

# Survey and Benchmark of Stream Ciphers for Wireless Sensor Networks

N. Fournel<sup>1</sup>, M. Minier<sup>2</sup>, S. Ubéda<sup>2</sup>

<sup>1</sup>LIP, ENS Lyon

Lyon - France

Nicolas.Fournel@ens-lyon.fr

<sup>2</sup>CITI - INSA de Lyon - ARES INRIA Project

Lyon - France

FirstName.Name@insa-lyon.fr

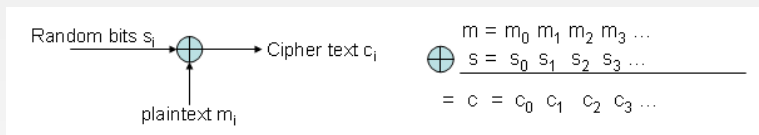
May 10, 2007

# Introduction

- The aim of this paper is to present **some benchmarks of stream ciphers** on a dedicated sensor (as previously done for block ciphers in [Law et al. 2006])
- Why use stream ciphers in WSNs ?
  - they could reach important flows for limited costs
  - They do not propagate errors in communication channels (one time pad)
  - Usually used in wireless communications (WEP, GSM,...)
  - useful for pairwise secure associations in WSNs

# General overview of stream ciphers (1/2)

- “one time pad” use

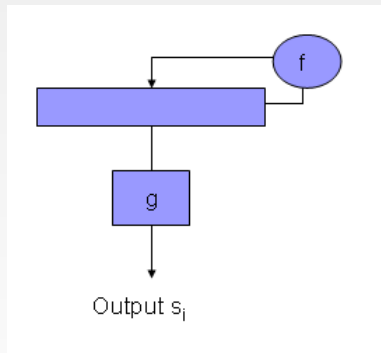


- The **random sequence**  $s_i$  is generated using a **stream cipher** (or pseudo-random generator)
- Initialized with a **common shared key**  $K$  and an  $IV$  that must be different for each message

# General overview of stream ciphers (2/2)

- A stream cipher is composed of three phases

- 1 An **initial state** of length  $L$  (at least 2 times the key length) fulfilled with  $K$  and the  $IV$  value
- 2 A “**warm-up**” phase that produces an internal state from the initial one
- 3 A phase that **generates the pseudo-random sequence** using:
  - a first function  $f$  that **updates** the internal state
  - a function  $g$  to **filter or combine** the state to produce the sequence.



# The stream ciphers studied here

- 1 The classical ones:
  - RC4
  - SNOW v2
  - AES-CTR
- 2 The ones submitted to the ongoing eStream European project, selected for the Focus Phase 2 (Profile 1 Software):
  - **Dragon, HC-128 and HC-256, LEX, Phelix, Py and Pypy, Salsa20 and Sosemanuk**

# The classical ones

- **RC4** [Rivest - 87]
  - uses an internal table of bytes with 256 values updated at each step. Used in WEP, SSL,...
  - **Security**: secure cipher when key schedule is strengthened
- **SNOW v2** [Ekdahl and Johansson - 02]
  - Uses a LFSR over  $GF(2^{32})$  and a FSM generating a 32-bit output
  - **Security**: One attack with an unreachable complexity against SNOW v2
- **AES-CTR** [AES - 01]
  - The AES block cipher used in a the particular mode of operation CTR (cipher the IV with AES + one time pad)
  - **Security**: If you manage to break the AES, you break AES-CTR...

# The eStream ones (1/3)

- **Dragon** [Dawson et al - 05]
  - Uses a NLFSR of 1024 bits with a non linear filtering function. Two versions: key of 128 bits or of 256 bits
  - **Security**: two attacks with unreachable complexities. Until now, Dragon is secure
- **HC-128 and HC-256** [Wu - 05]
  - Uses two secret tables of 1024 32 bits words updated at each clock generating a 32-bit output
  - **Security**: Until now, no attacks against the 2 versions
- **LEX** [Biryoukov - 05]
  - Extracts parts of the internal state of the AES after certain rounds
  - **Security**: have been tweaked to enter in Phase 2. No attack against the new version

# The eStream ones (2/3)

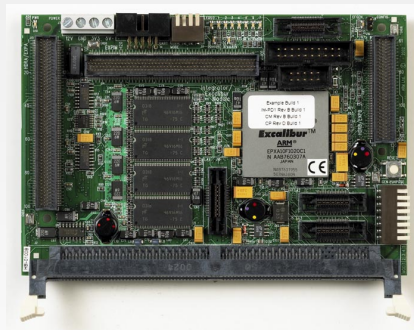
- **Phelix** [Whiting et al. - 05]
  - Uses 9 32-bit words updated 20 times to produce a 32-bit output at each clock
  - **Security**: a very recent attack [Wu et al - 07] with a reachable complexity ( $2^{42}$  operations)
- **Py and Pypy** [Biham, Seberry - 05]
  - Uses same principles than RC4: two rolling tables updated at each clock
  - **Security**: recently successfully attacked [Souradyuti - 06] (complexity:  $2^{85}$ )

# The eStream ones (3/3)

- **Salsa20** [Bernstein - 05]
  - Uses a hash function with input/output of length 64 bytes used in the CTR mode
  - **Security**: no attack against Salsa20
- **Sosemanuk** [Berbain et al - 05]
  - Uses principles of SNOW v2 and some part of the block cipher Serpent
  - **Security**: no attack with a reasonable complexity

# The dedicated platform (1/2)

- 1 The processor: a 32-bit micro-controller, the **ARM922T**
- 2 Two levels of **AMBA bus** for accessing
  - Classical peripherals
  - Memory levels
- 3 **Memories** are organized in a **three level hierarchy**
  - two 8 kB separated caches
  - two 256 kB and two 128 kB scratch pad memories (not used in the benchmarks)
  - main memory: a 128 MB SDRAM



# The dedicated platform (2/2)

- **Operating system:** Mutex [Pétrot et al - 03]
- **Compiler:** GCC targeted to ARM processors
- A full architecture **simulator** (for energy consumption): the open source Skyeye and eSimu

# Methodology used for the benchmarks

- The one provided with the eStream testing framework [De Cannière - 06]
- We then test (in cycles/byte)
  - 1 The time required to perform the **key-setup** and the **IV-setup**
  - 2 Encryption rate for **ciphering a long stream** in chunks of about 4 kB
  - 3 **Packet encryption rate** for packet lengths equal to 40, 576 and 1500 bytes
  - 4 **Agility**: after initiating a large number of sessions, encrypts plaintexts of length 256 bytes

# CPU cycles consumption

Algo.(Key, IV)	cycles/byte				cycles/key Key setup	cycles/IV IV setup	cycles/byte agility
	Stream	40 B	576 B	1500 B			
Copy (80, 80)	2.19	3.72	1.00	7.58	4.40	4.19	7.78
RC4 (128, 0)	26.97	610.95	58.53	33.29	76.41	23581.61	21.24
SNOW v2 (128, 128)	25.08	66.38	16.82	23.71	163.41	2273.35	20.87
AES CTR (128, 128)	206.19	131.52	198.73	195.76	636.49	157.52	202.23
DRAGON (128, 128)	30.89	177.05	69.76	64.91	421.42	4497.61	33.60
HC-256 (128, 128)	27.00	6044.76	446.11	183.17	141.75	198126.10	49.30
HC-128 (128, 128)	19.35	1484.72	112.12	53.70	141.76	58194.93	31.67
LEX (128, 128)	47.07	71.41	40.32	41.92	501.41	1415.57	50.71
Phelix (128, 128)	25.61	90.15	28.36	26.77	1271.42	2154.61	26.99
Py (128, 64)	214.25	349.23	47.58	60.88	7713.83	9327.43	64.40
Pypy (128, 56)	44.78	360.95	103.91	74.72	7713.82	9660.11	73.46
Salsa20 (128, 64)	57.54	84.57	55.05	73.07	367.70	118.07	72.60
SOSEMANUK (128, 64)	14.81	385.63	37.95	30.48	16374.01	1264.09	20.78

**Table:** Number of CPU cycles for the stream ciphers using the testing framework

# Energy consumption

Algo. (Key, IV)	nJ/byte				nJ/key	nJ/IV	nJ/byte
	Stream	40 B	576 B	1500 B	Key setup	IV setup	agility
Copy (80, 80)	38.32	60.85	16.84	142.07	70.54	67.29	145.35
RC4 (128, 0)	465.17	9843.25	948.49	542.06	1243.66	379636.24	354.43
SNOW v2 (128, 128)	438.34	1093.46	280.59	414.20	2656.66	41749.08	365.26
AES CTR (128, 128)	3587.00	2197.89	3437.36	3384.26	11378.81	2861.89	3499.45
DRAGON (128, 128)	514.26	2912.69	1144.53	1064.58	6846.80	74109.24	575.67
HC-256 (128, 128)	471.39	102473.69	7577.28	3112.48	2540.02	2705307.85	864.11
HC-128 (128, 128)	342.29	24264.78	1838.04	897.21	2540.20	950661.16	559.97
LEX (128, 128)	804.03	1186.80	670.42	714.16	8250.66	23850.60	868.13
Phelix (128, 128)	421.15	1470.51	461.14	454.71	20622.78	35111.32	461.26
Py (128, 64)	3894.22	5822.63	827.52	1101.62	145194.31	154181.03	1141.65
Pypy (128, 56)	817.35	6008.43	1859.92	1361.36	145194.15	161834.16	1300.67
Salsa20 (128, 64)	952.19	1394.11	907.17	1275.82	6884.19	2215.93	1268.12
Sosemanuk (128, 64)	247.93	6727.04	648.50	528.97	286119.29	20860.01	365.30

**Table:** Number of nJ for the stream ciphers using the testing framework

# Code and data memory sizes

Algo.	Empty	Copy	RC4	SNOW v2.0	AES-CTR	DRAGON	HC-256
Code size	4992	5040	6064	11152	17456	8512	14432
Data size	480	752	692	6836	13020	2740	692
Algo.	HC-128	LEX	Phelix	Py	Pypy	Salsa20	SOSEMANUK
Code size	12496	13072	9968	8736	8512	6560	21968
Data size	692	5852	724	35248	35248	724	3164

**Table:** Code Memory Requirements and Data Memory Requirements in bytes and in decimal notations

# Analysis (1/2)

- Most of stream ciphers stay more efficient than **AES-CTR**
  - Code and data memory sizes of AES-CTR is **among biggest**
  - → So using stream ciphers in sensor network applications could be a good solution **to achieve high encryption speed** in high constraint environments.

# Analysis (2/2)

- **Comparison** between our benchmarks and the ones given of the eStream page for traditional architectures (Pentium 4,...)
  - on our platform, **Py and Pypy** do no longer work rapidly whereas SNOW v2.0, SOSEMANUK or HC-128 stay relatively fast
  - → comes from the **intrinsic structure** of the ciphers Py and Pypy: use of two rolling tables of 256 bytes with many memcopy
  - → the number of **DC-misses** is huge
  - number of cycles required by the **Keysetup of SOSEMANUK** is very huge because the code is in fact optimized for more powerful architecture (unrolled loops)

# Conclusion

- Benchmarks on similar processors presented at SASC 2007 in february:
  - [Paar et al - 07]: benchmarks of some of them on **8-bit AVR micro-controllers**
  - [Lauradoux - 07]: benchmarks with optimized code of some of them on an **ARM920T**
- Further works:
  - Provide some benchmarks for the same ciphers on a **MSP430 16-bit micro-controller**
- eStream entered in its 3rd phase: 8 algorithms selected
  - CryptMT (CryptMT Version 3), Dragon, HC (HC-128 and HC-256), LEX (LEX-128, LEX-192 and LEX-256) NLS (NLSv2, encryption-only), Rabbit, Salsa20, SOSEMANUK