Writing your own kernel crypto accelerator driver

For ELC-E 2020

Tero Kristo @ TI

Who am I

- Worked 9 years for Texas Instruments on Linux kernel development
- Lead for TI baseport team for ~5 years
- About 600 patches merged upstream
  - About 60 of these in crypto drivers
- Maintainer for couple of TI related drivers/subsystems in upstream Linux

- Linkedin: https://www.linkedin.com/in/tero-kristo-49068a/

Contents

Texas Instruments

1. Introduction

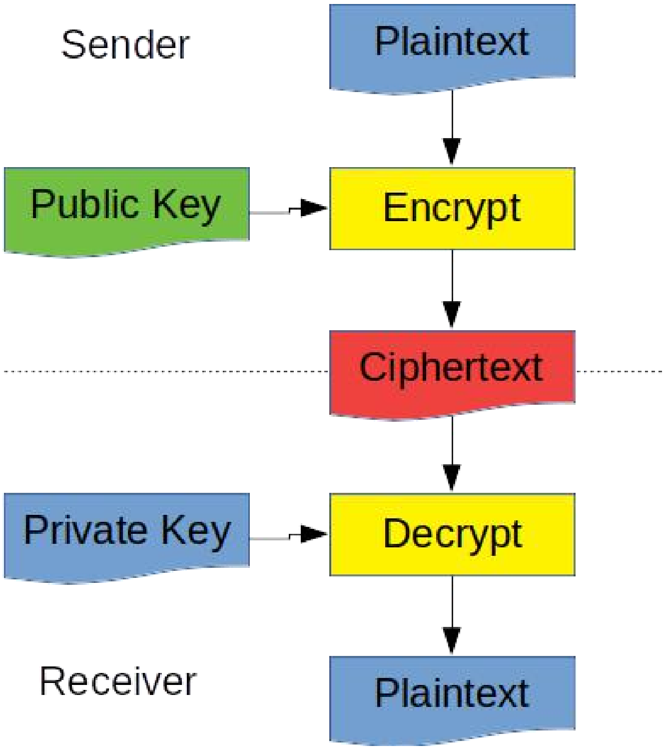
Cryptography overview

- What is cryptography?
  - Relatively complex mathematical algorithms to convert data into something unintelligible
- Why cryptography?
  - Authentication (no spoofing of identity)
  - Confidentiality (no eavesdropping)
  - Integrity (no tampering of data)
- Different algorithms for different use-cases

Texas Instruments

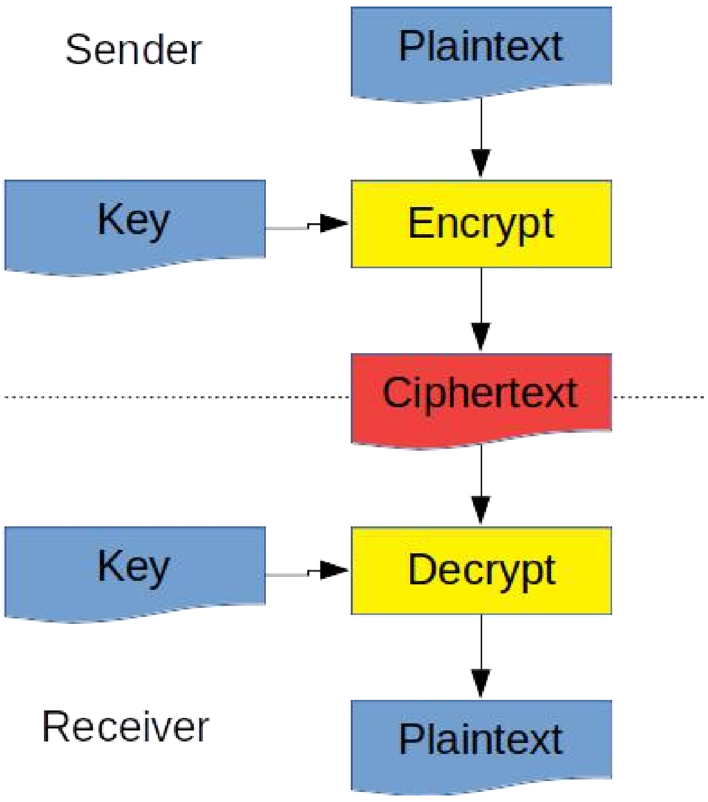
Authentication

- Asymmetric ciphers
  - RSA, DSA etc.
  - Has two keys, private and public
  - Public key can be shared freely
- Applications: digital signing, secure boot etc.



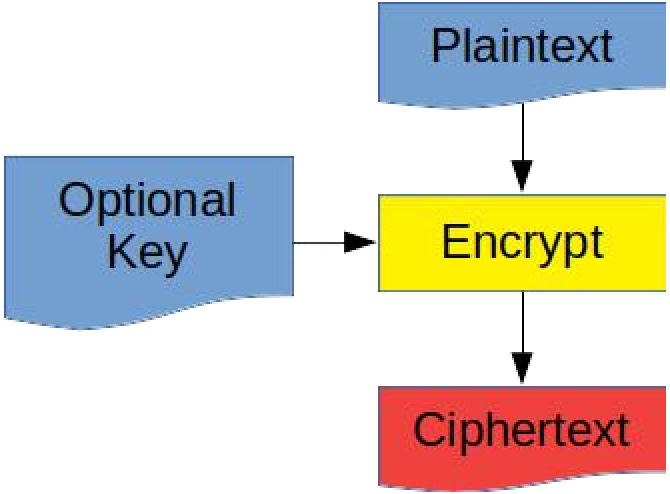
Confidentiality

- Symmetric ciphers
  - AES, DES etc.
  - Much faster than asymmetric ones
- Private key
  - Must be shared somehow secretly
- Applications: HDD encryption, secure messaging, IPSec etc.



Integrity

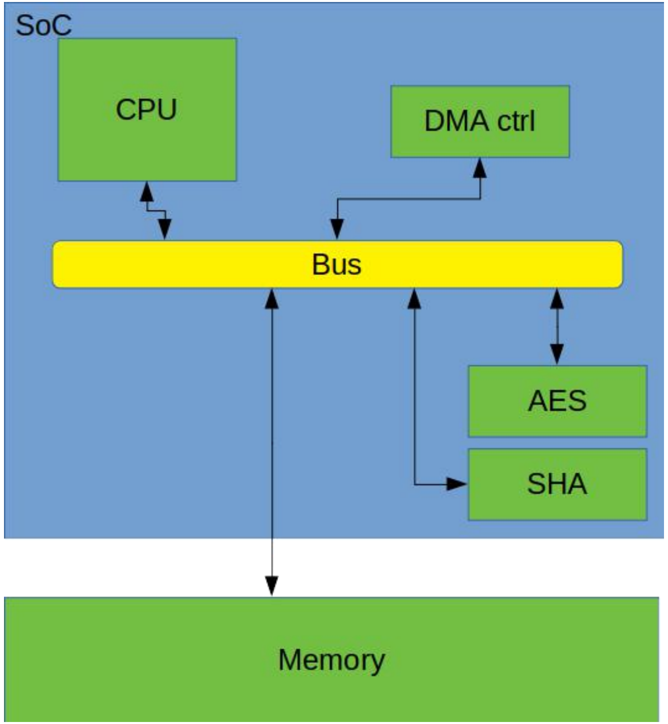
- Hash algorithms
  - MD5, SHA etc.
- Applications: image integrity checking, password storage etc.
- Impossible to generate original data from ciphertext



2. Implementation

Texas Instruments

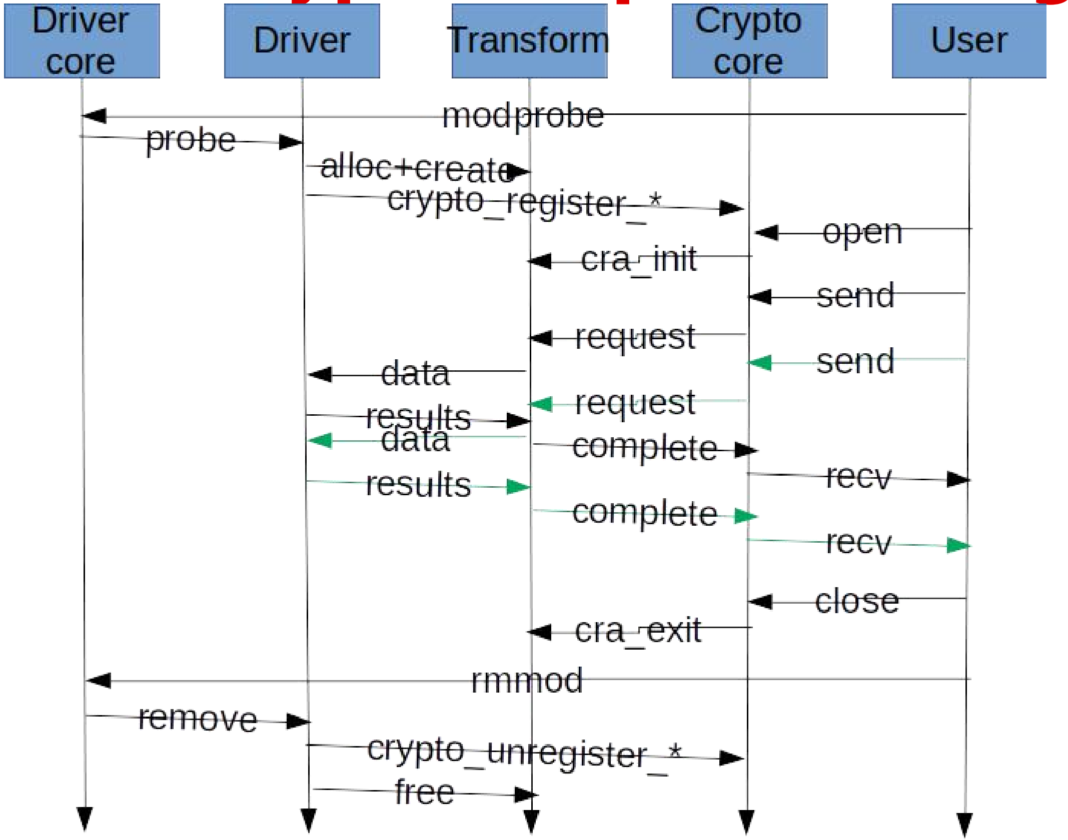
Simplified system architecture



Crypto API driver level concepts

- Transform
  - A single algorithm implementing a cryptographic operation
  - Either a hash, cipher, compression or random number generator (or AEAD)
  - Initialized/removed via the cra_init / cra_exit calls
- Request
  - A single crypto handling request containing data (might be zero length) to be processed
  - Single transform provides a few operations with whom the requests get processed
  - Results are typically provided back via asynchronous completion
- Both provide their contexts for driver level data storage
  - Don’t mix up these two
- Their lifetime is also different

High level crypto sequence diagram



Texas Instruments

Kernel APIs for creating a new algorithm

- int crypto_register_skcipher(struct skcipher_alg *alg)
  - For symmetric ciphers like AES, 3DES
- int crypto_register_ahash(struct ahash_alg *alg)
  - For hash algorithms: SHA1, SHA2 etc.
- int crypto_register_aead(struct aead_alg *alg)
  - For AEAD algorithms which are combined suites like SHA256 + AES (authenc(hmac(sha256),cbc(aes)))
- Plenty of others available also, but these are the ones we are interested in this presentation
- Once proper register function has been selected, just need to fill the *alg container

Hash operations

- Hash needs to register following driver APIs via the cookie
- init(struct ahash_req *req)
  - Initialize HW, hash state, internal data
- update(struct ahash_req *req)
  - Send new data to accelerator
- final(struct ahash_req *req)
  - Close current hash and return the result
- digest(struct ahash_req *req)
  - Combination of init/update/final
- export(...) + import(struct ahash_req *req, void *data)
  - Export current hash + status to continue it later

Hash notes

- Both export and import must be implemented
  - This might be tricky on some hw, so may resort to SW fallback only
- Register proper statesize
  - Using too small size will ensure strange problems with memory handling
  - Using too large will get the algorithm rejected by the crypto core
- Use SW fallback for small payload sizes
  - Setting up DMAs etc. can be expensive per packet
  - Can provide pretty large performance boost in some use cases
- Data will be sent over in multiple chunks (repeated update calls)
  - Complex buffering may be needed to handle things properly

Cipher / AEAD operations

- cipher and AEAD need to register following via the cookie
- setkey(struct crypto_{aead|skcipher]} *tfm, u8 *key, int keylen)
  - Set the encryption key for the algorithm
- encrypt(struct {skcipher|aead}_request *req)
  - Encrypt a chunk of data
- decrypt(struct {skcipher|aead}_request *req)
  - Decrypt a chunk of data
- Additionally, AEAD needs to register this:
- setauthsize(struct crypto_aead *tfm, int authsize)
  - Set the authentication data size for AEAD

Cipher / AEAD notes

- Register proper state/reqsizes
  - Similar to hashes, wrong sizes here induce difficult to debug problems
- Cipher typically easier to implement than hashes due to data being sent over in single chunk
- With small payload, use SW fallback similar to hashing

Texas Instruments

Testing support

- Crypto self tests done by crypto core
  - CONFIG_CRYPTO_MANAGER_DISABLE_TESTS=n (note inversion!)
  - CONFIG_CRYPTO_MANAGER_EXTRA_TESTS=y
  - Results in the boot log if any failures seen
  - /proc/crypto shows the status as unknown for any failed transforms
  - tip: if testing hangs during boot, try adding timeout to the crypto_wait_req to see what your driver was doing: https://patchwork.kernel.org/patch/11195553/
- Crypto test module
  - CONFIG_CRYPTO_TEST=m
  - modprobe tcrypt.ko mode=<mod> sec=<sec>
  - mode=600 for AES, mode=423 for SHA

Testing support (cont.)

- openssl testing
  - via either AF_ALG or devcrypto
  - e.g. openssl speed -evp aes256 -engine devcrypto
  - For devcrypto, must have cryptodev.ko installed
- IPSec testing
  - via e.g. strongswan suite
  - use iperf3 or something similar on top to test throughput

Driver optimization tips

- Combine processing if possible
  - Combine small data chunks to larger ones
  - Combine multiple interrupts and process them in batches
  - Combine multiple DMA xfers into single one
- Parallelism
  - Queue multiple requests simultaneously to HW if possible
- SW fallback usage
  - For small data chunks just execute the processing with SW fallback algorithm
  - Setting up DMA and processing the IRQ is expensive for small payloads
- Avoid scheduling
  - If you have more data waiting when finalizing old one, attempt to queue next chunk immediately (use crypto engine to do this automatically)

3. Test Results

HW used

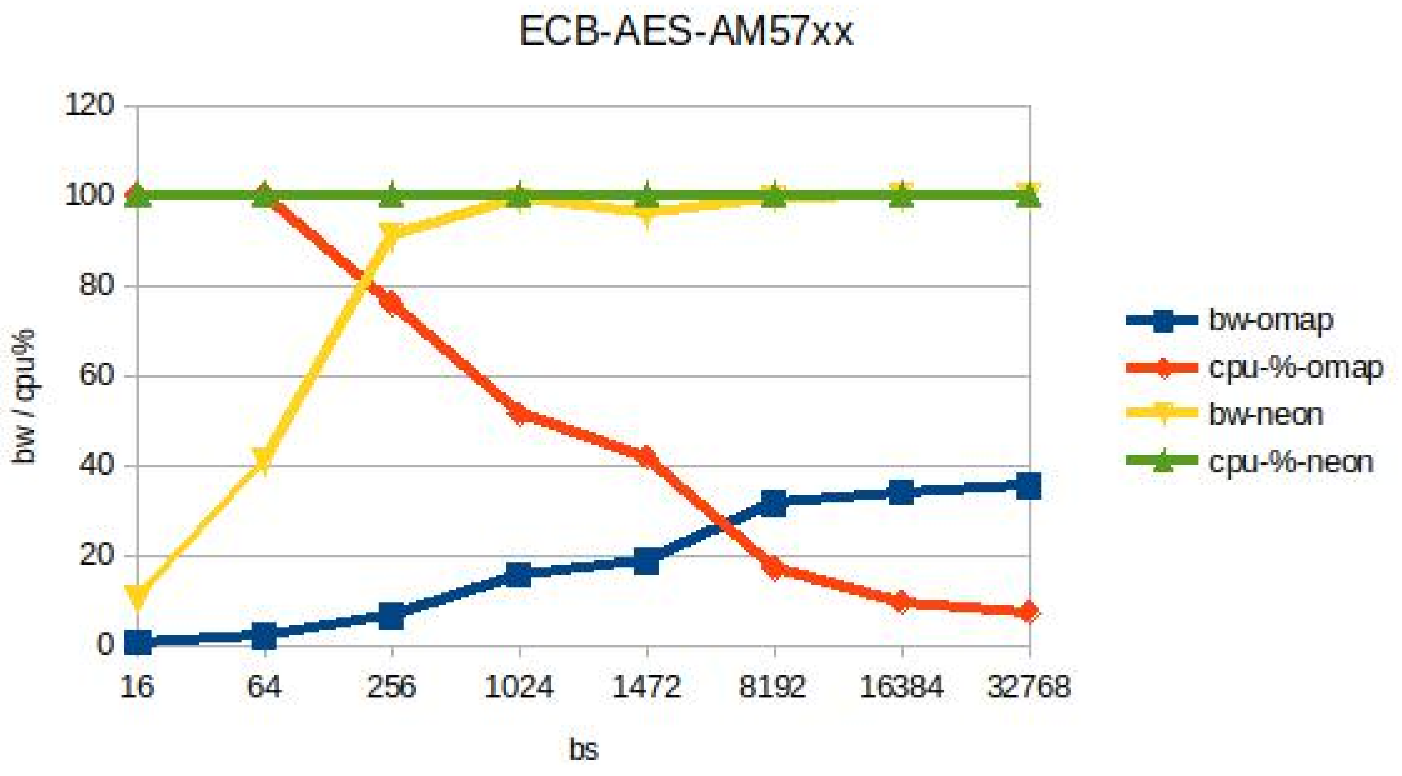
- Tested on couple of TI platforms
- AM57xx EVM
  - Cortex A15 x 2 @ 1.5GHz
  - ARMv7 architecture
  - NEON acceleration (-neon drivers)
  - TI OMAP family crypto IPs in use
- J721e EVM
  - Cortex A72 x 2 @ 2GHz
  - ARMv8 architecture
  - Crypto Extensions in use (-ce drivers)
  - TI SA2UL crypto accelerator block

Testing done

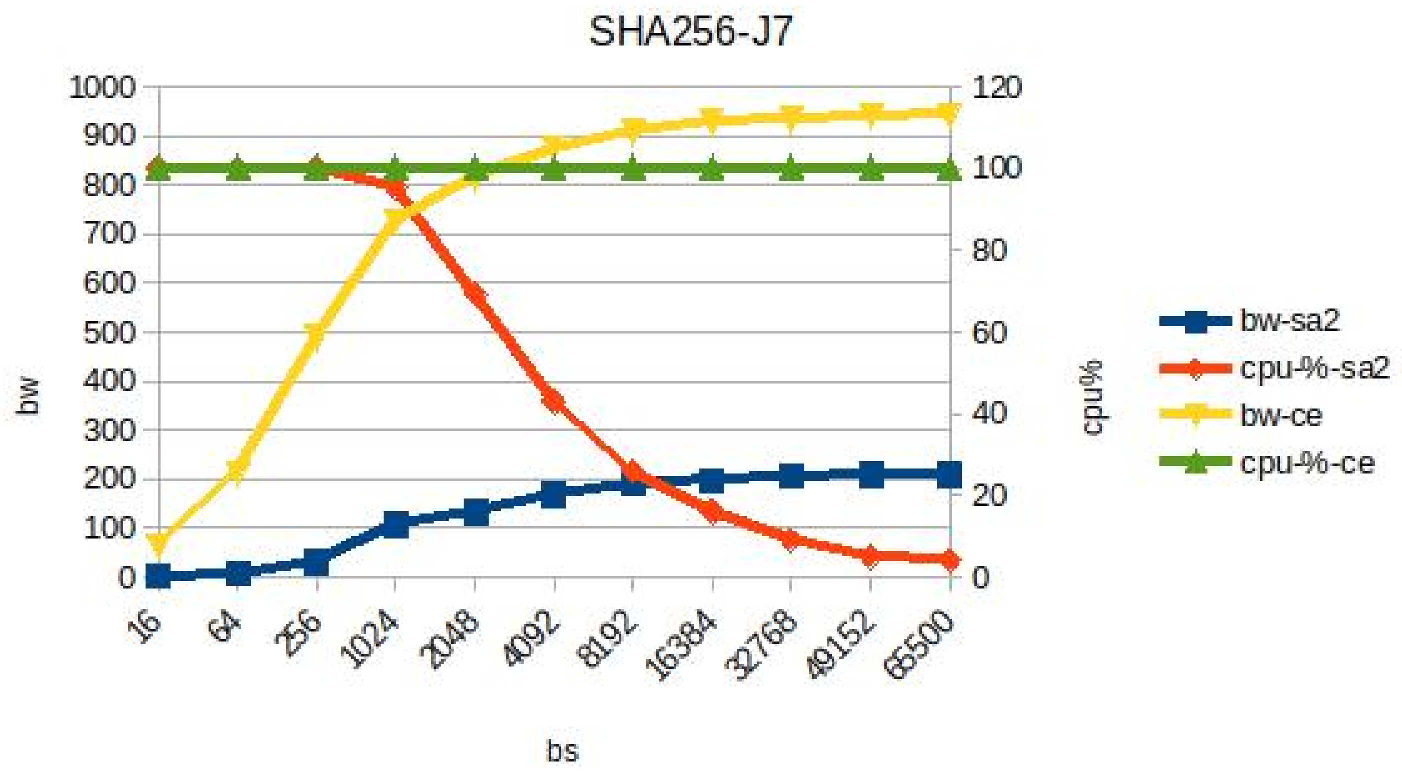
- Tested both HW accelerated and SW mode crypto
- Tcrypt.ko testing
  - Basic usage: modprobe tcrypt.ko mode=<mode> sec=1
  - SHA256 and AES-ECB with multiblock mode (AES=600, SHA=423)
  - captured results for 128b key
  - Slightly modified
    - Larger than normal block sizes used (upto 64K)
- CPU load measured additionally in all tests

Tcrypt results for AM57xx (1/2)

Tcrypt results for AM57xx (2/2)

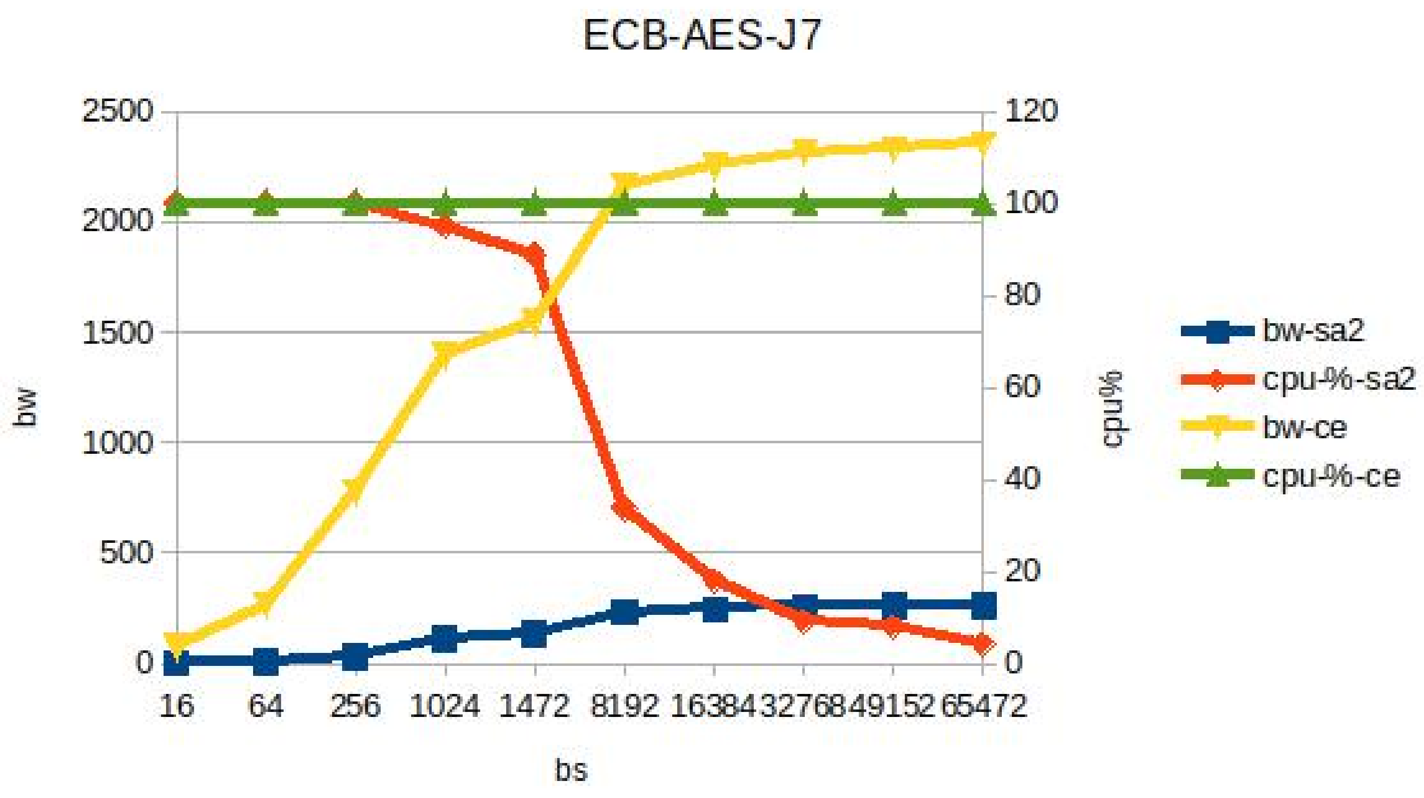


Tcrypt results for J7 (1/2)



Texas Instruments

Tcrypt results for J7 (2/2)



Thank you!