Grain-128AEAD - Status Document
September, 2020

Martin Hell[1], Thomas Johansson[1], Alexander Maximov[2], Willi Meier[3], Jonathan Sönnerup[1], and Hirotaka Yoshida[4]

[1]Lund University, Sweden
[2]Ericsson AB, Sweden
[3]FHNW, Switzerland
[4]AIST, Japan

1 Introduction

This short note summarizes our implementation efforts for the Grain-128AEAD stream cipher, and also discusses some advantages of using Grain-128AEAD.

2 Software Implementations

The provided reference implementation is, together with the specification document, primarily designed to give an easy to grasp overview of the cipher. No serious optimization attempts are included with the reference implementations. Instead, optimized software implementations are provided separately. The optimizations are discussed in detail in [2] and target the two most constrained architectures used in the FELICS-AEAD framework [1]. We refer to that document for details and provide only a very short summary here.

The implementations have been included in the framework and the results are those delivered by FELICS-AEAD. The two processors used were the 8-bit AVR ATmega 128 and the 16-bit MSP430F1611. Both these architectures provided optimization challenges and we give four different implementations, one for each processor and targeting either fast execution or small code size. The implementations can be found in the FELICS-AEAD framework.

There is often a tradeoff between execution time and code size, so we also provide results for a more balanced choice, where we use the product of the code size and execution time as a metric. The FELICS-AEAD framework includes different scenarios. The results in Table 1 gives our results for a balanced choice between size and speed, and for the scenario with authenticated encryption of 1224 bytes of payload and 40 bytes associated data. In the table, Grain is compared to other algorithm available in the FELICS-AEAD framework. In [2] we give more results, e.g., that Grain-128AEAD can be implemented with code size 1100 bytes on the 8-bit architecture AVR and 926 bytes on the 16-bit architecture MSP.

Table 1: Balanced choices, using smallest Code-size × min total time.

Name	RAM state	Code size	RAM stack	Total time (cycles)	Code size × Total time
AVR balanced choice					
ACORN	37	3024	81	396798	$2^{30.16}$
AES-GCM	228	2338	96	1460677	$2^{31.67}$
ASCON	40	24590	63	53272	$2^{30.29}$
(Alt.) ASCON	40	3724	122	362589	$2^{30.33}$
Grain	49	1734	38	263101	$2^{28.77}$
Ketje-Jr	25	5156	190	311949	$2^{30.58}$
NORX	64	5028	201	124062	$2^{29.22}$
MSP balanced choice					
ACORN	37	1750	64	676228	$2^{30.14}$
AES-GCM	228	1952	126	2174330	$2^{31.98}$
ASCON	40	5572	336	417711	$2^{31.12}$
Grain	50	1358	44	184104	$2^{27.90}$
Ketje-Jr	25	6248	196	335624	$2^{30.97}$
NORX	64	4216	212	71419	$2^{28.17}$

3 Hardware Implementations

The design document included a rough estimate of the hardware performance of Grain-128AEAD. In [3], hardware implementations targeting ASICs were presented and discussed. The code was implemented in VHDL, and synthesized using Synopsys Design Compiler 2013.12 along with a 65 nm library from ST Microelectronics. Both high speed and low power implementations were provided.

3.1 High Speed Implementation

In order to reach high clock speeds, several RTL optimization techniques were utilized, such as Galois transforms, pipelining, optimization of control logic, and unrolling. Different synthesis options were also explored. The synthesis script utilizes clock gating and high speed transistors (LVT).

The best high-speed results are shown in Table 2. The throughput is increased for every level of parallelization at the expense of power consumption. The fastest implementation is the 64 times parallelized one, with a throughput of 33.6 Gb/s, at the cost of a huge area increase and a power increase. A parallelization level of 4 yields the most power efficient implementation. The most area efficient implementation is given when $n = 32$. This is due to the architectural design of the Grain family, which allows for relatively cheap parallelization up to 32x.

Using fixed message lengths, the energy consumption is calculated. We encrypt 1 and 1000 blocks, where each block is 64 bits, and the results are shown in Table 3. Again, a parallelization level of 4 results in the lowest energy consumption, closely followed by the 32x version.

Table 2: Results for the high-speed implementation. The throughput per area is given in kbit/s per GE. The throughput per power is given in Gb/s per mW.

n	**Period**	**Freq.**	**Thrp.**	**Area**	**Power**	**Thrp. / Area**	**Thrp. / Power**
	(ns)	(GHz)	(Gb/s)	(GE)	(mW)		
1	0.40	2.5	1.25	2645	0.25	472	5.00
2	0.43	2.32	2.32	2695	0.23	861	10.09
4	0.47	2.13	4.26	3335	0.29	1277	14.69
8	0.46	2.17	8.68	4448	0.67	1951	12.96
16	0.48	2.08	16.64	7118	1.55	2338	10.74
32	0.64	1.56	24.96	9206	1.78	2710	14.02
64	0.95	1.05	33.60	16958	2.76	1982	12.17

Table 3: Energy consumption for the high-speed implementation.

Energy (nJ)	x1	x2	x4	x8	x16	x32	x64
1 Block	0.064	0.032	0.022	0.025	0.030	0.023	0.026
1000 Blocks	12.85	6.35	4.38	4.95	5.98	4.58	5.26

3.2 Low Power Implementation

When targeting a low power design, a clock period must be defined. Here, we chose a clock frequency of 100 kHz, which is close to proximity cards that run at 125 kHz. The synthesis script utilizes clock gating and low power transistors (HVT).

The results for the low power implementation is shown in Table 4. We see that the power consumption is roughly a factor 1000 lower compared to the high speed implementation, with an area decrease between 10-20%. Even though the power increases for every value of n, the total energy consumption decreases, since the computation can be done in shorter time. The most energy efficient implementation is the 64-parallelized version, but again with a substantial increase of area due to the design not natively supporting parallelization of degree higher than 32.

Table 4: Result for low power implementation at 100 kHz.

n	**Area** (GE)	**Power** (μW)	**Energy** (nJ)	
			1 block	1000 blocks
1	2375	0.23	1.47	296
2	2588	0.28	0.90	180
4	2950	0.29	0.46	93.2
8	3692	0.31	0.25	49.8
16	5053	0.39	0.16	31.3
32	7950	0.46	0.09	18.5
64	13800	0.63	0.06	12.7

4 Use Cases and Advantages

Grain-128AEAD was designed to be an efficient and secure cipher in hardware. However, the cipher performs also well in software on constrained devices, as indicated in Section 2. Advantages speaking in favor of Grain-128AEAD are at least the following:

- The algorithm allows for nice implementation tradeoffs, as it has good performance both in hardware and in software implementations on constrained CPUs. Regarding industrial relevance, Grain-128AEAD can be used in products that allow for both hardware and software implementations. Hardware implementations also have a natural tradeoff between gate complexity and speed.
- Variants of Grain-128AEAD are successfully fielded already. The design is very close to Grain-128a, which is an ISO standard for RFID systems (ISO/IEC 29167-13:2015). In [4], some results of memory-optimized implementations of Grain-128a requiring 84 RAM bytes on ARM Cortex-M3 are presented. Existing implementation knowledge and know-how can be reused with small modifications. This indicates that Grain-128AEAD can be suitable for the use case of PKES (Passive Keyless Entry and Start) system in an automotive domain.
- The Grain family of stream ciphers has received intense cryptanalysis over a long time.
- Grain-128AEAD is the only remaining stream cipher in NIST round 2. It is good with diversity among designs for the remaining candidates.
- We finally highlight some security properties that we believe are in favour of Grain-128AEAD in comparison with ciphers based on the duplex construction. It is often stated in specifications, that in order to fulfill the security claims implementations must make sure that the nonce is never repeated for two encryptions using the same key. Furthermore, it is specified that decrypted plaintexts are only released after successful verification of the tag. We note that for both these assumptions, violating one of them

may lead to a **state recovery** for duplex-based constructions. For a synchronous stream cipher like Grain-128AEAD, however, such violation can at most lead to a leakage of plaintext information, never a state recovery.

In particular, not releasing the plaintext before the tag has been verified may be a large practical burden in some applications and one might find use cases where plaintext has to be released before tag verification takes place. Then a synchronous stream cipher will be more suitable.

Similar arguments are valid in the case when the user excludes the authentication mechanism and only encrypts with an AEAD primitive.

References

[1] Luan Cardoso dos Santos, Johann Großschädl, and Alex Biryukov. FELICS-AEAD: Benchmarking of lightweight authenticated encryption algorithms, 2019. Lightweight Cryptography Workshop.

[2] Alexander Maximov and Martin Hell. Software evaluation of Grain-128AEAD for embedded platforms. Cryptology ePrint Archive, Report 2020/659, 2020. https://eprint.iacr.org/2020/659.

[3] Jonathan Sönnerup, Martin Hell, Mattias Sönnerup, and Ripudaman Khattar. Efficient hardware implementations of Grain-128AEAD. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology – INDOCRYPT 2019*, pages 495–513. Springer International Publishing, 2019.

[4] Yuhei Watanabe, Hideki Yamamoto, and Hirotaka Yoshida. Towards minimizing RAM requirement for implementation of grain-128a on ARM cortex-m3. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 103-A(1):2–10, 2020.