DryGASCON status update

NIST LWC round 2 candidate

Sébastien Riou

Tiempo S.A.S., France, sebastien.riou@tiempo-secure.com

Abstract. This paper gives a status update on DryGASCON submission to NIST's LWC standardization process.

Keywords: AEAD · Side Channels · Fault attacks · Leakage resilience

1 Clarification on AEAD instance names

DryGASCON AEAD has an unusual feature: it supports 3 key sizes (aka "profiles") for each security level. This has led to some confusion, this section introduce names for each variant described in the submission document.

The differences between the profiles are explained in page 7 of the round 2 submission document.

Table 1: DryGASCON AEAD named instances

Security level in bits	Key size in bytes	Instance name	Profile
128	16	DryGASCON128k16	Small
	32	DryGASCON128k32	Fast
	56	DryGASCON128k56	Full
256	32	DryGASCON256k32	Small
	48	DryGASCON256k48	Fast
	88	DryGASCON256k88	Full

2 Implementations

2.1 Software implementations

We use the names reported in the result page of the benchmark hosted at lwc.las3.de. We limit the list to the most efficient implementations.

2.1.1 rhys

The implementation is done in C99. When compiling for Microchip AVR CPUs, the *GASCON* primitive is implemented in assembly. This implementation is the fastest on AVR CPU. link: Authors' repository

Supported instances:

- DryGASCON128k16
- DryGASCON256k32
- DryGASCON128 hash
- DryGASCON256 hash

2.1.2 add_arm_cortex-m

This implementation is done in C99, it targets little endian CPUs. When compiling for ARM Cortex-M CPUs, the F and G primitives are implemented in assembly. link: las3.de submitted package

Supported instances:

- DryGASCON128k16
- DryGASCON128k32
- DryGASCON128k56
- DryGASCON128 hash

Remark 1. In future submission packages this implementation will be renamed "opt_arm_cortex-m".

This implementation has been submitted for inclusion in Rhys Weatherley's benchmark. (see this pull request). At the time of writing this has need been done so we present here our measurement using that benchmark framework:

Table 2: rhys vs add_arm_cortex-m on STM32F411RE (ChaCha-poly unit)

implementation	variant	encrypt 16 bytes	encrypt 128 bytes
rhys	DryGASCON128k16	0.35x	0.19x
add_arm_cortex-m pure C	DryGASCON128k16	0.54x	0.31x
	DryGASCON128k32	0.55x	0.31x
	DryGASCON128k56	0.55x	0.31x
add_arm_cortex-m with asm	DryGASCON128k16	0.98x	0.58x
	DryGASCON128k32	1.00x	0.58x
	DryGASCON128k56	1.01x	0.58x

Remark 2. The figures are lower than in the online benchmark because the benchmarking platform is different. Online benchmark use a device based on ARM-Cortex-M3 CPU, here we use the STM32F411RE which is based on ARM-Cortex-M4 CPU. This seems to give an advantage to ChaCha-poly cipher and therefore lowers the score of all benchmarked ciphers.

2.1.3 le32

This implementation is done in C99, it targets little endian CPUs. All instances are supported. link: Authors' repository

2.1.4 Side channel protection

As DryGASCON provides side channel and fault protection at the algorithmic level, none of the implementation includes expensive countermeasures such as masking. Nevertheless not all implementation are equal with respect to side channel attacks. For example, the C reference implementation manipulates all values byte by byte to be independant of system endianness.

We expect all implementations discussed in this document to be protected against side channel attacks. To study side channel attacks on DryGASCON, we recommend to target the **add_arm_cortex-m** implementation. This implementation contains the full F primitive as assembly code. This avoids to depend on compiler version and parameters.

2.2 Hardware implementations

2.2.1 Full hardware implementation

This implementation is compliant to the LWC hardware API. It has been submitted to both the FPGA and ASIC benchmarks, we do not have the results yet. link: GitHub repository

2.2.2 Accelerator for microcontrollers

In microcontrollers, it is popular to implement only core primitives in hardware and let the software handle the higher levels of algorithms. For DryGASCON, that means supporting F and G in hardware and let software manage key setup, domain separators, the number of rounds and so on. This strikes a trade off between performances, area and development effort. It also provides flexibility, allowing to use the hardware for future algorithm which would rely on the same primitives.

The accelerator connects as a standard slave on APB3 bus. It is implemented as a single file, easing integration.

link: GitHub repository

Remark 3. FPGA synthesis results are given in the submission document in section "13.1 Hardware implementations".

2.2.3 Side channel protection

As DryGASCON provides side channel and fault protection at the algorithmic level, none of the implementation includes expensive countermeasures such as masking.

2.3 Python implementation

The package drysponge contains CLI programs and expose an API to compute all DryGASCON v1 variants. It allows to print out intermediate values.

3 Targeted use cases

3.1 Secure element integrated in SOC

Advanced technology nodes such as 16nm and below do not have on-chip NVM. More precisely, they lack "multiple times programable, non volatile memory" or MTP-NVM (EEPROM, flash and MRAM are MTP-NVMs, fuse and ROM are not). This is a problem when a SOC integrates a secure element (aka secure enclave, HSM, root of trust...). Typically a secure element is integrated in a SOC to handle payments or strong authentication (in order to support billing, like in the case of SIM applications). Those applications requires some amount of MTP-NVM, typical range is between 128KB (low end payment) to 2MB (high end SIM). Since that type of memory is not available on-chip, integrated secure elements use an external one (typically a QPSI flash). Since the data in a QSPI flash is easily accessible, the secure element use an internally generated secret key and an AEAD algorithm to guarantee confidentiality and integrity. Typically the memory is managed by chunks maching the internal cache line size, usually between 32 to 256 bytes. Typically there is at most one block of associated data. Last but not least, some industry groups requires 256 bit security. The AEAD or at least the core primitive is typically implemented in hardware.

In this use case, the AEAD is fully exposed to side channel and fault injection attacks. The attacker may control the encryption input data since it can be some data provided

externally. The attacker is able to replay decryptions unlimited number of times and may be able to observe decryption output in some cases.

Due to the risk of fault attack, AES-CCM is prefered over AES-GCM (GCM tag can be correct even though the decryption result is wrong since the tag is computed from the ciphertext). Using AES-OCB would provide a speed up however the patent situation is a show stopper for hardware IP providers and it is not currently in the list of "NIST approved" algorithm, so it would not be accepted in the market (some industry groups explicitely require the use of "NIST approved" algorithms). Even if those show stoppers would be removed, AES-OCB still requires side channel protection of the AES. This is a major drawback since side channel protection is expensive (development effort, risk of vulnerability revealed only on silicon, higher area and power consumption, lower performances...).

DryGASCON would bring a significant improvement over any AES based AEAD for this use case. We would recommend in particular a hardware implementation of DryGASCON128k32 or DryGASCON256k48. The resulting implementation is much more efficient overall than an AES fully protected against both side channel and fault attacks which also requires a significant amount of random numbers. The implementation and security validation efforts are also greatly reduced as the side channel and fault attack protection comes for free with DryGASCON.

4 DryGASCON vs current NIST standards

Section 3.1 discussed already the advantages of DryGASCON AEAD versus AES-GCM, AES-CCM and any other AES based AEAD standardized so far. The performance advantage is even more significant when considering software implementation on MCUs if they have to be protected against side channel and/or fault attacks. lwc.las3.de results show that DryGASCON128 is faster than unprotected AES-GCM on ARM-Cortex-M CPUs.

SHA2 is even harder to protect against side channel attacks than AES. This is due to its 'ARX' nature. This is well explained in Why Keccak is not ARX. This is unfortunate because in the context of HMAC side channel attacks are applicable.

SHA3 is easier to protect against side channel attacks than SHA2 and AES. It requires nevertheless careful implementation of countermeasures which are not easy to verify by simulation. In the context of implementations on ASIC, side channel weaknesses are typically found after the ASIC is produced. This means that having side channel protection at implementation level is a huge risk. Often the only way to fix the weakness is to modify the design and launch a respin of the chip.

DryGASCON solve this problem by providing protection at algorithmic level. ASIC designers only have to ensure correct functionality, something which is trivially done using simulation.

Last but not least, DryGASCON, even in its 256 bit version, is significantly smaller than unprotected SHA3 (SHA3 state size is almost twice as large).

5 Tweaks

5.1 XOF mode

DryGASCON v1 includes a hash mode but not a XOF mode. This is something we wish to add in the round 3 submission if we have this opportunity. In this mode, DryGASCON performances shall be very close to ASCON performances as it would involve only the G function.

5.2 KSneq32v2: uniformization of key profiles

For each security level we defined three profiles. The full profile match the natural key size of the primitive. The fast profile reduce the key size as much as possible while keeping the key setup fast and straightforward. The small profile reduce the key size to the strict minimum such that the key size match the security level. The small profile allows to use DryGASCON as a drop in replacement of most other AEAD, including the ones based on AES128 and AES256. One inevitable drawback of the small profile is that such a small key is more vulnerable to side channel attacks during the key loading phase. For that reason, we recommend to use the fast profile. For cases in which it is mandatory to stick to minimum key sizes, the small profile as specified in "DryGASCON v1" is fine however it is somewhat complex and it is not compatible with the fast profile. That means that all users have to use the same key profile for a given application. This is workable but it would be much better if the choice of the key profile could be considered as an implementation choice rather than as a choice of algorithm. This would allow a server to use compact 128 bits keys while connected objects could use fast 256 bits keys.

The tweak describe here allows to map all keys from the small key profile to keys in the fast and full profile, therefore achieving 'upward' interoperability.

Algorithm 1 KSneq32v2

Input: K secret key, $width \in \{minwidth, fastwidth, fullwidth\}$
Output: (c, x)

if $width == fullwidth$ **then**
  $c \quad Sel(K, c.width, 0)$
  $x \quad K \gg c.width$
else
  $kwords \quad minwidth/32$
  for $i \in 0, \cdots, c.width/32$ **do**
    $Sel(c, 32, i) \quad Sel(K, 32, i \mod kwords)$
  end for
  if $width == fastwidth$ **then**
    $x \quad K \gg minwidth$
  else
    $x \quad GASCON(c, 0)$
    $x \quad Sel(x, x.width, 0)$
  end if
end if
$match = 0$
for $i \in 0, \cdots, xwords - 1$ **do**
  for $j \in i + 1, \cdots, xwords$ **do**
    if $Sel(x, 32, i) == Sel(x, 32, j)$ **then**
      $match = 1$
    end if
  end for
end for
if $match == 1$ **then**
  $mask \quad$ 0xFFFFFFFF $\ll \log_2(xwords)$
  for $i \in 0, \cdots, xwords$ **do**
    $Sel(x, 32, i) \quad (Sel(x, 32, i)\ \&\ mask) \mid i$
  end for
end if
return (c, x)

Remark 4. The test vectors for the full and fast profiles remain unchanged.

Remark 5. The small profile is simplified and can be implemented in constant time.

Remark 6. In the small profile, this version injects 2 known bits into each 32 bit word of x. This may reduce the level of protection against side channel attack. In practice this injection of known bits happens with probablity 2^{-30}, meaning that it is rather unlikely even for fleet of millions of connected objects. We do not see this as a problem since we recommend to use the other profiles in applications which target side channel protection.

Remark 7. In the full and fast profiles, DryGASCON v1 accept only keys with special format for the 128 msb. This tweak allows to accept any key by 'fixing up' the keys that do not have the required property. Applications which requires protection against side channel shall avoid using such 'fixed up' keys. This is trivial to do, one has to check that the 'match' variable is 0 (see lines 16 to 23).

5.3 Allow precomputation over associated data

DryGASCON v1 enforce to process the NONCE before the associated data. Swapping the order of computation would allow precomputation over the associated data, something that may be useful in some applications. This would involve removing support for the "static data" feature to avoid a collision of domain separators. This is not a problem since that feature would be somewhat redundant with what this change provides.

6 The case for merging with ASCON and ISAP

ASCON is an interesting candidates as it combines good performances, ease of masking and reasonable hardware ressources. In addition, its primitive is reused in ISAP which brings algorithmic security against DPA, at the cost of much slower performance though.

Comparison with DryGASCON:

Table 3: DryGASCON vs ASCON vs ISAP

	DryGASCON128	Ascon-128	ISAP-A-128a
Performances without countermeasures*	Medium	Fastest	Slowest
Performances with countermeasures*	Fastest	Slowest	Medium
Support 256 bit security level	Yes	No	No
Support Hash	Yes	Yes	No
TAG protects plain-text	Yes	No	No

Remark 8. The stars indicate lines in which the ranking is debatable. It reflects only our expectations in the context of real world use cases.

Remark 9. The "TAG protects plain-text" property is important when considering fault attacks. Without it, implementers have to add costly countermeasures for decryption.

If ASCON-128 is standardized, we hope ISAP-A-128a will be too. This would allow semiconductor manufacturers to support both with a single hardware accelerator. A MCU with such accelerator would therefore appeal to applications with need of performance without physical security AND to applications with need of physical security.

Our expectation is that DryGASCON is safer and significantly faster than ISAP-A-128a, unfortunately DryGASCON does not use exactly the same permutation as ASCON. DryGASCON is built on a tweaked version of the ASCON permutation we called $GASCON_{C5}$. The tweaks have been introduced to meet requirements of the drysponge contruction and have the nice side effect to speed up execution of software implementation on 32 bits CPUs. Much of the cryptanalysis done on ASCON permutation is applicable to $GASCON_{C5}$. It

follows that the $GASCON_{C5}$ permutation can be safely used with the ASCON AEAD, Hash and XOF modes. We will refer to this combination simply as GASCON. Similarly $GASCON_{C5}$ permutation can be safely used with the ISAP AEAD, we will refer to this combination simply as GISAP.

GASCON and GISAP would be faster than ASCON and ISAP on 32 bit CPUs because the $GASCON_{C5}$ permutation is using 32 bit friendly "bit interleaved rotations" instead of the regular 64 bit rotations. On little endian CPUs (which seems to be the trend in embedded systems), another speed up is provided as $GASCON_{C5}$ can take advantage of the memory layout. Proof of concept implementations benchmarked using SUPERCOP confirm this. We also did another proof of concept based on the ASCON implementation from https://github.com/rweather/lightweight-crypto/. The results in table 4 show a significant speed up in favor of GASCON for all variant of ASCON.

Table 4: GASCON vs ASCON on STM32F411RE (ChaCha-poly unit)

	encrypt 16 bytes	encrypt 128 bytes
ASCON-128	1.07x	0.84x
GASCON-128	1.29x	0.92x
ASCON-128a	1.14x	1.08x
GASCON-128a	1.46x	1.25x
ASCON-80pq	1.06x	0.84x
GASCON-80pq	1.26x	0.92x

Standardizing DryGASCON, GASCON and GISAP would allow semiconductor manufacturers to support the three modes with a single hardware accelerator. Users would typically use GASCON or DryGASCON depending if they need physical security or not. The GISAP mode being available as a backup solution in the event of a devastating attack appearing on DryGASCON.

Remark 10. Our proposal for GASCON and GISAP is limited to the 128 bit security level. An adaptation to the 256 bit security level is certainly possible but would involve careful analysis to select parameters like the number of rounds. The absence of attacks on DryGASCON256 with a given number of round is not implying the absence of attacks on what would be named GASCON256 because the drysponge construction is shielding the permutation of most attacks.