芯来科技
NUCLEI

安全加解密算子RVK性能优化

舒卓
Nuclei Technology

Contents:

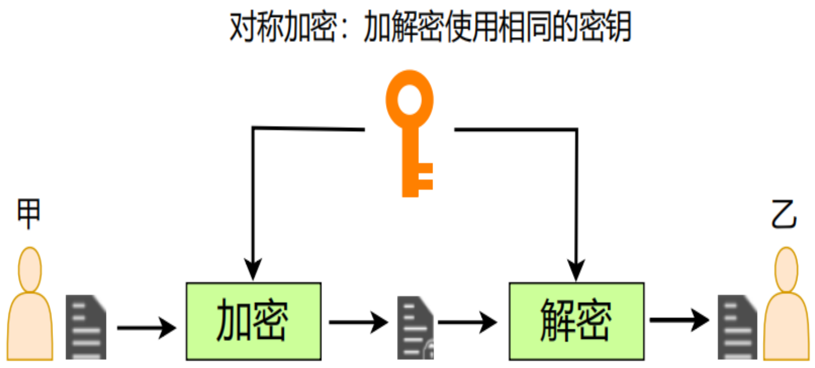
- 安全加解密算法简介
- RISC-V Crypto 扩展（Scalar K 与 Vector K）
- 使用 K 扩展加速安全加解密算法
- 在 Nuclei Evalsoc 上实测的提升效果

安全加解密算法简介

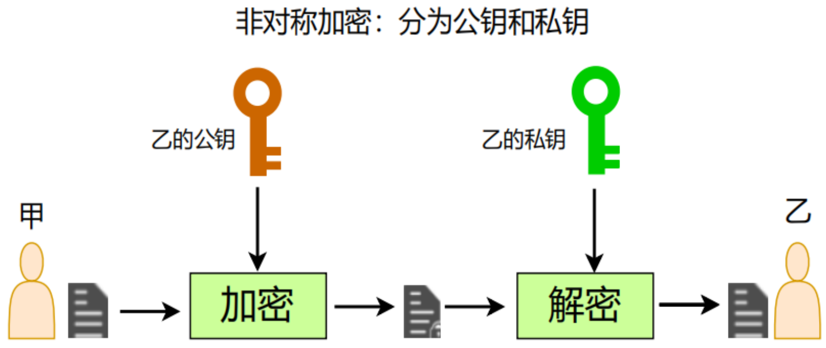
芯来科技 NUCLEI

安全加解密算法 主要分为三大类：对称加解密、非对称加解密与哈希函数

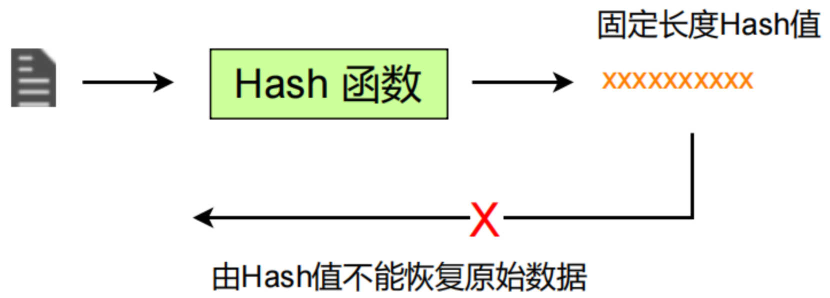
- 对称加解密（DES/3DES/AES/SM4 等）



- 非对称加解密（RSA/ECC/SM2 等）



- 哈希函数（MD5/SHA-1/SHA-2/SM3 等）



RISC-V Crypto 扩展

RISC-V Crypto 扩展包括 Scalar K 与 Vector K 扩展，相比于纯软件实现，使用 K 扩展指令可以提升加解密算法的速度，并降低应用程序的Code size

- **Scalar K 扩展**，包含 zbk(位操作)、zkn(NIST)、zks(商密) 等子扩展
  - ✓ 优点：基于通用寄存器，实现开销较小，适用于小核
  - ✓ 缺点：性能提升有限，比 Vector K 差
- **Vector K 扩展**，包含 zvkb(位操作)、zvkn(NIST)、zvks(商密) 等子扩展
  - ✓ 优点：基于 RVV 扩展实现，性能较高，适用于大核
  - ✓ 缺点：Vector K 扩展依赖 RVV 扩展（即VPU），面积开销较大

RISC-V Crypto 扩展-支持的加解密算法

芯来科技
NUCLEI

RISC-V Crypto 扩展支持以下算法，以及不同的模式：

- AES-128/192/256 GCM/CCM/ECB/OCB/OFB/CFB/CFB1/CFB8/CTR
- AES-128/256-XTS
- SHA-224/SHA-256
- SHA-384/SHA-512
- SM3
- SM4 CBC/ECB/CCM/CFB/CFB1/CFB8/CTR/GCM/OFB/XTS

RISC-V Crypto 不支持以下算法，原因是这些算法不再安全：

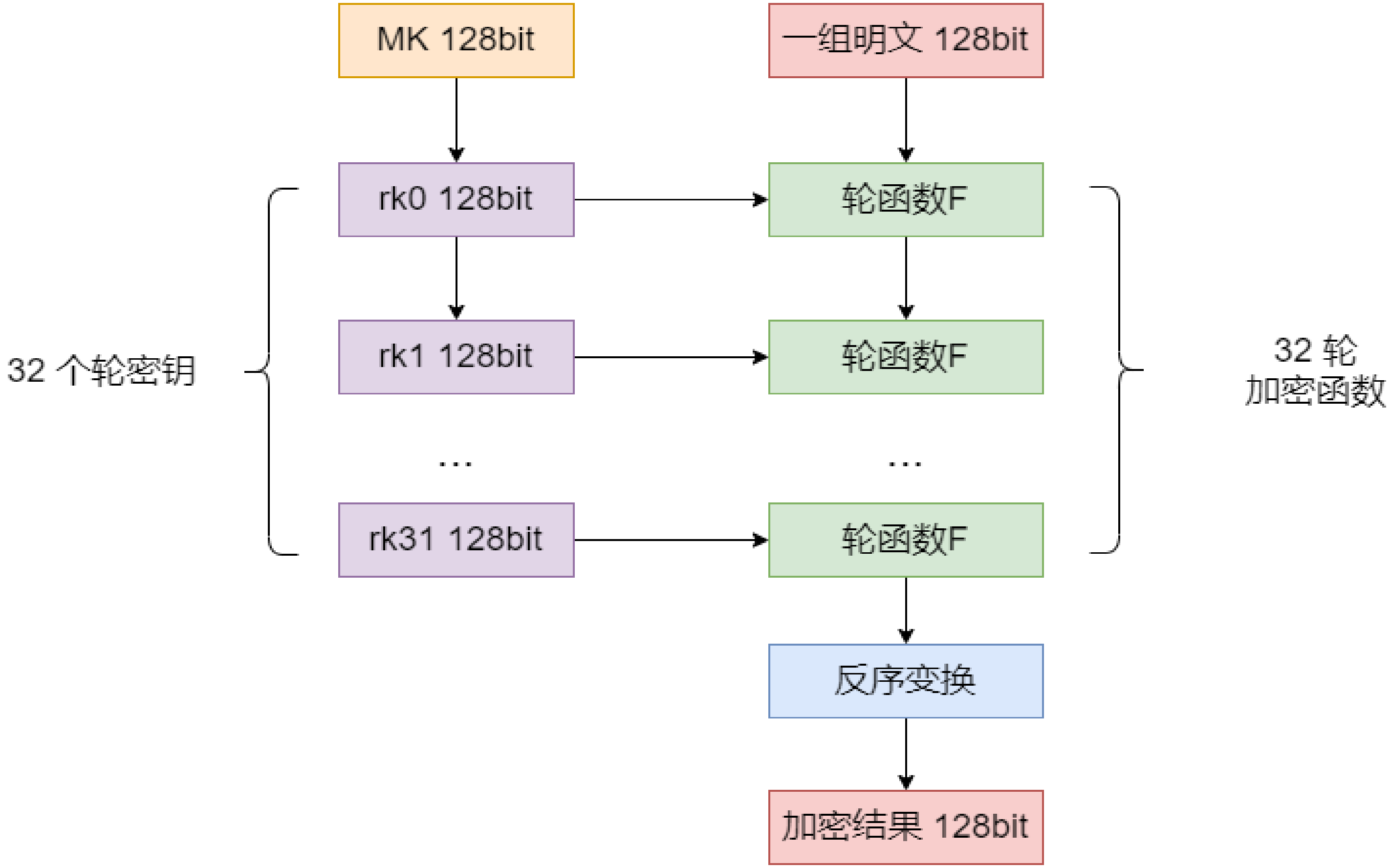
- DES/3DES
- MD5
- SHA-1

使用 K 扩展加速安全加密算法-SM4

芯来科技 NUCLEI

以 SM4 算子为例来展示使用K扩展优化的流程

SM4 加解密算法原理如图：



使用 K 扩展加速安全加密算法-Scalar K

芯来科技 NUCLEI

使用 Scalar K 扩展加速 SM4-密钥扩展部分：

```c
#define ROL32(X,Y) ((X<<Y) | (X >> (32-Y)))

#define TAU(A) (((uint32_t)SBOX[(A >> 24) & 0xFF] << 24) | \
               ((uint32_t)SBOX[(A >> 16) & 0xFF] << 16) | \
               ((uint32_t)SBOX[(A >>  8) & 0xFF] <<  8) | \
               ((uint32_t)SBOX[(A >>  0) & 0xFF] <<  0) ) \

static inline uint32_t Tp(uint32_t X) {
    uint32_t x1 = TAU(X);
    uint32_t x2 = x1 ^ ROL32(x1,13) ^ ROL32(x1,23);
    return   x2;
}

// 某一轮秘钥扩展ref 实现(代码有删减)
{
    ...
    t  = K1 ^ K2 ^ K3 ^ ckp[0];
    K0 = K0 ^ Tp(t);

    t = K2 ^ K3 ^ K0 ^ ckp[1];
    K1 = K1 ^ Tp(t);

    t = K3 ^ K0 ^ K1 ^ ckp[2];
    K2 = K2 ^ Tp(t);

    t = K0 ^ K1 ^ K2 ^ ckp[3]
    K3 = K3 ^ Tp(t);
    ...
}
```

```c
static inline unsigned long ssm4_ks4(unsigned long rs1, unsigned long rs2) {
    rs1 = _sm4ks(rs1, rs2, 0);
    rs1 = _sm4ks(rs1, rs2, 1);
    rs1 = _sm4ks(rs1, rs2, 2);
    rs1 = _sm4ks(rs1, rs2, 3);
    return rs1;
}

// 某一轮秘钥扩展(使用 scalar K优化)
{
    ...
    t  = K1 ^ K2 ^ K3 ^ ckp[0];
    K0 = ssm4_ks4(K0, t);

    t  = K2 ^ K3 ^ K0 ^ ckp[1];
    K1 = ssm4_ks4(K1, t);

    t  = K3 ^ K0 ^ K1 ^ ckp[2];
    K2 = ssm4_ks4(K2, t);

    t  = K0 ^ K1 ^ K2 ^ ckp[3];
    K3 = ssm4_ks4(K3, t);
    ...
}
```

Zks 子扩展定义的 sm4ks 指令

使用 K 扩展加速安全加密算法-Scalar K

芯来科技 NUCLEI

使用 Scalar K 扩展加速 SM4-轮函数迭代加密部分：

```c
#define ROL32(X,Y) ((X<<Y) | (X >> (32-Y)))

#define TAU(A) (((uint32_t)SBOX[(A >> 24) & 0xFF] << 24) | \
                ((uint32_t)SBOX[(A >> 16) & 0xFF] << 16) | \
                ((uint32_t)SBOX[(A >>  8) & 0xFF] <<  8) | \
                ((uint32_t)SBOX[(A >>  0) & 0xFF] <<  0) ) \

static inline uint32_t T(uint32_t X) {
    uint32_t x1 = TAU(X);
    uint32_t x2 = x1 ^ ROL32(x1, 2) ^ ROL32(x1,10) ^
                       ROL32(x1,18) ^ ROL32(x1,24) ;
    return   x2;
}

// 某一轮加密 ref 实现 (代码有删减)
{
     ...
    t = X1 ^ X2 ^ X3 ^ rkp[0];
    X0   = X0 ^ T(t);

    t = X2 ^ X3 ^ X0 ^ rkp[1];
    X1   = X1 ^ T(t);

    t = X3 ^ X0 ^ X1 ^ rkp[2];
    X2   = X2 ^ T(t);

    t = X0 ^ X1 ^ X2 ^ rkp[3];
    X3   = X3 ^ T(t);
    ...
}
```

```c
static inline unsigned long ssm4_ed4(unsigned long rs1, unsigned long rs2) {
    rs1 = _sm4ed(rs1, rs2, 0);
    rs1 = _sm4ed(rs1, rs2, 1);
    rs1 = _sm4ed(rs1, rs2, 2);
    rs1 = _sm4ed(rs1, rs2, 3);
    return rs1;
}

// 某一轮加密使用 scalar K优化 (代码有删减)
{
    ...
    t  = X1 ^ X2 ^ X3 ^ rkp[0];
    X0 = ssm4_ed4(X0, t);

    t  = X2 ^ X3 ^ X0 ^ rkp[1];
    X1 = ssm4_ed4(X1, t);

    t  = X3 ^ X0 ^ X1 ^ rkp[2];
    X2 = ssm4_ed4(X2, t);

    t  = X0 ^ X1 ^ X2 ^ rkp[3];
    X3 = ssm4_ed4(X3, t);
    ...
}
```

Zksed 子扩展定义的 sm4ed 指令

使用 K 扩展加速安全加密算法-Vector K

使用 Vector K 扩展加速 SM4算法：

```
// void sm4_expandkey_zksed_zvkb(const u8 user_key[16], u32 rkey_enc[32],
//                  u32 rkey_dec[32]);
SYM_FUNC_START(sm4_expandkey_zvksed_zvkb)
    vsetivli    zero, 4, e32, m1, ta, ma

    // Load the user key.
    vle32.v     v1, (a0)
    vrev8.v     v1, v1

    // XOR the user key with the family key.
    la      t0, FAMILY_KEY
    vle32.v     v2, (t0)
    vxor.vv     v1, v1, v2

    // Compute the round keys.  Store them in forwards order in rkey_enc
    // and in reverse order in rkey_dec.
    addi        a2, a2, 31*4
    li      t0, -4
    .set        i, 0
.rept 8
    vsm4k.vi    v1, v1, i
    vse32.v     v1, (a1)    // Store to rkey_enc.
    vsse32.v    v1, (a2), t0    // Store to rkey_dec.
.if i < 7
    addi        a1, a1, 16
    addi        a2, a2, -16
.endif
    .set        i, i + 1
.endr

    ret
SYM_FUNC_END(sm4_expandkey_zvksed_zvkb)
```

```
// void sm4_crypt_zvksed_zvkb(const u32 rkey[32], const u8 in[16], u8 out[16]);
SYM_FUNC_START(sm4_crypt_zvksed_zvkb)
    vsetivli    zero, 4, e32, m1, ta, ma

    // Load the input data.
    vle32.v     v1, (a1)
    vrev8.v     v1, v1

    // Do the 32 rounds of SM4, 4 at a time.
    .set        i, 0
.rept 8
    vle32.v     v2, (a0)
    vsm4r.vs    v1, v2
.if i < 7
    addi        a0, a0, 16
.endif
    .set        i, i + 1
.endr

    // Store the output data (in reverse element order).
    vrev8.v     v1, v1
    li      t0, -4
    addi        a2, a2, 12
    vsse32.v    v1, (a2), t0

    ret
SYM_FUNC_END(sm4_crypt_zvksed_zvkb)
```

使用 K 扩展加速安全加密算法-3种使用方式

芯来科技
NUCLEI

Nuclei 目前提供 3种 方式来使用 K 扩展:

- MbedTLS

  Nuclei 基于 mbedtls 适配，裸机，适用于小型MCU

  链接地址：https://github.com/Nuclei-Software/mbedtls/tree/nuclei/v3.3.0/accelerator

- Linux Crypto框架

  Linux 6.9 主线已经支持 Vector K(目前不支持scalar K)， 运行在 kernel space

  链接地址：https://github.com/torvalds/linux/tree/v6.9-rc1/arch/riscv/crypto

- OpenSSL

  OpenSSL 主线已经支持 Vector K(目前不支持scalar K) ，运行在 user space

  链接地址：https://github.com/openssl/openssl

在Nuclei Evalsoc上实测的提升效果(MbedTLS)

芯来科技 NUCLEI

基于MbedTLS 测试的数据，单位是处理1字节所用的 cycle 数：

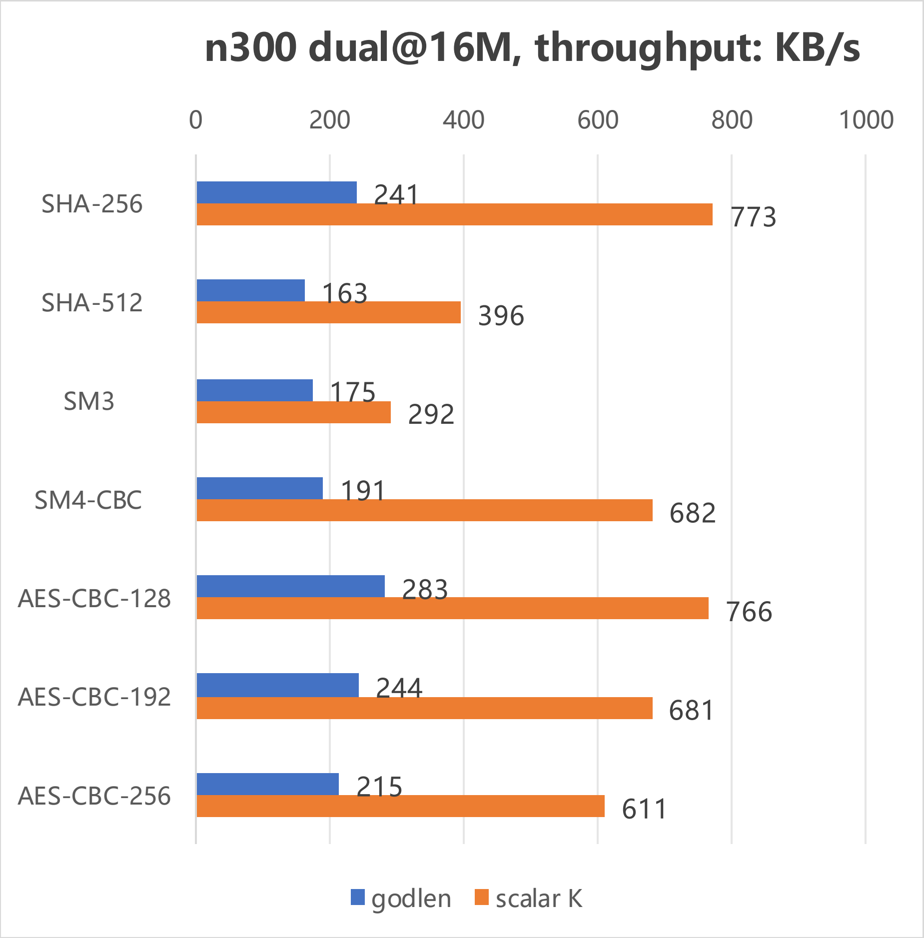
算子	scalar K (n300 dual)	scalar K (ux900)	vector K (ux900)
SHA-256	20 cycles/byte	27 cycles/byte	1 cycles/byte
SHA-512	39 cycles/byte	9 cycles/byte	2 cycles/byte
SM3	53 cycles/byte	42 cycles/byte	4 cycles/byte
AES-128-CBC	20 cycles/byte	5 cycles/byte	1 cycles/byte
AES-256-CBC	25 cycles/byte	7 cycles/byte	2 cycles/byte
SM4-CBC	22 cycles/byte	29 cycles/byte	5 cycles/byte

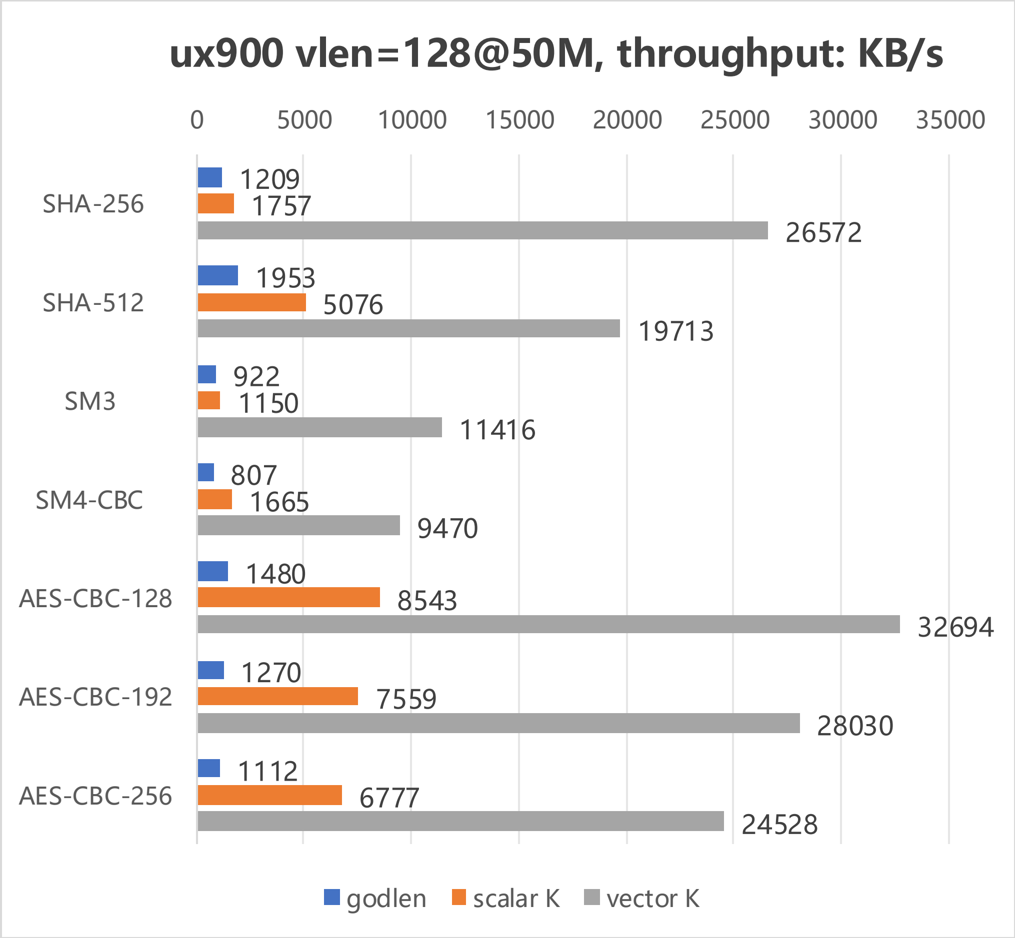
Note: ux900 vlen=128 dlen=128 @50M

在Nuclei Evalsoc上实测的提升效果(MbedTLS)

芯来科技 NUCLEI

基于MbedTLS 测试的吞吐率：

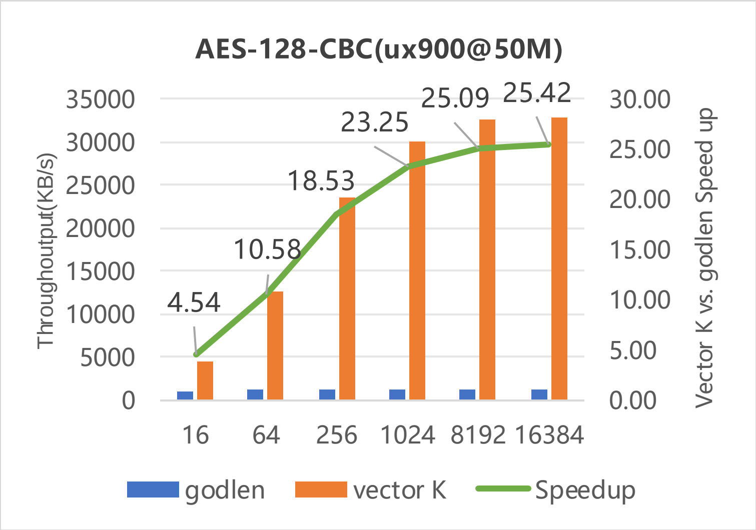


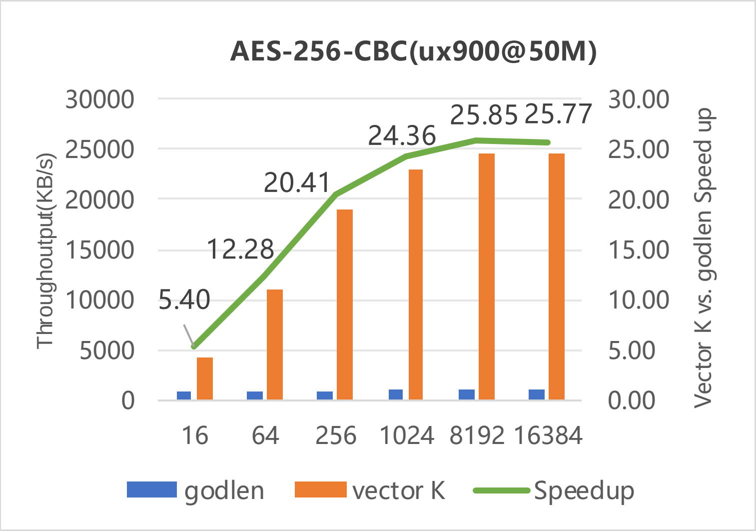


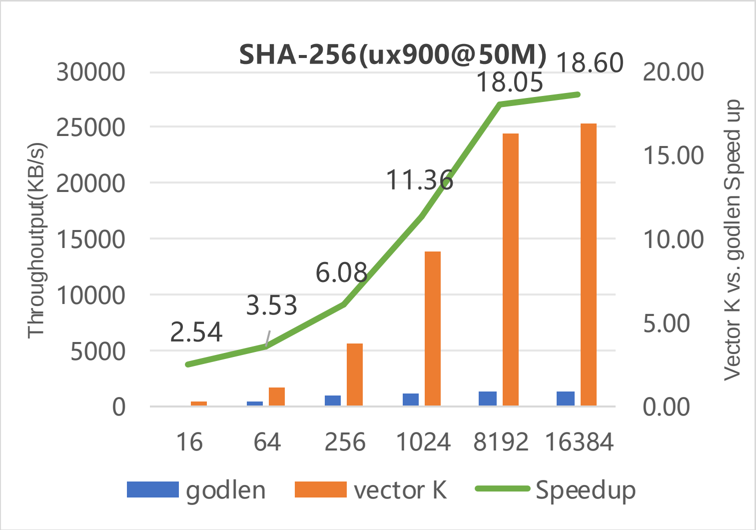
Note: ux900 vlen=128 dlen=128 @50M

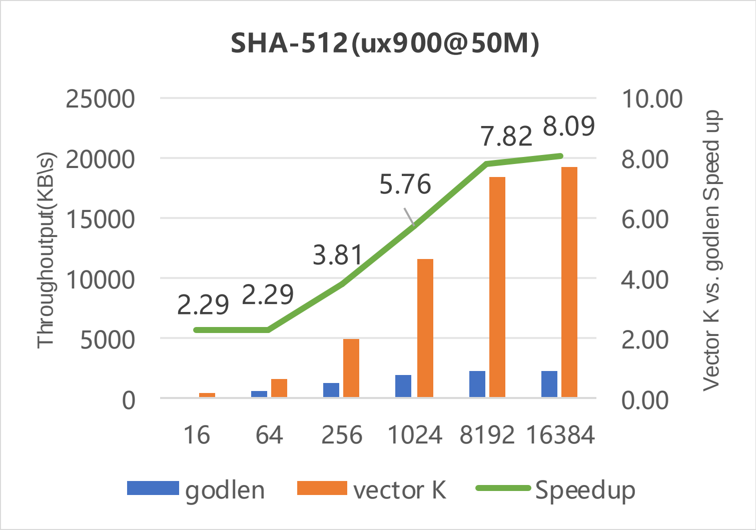
在Nuclei Evalsoc上实测的提升效果(OpenSSL)

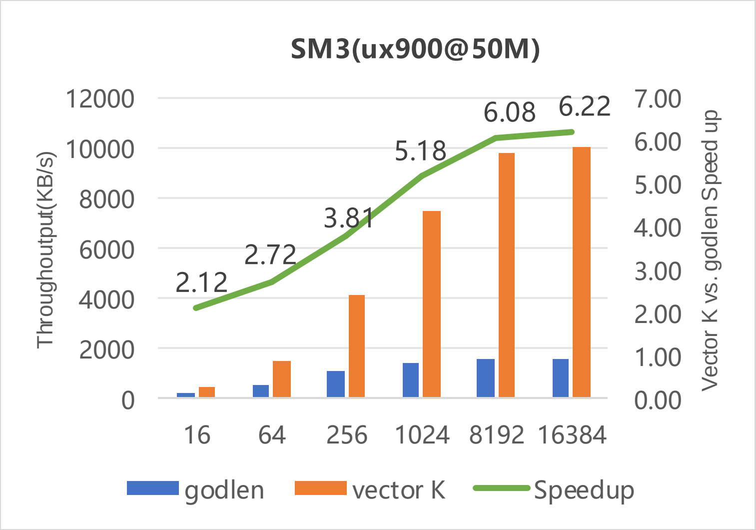
芯来科技 NUCLEI

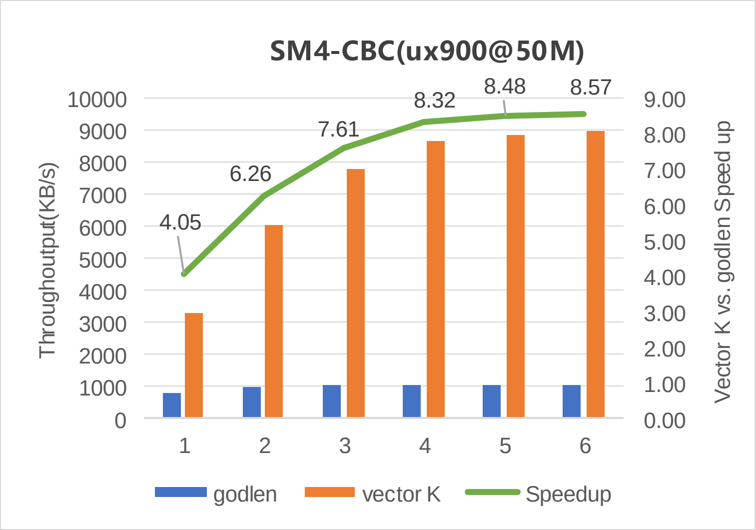












Note: ux900 vlen=128 dlen=128 @50M

芯来科技
NUCLEI

THANK YOU