Status Update on Ascon v1.2

Christoph Dobraunig, Maria Eichlseder, Florian Mendel and Martin Schläffer

https://ascon.iaik.tugraz.at
ascon@iaik.tugraz.at

30 September 2022

1 Overview on third-party analysis

In this section, we give an overview of third-party analysis of Ascon. For the results of the designers, we refer to the Ascon design document [23].

1.1 Analysis on AEAD only reducing rounds

In Table 1, we give an overview of third-party results that analyze the authenticated encryption variants of Ascon by reducing the rounds of the underlying permutation. As we can see, the results of Table 1 indicate a comfortable security margin.

Table 1: Best third-party analyses of Ascon-128, Ascon-128a, and Ascon-80pq achieved by only reducing the number of rounds of their underlying permutation.

Type	Target	Rounds	Time	Method	Reference
Key recovery	Ascon initialization	7 / 12	2^{123}	Cube	[48]
	Ascon initialization	6 / 12	2^{40}	Cube-like	[38]
	Ascon initialization	5 / 12	2^{31}	Diff.-linear	[54]
Forgery	Ascon-128a finalization	3 / 12	2^{20}	Differential	[27]
State recovery	Ascon-128a iteration	3 / 8	2^{117}	Differential	[27]
	Ascon-128a iteration	2 / 8	–	Sat-Solver	[25]

1.2 Analysis on AEAD by reducing rounds and under misuse

In Table 2, we show analysis results on Ascon that further weaken the algorithm outside of its specification and security claim by repeating the nonce, using more than 2^{64} blocks of data per key, exceeding 2^{128} time complexity, or in a weak-key setting.

The third-party results show that the initialization and finalization hold up very well even in the analyzed misuse scenarios only allowing for application on round-reduced variants. As we can further see in Table 2, it is possible to recover the internal state of Ascon-128 and Ascon-80pq under massive repetition of the nonce and an attacker who is able to choose the plaintexts according to their needs. Due to the initialization and finalization being keyed, the only immediate consequence of such a state recovery is the loss of confidentiality for the plaintexts that have been encrypted under the nonce that has been massively repeated. As shown in [14, 15], it requires cryptographically significant additional effort ($2^{97.5}$ for Ascon-128a [15] and $2^{129.5}$ for Ascon-80pq [14]) and additional queries under repeated nonces to turn a state recovery into a key recovery that defeats authenticity and the confidentiality of plaintexts encrypted under arbitrary nonces.

Table 2: Best third-party analysis of Ascon-128, Ascon-128a, and Ascon-80pq achieved by additional weakening of the algorithm outside of its specification or security claim. We annotate this additional weakening by 🚫 = nonce misuse, ⚠ = exceeds data limit of 2^{64} blocks, ⌛ = time exceeds 2^{128}, and 🔑 = using a weak key.

Type	Target	Rounds	Time	Method	Reference
Key recovery	Ascon initialization	7 / 12	2^{97} 🚫	Cube-like	[37]
	Ascon initialization	7 / 12	2^{104} ⚠	Cube-like	[38]
	Ascon initialization	7 / 12	2^{97} 🔑	Cube-like	[49]
	Ascon initialization	7 / 12	2^{77} 🔑⚠	Cube-like	[38]
	Ascon initialization	6 / 12	2^{74} ⚠	Cond. HDL	[31]
	Ascon-128a iteration	7 / 8	2^{118} 🚫⚠	Cond. cube	[15]
	Ascon-80pq iteration	6 / 6	2^{130} 🚫⌛	Cond. cube	[14]
Forgery	Ascon-128 finalization	6 / 12	2^{33} 🚫	Cube tester	[37]
	Ascon-128 finalization	4 / 12	2^{97} ⚠	Differential	[27]
State recovery	Ascon-128 iteration	6 / 6	2^{40} 🚫	Cond. cube	[4]
	Ascon-128 iteration	6 / 6	2^{45} 🚫	Cond. cube	[14]
	Ascon-128 iteration	5 / 6	2^{66} 🚫	Cube-like	[37]
	Ascon-128a iteration	7 / 8	2^{118} 🚫⚠	Cond. cube	[15]

1.3 Analysis on Ascon-Hash and Ascon-Xof only reducing rounds

The hash functions and extendable output functions of the Ascon family have a relatively high capacity of 256 bits compared to their 64-bit rate. Hence, as also backed by the third-party results of Table 3, an attacker is very limited in degrees of freedom, resulting in a comfortable security margin. The published results on Ascon-Hash and Ascon-Xof are also directly applicable to Ascon-Hasha and Ascon-Xofa.

Table 3: Best third-party analysis of Ascon-Hash and Ascon-Xof.

Type	Target	Output	Rounds	Time	Method	Ref.
Collision	Ascon-Xof finalization	64	2 / 12	2^{15}	Differential	[58]
	Ascon-Xofa finalization	64	2 / 12	2^{15}	Differential	[58]
	Ascon-Hash iteration	256	2 / 12	2^{125}	Differential	[58]
	Ascon-Hash iteration	256	2 / 12	2^{103}	Differential	[27]
	Ascon-Hasha iteration	256	2 / 8	2^{125}	Differential	[58]
	Ascon-Hasha iteration	256	2 / 8	2^{103}	Differential	[27]

1.4 Analysis of permutation

As already stated in the design document [23], we emphasize that we do not require ideal properties for the underlying permutations of our designs. Non-random properties of the permutations are known and do not afflict the claimed security properties of the entire algorithms. Nevertheless, for designing algorithms based on Ascon's permutation, it is necessary to understand the permutations' cryptographic properties. Hence, we give an overview of third-party results regarding Ascon's permutation in Table 4.

Table 4: Third-party analysis of Ascon's permutation. (👁 = non-black-box distinguisher, 🛠 = chosen constants at input (integral) or middle (zero-sum, i.e., no zero-sum partition))

Goal	Target	Rounds	Time	Method	Reference
Distinguisher	Permutation	12 / 12	2^{55} 👁🛠	Zero-sum	[31]
	Permutation	8 / 12	2^{46} 🛠	Integral	[31]
	Permutation	7 / 12	2^{65}	Integral	[56]
	Permutation	7 / 12	2^{60}	Integral	[48]
	Permutation	7 / 12	2^{34} 👁	Limited-Birthday	[27]
	Permutation	5 / 12	2^{109}	Truncated Differential	[55]
	Permutation	5 / 12	2^{80}	Rectangle	[27]
	Permutation	3 / 12	–	Subspace Trails	[35]

2 New proofs/arguments supporting the security claims

All variants of Ascon are based on either the sponge or the duplex construction. Hence, they profit from the vast analysis already performed for these constructions [8, 19]. An overview of the relevant work can be found in Ascon's design document [23, Section 6].

Improved preimage security bounds. A new paper by Levevre and Mennink [36] proves that Ascon-Hash could generically achieve 192-bit preimage security, which is higher than anticipated based on previous bounds.

Improved bounds on characteristics. Several teams have worked on proving improved bounds on characteristics. Erlacher et al. [26] proved bounds on the minimum number of linearly or differentially active S-boxes for 4, 6, 8, 12 rounds using SAT solvers. For the 12-round initialization and finalization, any characteristic has differential probability or squared correlation $\leq 2^{-216}$; this bound is very likely not tight, but provides ample security margin. For the 6-round (8-round) data absorption, the bound is $\leq 2^{-108}$ ($\leq 2^{-144}$) and thus not exploitable within the data limit, again likely not tight. Makarim and Rohit [39] proved that the previously best known differential characteristic for 3 rounds with probability 2^{-40} is optimal, using a hybrid SMT/MILP approach. They also slightly improve the best known characteristics (from 44 to 43 differentially active S-boxes for 4 rounds and 78 to 72 for 5 rounds, though with worse probability; and from squared correlation 2^{-186} to 2^{-184} for 5 rounds), allowing a clearer estimate of how far from tight the current bounds likely are. The results are summarized in Table 5.

Table 5: Bounds on the number of active S-boxes (min #S) and probability or squared correlation (max p, max c^2) of the round-reduced Ascon permutation, listing both bounds and currently best known characteristics. Bold values are new.

R	Differential characteristics				Linear characteristics			
	min #S		max p		min #S		max c^2	
	bound	known	bound	known	bound	known	bound	known
1	1	1	2^{-2}	2^{-2}	1	1	2^{-2}	2^{-2}
2	4	4	2^{-8}	2^{-8}	4	4	2^{-8}	2^{-8}
3	15	15	$2^{\mathbf{-40}}$	2^{-40}	13	13	2^{-26}	2^{-28}
4	$\geq \mathbf{36}$	43	$\leq 2^{\mathbf{-72}}$	2^{-107}	$\geq \mathbf{36}$	43	$\leq 2^{\mathbf{-72}}$	2^{-98}
5	$\geq \mathbf{37}$	72	$\leq 2^{\mathbf{-74}}$	2^{-190}	$\geq \mathbf{37}$	67	$\leq 2^{\mathbf{-74}}$	2^{-184}
6	$\geq \mathbf{54}$		$\leq 2^{\mathbf{-108}}$		$\geq \mathbf{54}$		$\leq 2^{\mathbf{-108}}$	
8	$\geq \mathbf{72}$		$\leq 2^{\mathbf{-144}}$		$\geq \mathbf{72}$		$\leq 2^{\mathbf{-144}}$	
12	$\geq \mathbf{108}$		$\leq 2^{\mathbf{-216}}$		$\geq \mathbf{108}$		$\leq 2^{\mathbf{-216}}$	

3 New software and hardware implementations

Since Ascon has been published in 2014, many hardware and software implementations are available. An overview can be found in the design document [23, Section 7]. This includes implementations that protect against side-channel attacks. Nevertheless, many new implementations have been published in the final year of the competition. In the following sections, we give an overview on these implementations and their improvements.

3.1 Software implementations

The main sources of optimized Ascon software implementations is the Github repository by the Ascon team [24]. These implementations have been submitted to the main benchmarking initiatives. Additionally, Rhys Weatherley has implemented a general Ascon suite and a specific one for Arduino devices. In the following, we list the available implementations of these repositories:

- `https://github.com/ascon/ascon-c` (Ascon team):
  - C: ref, speed/area optimized, trade-offs, combined
  - ASM: ARMv6, ARMv6m, ARMv7m, ESP32, RV32, AVR
  - Masked C+ASM: 2-4 shares, leveled (evaluated)
  - Modes: AEAD, Hash, XOF, MAC, PRF
- `https://github.com/rweather/ascon-suite` and `https://github.com/rweather/ascon-arduino` (Rhys Weatherley)
  - 8/32/64-bit C, AVR, ARM, RISC-V, m68k, Xtensa (ESP32)
  - Framework to generate C/ASM/masked implementations
  - Masked 8-bit AVR and 32/64-bit C implementations
  - Modes: AEAD, Hash, HKDF, ISAP, KMAC, PBKDF2, PRNG, SIV, XOF

The updates in these repositories include new algorithms, implementations optimized for speed and size as well as masked implementations to protect against side-channel attacks. Most importantly, we

- improved the performance (0-65%) and code size (0-65%) on low-end 32-bit and 8-bit devices (ARMv6m, ARMv7m, ESP32 [3], RV32 [3], AVR [12]),
- improved the code size of implementations combining AEAD with hashing (overhead for AEAD with hashing compared to AEAD is only about 15%),
- improved the round function using fewer instructions for the S-box [11],
- added evaluated masked and leveled ARMv6 implementations [21],
- added Ascon algorithms with a bit-interleaved interface,
- added implementations for the NIST LWC candidates Ascon-Hasha and Ascon-Xofa,
- and added implementations for Ascon-Mac, Ascon-Prf and Ascon-Prfs [22].

3.1.1 Software benchmarks

The performance of many implementations mentioned in the previous section have been benchmarked by the following benchmarking initiatives:

- https://bench.cr.yp.to/
- https://lwc.las3.de/
- https://rweather.github.io/lwc-finalists/
- https://rweather.github.io/arduinolibs/crypto.html
- https://github.com/usnistgov/Lightweight-Cryptography-Benchmarking

We refer to the benchmarking initiatives instead of listing all results here. Note that updates of [24] in the final round have only been benchmarked by the first two initiatives yet. We hope that the latest improvements will be included by the others as well until the end of the final round. Nevertheless, we provide a first analysis to show the improvements made to [24] since the end of round 2 for different low-end devices here.

Table 6 includes a preliminary code size comparison of different Ascon algorithms with Aes-Gcm and Sha-256 using the benchmarking framework by NIST [45][1]. Table 7 provides an overview of the improvements made to Ascon compared to the round 2 code, Aes-Gcm and the best other finalists on each device and category using the benchmarking results of [47]. The benchmarks show excellent performance and area numbers for Ascon on a wide range of different platforms. Compared to current NIST standards and other NIST LWC finalists, Ascon performs better in most situations.

Table 6: Code size (in bytes) for the smallest implementations of Ascon and common NIST standards. Numbers have been generated using [45] with platformio 6.1.4 and the latest version of [24].

	ATmega328P	ATmega4809	SAM D21	nRF52840	PIC32MX	ESP8266
Aes-Gcm	3188	3102	1648	1776	2536	2412
Ascon-128	2754	2820	1300	1144	1820	1004[1]
Ascon-128a	3026	2894	1424	1192	1876	1084[1]
Sha-256	2432	2322	944	936	1628	1116
Ascon-Hash	1836	1876	760	688	1244	652[1]
Ascon-Hasha	1836	1876	760	688	1244	652[1]

Also, for Ascon, the area overhead for AEAD with hashing is very small compared to, e.g., Aes-Gcm with Sha-256 or Aes-Gcm with SHA-3. Table 8 shows a preliminary evaluation of combined Ascon implementations on different low-end devices using [24]. In this case, the overhead of AEAD with hashing is only about 12-16% (about 500 Bytes for AVR or about 200 Bytes for ARM) compared to AEAD only implementations. Using Table 6, the overhead for Aes-Gcm + Sha-256 compared to only Aes-Gcm seems to be in the range of about 45-75%.

An efficient way to implement Ascon on 32-bit platforms is using bit interleaving. If performance is critical, the data/key/nonce can be stored/transmitted in bit interleaved format. This allows to improve the performance of Ascon by another 10-20%. The same improvement is possible using, e.g., funnel shifts (on ESP32) or dedicated bit

[1]For ESP8266, the asm_esp32 implementation of [24] does not build correctly with xtensa-lx106-elf-gcc 10.3.0 (installed by platformio 6.1.4), while it does build with xtensa-lx106-elf-gcc 10.3.0-1ubuntu1+8ubuntu1 on Ubuntu 22.04. The latter version has been used to generate the results.

Table 7: Performance (in µs) and code size (in bytes) for the fastest and smallest implementations of Aes-Gcm, any other primary candidate or finalist, Ascon-128, Ascon-128a, and Ascon-128 at the end of round 2. Performance numbers have been extracted from [47]. The time and area for performing no crypto operation has been removed.

	Time					Area				
	Uno	F1	ESP	F7	R5	Uno	F1	ESP	F7	R5
Aes-Gcm	-	332.8	67.2	35.8	23.7	-	9908	14832	9836	14272
Best primary	1985	59.8	38.7	15.6	15.6	1606	2692	2048	1356	1984
Best finalist	1448	59.6	34.9	11.9	15.7	1604	1804	2048	1356	1728
Ascon-128	2337	76.7	22.3	13.8	8.5	2552	2157	1120	1180	1792
Ascon-128a	1981	66.4	18.4	11.8	7.3	2544	2252	1200	1240	1792
(@ round 2)	2719	85.5	65.8	15.0	8.5	3676	2556	3072	1640	1792

Table 8: Code size (in bytes) for the smallest implementations of Ascon for AEAD, hashing, and combined implementations. Numbers have been generated using [24].

	ATmega328P	Cortex-M0	Cortex-M4
Ascon-128	2774	1298	1342
Ascon-128a	3062	1334	1418
Ascon-Hash	1738	776	718
Ascon-Hasha	1738	776	718
Ascon-128 + Ascon-Hash	3198	1456	1542
Ascon-128a + Ascon-Hasha	3524	1510	1638

interleaving instructions (ARM Custom Datapath Extension, RISC-V Bitmanip Extension). To demonstrate this potential improvement, implementations of algorithms without bit interleaving have also been submitted to the benchmarking platforms.

3.2 Hardware implementations

For Ascon, many different hardware implementations exist. In the case of implementations without countermeasures against implementation attacks, we have the following recent implementations:

- The implementations by Robert Primas (`https://github.com/ascon/ascon-hardware`) providing round-based implementations for 1, 2, 3, and 4 rounds per clock cycle for authenticated encryption and hash variants of Ascon and combinations of them.
- Implementations from Virginia Tech by Behnaz Rezvani advised by William Diehl (`https://github.com/vtsal/ascon_lwc_aead_hash`).
- Implementations from George Mason University by Rishub Nagpal advised by Kris Gaj and Jens-Peter Kaps (`https://github.com/GMUCERG/Ascon`).
- Implementations from George Mason University by Kamyar Mohajerani advised by Kris Gaj and Jens-Peter Kaps (`https://github.com/kammoh/bluelight/tree/api3/Ascon`)
- A fast and lightweight Ascon round instruction for 32-bit ARM/RV32 by Stefan Steinegger and Robert Primas, able to achieve a 50x speedup with only 4.7kGE by reusing 10 registers of the RI5CY CPU [53].

3.2.1 Hardware benchmarks

A good benchmark of NIST LWC entries and AES-GCM can be found in [1]. This benchmark is done for 5 different cell libraries and 2 different design flows. We think that this work gives a good overview of the relative performance of the various candidates and refer to the very informative figures for further details. However, since the benchmark seems to be non-continuous, it is only a snapshot of the then available (submitted) implementations and does not picture improved implementations of candidates and design tweaks of some of the competitors of ASCON.

3.2.2 FPGA benchmarks

To compare ASCON with other candidates including AES-GCM on FPGAs, we rely on third-party results [40]. Please note that the benchmark [40] was performed for 2nd round candidates. Hence, newer implementations and the effects of changes in the specifications of some competitors are not considered. As we can see in Table 10 and Table 9, ASCON performs best, or close to best on many platforms.

Table 9: Throughput ([Mbit/s]) of best ASCON implementations compared to best other candidate and AES-GCM. Only Plaintext [40].

FPGA	Xilinx Artix®-7		Intel® Cyclone® 10 LP		Lattice ECP5	
length	16 Byte	long	16 Byte	long	16 Byte	long
AES-GCM	–	2455.30	–	1407.50	–	1258.60
Best other	601.40	5458.30	318.30	2665.80	**263.90**	**2222.50**
ASCON-128	637.30	3744.00	**367.10**	2157.00	243.00	1427.50
ASCON-128a	**699.70**	**6297.60**	337.10	**3031.00**	245.90	1666.30

Table 10: Throughput ([Mbit/s]) of best ASCON implementations compared to best other candidate and AES-GCM. Associated data and Plaintext [40].

FPGA	Xilinx Artix®-7		Intel® Cyclone® 10 LP		Lattice ECP5	
length	16 Byte	long	16 Byte	long	16 Byte	long
AES-GCM	–	2700.80	–	1548.30	–	1384.40
Best other	601.40	**6569.80**	**318.30**	**3563.20**	**263.90**	**3148.50**
ASCON-128	544.60	3744.00	313.70	2157.00	207.60	1427.50
ASCON-128a	**629.80**	6297.60	303.10	3031.00	205.40	1666.30

3.3 Implementations that protect against side channel attacks

ASCON has been designed with protected implementations in mind. For example, most implementations are constant time by default which is shown in the results at https://bench.cr.yp.to/. ASCON is also very flexible, and several hardware and software options can be used to protect ASCON against side-channel attacks.

Another advantage of ASCON is its small state size. Therefore, masking can be implemented more efficiently. On RV32, two shares easily fit in the register file. Software shares can also be stored and computed in a rotated form with limited performance impact on ARM platforms. This reduces the side-channel leakage on real devices. Furthermore, the non-linear ASCON S-box can be efficiently masked with fewer instructions and less (or no fresh) randomness using the Toffoli gate, as discussed in [18, 52].

Additionally, Ascon provides the option to use leveled implementations proposed by Adomnicai et al. [2] and recently discussed by Bellizia et al. [6]. In leveled implementations, a higher protection order is used for the initialization and finalization than for other parts of an algorithm. In particular, in scenarios where the number of decryption failures or queries can be limited or plaintext confidentiality is not critical, such implementations might be of interest. For long messages, such implementations can achieve performances similar to that of unprotected implementations.

3.3.1 Protected software implementations

A preliminary comparison of plain, leveled and masked implementations of Ascon with bitsliced AES-CTR is shown in Table 11. Leveled implementations (2-1-2) use 2 shares for Init/Final and 1 share for AD/PT/CT processing. Unfortunately, not many side-channel protected software implementations of Aes-Gcm are publicly available to provide a detailed comparison. Combining the masked Aes-Ctr and plain Aes-Gcm performance suggest that masked Ascon is at least twice as fast.

Additionally, protected implementations of Ascon have been submitted to the benchmarking platforms. At the time of writing this report, the 2-share protected ARMv6 implementation was even faster and smaller than most plain primary finalists on the F7 device at [47]. However, note that this dedicated implementation on this device has not been security-evaluated and device-specific fixes might be needed.

Table 11: Software performance of plain and masked bitsliced Aes-Ctr and plain, leveled, and masked Ascon on 32-bit ARM Cortex-M4 microcontrollers in cycles/byte. Underlined values have been security evaluated by [17] and/or using the framework of [21].

GCC flags	asm	-O2	-Os	-O2	-Os	-O2	-Os	-O2	-Os
# shares	1	1	1	2-1-2	2-1-2	2	2	3	3
Ascon-128 [24]	59	69	97	94	102	287	369	531	722
Ascon-128a [24]	42	47	71	67	73	196	252	362	491
Aes-Ctr [50, 51]	101					545			

3.3.2 Protected hardware implementations

Regarding hardware implementation that include protection mechanisms against side channel attacks, two teams have submitted to the ongoing side-channel evaluation orchestrated by the team of the George Mason University (see `https://cryptography.gmu.edu/athena/LWC/LWC_Finalists_protected_HW_implementations.html`):

- Nicolai Müller has automatically generated masked hardware [33] of all finalists including Ascon using PINI-secure composable HPC2 [13] at `https://github.com/Chair-for-Security-Engineering/LWC-Masking`.
- Masked hardware implementations have been published by Robert Primas and Rishub Nagpal at `https://github.com/ascon/ascon-hardware-sca`. The implementation has been successfully formally verified in the glitch-extended probing model using the tool Coco [28].

Note that the implementations have not been optimized for low randomness requirements. Similar to masked software implementations, Ascon can be masked with minimal (or no) fresh randomness, depending on the number of masks and required protection order.

Recently, also a new method for low-latency masking has been proposed, implementing Ascon's permutation as an example [41]. Finally, the side-channel security of Ascon and other LWC finalists providing mode-level security has been evaluated in [57].

4 Platforms and metrics in which Ascon performs better than current NIST standards

4.1 Authenticated encryption with associated data

If we compare Ascon's authenticated encryption schemes with the current widely used NIST standard Aes-Gcm [20, 44], Ascon has the benefit that we could learn from the issues of Aes-Gcm in practical deployment. This includes the increased relevance of side-channel attacks [7, 34] and also resilience against implementation mistakes [32].

Ascon has been designed to allow naturally for fast, constant-time, and bitsliced software implementations, because non-constant-time table-based implementations have turned out to be very vulnerable against timing-based attacks [7]. As a consequence, third-party benchmarks [45, 46, 47] show that Ascon is faster than Aes-Gcm on a wide variety of platforms ranging from 8 to 64-bits.

Since the round function of Ascon allows for a very compact description as binary circuit and the state-size of 320 bits is small for an AEAD-scheme aiming at 128-bit security, hardware implementations are also very efficient. For hardware/ASIC implementations, third-party benchmarks [1] show an advantage over Aes-Gcm considering area, energy, and throughput. If we consider masking as a countermeasure against side-channel attacks, the low algebraic degree of 2 of one round of Ascon allows for very efficient masked implementations in hardware [29] and software[2], which is another advantage over Aes-Gcm. Furthermore, Ascon can be masked with almost no or less fresh randomness than Aes-Gcm, as discussed in [18, 52].

Another interesting feature of Ascon is its keyed initialization and finalization. Keying the initialization and finalization protects against trivial key recovery and forgery attacks even if an attacker somehow gets knowledge of an internal state during the data procession of Ascon. This feature increases the resilience of Ascon even against attacks with heavy nonce misuse; in particular, key-recovery still requires significant effort [5, 16]. Furthermore, the keyed initialization and finalization feature allows for leveled implementations, where the masking degree is reduced during the data processing phase to allow for more efficient implementations [6]. Both features are not present in Aes-Gcm, which allows for forgeries after a few repeated nonces [32].

4.2 Hashing

Both SHA-3 [10, 42] and Ascon-Hash are permutation-based designs using the sponge construction [9], bringing in the flexibility to easily define functionality besides classical hashing, like extendable output functions (Ascon-Xof) or message authentication codes [22]. Hence, the main difference in performance is rooted in the underlying permutation. Here, Ascon uses a 320-bit permutation with a maximum of 12 rounds, while SHA-3 is based on the 1600-bit permutation Keccak-f with 24 rounds. Both round functions have a similar number of Boolean operations per processed bit. Due to the massive difference in state-size, Ascon's permutation can be expected to have a much lower area footprint in hardware implementations and less register pressure in software implementations.

Comparing Ascon with Sha-256, we see that Ascon does not have many of the undesirable properties of Sha-256 like message extension. Thus, there is no need of

[2]https://github.com/ascon/simpleserial-ascon

using HMAC when wanting to have keyed functionality that requires at least two calls to Sha-256. Also, Ascon can be used together with KMAC [43] to provide pseudorandom functions (PRFs), message authentication codes (MACs) and keyed hash functions with variable-length outputs. For improved performance, dedicated PRF and MAC modes are possible using Ascon as suggested in [22].

While Sha-256 it is very suitable for software implementations due to the heavy use of modular additions, rotations and XORs, it comes short when considering hardware implementations. For example, a straightforward hardware implementation of Sha-256 using one round per cycle results in a performance of 1 c/b (64 rounds for 64 bytes) and has a state size of 1024 bits. On the other hand, a similar round-based 1 c/b implementation of Ascon-Hasha (8 rounds for 8 bytes) only has a state size of 320 bits, less than one third. Moreover, the round-based implementation can be shared with the AEAD mode of Ascon.

5 Target applications and use-cases for Ascon

Ascon follows a very balanced approach providing excellent performance/size trade-offs for a wide variety of software platforms from high-end to low-end and also for dedicated hardware designs. Furthermore, Ascon can keep its excellent performance even for short messages.

In addition, Ascon has been designed with robustness and implementation attacks in mind. Hence, it allows for masking with a very low overhead [18, 30] and even leveled implementations [6]. Moreover, even if an attacker somehow manages to recover an internal state during data processing (e.g., due to side-channel attacks), this does not directly lead to the recovery of the secret key or to constructing trivial forgeries. These properties of the mode set Ascon apart from many other lightweight designs.

Taking all into account, Ascon is not only highly suited for scenarios where lightweight devices communicate with lightweight devices, but also for scenarios where many lightweight devices communicate with high-end devices (e.g., a back-end server), a typical use-case in many applications including the Internet of Things (IoT). This is especially true in scenarios where protection against side-channel attacks is needed.

In these settings, we consider both Ascon-128 and Ascon-128a to be equally well-suited and secure choices.

Acknowledgments. The authors would like to thank all researchers contributing to the design, analysis, and implementation of Ascon. In particular, we want to thank Hannes Gross and Robert Primas for all their support and various implementations of Ascon.

References

[1] M. D. Aagaard and N. Zidaric. “ASIC Benchmarking of Round 2 Candidates in the NIST Lightweight Cryptography Standardization Process”. Cryptology ePrint Archive, Paper 2021/049. 2021. URL: https://eprint.iacr.org/2021/049.

[2] A. Adomnicai, J. J. Fournier, and L. Masson. “Masking the Lightweight Authenticated Ciphers ACORN and Ascon in Software”. Cryptology ePrint Archive, Report 2018/708. https://eprint.iacr.org/2018/708. 2018.

[3] F. Bachmann. “Optimized C and Assembly Implementations for ESP32 and RISC-V”. Bachelor’s Thesis (work in progress). 2022. URL: https://github.com/Ferdi265/ascon-c.

[4] J. Baudrin, A. Canteaut, and L. Perrin. “Practical cube-attack against nonce-misused Ascon”. FSE 2022 Rump Session. 2022. URL: https://youtu.be/avBHsIM_5DA?t=2582.

[5] J. Baudrin, A. Canteaut, and L. Perrin. “Practical cube-attack against nonce-misused Ascon”. NIST Lightweight Cryptography Workshop. 2022. URL: https://csrc.nist.gov/csrc/media/Events/2022/lightweight-cryptography-workshop-2022/documents/papers/practical-cube-attack-against-nonce-isused-ascon.pdf.

[6] D. Bellizia, O. Bronchain, G. Cassiers, V. Grosso, C. Guo, C. Momin, O. Pereira, T. Peters, and F.-X. Standaert. “Mode-Level vs. Implementation-Level Physical Security in Symmetric Cryptography – A Practical Guide Through the Leakage-Resistance Jungle”. In: Advances in Cryptology – CRYPTO 2020. Vol. 12170. LNCS. Springer, 2020, pp. 369–400. URL: https://doi.org/10.1007/978-3-030-56784-2_13.

[7] “Cache-timing attacks on AES”. 2004. URL: https://cr.yp.to/papers.html#cachetiming.

[8] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. “On the Indifferentiability of the Sponge Construction”. In: Advances in Cryptology - EUROCRYPT 2008. Vol. 4965. LNCS. Springer, 2008, pp. 181–197. URL: https://doi.org/10.1007/978-3-540-78967-3_11.

[9] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. “Sponge functions”. Ecrypt Hash Workshop 2007. 2007. URL: http://sponge.noekeon.org/SpongeFunctions.pdf.

[10] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. “The Keccak Reference”. Submission to NIST (Round 3). 2011. URL: https://keccak.team.

[11] F. Campos, L. Jellema, M. Lemmen, L. Müller, D. Sprenkels, and B. Viguier. “Assembly or Optimized C for Lightweight Cryptography on RISC-V?” In: CANS 2020. Vol. 12579. LNCS. Springer, 2020, pp. 526–545.

[12] L. Cardoso and J. Groszschaedl. “An Evaluation of the Multi-Platform Efficiency of Lightweight Cryptographic Permutations”. In: SecITC 2021. Vol. 13195. LNCS. Springer, 2022.

[13] G. Cassiers, B. Grégoire, I. Levi, and F.-X. Standaert. “Hardware Private Circuits: From Trivial Composition to Full Verification”. Cryptology ePrint Archive, Paper 2020/185. https://eprint.iacr.org/2020/185. 2020. URL: https://eprint.iacr.org/2020/185.

[14] D. Chang, D. Hong, and J. Kang. “Conditional Cube Attacks on Ascon-128 and Ascon-80pq in a Nonce-misuse Setting”. IACR Cryptology ePrint Archive, Report 2022/544. 2022. URL: https://eprint.iacr.org/2022/544.

[15] D. Chang, J. Kang, and M. S. Turan. “A New Conditional Cube Attack on Reduced-Round Ascon-128a in a Nonce-misuse Setting”. NIST LWC Workshop 2022. 2022.

[16] D. Chang, J. Kang, and M. S. Turan. “A New Conditional Cube Attack on Reduced-Round Ascon-128a in a Nonce-misuse Setting”. NIST Lightweight Cryptography Workshop. 2022. URL: https://csrc.nist.gov/csrc/media/Events/2022/lightweight-cryptography-workshop-2022/documents/papers/a-new-conditional-cube-attack-on-reduced-round-ascon-128a.pdf.

[17] Cryptographic Engineering Research Group, George Mason Univeristy. “Assignments, Commitments, and Reports of the LWC Side-Channel Security Evaluation Labs Targeting Software Implementations”. 2022. URL: https://cryptography.gmu.edu/athena/LWC/Lab_Implementation_Matching_SW.html.

[18] J. Daemen, C. Dobraunig, M. Eichlseder, H. Gross, F. Mendel, and R. Primas. “Protecting against Statistical Ineffective Fault Attacks”. In: IACR Transactions on Cryptographic Hardware and Embedded Systems 2020.3 (2020), pp. 508–543.

[19] J. Daemen, B. Mennink, and G. Van Assche. “Full-State Keyed Duplex with Built-In Multi-user Support”. In: Advances in Cryptology - ASIACRYPT 2017. Vol. 10625. LNCS. Springer, 2017, pp. 606–637. URL: https://doi.org/10.1007/978-3-319-70697-9_21.

[20] J. Daemen and V. Rijmen. “The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition”. Information Security and Cryptography. Springer, 2020. URL: https://doi.org/10.1007/978-3-662-60769-5.

[21] F. Dietrich, C. Dobraunig, F. Mendel, R. Primas, and M. Schläffer. “Masked Ascon Software Implementations”. 2022. URL: https://github.com/ascon/simpleserial-ascon.

[22] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. “Ascon PRF, MAC, and Short-Input MAC”. Cryptology ePrint Archive, Paper 2021/1574. https://eprint.iacr.org/2021/1574. 2021. URL: https://eprint.iacr.org/2021/1574.

[23] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. “Ascon v1.2 (Submission to NIST)”. NIST Finalist, https://csrc.nist.gov/Projects/lightweight-cryptography/finalists. 2021.

[24] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. “Reference and optimized C and ASM implementations of Ascon”. 2022. URL: https://github.com/ascon/ascon-c.

[25] A. D. Dwivedi, M. Klouček, P. Morawiecki, I. Nikolič, J. Pieprzyk, and S. Wójtowicz. “SAT-based Cryptanalysis of Authenticated Ciphers from the CAESAR Competition”. In: SECRYPT ICETE 2017. SciTePress, 2017, pp. 237–246. IACR: 2016/1053.

[26] J. Erlacher, F. Mendel, and M. Eichlseder. “Bounds for the Security of Ascon against Differential and Linear Cryptanalysis”. In: IACR Transactions on Symmetric Cryptology 2022.1 (2022), pp. 64–87.

[27] D. Gérault, T. Peyrin, and Q. Q. Tan. “Exploring Differential-Based Distinguishers and Forgeries for ASCON”. In: IACR Transactions on Symmetric Cryptology 2021.3 (2021), pp. 102–136.

[28] B. Gigerl, V. Hadzic, R. Primas, S. Mangard, and R. Bloem. “Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs”. In: USENIX Security Symposium. USENIX Association, 2021, pp. 1469–1468.

[29] H. Groß and S. Mangard. “Reconciling d+1 Masking in Hardware and Software”. In: Cryptographic Hardware and Embedded Systems - CHES 2017. Vol. 10529. LNCS. Springer, 2017, pp. 115–136. URL: https://doi.org/10.1007/978-3-319-66787-4_6.

[30] H. Gross and S. Mangard. “A unified masking approach”. In: Journal of Cryptographic Engineering 8.2 (2018), pp. 109–124.

[31] K. Hu and T. Peyrin. “Revisiting Higher-Order Differential(-Linear) Attacks from an Algebraic Perspective: Applications to Ascon, Grain v1, Xoodoo, and ChaCha”. NIST LWC Workshop 2022. 2022.

[32] A. Joux. “Authentication Failures in NIST version of GCM”. URL: https://csrc.nist.gov/CSRC/media/Projects/Block-Cipher-Techniques/documents/BCM/Comments/800-38-series-drafts/GCM/Joux_comments.pdf.

[33] D. Knichel, A. Moradi, N. Müller, and P. Sasdrich. “Automated Generation of Masked Hardware”. Cryptology ePrint Archive, Paper 2021/569. https://eprint.iacr.org/2021/569. 2021. URL: https://eprint.iacr.org/2021/569.

[34] P. C. Kocher, J. Jaffe, and B. Jun. "Differential Power Analysis". In: Advances in Cryptology - CRYPTO '99. Vol. 1666. LNCS. Springer, 1999, pp. 388–397. URL: https://doi.org/10.1007/3-540-48405-1_25.

[35] G. Leander, C. Tezcan, and F. Wiemer. "Searching for Subspace Trails and Truncated Differentials". In: IACR Transactions on Symmetric Cryptology 2018.1 (2018), pp. 74–100.

[36] C. Lefevre and B. Mennink. "Tight Preimage Resistance of the Sponge Construction". Cryptology ePrint Archive, Paper 2022/734, to appear in CRYPTO 2022. 2022. URL: https://eprint.iacr.org/2022/734.

[37] Y. Li, G. Zhang, W. Wang, and M. Wang. "Cryptanalysis of round-reduced ASCON". In: SCIENCE CHINA Information Sciences 60.3 (2017), p. 38102.

[38] Z. Li, X. Dong, and X. Wang. "Conditional Cube Attack on Round-Reduced ASCON". In: IACR Transactions on Symmetric Cryptology 2017.1 (2017), pp. 175–202. IACR: 2017/160. URL: https://github.com/lizhengcn/Ascon_test.

[39] R. H. Makarim and R. Rohit. "Towards Tight Differential Bounds of Ascon: A Hybrid Usage of SMT and MILP". In: IACR Transactions on Symmetric Cryptology 2022.3 (2022), pp. 303–340.

[40] K. Mohajerani, R. Haeussler, R. Nagpal, F. Farahmand, A. Abdulgadir, J.-P. Kaps, and K. Gaj. "FPGA Benchmarking of Round 2 Candidates in the NIST Lightweight Cryptography Standardization Process: Methodology, Metrics, Tools, and Results". Cryptology ePrint Archive, Paper 2020/1207. version 2021-02-24. 2020. URL: https://eprint.iacr.org/2020/1207.

[41] R. Nagpal, B. Gigerl, R. Primas, and S. Mangard. "Riding the Waves Towards Generic Single-Cycle Masking in Hardware". In: IACR Transactions on Cryptographic Hardware and Embedded Systems 2022.4 (2022), pp. 693–717. URL: https://doi.org/10.46586/tches.v2022.i4.693-717.

[42] National Institute of Standards and Technology. "FIPS PUB 180-3: Secure Hash Standard". Federal Information Processing Standards Publication 180-3, U.S. Department of Commerce. 2008. URL: http://csrc.nist.gov/publications/fips/fips180-3/fips-180-3_final.pdf.

[43] National Institute of Standards and Technology. "NIST SP 800-185: SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, ParallelHash". NIST Special Publication 800-185, U.S. Department of Commerce. 2016.

[44] National Institute of Standards and Technology. "NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC". Nov. 2007. URL: https://doi.org/10.6028/NIST.SP.800-38D.

[45] NIST LWC team. "Microcontroller Benchmarking". 2022. URL: https://github.com/usnistgov/Lightweight-Cryptography-Benchmarking.

[46] S. Renner, E. Pozzobon, and J. Mottok. "A Hardware in the Loop Benchmark Suite to Evaluate NIST LWC Ciphers on Microcontrollers". In: Information and Communications Security - 22nd International Conference, ICICS 2020. Vol. 12282. LNCS. Springer, 2020, pp. 495–509. URL: https://doi.org/10.1007/978-3-030-61078-4_28.

[47] S. Renner, E. Pozzobon, and J. Mottok. "NIST LWC Software Performance Benchmarks on Microcontrollers". 2022. URL: https://lwc.las3.de/.

[48] R. Rohit, K. Hu, S. Sarkar, and S. Sun. "Misuse-Free Key-Recovery and Distinguishing Attacks on 7-Round Ascon". In: IACR Transactions of Symmetric Cryptology 2021.1 (2021), pp. 130–155.

[49] R. Rohit and S. Sarkar. “Diving Deep into the Weak Keys of Round Reduced Ascon”. In: IACR Transactions on Symmetric Cryptology 2021.4 (2021), pp. 74–99. URL: https://doi.org/10.46586/tosc.v2021.i4.74-99.

[50] P. Schwabe and K. Stoffelen. “All the AES You Need on Cortex-M3 and M4”. In: SAC 2016. Vol. 10532. LNCS. Springer, 2016, pp. 180–194.

[51] P. Schwabe, K. Stoffelen, and M. J. Kannwischer. “Fast AES on ARM Cortex-M3 and M4”. In: (accessed: 09/2022). URL: https://github.com/Ko-/aes-armcortexm.

[52] A. R. Shahmirzadi and A. Moradi. “Second-Order SCA Security with almost no Fresh Randomness”. In: IACR Trans. Cryptogr. Hardw. Embed. Syst. 2021.3 (2021), pp. 708–755. URL: https://doi.org/10.46586/tches.v2021.i3.708-755.

[53] S. Steinegger and R. Primas. “A Fast and Compact RISC-V Accelerator for Ascon and Friends”. In: CARDIS. Vol. 12609. Lecture Notes in Computer Science. Springer, 2020, pp. 53–67.

[54] C. Tezcan. “Analysis of Ascon, DryGASCON, and Shamash Permutations”. In: International Journal of Information Security Science 9.3 (2020), pp. 172–187. URL: https://www.ijiss.org/ijiss/index.php/ijiss/article/view/762.

[55] C. Tezcan. “Truncated, Impossible, and Improbable Differential Analysis of Ascon”. In: ICISSP 2016. SciTePress, 2016, pp. 325–332. IACR: 2016/490.

[56] Y. Todo. “Structural Evaluation by Generalized Integral Property”. In: EUROCRYPT 2015. Vol. 9056. LNCS. Springer, 2015, pp. 287–314. IACR: 2015/090.

[57] C. Verhamme, G. Cassiers, and F.-X. Standaert. “Analyzing the Leakage Resistance of the NIST’s Lightweight Crypto Competition’s Finalists”. In: CARDIS. Lecture Notes in Computer Science. https://perso.uclouvain.be/fstandae/PUBLIS/279b.pdf. Springer, 2022.

[58] R. Zong, X. Dong, and X. Wang. “Collision Attacks on Round-Reduced Gimli-Hash/Ascon-Xof/Ascon-Hash”. IACR Cryptology ePrint Archive, Report 2019/1115. 2019. IACR: 2019/1115.