 * safely casted to the public interface iface.
 * _name is provided a function pointer, but will get optimized out by GCC.
 */
#define METHOD(iface, name, ret, this, ...) \
    static ret name(union {iface *_public; this;} \
    __attribute__((transparent_union)), ##__VA_ARGS__); \
    static typeof(name) *_##name = (typeof(name)*)name; \
    static ret name(this, ##__VA_ARGS__)
```

strongSwan 中 plugin 需要实现系统定义的 plugin_t 接口：



以新增国密插件为例，实现了以下接口：

get_name：用于返回插件名

```
METHOD(plugin_t, get_name, char*,
    private_gmalg_plugin_t *this)
{
    return "gmalg";
}
```

get_features：用于在创建时加载plugin所支持的特性

```
METHOD(plugin_t, get_features, int,
    private_gmalg_plugin_t *this, plugin_feature_t *features[])
{
    static plugin_feature_t f[] = {
        /* we provide GmSdf threading callbacks */
        PLUGIN_PROVIDE(CUSTOM, "gmalg-threading"),
        /* crypters */
        PLUGIN_REGISTER(CRYPTER, gmalg_crypter_create),
            PLUGIN_PROVIDE(CRYPTER, ENCR_SM1_ECB, 16),
            PLUGIN_PROVIDE(CRYPTER, ENCR_SM1_CBC, 16),
            PLUGIN_PROVIDE(CRYPTER, ENCR_SM4_ECB, 16),
            PLUGIN_PROVIDE(CRYPTER, ENCR_SM4_CBC, 16),
            /* chen hao_ipsec with GM 2019-1-17, begin */
            //PLUGIN_PROVIDE(CRYPTER, ENCR_NULL, 0),
            /* chen hao_ipsec with GM 2019-1-17, end */
        /* hashers */
        PLUGIN_REGISTER(HASHER, gmalg_hasher_create),
            PLUGIN_PROVIDE(HASHER, HASH_SM3),
```

destroy：对象销毁时释放内存

```
METHOD(plugin_t, destroy, void,
    private_gmalg_plugin_t *this)
{
    free(this);
}
```

2.3.3. 实现 plugin 中加密算法接口

插件接口定义中，在get_features接口中，需要指定features。以国密插件为例，定义了加密算法SM4，并指定使用gmalg_crypter_create进行加密算法对象的构造。

```
gmalg_crypter_t *gmalg_crypter_create(encryption_algorithm_t algo,
                                      size_t key_size)
{
    private_gmalg_crypter_t *this;

    INIT(this,
        .public = {
            .crypter = {
                .encrypt = _encrypt,
                .decrypt = _decrypt,
                .get_block_size = _get_block_size,
                .get_iv_size = _get_iv_size,
                .get_key_size = _get_key_size,
                .set_key = _set_key,
                .destroy = _destroy,
            },
        },
    );
```

```
    /* chen hao_ipsec with GM 2019-1-17, begin */
    switch (algo)
    {
        default:
        {
            DBG0(DBG_LIB, "Unknown alg[%d]!!!", algo);
            return NULL;
        }
        case ENCR_SM1_ECB:
        case ENCR_SM1_CBC:
        {
            DBG0(DBG_LIB, "SM1 NOT IMPLEMENTED!!!");
            return NULL;
        }
        case ENCR_SM4_ECB:
        case ENCR_SM4_CBC:
        {
            break;
        }
    }
```

```
    if (key_size == 0)
    {
        DBG0(DBG_LIB, "Key size is wrong, correct it here to 128");
        //should fix this maybe in other place
        //16 bytes, 128 bits
        key_size = GMALG_SM_KEY_SIZE;
    }
    /* chen hao_ipsec with GM 2019-1-17, end */

    this->algo = algo;
    this->key = chunk_alloc(key_size);
    GMALG_OpenDevice(&this->hDeviceHandle);

    return &this->public;
} « end gmalg_crypter_create »
```

加密算法需要实现strongSwan系统下crypter_t类型的接口：

加密函数encrypt：

```
METHOD(crypter_t, encrypt, bool,
    private_gmalg_crypter_t *this, chunk_t data, chunk_t iv, chunk_t *dst)
{
    return crypt(this, data, iv, dst, 1);
}
```

解密函数decrypt：

```
METHOD(crypter_t, decrypt, bool,
    private_gmalg_crypter_t *this, chunk_t data, chunk_t iv, chunk_t *dst)
{
    return crypt(this, data, iv, dst, 0);
}
```

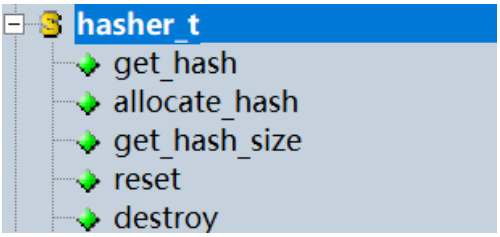
其余接口不一一列举，只需将以上接口都实现，就可以完成plugin中加密算法feature的实现。

2.3.4. 实现 plugin 中哈希算法接口

插件接口定义中，在get_features接口中，需要指定features。以国密插件为例，定义了哈希算法SM3，并指定使用gmalg_hasher_create进行哈希算法对象的构造。

```
gmalg_hasher_t *gmalg_hasher_create(hash_algorithm_t algo)
{
    private_gmalg_hasher_t *this;

    INIT(this,
        .public = {
            .hasher = {
                .get_hash = _get_hash,
                .allocate_hash = _allocate_hash,
                .get_hash_size = _get_hash_size,
                .reset = _reset,
                .destroy = _destroy,
            },
        },
    );

    this->algo = algo;
    GMALG_OpenDevice(&this->hDeviceHandle);
    GMALG_HashInit(this->hDeviceHandle, NULL, NULL, 0);

    return &this->public;
} « end gmalg_hasher_create »
```

哈希算法需要实现strongSwan系统下hasher_t类型的接口：



获取哈希值的函数get_hash：

```
METHOD(hasher_t, get_hash, bool,
    private_gmalg_hasher_t *this, chunk_t chunk, uint8_t *hash)
{
    GMALG_HashUpdate(this->hDeviceHandle, chunk.ptr, chunk.len);

    if (hash)
    {
        u_int len;
        GMALG_HashFinal(this->hDeviceHandle, hash, &len);
        GMALG_HashInit(this->hDeviceHandle, NULL, NULL, 0);
    }
    return TRUE;
}
```

其余接口不一一列举，只需将以上接口都实现，就可以完成plugin中哈希算法feature的实现。

2.3.5. 实现 plugin 中其他算法接口

除了加密算法，哈希算法外，根据IPSec协议，还需实现一些其他接口，但是实现方式与以上章节中描述内容类似。可能需要的features对应的接口：

- 随机数产生 → rng_t

- 密钥交互算法 → diffie_hellman_t
- 私钥相关算法 → private_key_t
- 公钥相关算法 → public_key_t