On Lightweight Security for Constrained Environments

Sönnerup, Jonathan

2020

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Sönnerup, J. (2020). *On Lightweight Security for Constrained Environments*. [Licentiate Thesis, Department of Electrical and Information Technology]. Lunds Universitet/Lunds Tekniska Högskola.

Total number of authors:
1

On Lightweight Security for Constrained Environments

Jonathan Sönnerup

LUND
UNIVERSITY

ISBN 978-91-7895-576-3 (printed)
ISBN 978-91-7895-577-0 (electronic)
Series of licentiate and doctoral theses
No. 131
ISSN 1654-790X

Jonathan Sönnerup
Department of Electrical and Information Technology
Lund University
Box 118
SE-221 00 Lund
Sweden

Typeset using LaTeX.
Printed in Sweden by Tryckeriet i E-huset, Lund, 2020.

Abstract

The market of connected devices, IoT devices in particular, is hotter than ever. Today, lightweight IoT devices are used in several sectors, such as smart cities, smart homes, healthcare, and the manufacturing industry. IoT solutions help increase productivity by predictive maintenance and resource management in the industry. Devices with voice interfaces are spreading rapidly in the home automation markets. Hospitals utilize these "smart" devices to monitor patients and present diagnostics data, aiding physicians in their work.

It is safe to say that we will be surrounded with more and more connected devices. This opens up to potential attacks, where adversaries may try to disrupt critical services or steal sensitive information. To combat this, data needs to be secured in different ways. This dissertation presents cryptographic algorithms and their performance in constrained environments.

First, a new lightweight cryptographic algorithm, Grain-128AEAD, is presented. Grain-128AEAD is a stream cipher designed to be implemented in hardware at a low cost while still being fast. The new design improves on earlier versions by making previous attacks more difficult.

Next, Grain-128AEAD is implemented in hardware using multiple optimization techniques to fit different criteria. Trade-offs between throughput, power, and area are evaluated to analyze the suitability for both constrained devices but also for server back-ends.

Finally, the overhead when adding confidentiality and authenticity for communication in an IoT device is evaluated. Here, modern lightweight protocols are utilized in multiple use-cases to give an overview of the overhead in terms of bytes, time, and energy.

Acknowledgements

They say that when you decide to become a Ph.D. student, you do not chose your subject, you chose your supervisor. At least, that was the case for me. The first course I ever took in security was taught by Martin. His deep knowledge and enthusiasm in the subject led me to slowly change my major from hardware design to security. Multiple courses later along with a master's thesis, I am now writing my dissertation, with Martin as my main supervisor. I can not express anything other than my fullest gratitude for giving me this opportunity to pursue this degree. I would like to thank my assistant supervisors Paul Stankovski Wagner and Martin Höst for their guidance and interesting discussions.

I want to thank all my colleagues and friends in the Security and Crypto group at the EIT department for all the discussions during lunch and fika, and other random small-talk in the corridors. I would like to thank some people in particular. Erik Mårtensson was the first colleague with whom I had the pleasure of sharing office. However, that did not last long. After many, many hours of talking and discussing, not getting any work done, we eventually had to split, albeit only to an office next door. At one point in time, Erik was officially banned from my new office, due to overly excessive writing of mathematical formulae on my whiteboard. Even though we had our differences, I have always enjoyed his company during our travels around the world. We were the Jim and Dwight in the office. I want to thank Linus Karlsson for being a good role model. He is always up-to-date with the latest technology, knowledge that he happily shares with others. I have spent many hours in Linus's office discussing subjects from teaching to the internals of C compilers, but also hiding from Erik. Finally, I would like to thank Jing Yang for putting up with me during these years. I never gave her an easy time with all the jokes and pranks, for which I am not sorry. During our travels, she kept me and Erik in check, while still always being happy and kind.

I wish to thank the administrative and technical staff for all help during these years. Special thanks to Elisabeth Nordström who had to endure my rants about the travel expenses, Erik Jonsson for letting me degrade the security of the deparment by allowing me to open ports in the firewall on a daily basis, and Bertil Lindvall for helping me out when I needed equipment and for all joyful discussions ranging from farming to Breaking Bad.

Finally, I want to thank all my friends and family who have helped and supported me during my years as a Ph.D. student. In particular, I want to mention all the crazy projects with Christoffer, and the philosophical discussions with Umar. To my mom and dad for raising me to become who I am today, to my grandparents who always believed in me, and to my loving girlfriend, thank you for your love and understanding.

Jonathan
Lund, May 2020

Contribution Statement

The following papers are included in this dissertation:

Paper I Martin Hell, Thomas Johansson, Willi Meier, Jonathan Sönnerup, and Hirotaka Yoshida. "An AEAD Variant of the Grain Stream Cipher". In *Codes, Cryptology and Information Security: Third International Conference, C2SI 2019, Rabat, Morocco*. Springer Nature Switzerland AG. pp. 55-71.

Paper II Jonathan Sönnerup, Martin Hell, Mattias Sönnerup, and Ripudaman Khattar. "Efficient Hardware Implementations of Grain-128AEAD". In *Progress in Cryptology – INDOCRYPT 2019 - 20th International Conference on Cryptology Proceedings, Hyderabad, India*. Springer Gabler. pp. 495-513.

Paper III Pegah Nikbakht Bideh, Jonathan Sönnerup, and Martin Hell. "Energy Consumption for Securing Lightweight IoT Protocols". This paper has been submitted but not yet published.

The responsibilities Jonathan Sönnerup had in each paper are summarized in the table below:

Paper	*Writing*	*Concepts*	*Implementation*	*Evaluation*
I	partial	partial	YES	-
II	YES	YES	yes	YES
III	yes	YES	YES	yes

Capital letters indicate roles where Jonathan Sönnerup had the primary responsibility.

In Paper I, Jonathan created a C implementation of the design, both a reference implementation and an optimized version. He contributed with implementation ideas and approaches, and parts of the hardware implementation, and wrote the corresponding section in the paper.

In Paper II, Jonathan had the primary responsibility for writing the paper. He contributed with optimization ideas and approaches, and parts of the implementation. Together with the other authors, he analyzed and evaluated the results.

In Paper III, Jonathan's primary responsibility was the implementation of the system and test suite and the embedded system, i.e., the client-side. Together with the other authors, he defined the overall system along with evaluating the results, where he was responsible for MQTT.

A further description of the papers' contributions *to the research field* is presented in Section 3.1.

Other Contributions

The following peer-reviewed publications have also been published during my Ph.D. studies, but are not included in this dissertation.

- Martin Höst, Jonathan Sönnerup, Martin Hell, and Thomas Olsson. "Industrial Practices in Security Vulnerability Management for IoT Systems – an Interview Study". In *International Conference on Software Engineering Research and Practice, SERP'18, Las Vegas, United States*, pp. 61–67.

- Jonathan Sönnerup and Martin Hell. "Evaluating Security of Software Through Vulnerability Metrics". In *International Conference on Security and Management, SAM'18, Las Vegas, United States*, pp. 79–85.

- Alexander Cobleigh, Martin Hell, Linus Karlsson, Oscar Reimer, Jonathan Sönnerup, and Daniel Wisenhoff. "Identifying, Prioritizing and Evaluating Vulnerabilities in Third Party Code". In *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW), Stockholm, Sweden*. pp. 208–211, IEEE.

- Sara Gunnarsson, Peter Larsson, Sara Månsson, Erik Mårtensson, Jonathan Sönnerup. "Engaging Students using GitHub as a Learning Management System". In *Lunds universitets pedagogiska utvecklingskonferens 2017, Lund, Sweden*. pp. 63–70.

Contents

Abstract iii

Acknowledgements v

Contribution Statement vi

Contents ix

1 Introduction 1
- 1.1 Dissertation Outline 2

2 Background 3
- 2.1 Constrained Devices and Lightweight Protocols 3
- 2.2 Cryptography 12
- 2.3 Hardware Design 23

3 Contributions and Conclusions 31
- 3.1 Contributions 31
- 3.2 Conclusions 33

References 35

Included Publications 43

I An AEAD Variant of the Grain Stream Cipher 45
- 1 Introduction 45
- 2 Design Details 47
- 3 Design Rationale 50
- 4 Security Analysis 53

5 Implementation 56
6 Conclusions 58
References 58
Appendix 63

II Efficient Hardware Implementations of Grain-128AEAD 65
1 Introduction 65
2 Grain-128AEAD 67
3 A Straightforward Approach 70
4 RTL Level Optimizations 71
5 Synthesis Level Optimization 76
6 Synthesis Results 77
7 Conclusions 82
References 83

III Energy Consumption for Securing Lightweight IoT Protocols 85
1 Introduction 86
2 Related Work 87
3 CoAP and MQTT 88
4 Experimental setup 89
5 Methodology and Use Case 91
6 Results and Discussion 93
7 Conclusions 101
References 102

Introduction

Ever considered that your smart-home could kill you? While that may be an exaggeration, connecting random things to the Internet might pose a significant problem.

Today, more and more "smart" devices are being manufactured and sold, both in industrial markets and to regular households. Heating, Ventilation, and Air Conditioning (HVAC) is one example of industrial entities transformed by the latest IoT technologies. Adding connectivity allows for a vast flow of information, both to and from the HVAC devices, such as weather feeds, sensor data. This may be used to perform data analysis on a larger scale, which may help reducing power consumption and increase efficiency. Connectivity also allows for remote management of the complete system.

Another example is self-driving cars. Connecting cars, or general automotive vehicles, to the Internet allows for distribution of software updates in real-time. Manufacturers may also analyze the performance and usage of the vehicles and adjust properties accordingly. Connectivity would aid the self-driving system by gathering relevant data of its surroundings, preventing potential accidents. It is not difficult to see that even though the benefits are many, by allowing remote communication, an attacker could send bogus data or also gain access to the control mechanisms in the car, allowing the attacker to cause fatal incidents.

A less obvious example of abusing an IoT device might be a temperature system, including sensors and heaters, in a villa controllable from a smartphone app. An attacker could use the current temperature information to deduct if the house is empty, to break in. If the house is equipped with an alarm system, this too may be exploited.

What, then, needs to be done to protect companies and households from having their IoT devices exploited? There is little surprise that the answer to this question is cybersecurity. However, there is no single solution to all problems. The infrastructure and computational resources of self-driving cars are entirely different from temperature sensors. For the smallest of devices, there are strict requirements on speed, physical area, and power consumption. Being that most smaller devices run on batteries, they can not afford any high-power solutions. Instead, one must design cryptographic primitives suitable in a constrained environment, without weakening the security. The design aspect ranges from the physical hardware design and implementation of circuits to the software that utilizes the hardware components.

1.1 Dissertation Outline

This dissertation is organized as follows. The following chapter, Chapter 2, introduces the topics related to the research conducted in this dissertation.

Section 2.1 presents the possibilities and limitations of constrained devices and their role in modern society. Here, we also present some current lightweight protocols adapted to keep the overhead and power consumption at a minimum. The section introduces the concepts used in Paper III. In Section 2.2, the area of cryptography is presented with a focus on symmetric ciphers, especially stream ciphers. The area of cryptography is the basis of all papers in this dissertation. Finally, in Section 2.3, the reader is introduced to the process of designing digital circuits, i.e., hardware design, and how to implement digital logic running at high speed and consuming low power. This section is closely related to Paper II, where a stream cipher is implemented in hardware.

Chapter 3 summarizes the contributions of the research, followed by conclusions.

Finally, the second part of this dissertation contains the publications.

Background

Today, products such as watches, clothes[1], smart grids, and industrial sensor networks are being connected to the Internet, aggregating and exchanging data. These products are commonly referred to as the Internet of Things, or IoT. Some of these products are made up of small and cheap embedded devices, such as microcontrollers (MCUs). Being connected to the Internet poses a risk for the devices, due to the exposure to attackers. To avoid exploitation and data leakage, one needs to implement security measures.

Such small devices usually lack the resources to perform heavy computation. Thus, to prevent or mitigate attacks, careful design and implementation is a must, ranging from the design process from the hardware chip to the selection of cryptographic algorithms and software implementation.

In this chapter, we introduce the general concepts behind constrained devices, cryptography, and hardware design.

2.1 Constrained Devices and Lightweight Protocols

Following IETF, a device is said to be constrained if it consists of a limited CPU, small memory such as ROM and RAM, low bandwidth, low energy consumption, or a combination of these [BEK14]. IETF also specifies different classes of constrained devices, ranging from class 0, including small sensors that do not directly communicate with the Internet, up to class 2, including devices that can run standard protocols but would benefit from running lightweight protocols due to energy and bandwidth.

Generally speaking, a constrained device can not, or should not, run the *usual* protocols such as TCP/IP, TLS, HTTP, and so on. Instead, many lightweight protocols have been developed to be used in a constrained device, described in Section 2.1.3.

A microcontroller unit (MCU) is a single integrated circuit (IC), comprising of a CPU along with memory and peripherals for I/O. Microcontrollers range from

[1]https://atap.google.com/jacquard/

the smallest architectures such as Atmel AVR 8-bit products, e.g., ATmega8A, to Espressif 32-bit microcontrollers with integrated WiFi and support for Over-the-Air (OTA) updates, e.g., ESP32. An MCU typically consists of the following:

- A central processing unit, CPU.
- Flash memory, for storing program instructions and non-volatile data.
- Volatile memory, RAM, for temporary data storage during run-time.
- General purpose input/output, GPIO, configurable to act as either an input, when reading, e.g., sensor data, or an output when controlling, e.g., LEDs or motors.
- Serial communication protocols, such as UART, I^{2}C, and SPI.
- Timers and PWM generators.
- Analog-to-Digital converters, ADCs.

Some microcontrollers also include crypto modules, i.e., hardware-accelerated cryptographic functions such as RSA, AES, ECC, RNG, and hash functions. Having these algorithms run in hardware instead of software saves energy and increases throughput. The general architecture of a microcontroller is depicted in Figure 2.1. Some common microcontrollers, along with an excerpt of their specifications, are shown in Table 2.1. As seen in the table, the devices are quite constrained in terms of speed and memory, with the ARM Cortex-M0+ being the most lightweight processor.

Table 2.1: Comparison of popular IoT development boards. All have an operating voltage of 3.3 V.

Board	CPU	Arch	Clock	RAM	Connectivity
Particle Photon	ARM Cortex M3	32-bit	120 MHz	128 KiB	WiFi
Particle Argon	ARM Cortex-M4F	32-bit	64 MHz	256 KiB	WiFi + BLE
ESP32	Tensilica Xtensa	32-bit	240 MHz	520 KiB	WiFi + BLE
Arduino MKR1000	ARM Cortex-M0+	32-bit	48 MHz	32 KiB	WiFi

It is not only microcontrollers who benefit from hardware acceleration. In 2010, Intel released the first set of CPUs with an instruction set for running AES in hardware, known as AES-NI [Gue06]. AES is such a widespread and common algorithm that the increase in speed and reduction in power consumption outweighs the hardware cost [CLG13].

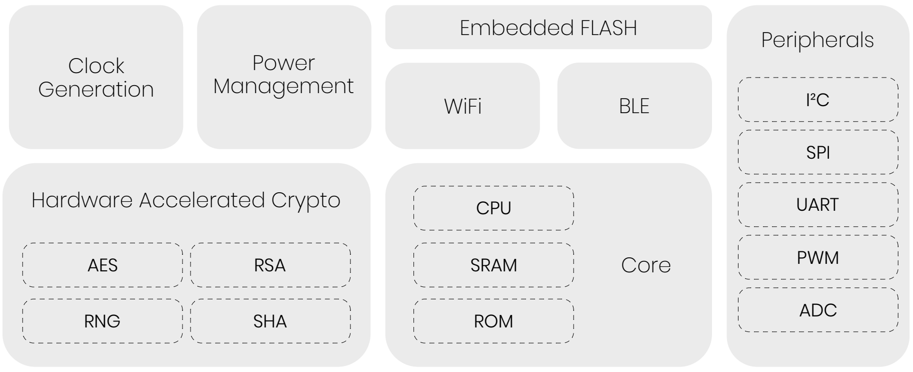


Figure 2.1: A general architecture of a microcontroller.

Today, most microcontrollers are programmed using C/C++, but even Python and Javascript may be used in some cases. However, due to the variety of microcontrollers, the standard implementations of programming languages might not be feasible. Many flavors of standard languages, such as Embedded C [Int08], nesC [Gay+03], and MicroPython[2] have been implemented to better fit embedded systems.

2.1.1 Low-Power Solutions

Since many IoT devices run on batteries, the circuits employ functionality to help reduce power consumption. A common solution is to use a set of sleep modes. A sleep mode can be thought of as a set of active components used in the system. The deeper the sleep, the fewer components are active, lowering the used power.

The technique for powering down certain parts of a circuit is known as clock gating. Clock gating implies that the clock signal is *gated*, i.e., activated or deactivated for some time. By disabling the clock pulses, the digital logic no longer switches states. It is the switching activity that consumes dynamic power.

An example of sleep mode levels is described next, assuming a microcontroller with communication capabilities, like in the architectural overview in Figure 2.1. **Note** that we here express power consumption in terms of current, which is common practice. While it is technically wrong to do so, the power consumption for a microcontroller is proportional to the current, since the voltage is kept constant.

Active mode Here, the device operates normally with all components in the active state. Active mode is the state where the device consumes the most power. It is not unusual for the power consumption to be a few 100 mA. For example, the ESP32 with the communication blocks active consumes around 30 mA

[2] https://docs.micropython.org/en/latest/

in idle state, and around 100 mA during transmission. A Raspberry Pi Zero consumes roughly 100 mA while being 600 mA for Raspberry Pi 4[3].

Sleep mode Communication is one of the most expensive operations in an embedded system. Hence, this is the first part to power down in sleep mode. The CPU is also paused in the sense that it does not execute any instructions. The high-frequency clock generator is active in order to quickly resume regular operation once we exit sleep mode via an interrupt, for example. Peripherals may be active, along with the SRAM, to allow computation and storage via DMA without the need for CPU intervention. A typical value for power consumption in this state is a few mA. An ESP32 in sleep mode consumes ca 1 mA[4].

Deep sleep This is as close to the system being completely shut down as we can get, without actually shutting down the system. The high-frequency clock is powered off, and only critical parts are active, such as the real-time clock and watchdog timer. The content previously stored in CPU registers and RAM is also erased. If some data needs to be saved, one can utilize the non-volatile flash memory. The power consumption in this state may be as low as a few µA. For example, the ATmega328P uses around 1.5 µA at 3 V [Atm].

2.1.2 Over-the-Air Updates

As software is being continuously developed with new features, bug fixes, or security patches, the already deployed systems need to be updated. In systems without communication capabilities, this is usually done by manually flashing a new firmware via UART, SPI, JTAG, or similar, requiring physical access.

Many IoT devices are deployed in areas that are physically cumbersome to access [Bar+08]. Other devices are located in home environments, measuring humidity, temperatures, power consumption, and so on. A home environment may make it hard to update the devices since ordinary residents do not possess the tools nor skills to perform a system update. In such cases, *Over-the-Air* updates may assist.

Over-the-air, or OTA, refers to the wireless medium used to distribute content, such as firmware. With OTA, there is no need for physical presence while performing an update, which solves many of the above problems. However, implementing OTA support is not a trivial task, not only from an implementation point of view but also from a management point of view. Some potential issues are given next.

- You need to handle devices that lose connection and which devices have and have not already been updated.

[3]`https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md`

[4]`https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/`

- Installing a new firmware may cause the system to hang or break.
- There needs to be sufficient space for temporarily storing the latest firmware before it can be applied.

Next, let us look at some known architectures for OTA.

Modern Android implements so-called A/B system updates[5]. Upon update, the new firmware is downloaded and stored in `/data`. After the full firmware is retrieved, it is installed in a partition different from the currently running firmware, called A and B partitions. The installation process does not interrupt the user as the current firmware is running until the user reboots the device. Upon booting, the new firmware is applied. If it fails, the device will load the previous firmware, known as roll-back. Since Android 8.0, it is now possible to use *streaming* A/B updates. This means that the firmware is written directly to partition B without having to store it in `/data` temporarily. To reduce the size of the new firmware, one may use compression algorithms such as Brotli[6].

IETF has proposed an update-architecture for the IoT [Mor+19]. The document highlights that the update process must ensure that the firmware has integrity protection to prevent modification or injection attacks. The update process must be able to provide confidentiality protection since the first step of an attacker is usually to obtain the firmware and reverse engineer it. Other requirements include:

- The firmware distribution mechanism must support a variety of protocols, such as UART, BLE, HTTP, and CoAP. Supporting multiple protocols adds redundancy and compatibility.
- The device should not accept old firmware, preventing an attacker from exploiting a previous vulnerability. This attack is known as a roll-back attack.
- Updating the bootloader should be kept at a minimum since failure to update the bootloader properly may ultimately break the system.
- Updates can be client-initiated via polling. Polling only consumes energy upon checking for new firmware, but the time difference between the release of new firmware and the client checking for it may be significant, causing a window of risk (or opportunity). A server-initiated update method requires the clients to have a persistent connection to the server, but the client may receive the firmware as soon as it is published.

All in all, this puts high demands on small devices, requiring careful and efficient implementation. Luckily, there exists a plethora of both circuits and libraries to handle the security requirements.

[5] `https://source.android.com/devices/tech/ota`

[6] `https://github.com/google/brotli`

2.1.3 Lightweight Protocols

As described earlier, the old and well-established protocols like HTTP, TCP, and so on are not always applicable in a constrained setting. Imagine that we are to send 100 kB of data from a mobile phone to a friend in a secure way. We may use TCP/IP along with TLS to achieve this. We make the following assumptions:

- We are using IPv4.
- The maximum transmission unit, MTU, is 1500 bytes.
- The maximum segment size, MSS, of a TCP packet is 1460. That is, we are not using any options in neither TCP nor IP.

This means that we need to send $\lceil \frac{100k}{1460} \rceil = 69$ packets. Using the numbers in [Mat20] means that the overhead is $69 \cdot 40 = 2760$ bytes for TCP and IP itself, along with $69 \cdot 30 = 2070$ bytes for TLS packet overhead, and around 5 kB for the TLS handshake. All in all, the total overhead is circa 10 kB bytes, which equals $\frac{10k}{100k} = 10\%$ overhead, which is not too bad. Now, imagine that we have an IoT device transmitting 100 bytes to a gateway. The data fits into a single packet, and the overhead is around 5 kB, using the same setup as before. However, this equals a $\frac{5k}{100} = 5000\%$ overhead.

A *lightweight* protocol tries to combat the issues of large overheads and computational costs by using less complex designs and small memory footprints. Lower complexity usually results in using fixed header lengths for ease of parsing and reducing the number of options that are part of the standard.

Next, we briefly present some of the most common lightweight protocols.

IEEE 802.15.4

The IEEE 802.15.4 standard defines low-rate wireless personal area networks, LR-WPANs [IEE16]. The standard specifies the physical layer along with the media access control (MAC) layer of the OSI model. It supports data rates up to 250 kb/s. The standard specifies the 2.4 GHz band as one of the allowed frequency bands for worldwide use.

6LoWPAN

6LoWPAN, short for IPv6 over Low-Power Wireless Personal Area Network, is a protocol for devices conforming to IEEE 802.15.4 [KMS07]. A LoWPAN may be characterized by the following:

- Devices are transmitting small-sized packets.
- Low bandwidth communication.
- The devices are battery operated.

- Devices may be put into sleep mode for longer periods.

6LoWPAN utilizes IPv6 as the network protocol due to already existing infrastructure. IP-based networks have been used and analyzed for a long time; hence management and diagnostics for these networks are well known. The connectivity to other IP-based devices is straightforward since there is no need for gateways or proxies.

ZigBee

ZigBee is a low-power wireless mesh network protocol based on IEEE 802.15.4, adding networking functionality, such as network formation and routing [Zig15]. ZigBee also defines a framework for user applications. ZigBee operates typically at 2.4 GHz, making it compatible in many markets.

ZigBee has been used in many commercial products[7] such as Philips Hue[8], IKEA's Trådfri products[9], and HVAC systems[10]. The most interesting product, in the author's opinion, is the Hue Tap switch from Philips, using *only* the kinetic energy from physically pushing the buttons. That is, there is no battery involved.

A ZigBee device acts as one of three types within the network:

Coordinator The coordinator node initializes the network by configuring the frequency and network ID. The coordinator becomes the parent of all nodes connecting to the network through it.

Router The routers handle the packet routing in the network. These nodes are not required, but it is common to use them.

End Device An end device only sends and receives messages; hence it does not perform any other actions in the network. Only end devices may be put into sleep mode. The parent node then buffers messages until the device is awake again.

Even though IEEE 802.15.4 specifies encryption at the MAC layer, it lacks key management schemes and how to handle authentication. ZigBee implements these schemes at higher layers. ZigBee uses AES-128 for encryption and specifies three methods for key distribution: pre-installation, placing keys on the device before deployment; transport, sending keys over the network to the devices; and establishment, where keys are negotiated over the network.

ZigBee specifies two security models, described next.

[7] `https://zigbeealliance.org/product_type/certified_product`

[8] `https://www2.meethue.com`

[9] `https://www.ikea.com/us/en/cat/smart-lighting-36812/`

[10] `https://www.airconditioning-systems.com/zigbee.html`

Centralized Security Networks In this model, there is a security component called the Trust Center responsible for authenticating devices joining the network. The Trust Center also generates network keys, which is periodically changed.

Distributed Security Networks This model is simpler but also less secure. There is no Trust Center, but only routers and end devices.

Even though the security in ZigBee is considered to be strong, there are potential weaknesses due to the actual implementation, but also due to many devices not being tamper resistant [Zil16]. Due to most ZigBee devices are constrained and battery-powered, they are susceptible to DoS and battery depletion attacks [Cao+16].

CoAP

The Constrained Application Protocol, or CoAP [SHB14], is a client-server based web transfer protocol designed to be used within constrained devices and networks. CoAP, similar to HTTP, is based on the REST model, where resources are available under a URL. Clients may then access the resources using methods like GET, POST, PUT, and DELETE. CoAP may be seen as a lightweight version of HTTP.

CoAP has the following features:

- Small header overhead
- Low parsing complexity
- Allows proxy and chaining
- Stateless HTTP binding, which allows access to CoAP resources via HTTP and vice versa
- Compatible with DTLS

The message header is of fixed length and only requires 4 bytes, using no options. CoAP runs over UDP due to UDP being more lightweight than TCP. Due to UDP not having mechanisms for reliable transmission, CoAP implements support for reliable messaging by marking a message as confirmable. A message marked as confirmable is retransmitted, using timeouts and exponential back-off, until the receiver has sent an acknowledgment message. For unreliable transmission, a message can be marked as non-confirmable instead.

Since CoAP runs over UDP, it is common to use DTLS for securing the communication channel. Another method for securing communication is to use OSCORE [Sel+19]. Since CoAP is mappable to HTTP via proxies, which terminates the (D)TLS connection, the proxy servers may access and manipulate the data. OSCORE provides CoAP with an end-to-end encrypted channel. OSCORE has a smaller overhead compared to DTLS, making it faster in some settings, as shown in [Gün+20].

CoAP is one of the protocols analyzed in Paper III with and without DTLS using different encryption schemes.

MQTT

The Message Queuing Telemetry Transport protocol, or MQTT [Ban+19], is a lightweight publish-subscribe protocol used to transport messages between devices. An MQTT network consists of two entity types: a message broker and clients. The broker acts as a router, receiving messages from clients, and routing the messages to the appropriate destination. A client is a device running an MQTT application, communicating with the broker to send and receive messages. MQTT usually typically runs over TCP/IP but supports other protocols as well.

Clients subscribe to specific topics. When a client sends a message with a specific topic, the broker distributes this to all clients subscribing to this topic. A client may both be a publisher and a subscriber.

An MQTT message packet consists of a header and payload, with a fixed-size header of 2 bytes. The length variable is a single byte, allowing for message sizes up to 127 bytes. For longer messages, MQTT supports a variable-length header using an encoding scheme. The largest supported message is approximately 256 MiB. The four first bits (MSB) in the header determines the message type, such as connect, publish, subscribe, and disconnect.

The security in MQTT is left to the implementer, but it is common to use TLS, allowing for password-based and certificate-based authentication. Management of keys and certificates for heterogeneous networks becomes cumbersome as the networks grow. The scalability issues have led to research on other approaches, such as using Attribute-Based Encryption (ABE) over elliptic curves, supporting broadcast encryption, as done in SMQTT [Sin+15].

MQTT-SN [ST13] is a continuation of the MQTT protocol, targeting small sensor nodes with minimal resources. MQTT-SN runs over UDP, which reduces the size of the messages. There is an MQTT-SN gateway translating between MQTT and MQTT-SN. Just like for MQTT, there have been several proposed security architectures for MQTT-SN, such as SMQTT-SN [Sin+15].

MQTT is the other protocol, along with CoAP, analyzed in Paper III.

2.2 Cryptography

The word cryptography originates from the Greek words *kryptós* "secret", and *graphein* "to write". In other words, *secret writing*.

The desire for hiding information has been around for a very long time - the oldest known use of cryptography dates back to 1500 B.C. [Kah96], used for hiding the formula of making pottery glaze. Monoalphabetic substitution ciphers were later invented, such as the Atbash cipher and the Caesar cipher. The ciphers use a fixed bijective mapping from an alphabet to itself, e.g., the letter "A" is always replaced with the letter "Q" and so on. Polyalphabetic ciphers build upon monoalphabetic, with the mapping being changed during the encryption or decryption process. For example, for the first five letters, "A" maps to "Q", while for the next five, we map "A" to "M", and so on.

In the modern age, there are two basic cryptographic primitives when it comes to encryption - asymmetric, known as public-key cryptography, and symmetric, known as symmetric-key cryptography.

2.2.1 Asymmetric Ciphers

In the asymmetric case, we have two different, mathematically linked, keys: a public key for encryption and a private key for decryption. It is hard to find the private key given the public key, whereas it is easy to get the public key if we have access to the private one. Next, we describe some commonly used asymmetric algorithms.

RSA

Dating back to 1977, RSA [RSA78] is one of the first published public-key algorithms and is still today widely used in communication protocols. The security of RSA is known as the RSA problem. The RSA problem is related to the integer factorization problem, which is believed to be difficult. At the time of writing, no published algorithm solves the integer factorization problem in polynomial time.

> **Definition** (Integer Factorization Problem). *Given a positive integer n, find the prime factorization of n, such that $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, p_i being pairwise distinct primes with $e_i \geq 1$.*

RSA consists of two parts: a public key consisting of the pair $\langle e, n \rangle$, and a private key d. In practice, more values are stored along with the private key, to speed up computations. The n parameter is calculated by multiplying a set of prime numbers, usually two, denoted p and q. The public exponent e is generated such that it is co-prime to $\phi(n)$, usually $2^{16} + 1$, with ϕ being Euler's totient function. This is sometimes replaced by Carmichael's totient function, $\lambda(n)$, for efficiency. Keeping e small allows for more efficient encryption, due to the reduced amount of computation carried out. The private key is calculated as the inverse of e mod $\phi(n)$. To summarize,

$$\begin{aligned} n &= p \cdot q, \\ e &= \{e \mid 2 < e < \phi(n),\ \gcd(e, \phi(n)) = 1\}, \\ d &\equiv e^{-1} \mod \phi(n). \end{aligned}$$

The encryption and decryption process, of a message m and ciphertext c, is then given by

$$\begin{aligned} c &= m^e \mod n, \\ m &= c^d \mod n. \end{aligned}$$

The security relates to the size of n. NIST recommends using at least 2048 bits in today's systems, with an estimated security level of 112 bits [Bar+19]. Plain RSA is un-padded and deterministic, making it vulnerable to several attacks. Schemes like RSA-OAEP [Mor+16] have been proposed, which adds padding and randomness. RSA is not only used for encryption, but also for providing digital signatures. In this case, the private key, d, is used for signing, and the public exponent, e, is used for verification. Due to the usually small value of e, encryption and signature verification is fast, whereas decryption and signature creation is more expensive.

The introduction of quantum computers and Shor's algorithm [Sho94] poses a serious problem against algorithms like RSA, since it would be possible to break RSA fairly efficiently [GE19]. Post-quantum resistant versions of RSA have been proposed, such as the one in [Ber+17], but are impractical and require a vast amount of computational power.

Diffie-Hellman

The first publicly known public-key algorithm was the Diffie-Hellman (DH) key exchange algorithm, proposed in 1976 [DH76]. The underlying structure of DH is based on modular exponentiation, and the security relates to the discrete logarithm problem, DLP.

> **Definition** (DLP). *Given a prime p, a generator g of $\mathbb{Z}_p^*$, and an element $\alpha \in \mathbb{Z}_p^*$, it is difficult to find the integer x, $1 \leq x \leq p-1$, such that $g^x \equiv \alpha \mod p$. The integer x is called the discrete logarithm of α in base g, denoted* $\log_g \alpha$.

Assume that Alice and Bob want to exchange a key using DH. They perform the following actions, with all operations performed mod p:

1. They agree on a finite cyclic group G with a primitive root (generator) g and a modulus p. These values are all public.

2. Alice chooses a secret random number $a \in \mathbb{Z}/p\mathbb{Z}$, sending g^a to Bob.
3. Bob chooses a secret random number $b \in \mathbb{Z}/p\mathbb{Z}$, sending g^b to Alice.
4. Alice now computes $(g^b)^a$, while Bob computes $(g^a)^b$. Alice and Bob now share the secret g^{ab}.

Due to the Pohlig-Hellman algorithm [PH78], the security of a group G is limited by the largest, prime order, subgroup of G. Hence, it is common to generate a so-called safe prime, using a Sophie Germain prime q to calculate $p = 2q+1$. This prime makes the order of the group G divisible only by 2 and q.

The number field sieve [Len+90] is one of the most effective attacks against discrete logarithms. It was used in the LogJam attack in 2015 to break 512-bit Diffie-Hellman groups [Adr+15] by performing precomputation based on p, leading them to attack any Diffie-Hellman instance using the prime p. The authors estimate that the attack is plausible even at 1024 bits, using nation-state resources. Finally, they recommend the usage of 2048 bit primes or switching to elliptic curves instead. NIST also recommends using at least 2048 bit primes for discrete logarithm based protocols[11].

Plain Diffie-Hellman (DH), also known as anonymous Diffie-Hellman, is vulnerable to Man-in-the-Middle (MitM) attacks due to the keys not being authenticated. One solution is to digitally sign the transmitted parameters, g^a and g^b, by using, e.g., RSA. If the keys used in DH are long-term keys, we call it static Diffie-Hellman. Ephemeral Diffie-Hellman (DHE) uses a new public key for every instance of the protocol. For DHE, if the long-term signing key is compromised, it does not affect the security of past sessions, resulting in Forward Secrecy (FS).

The current best practice for (D)TLS is to only use forward-secrecy-only ciphers [SHS15]. In TLS 1.3, all public-key based key exchange schemes provide forward secrecy [Res18].

Elliptic Curve Cryptography

Instead of performing calculations in the finite group $\mathbb{Z}_p$, we can define an *elliptic curve* over a field K. An elliptic curve is defined as a set of solutions of an elliptic function, e.g., the curve, in Weierstrass form,

$$y^2 = x^3 + ax + b, \quad a, b \in \mathrm{GF}(q), \tag{2.1}$$

q being an odd prime. The points on the curve form an ordered pair $\langle x, y \rangle$, with the coordinates being elements of GF(q) that satisfies the curve equation. There is a particular point, $\mathcal{O}$, known as the point at infinity. This set of points forms a group under addition, using the chord-tangent process.

The strength of elliptic curves is related to discrete logarithms, known as the elliptic curve discrete logarithm problem, ECDLP. That is, for an integer m, we

[11] `https://www.keylength.com/en/4/`

denote the summation of a point P with itself m times, as $[m]\,P$. This multiplication is easy to compute, but it is believed to be hard to invert.

The best-known attack against a general elliptic curve is the parallel Pollard's Rho algorithm [VW99], with a complexity of about $\mathcal{O}(\sqrt{q})$. To achieve a security level of 128 bits, we need to let $q \approx 2^{256}$. Note that this is much smaller than the group size of Diffie-Hellman.

The key generation for ECC is shown to be significantly faster compared to RSA. The time for generating ECC keys is linear with respect to the key size, while it is exponential for RSA [JA04]. It is also shown that signature verification is faster for RSA, due to the small e. The smaller key size for ECC also results in reduced memory footprint and less bandwidth during transmission.

The are multiple standardized elliptic curves, which are believed to be secure, described next [Che+19]. Curves recommended by NIST include P-224, P-256, P-384, and P-521. These curves are all so-called Weierstrass curves, given in Eq. 2.1. The prime field used in the NIST curves are based on generalized Mersenne primes.

Definition (Generalized Mersenne Prime). *A generalized Mersenne prime is a prime of the form*

$$2^{c_n} + \left(\sum_{i=1}^{n-1} -1^{b_i} 2^{c_i} \right) - 1,$$

where

$$b_i \in \{0, 1\},$$

$$c_1 > 0,\ c_{i-1} < c_i,\ c_n \geq n.$$

This allows for efficient modular reduction without using division [Sol+99].

The elliptic curve Curve25519 [Ber06] is a Montgomery curve, which allows for fast x-coordinate point operations, which may offer better performance than the NIST curves. A Montgomery curve is defined by

$$By^2 = x^3 + Ax^2 + x, \quad A, B \in K \tag{2.2}$$

over a field K, usually $\mathrm{GF}(q)$.

Brainpool curves [LM10] use verifiable pseudo-random primes, compared to the NIST curves. These primes may prevent attacks using backdoored designs. However, due to the randomness, fast reduction algorithms are no longer possible, leading to performance penalties[12].

As new attacks are invented, we increase the parameters of the cryptographic algorithms. While this may be a good enough solution to make the algorithms

[12]`https://tls.mbed.org/kb/cryptography/elliptic-curve-performance-nist-vs-brainpool`

secure, the more complex the algorithms are, the more resources it requires to perform the computations. This complexity is not desirable in small, constrained systems. This is a case where the system architect needs to decide to either increase parameters or replace the algorithms with new, hopefully more lightweight, ones.

Let us now shift our focus to symmetric ciphers. In a symmetric cipher, a key is generated and shared between entities. The key must be kept a secret for other parties not to be able to decrypt the data. Symmetric ciphers are categorized as either block ciphers or stream ciphers, described next.

2.2.2 Block Ciphers

A block cipher operates on blocks of text, mapping n-bit plaintext to n-bit ciphertext, where n is the block size. For example, AES uses a block size of 128 bits. The encryption function $C = E_K(P)$ of a block cipher must be invertible, with the inverse function, $P = D_K(C)$, being the decryption function, for a given key, K, and plaintext, P. How messages larger than a block are handled is described next.

For messages exceeding the block size, n, we split the message into n-bit chunks. The way the chunks are processed is known as the *mode of operation* of a block cipher. In the simplest case, known as electronic-codebook, or ECB, each chunk is processed separately. This mode has weaknesses, e.g., if two blocks of plaintexts are identical, so will the corresponding ciphertexts be. Modern modes of operation are based on block chaining or counters, such as CBC and CTR mode.

If a scheme also allows for authenticating unencrypted data, we call it an authenticated encryption with associated data, or AEAD, scheme [Rog02]. Authenticated encryption has been studied for a long time with several proposed ways of achieving secure designs. A message authentication code (MAC) is a piece of data providing integrity and authenticity to a message. A MAC is generated and verified using a shared secret key. It is a symmetric version of a digital signature. MACs can be constructed using cryptographic primitives such as block ciphers and universal hash functions [CW79]. Generally, there are three ways of combining a symmetric encryption scheme and a MAC: *Encrypt-then-MAC*, EtM; *Encrypt-and-MAC*, E&M; *MAC-then-Encrypt*, MtE. The authors in [BN00] show that the MtE and E&M schemes are insecure in some settings.

Modern AEAD schemes include the GCM [MV04] and CCM [WHF03] modes, both recommended to use according to NIST [Dwo04; Dwo07], described next.

GCM This mode combines counter mode encryption with Galois mode authentication, calculated over a field GF(2^w), commonly GF(2^{128}) defined by

$$\mathbb{F}_2\left[x\right]/(x^{128} + x^7 + x^2 + x + 1).$$

Like the counter mode, GCM uses an incrementing counter and an IV as input to the block cipher, which essentially turns it into a stream cipher. For authentication, the GHASH [WC81] function is used, which requires multiplications in the above described field.

GCM allows for parallel computation and efficient use of CPU pipelines, making it fast and efficient, both in software and in hardware [MV05].

CCM This mode combines the counter mode encryption with the CBC-MAC authentication code. This combination is an application of the Authenticate-then-Encrypt scheme. The authors in [KR11b] show that CCM is slightly faster than GCM for smaller messages on x86 and ARM Cortex-A8, but slower for larger message sizes.

CCM-8 uses eight octets for the authentication tag, compared to the usual 16, which reduces the message size at the cost of security.

CCM* is a variant used in low-rate wireless networks, IEEE 802.15.4. Besides the normal CCM operations, CCM* also supports an encryption-only mode.

Let us now dive deep into the area of stream ciphers. Even though there exists public-key stream ciphers such as the Blum-Goldwasser scheme [BG84], we only look at the symmetric case.

2.2.3 Stream Ciphers

A stream cipher operates on a stream of symbols, see Figure 2.2 for a general architecture. Compared to a block cipher, a stream cipher has a state which is updated as it produces keystream. A stream cipher is an approximation of the Vernam cipher, which is known to be information-theoretically secure [Sha49]. A cryptographic primitive is said to be information-theoretically secure if there exists a proof that an attacker, even with infinite computational power, can not break it. More formally,

$$H(M|C) = H(M),$$

where $H(X)$ is the entropy of X. That is, an attacker observing a ciphertext does not gain any information about the plaintext.

The Vernam cipher uses a key of the same size as the message, a requirement that is not applicable to *most* modern use cases. Instead, a limited finite-state is used to derive a keystream used for encrypting a message. Since the state is finite and the algorithm is deterministic, a conventional stream cipher is not information-theoretically secure. Instead, a stream cipher is an instantiation of a pseudo-random generator, PRG, and offers semantic security [GM84; Gol09].

The general advantage of a stream cipher compared to a block cipher, in hardware, is higher speed and lower circuit complexity. Stream ciphers are also capa-

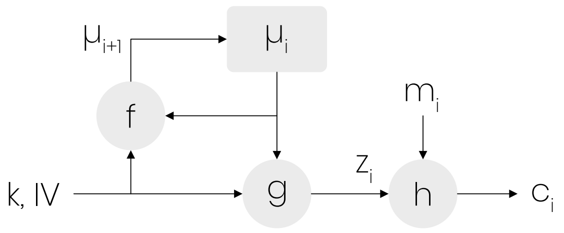


Figure 2.2: General model of a stream cipher, with the initial state μ_0, the key k, the initialization vector IV, the next-state function f, the keystream generator g, the keystream z_i, the output function h, along with the plaintext m_i and the ciphertext c_i.

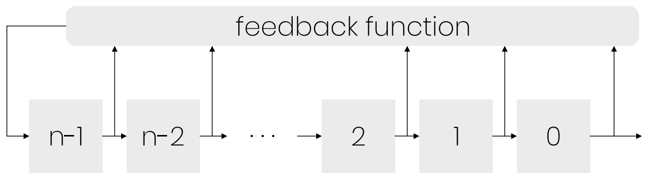


Figure 2.3: An LFSR in Fibonacci configuration.

ble of processing streaming data, e.g., network or mobile traffic, on-the-fly, while block ciphers need to buffer the data up to whole blocks.

There are multiple ways of constructing a stream cipher. One common approach is to use a linear-feedback shift register (LFSR) design. A shift register is a composition of multiple flip-flops connected in series. The feedback function updates the LFSR input with a value based on the previous state. It is common to use the linear XOR gates to construct the feedback functions. The flip-flops which are part of the feedback are called *taps*.

There are two ways of arranging the flip-flops and the feedback functions: Fibonacci and Galois configuration. In the Fibonacci configuration, the flip-flops are connected directly in series, and only the input of the last flip-flop is updated with the output from the feedback, see Figure 2.3. In Galois configuration, the XOR gates are located between flip-flops, controlled by the output of the LFSR. For an LFSR, a Fibonacci configuration may always be transformed into its corresponding Galois configuration. Due to their differences, the initial state may be different in order to produce the same output sequence. The Galois configuration is usually more efficient in software since it allows operations on multiple bytes simultaneously.

The feedback function may be expressed as a polynomial in a finite field mod 2, where the taps correspond to a monomial with coefficient 1. This construction allows us to use the mathematical theory about finite field arithmetic. We know that a primitive polynomial produces a maximal cycle length, which allows us to construct an LFSR producing the longest possible sequence before the cycle repeats. A sequence produced by a maximal LFSR is known as *maximum length sequence*, or *m-sequence*. An m-sequence is a pseudo-random sequence with many interesting properties related to randomness [Gol+67]. Some properties are given below.

- The sequence is almost perfectly balanced. That is, there are 2^{n-1} ones and $2^{n-1} - 1$ zeros. The zero-state only generates zeros; hence it is not particularly useful.
- Half of the runs are of length 1, $\frac{1}{2^n}$ of the runs are of length n. There is also a single run of n ones and $n - 1$ zeros.
- If a sequence is shifted by any non-zero number, the new sequence will have $2^{n-1} - 1$ elements the same as the original sequence, and 2^{n-1} elements different. Interpreting the sequence as containing +1's and -1's instead of 1 and 0, the normalized autocorrelation function is given by

  $$R(m) = \begin{cases} 1 & \text{if } m = 0, \\ -\frac{1}{N} & \text{if } 0 < m < N. \end{cases}$$

  with N being the period of the sequence.
- Adding two phase-shifted m-sequences results in another phase-shift of the same m-sequence.

There are many applications of m-sequences, such as measuring impulse responses in order to create reverberation effects for musical instruments [Val+12]. The impulse response is extracted from the measured system by a circular cross-correlation with the m-sequence. This operation works due to the autocorrelation of an m-sequence approximates the unit impulse, i.e., a Kronecker delta.

Even though m-sequences and general LFSR-based designs are widely used and work well, one can not only use an LFSR to generate secure keystream, due to its intrinsic linearity. The famous Berlekamp-Massey (BM) algorithm finds the shortest LFSR producing a given finite sequence [Ber68; Mas69] in $\mathcal{O}(n^2)$ time, only requiring $2n$ consecutive bits to reconstruct an LFSR with length n. To mitigate these attacks, different approaches are taken to introduce non-linearity, described next.

Non-linear Combining Functions

Here, we take the output of multiple LFSRs and combine the outputs via a non-linear Boolean function, see Figure 2.4. With n LFSRs in $\mathbb{F}_q$, the function is the mapping $\mathbb{F}_q^n \to \mathbb{F}_q$. The output can be given as

$$z = f(L_1, L_2, \ldots, L_n).$$

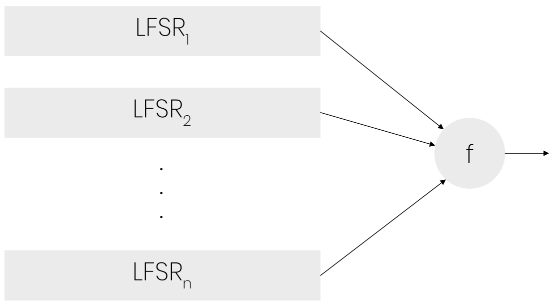


Figure 2.4: General architecture of a stream cipher based on a non-linear combiner design.

A Boolean function needs certain properties to be applicable in stream cipher applications. Here, we present some of these properties, as given in [SM00].

Balanced A Boolean function, f, of n variables is said to be balanced if the Hamming weight $wt(f) = 2^{n-1}$. That is, there is an equal distribution of ones and zeros.

Nonlinearity A Boolean function, f, of n variables is said to have nonlieарity m if the smallest Hamming distance between f and all n-variable affine functions is m.

Algebraic degree A Boolean function, f, of n variables can be represented as a polynomial in the algebraic normal form, ANF. The degree of this polynomial is called the algebraic degree, denoted $deg(f)$.

Correlation immunity A Boolean function, f, of n variables is said to be correlation immune of order k if all sets of k variables, or fewer, are statistically independent of the output of f. That is, the random variable

$$Z = f\left(X_0, \ldots, X_{n-1}\right)$$

is statistically independent from any random vector

$$\left(X_{i_1}, \ldots, X_{i_k}\right).$$

A balanced Boolean function with correlation immunity of order k is said to be k-resilient.

This leads to high linear complexity for the keystream; hence the BM algorithm becomes computationally hard. The Geffe generator [GP73] is an example of an architecture utilizing a non-linear combiner function. The problem with the stateless combiner functions is that they are susceptible to correlation attacks such as [MS89]. To counter these types of attacks, the authors proposed Boolean functions with no good linear approximation, and which are also correlation immune, such as stateful combiners.

E_0 is a stream cipher used in Bluetooth [Blu19]. Although it is recommended to use AES-CCM when possible, E_0 is kept for legacy reasons. E_0 consists of four LFSRs with lengths 25, 31, 33, and 39, all using primitive polynomials. The output is generated by an FSM, called a summation combiner, thus being a stateful combiner. By using only a small number of memory cells, E_0 is still vulnerable to attacks, such as that in [Cou04].

Another approach for increasing non-linearity is to use a nonlinear-feedback shift register, NFSR. An NFSR uses a feedback function, which is a non-linear function of the FSR state, usually by including AND and OR gates. Such construction is found in the Achterbahn cipher.

The Achterbahn stream cipher exists in two versions: Achterbahn-80 and Achterbahn-128, consisting of 11 and 13 NFSRs, respectively, with lengths ranging from 21 to 33. The output of the cipher is fed into a non-linear combiner function. For example, the combiner function, F, used in Achterbahn-128 has properties such as:

- It is balanced
- It is correlation-immune of order 8
- It has non-linearity 3584

Due to all NFSRs being rather short and independent, attacks, where the output functions are linearized, are shown to be effective [JMM06]. The authors in [HJ07] observed a stronger dependency between the input to the Boolean function and its output than previously assumed. This dependency led the authors to mount an attack requiring less complexity than brute force.

Non-linear Filter

Instead of using multiple LFSRs, we can use a single LFSR and use the state values to be the input to a non-linear output function, see Figure 2.5. That is, given the internal states $s_1, s_2, \ldots, s_L$, the output is given by

$$z = f(s_1, s_2, \ldots, s_L).$$

A non-linear filter design may be transformed into a non-linear combiner design by replicating the LFSR L times and shifting the initial state accordingly. The properties of the filter function are equal to the properties described for the combiner case.

The SNOW cipher, introduced in [EJ00], is a word-oriented stream cipher based on non-linear filter design. It has been updated as SNOW 2.0 in [EJ02],

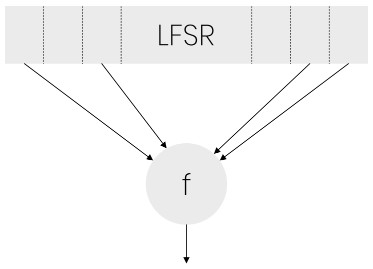


Figure 2.5: General architecture of a stream cipher based on a non-linear filter design.

and as SNOW 3G in [3GP06]. Due to the non-random behavior of the ciphers, the authors in [KY11] were able to mount an attack based on related-key pairs. The latest version, SNOW-V, was proposed in [Ekd+19]. Instead of operating on bits, SNOW-V uses symbols in $\mathbb{F}_{2^{16}}$. SNOW uses two LFSRs of length 16 from where symbols enter an FSM, similar to E_0. The FSM consists of three 128-bit registers: R1, R2, and R3. Two blocks from the LFSR part are used as input to the FSM, which produces a 128-bit keystream. A tag is generated using GMAC. The FSM also uses an S-box based on the Rijndael (AES) round function, to provide strong diffusion. Utilizing AES building blocks allows for fast software implementations due to AES instructions, such as `_mm_aesenc_si128`, being hardware-accelerated in most modern high-end architectures.

Due to linearity, stream ciphers based on LFSRs are susceptible to attacks such as algebraic attacks [Cou03b] and fast correlation attacks. The authors in [BL06] analyze the security on non-linear filters and combiners, based on known families of attacks, such as trade-off attacks, Berlekamp-Massey attacks, distinguishing attacks, fast correlation attacks, and algebraic attacks. The authors conclude that it is more challenging to protect combiner functions against certain attacks; hence they favor the non-linear filter generator.

The Grain family of stream ciphers is built on the non-linear filter design but uses one LFSR and one NFSR to further increase the non-linearity.

2.3 Hardware Design

1947 marks a valuable time in history - the birth of the transistor. The creators: John Bardeen, Walter Brattain, and William Shockley were rewarded with the Nobel Prize in Physics 1956. The transistor revolutionized the field of electronics, replacing vacuum tubes, making it easier and cheaper to manufacture electronic devices such as computers and radios.

Today, the most common type of transistor used in electronics is the Metal–Oxide–Semiconductor Field-Effect Transistor, or MOSFET for short. Estimations show that the MOSFET is the most manufactured component in the history of humanity[13], with a count of 1.3×10^{22}. In digital circuits, it is most common to use Complementary MOS, CMOS, which consists of symmetrical pairs of n-type and p-type MOSFETs. CMOS circuits are known for having excellent characteristics, such as low power consumption, high noise immunity, and low propagation delay.

Integrated circuits range from general to application-specific devices. General programmable devices include microcontrollers, usually found in embedded systems, such as Atmel AVR, processors such as those in a modern computer, e.g., AMD Ryzen. Domain-specific devices can perform general computations but are more focused on a specific area, or domain, such as Digital Signal Processors (DSPs) and Graphics Processing Units (GPUs), e.g., Nvidia RTX. Field-Programmable Gate Arrays (FPGAs) are re-programmable devices used for implementing both general and specific applications. In contrast, an Application-Specific Integrated Circuit (ASIC) is a hardware component implementing specific functionality very efficiently but can not be re-configured.

Next, we describe the synthesis process, along with optimization techniques for speed, area, and energy.

2.3.1 Hardware Synthesis

Designing hardware is easy; designing efficient hardware is hard. Today, hardware is implemented using a Hardware Description Language (HDL) such as VHDL and Verilog for describing the circuit at a register-transfer level (RTL), an abstraction model for the flow of data between registers. Just like compiling high-level source code into executable machine code in software design, the process is similar for hardware. Hardware synthesis is the process where an abstract description of the circuit, from HDL, gets translated into logic gates.

To produce a list of components and their connections, i.e., a netlist, we need to map a standard-cell library. This library contains all building blocks needed to realize the design, such as AND gates, XOR gates, and flip-flops. The exact implementation of the gates is vendor-specific, which leads to different results

[13]`https://computerhistory.org/blog/13-sextillion-counting-the-long-winding-road-to-the-most-frequently-manufactured-human-artifact-in-history/`

in terms of speed, area, and power. Conventional technologies used in industry are 28 nm and 65 nm, but even older technologies such as 90 nm and 130 nm occur. One exception is the desktop and server market, where modern circuits use a process technology of only a few nanometers, such as the 3rd generation Ryzen architectures with a 7 nm technology.

As previously stated, the synthesis tool is responsible for translating the RTL code into physical logic gates, e.g., NAND and NOR, using as "efficient" implementation as possible. By efficient, we usually refer to the number of gate equivalents (GE). A GE is the area of a 2-NAND gate. Although the Boolean circuit minimization problem is $\sum_2^{\mathrm{P}}$-complete [BU08], there exist efficient algorithms, using potential non-optimal heuristics, to facilitate the minimization process, such as Espresso [LS87]. The algorithm utilizes Boolean cubes to represent the ON-, OFF- and DC-sets of the function.

2.3.2 Optimizing Speed

To increase the speed of a design, the first step is to find the bottlenecks. In hardware design, these bottlenecks are known as critical paths. The critical path denotes the longest path a signal must propagate between two flip-flops. This path determines the maximum clock frequency. If a higher clock is used, the signal on the critical path will not have propagated to its destination; hence the circuit will behave erroneously. Once the critical path has been identified, the next step is to make it shorter without altering the circuit behavior. Next, we present some conventional optimization techniques for speed.

Pipelining

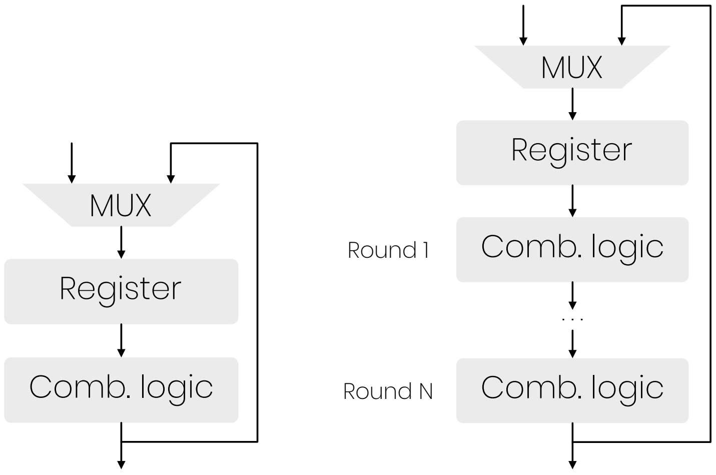
The fundamental concept of pipelining is simple; insert a flip-flop in the critical path, optimally dividing the path in two. If the critical path remains in the same path, the clock frequency may now be doubled. Of course, it is not as easy as this in reality.

First of all, we must make sure that pipelining is applicable. This is done by depicting the design as a graph, with arrows pointing in the data-flow direction. We are only allowed to insert pipeline steps in the feedforward cutset [Pro01]. A cutset is a set of edges such that if they are removed (cut), it results in two disjoint graphs. If data only flows forward on all cutsets, we call it a feedforward cutset. By inserting a pipeline step, we delay the computations by one clock cycle and introduce more hardware.

Even though the pipeline is strategically placed, there might be new paths that become the next critical path. For example, assume that the first critical path has a delay of 1000 ps. We insert a pipeline, dividing the path into two. This yields a 500 ps delay. Now, the next critical path might be somewhere else in the system, with a delay of 995 ps. The effective speed in the system is limited by 995 ps, not 500 ps. Pipelining, while simple, requires a thorough analysis of the whole system.

Loop Unrolling

Loop unrolling is a technique for improving the execution speed of a loop at the expense of the area. Unrolling a loop by a factor N, the body of the loop is repeated N times, and the total amount of loop iterations is reduced, see Figure 2.6. This reduces the amount of overhead from the loop itself.



(a) Normal loop. **(b)** Loop unrolling by a factor N.

Figure 2.6: Implementation of a loop in hardware.

There is, of course, a trade-off to be made between area and speed, and it is not always obvious what unrolling factor N results in the "best" design. An example of a design using unrolling is found in [GC09], where the authors implement and analyze AES with different levels of unrolling, showing that the throughput increases with N, but higher degrees of unrolling has a minimal increase in speed but a high area penalty.

2.3.3 Optimizing Energy

Minimizing energy consumption is an essential goal in circuit design, especially for circuits used in constrained systems. One important note to make here is that low power does not necessarily imply low energy.

Assume that we have a system performing some computation in time t_1, consuming the power P_1. We manage to reduce the power consumption to $P_2 = 0.5P_1$, which increases the computation time to $t_2 = 4t_1$. Now the total energy consumption after reducing the power is actually twice the initial energy consumption, i.e., $E_2 = 2E_1$. Since the lifetime of a battery is proportional to the energy consumption, we should stick to the initial implementation.

The relation between power and energy is not obvious; it depends on the system design and optimization techniques used. Analysis is a must to achieve the required result.

Loop unrolling is not only used for increasing the speed of a circuit. It may also lower energy consumption, as shown in [BBR15].

Clock Gating

Clock gating is a technique where the clock signal to a circuit, or part thereof, is disabled. This results in the flip-flops not being switched, which reduces the dynamic power consumption of the transistors. As described in Section 2.1.1, clock gating may be used to implement sleep modes in a system.

Studies show that the clock signals in digital circuits consume a significant fraction of the total power. Applying a clock gating scheme may have a tremendous impact on the total power consumption of the system [Mah+09].

2.3.4 Optimizing Area

The cost of the die is proportional to its area[14]. Hence, it is beneficial to make the chip as small as possible. Next, we present conventional area-optimization techniques.

Logic Reuse

One way to save area is to reuse existing functionality by time-multiplexing. The goal is to determine the smallest amount of basic functional blocks required to implement the functionality. Assume that we need to implement the following function,

$$y = x_1 + x_2 + x_3 + x_4.$$

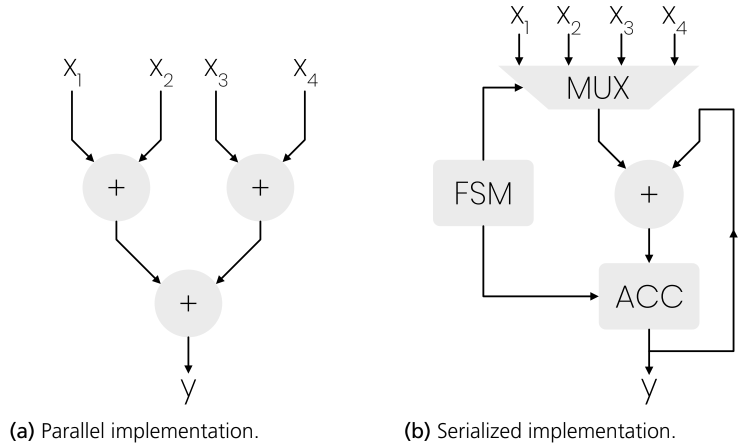
A naïve implementation requires three adders. The corresponding hardware architecture is shown in Figure 2.7a.

Another description of the function would be given as we can instead describe the function as

$$y = \sum_{i=1}^{4} x_i,$$

thereby using only a single adder, but in a loop over time. This reduces the area of the adders, at the cost of latency. However, we need a state machine to control the flow of data in the second case, since it executes serially. The corresponding architecture is shown in Figure 2.7b. As seen in the figure, for this small example, it is not apparent which of the designs would yield a smaller area. For more complex

[14]`https://mycmp.fr/technologies/price-list.html`



(a) Parallel implementation. **(b)** Serialized implementation.

Figure 2.7: Hardware implementation of an adder circuit. The serialized implementation uses an accumulator (ACC) to store intermediate values adding up to the final sum. A finite state machine (FSM) is used for controlling the dataflow.

functions, there could be lots to gain in area reduction, but this has to be carefully analyzed.

Custom Logic

During synthesis, the tool maps the RTL code to a specific cell library. This library contains all fundamental gates necessary to realize any logic function, such as NAND, NOR, XOR, multiplexers, and flip-flops. The gates are usually implemented using CMOS logic.

When talking about area optimization, one usually refers to the number of fundamental gates, like those listed above. However, even if any logic function can be described in terms of a minimal amount of NAND and NOR gates, it does not necessarily mean the realization uses a minimal number of transistors.

The most common approach when designing logic gates is to use complementary MOS, CMOS, technology. A CMOS gate consists of two parts - a *pull-up network* (PUN) and a *pull-down network* (PDN). The PUN provides a connection from the power supply, V_{dd}, to the output when the corresponding Boolean function equals 1. The PDN connects the output to ground, V_{ss}, when the function equals 0. The two networks are mutually exclusive, in the sense that only one network is active at a time, sourcing or sinking current. The PUN consists of PMOS transistors due to their ability to source current. Conversely, the PDN consists of NMOS transistors due to their ability to sink current [RCN02].

NMOS transistors in series correspond to a NAND function, whereas NMOS in parallel resembles a NOR function. Similarly, PMOS transistors in series correspond to a NOR function, while PMOS in parallel realizes a NAND function. By

Table 2.2: Common transistor count for NAND and NOR gates.

Gate	Transistors
2-NAND	4
3-NAND	6
2-NOR	4

De Morgan's theorems, one can show that the PUN and PDN are *dual* networks, hence one only needs to construct one of the networks, with the other one being easily derived. CMOS logic is inverting and can only implement functions such as NAND and NOR. An additional inverter is required to implement AND and OR functions.

The power of synthesizing CMOS gates directly, not using standard gates, is apparent when the gates get complex. Consider the Boolean function

$$F = \overline{D + A \cdot (B + C)}.$$

Minimizing the expression in disjunctive form yields

$$F = \overline{\overline{BCD} + \overline{AD}},$$

which requires a 3-input NAND, a 2-input NAND, and a NOR gate. From Table 2.2, we see that the cost of the function, F, in number of transistors is 14. Using methods for designing complex CMOS gate, we can implement the PDN using four transistors where B and C are in parallel, A is in series with B and C, and finally, D is in parallel with A, B, and C. Due to the PUN being the dual to the PDN, it uses an equal amount of transistors. Thus, we conclude that F can be implemented using eight transistors, which is $\approx 43\%$ less than before.

It is worth noticing that the CMOS design of a Boolean function minimizes the transistor count, using that technology. That being said, other technologies may be used, resulting in an even lower transistor count.

Transistor Technology

A popular alternative to CMOS logic is pass-transistor logic (PTL), intending to minimize the number of transistors required to implement a Boolean function. The difference is that we allow the input signals to drive both the gates and the source-drain terminals, whereas in CMOS logic, we only let the gates be driven by the input signals. Using this approach enables us to design a 2-AND gate using only four transistors, instead of six.

One drawback with this approach is that an NMOS, as stated earlier, efficiently sinks current, but is poor at sourcing current. That is, the NMOS can not pull a node high, but will only output $V_{dd} - V_{th}$. A widely used solution to the problem with dropped voltage is to utilize so-called *transmission gates*. The idea is to use an NMOS to pull down (sink) and a PMOS to pull up (source). This is realized by placing the NMOS and PMOS in parallel, with complementary control signals to the gates. This allows for efficient implementation of complex gates without the voltage drop problems. For example, an XOR gate may be realized using only six transistors, compared to 12 for a complementary implementation [WFF94]. These kinds of transistor-reducing techniques were utilized by the designers of the Zilog Z80 CPU [Shi13].

Contributions and Conclusions

3.1 Contributions

In this chapter, we summarize the contributions of the papers included in this dissertation. This chapter ends with some conclusions.

3.1.1 An AEAD Variant of the Grain Stream Cipher

In Paper I, a new version of the stream cipher Grain is presented, Grain-128AEAD. Following the latest recommendations from NIST, Grain offers authenticated encryption with associated data at a 128-bit security level. Grain is designed to be lightweight to fit in the most constrained environments. At the time of writing, Grain-128AEAD is currently one of 32 ciphers managing to proceed to round 2 of the NIST lightweight cryptography standardization process[1].

To mitigate attacks on previous versions of Grain, Grain-128AEAD no longer supports encryption only. The authentication tag has been extended to 64 bits. During the initialization, the key is re-introduced in order to randomize the state, thwarting key-recovery attacks. The keystream is also limited to 2^{80}, to make attacks using linear approximations more difficult.

With the new design, we present a straightforward hardware implementation of the cipher, using a 65 nm library in VHDL. We use three different levels of parallelizations: 1, 2, and 32 times, in two scenarios. First, the clock frequency is fixed to 100 kHz, matching the frequency to that of RFID. This yields very low power consumption, 313 nW in the best case, and 574 nW in the worst. Next, we synthesize for maximum clock frequency, thus throughput. The highest throughput is reached for a parallelization level of 32, with 10.59 Gbit/s.

[1]`https://csrc.nist.gov/projects/lightweight-cryptography`

3.1.2 Efficient Hardware Implementations of Grain-128AEAD

In Paper II, we present an in-depth analysis of hardware implementations of Grain-128AEAD. Synopsys design compiler is utilized along with a 65 nm library. When evaluating the designs, three properties are measured: area in terms of gates, power consumption, and throughput.

We utilize several optimization techniques, such as Galois transforms, pipelining, FSM optimizations, and unrolling. The first step is to implement a straightforward version of the architecture, for comparison, along with a simple FSM to keep track of the phases. We then implement parallelized versions by powers of two, from 1 to 32, and compare throughput, area, and power.

During synthesis, there are multiple compiler optimization flags to be used, such as structuring, flattening, and ungrouping. We show that for our implementations, only the ungrouping option affects the results. However, we note that for different designs, all options should be analyzed. Another parameter to specify is the transistor type. In this paper, we utilize LVT and HVT transistors. LVT transistors have a lower threshold voltage, resulting in a higher maximum clock frequency but also higher static power consumption. For HVT transistors, the outcome is the opposite - low static power consumption, but lower maximum clock frequency. We use LVT for the high-speed implementations and HVT for the low-power implementations. The results show that the low-power implementation has a higher energy consumption since computations take a longer time.

Finally, we conclude that Grain-128AEAD is a fast and energy-efficient cipher, with a small hardware footprint, suitable for IoT environments with a need for authenticated encryption.

3.1.3 Energy Consumption for Securing Lightweight IoT Protocols

In Paper III, we measure and analyze the energy overhead when adding security to CoAP and MQTT. For CoAP, we use DTLS 1.2, whereas, for MQTT, we use TLS 1.2.

Our experimental setup consists of an ESP-32 IoT device, along with an Otii Arc for measuring energy, and a router connected to a desktop hosting CoAP and MQTT server applications.

We split the (D)TLS analysis into a handshake part and a data transmission part. In the handshake, we compare the impact of using different cipher suites. We show that RSA performs better than ECDSA when no client certificate is used, while being worse when using client certificates. This is due to RSA signing being much more costly than verification. We also note that ECDHE is much more efficient than the corresponding DHE suites. The most energy and time-efficient cipher suites are the PSK ones since there is no need for key negotiation. However, PSK suites require keys to be distributed to the devices or pre-loaded, which is not always possible.

We measure the energy difference between modes of operation using AES, along with different key sizes when sending bulk data. There are only minor differences in energy when using various modes of operation. We see that MQTT performs better for larger payload sizes compared to CoAP. Due to the communication overhead, it is more beneficial to send aggregated data compared to sending smaller packets, both for CoAP and MQTT. When sending a full firmware, a packet size of ≈ 1 MiB, MQTT is roughly four times more energy-efficient than CoAP.

For lossy networks, the benefits of using TCP compared to UDP is obvious. CoAP requires $5 - 10$ times more energy to transmit a fixed-size packet at a given loss rate.

In conclusion, we present real-world measurements of the cost of security for CoAP and MQTT under different settings. We recognize that the measurements are hardware, software, and environment-specific, but we aim to give a better understanding of the relative difference rather than the absolute numbers.

3.2 Conclusions

This dissertation aims to present new results on lightweight cryptography and its use in constrained devices. For resource-constrained devices being connected to the Internet, there is a need for protocols of high security, while being energy-efficient.

We look at stream cipher design and present a new version of the lightweight cipher Grain, designed to be small in terms of circuit area and using low power while offering a 128-bit security level. The latest version also provides AEAD, for ensuring confidentiality and authenticity.

Further, we implement and analyze Grain-128AEAD in VHDL using a 65 nm standard cell library. Using multiple optimization techniques, we come up with various implementations of Grain, targeting different requirements and constraints. We show that Grain-128AEAD is suitable both for the smallest of devices, but also capable of handling large data streams at a high pace in, e.g., server environments.

Choosing cryptographic algorithms for a system is difficult since the impact of an algorithm depends on many variables. The engineer should research comparisons and analyze the relative differences between the algorithms. It is also important to understand the circumstances under which the comparisons have been carried out, e.g., if the system utilizes hardware acceleration, which libraries have been used, and so on. If the conditions change ever so slightly, it may have a considerable impact on the result; hence the engineer must carefully plan for various scenarios.

The security of an algorithm lie not only in the mathematical description, but also in the actual implementation. This means that both the algorithms and the implementaions must stay up-to-date in today's rapid development, as attacks never get worse, they only get better.

If shit were to hit the fan and you end up with a broken stream cipher, you can always convert it into a guitar pedal.

References

[3GP06] 3GPP. *Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 2: SNOW 3G Specification*. Tech. rep. Sept. 2006.

[Adr+15] D. Adrian et al. “Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice”. In: *22nd ACM Conference on Computer and Communications Security*. Oct. 2015.

[Atm] Atmel. *ATmega328P*. URL: `http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf` (visited on 05/12/2020).

[Ban+19] A. Banks et al. *MQTT Version 5.0*. Tech. rep. Mar. 2019.

[Bar+08] G. Barrenetxea et al. “The Hitchhiker’s Guide to Successful Wireless Sensor Network Deployments”. In: *SenSys’08 - Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems* (Jan. 2008).

[Bar+19] E. Barker et al. *Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography*. Technical report. Mar. 2019.

[BBR15] S. Banik, A. Bogdanov, and F. Regazzoni. *Exploring Energy Efficiency of Lightweight Block Ciphers*. Cryptology ePrint Archive, Report 2015/847. `https://eprint.iacr.org/2015/847`. 2015.

[BEK14] C. Bormann, M. Ersue, and A. Keranen. *Terminology for Constrained-Node Networks*. RFC 7228. RFC Editor, May 2014.

[Ber+17] D. J. Bernstein et al. *Post-quantum RSA*. Cryptology ePrint Archive, Report 2017/351. `https://eprint.iacr.org/2017/351`. 2017.

[Ber06] D. J. Bernstein. “Curve25519: new Diffie-Hellman speed records”. In: *International Workshop on Public Key Cryptography*. Springer. 2006, pp. 207–228.

[Ber68] E. Berlekamp. “Nonbinary BCH decoding (Abstr.)” In: *IEEE Transactions on Information Theory* 14.2 (1968), pp. 242–242.

[BG84] M. Blum and S. Goldwasser. “An efficient probabilistic public-key encryption scheme which hides all partial information”. In: *Workshop on the Theory and Application of Cryptographic Techniques*. Springer. 1984, pp. 289–299.

[BL06] A. Braeken and J. Lano. “On the (Im)Possibility of Practical and Secure Nonlinear Filters and Combiners”. In: *Selected Areas in Cryptography*. Ed. by B. Preneel and S. Tavares. Springer Berlin Heidelberg, 2006, pp. 159–174.

[Blu19] Bluetooth SIG. *Bluetooth Core Specification*. TJA1043. Rev. 5.2. Bluetooth SIG. Dec. 2019.

[BN00] M. Bellare and C. Namprempre. “Authenticated encryption: Relations among notions and analysis of the generic composition paradigm”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2000, pp. 531–545.

[BU08] D. Buchfuhrer and C. Umans. “The Complexity of Boolean Formula Minimization”. In: *Automata, Languages and Programming*. Ed. by L. Aceto et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 24–35.

[Cao+16] X. Cao et al. “Ghost-in-ZigBee: Energy Depletion Attack on ZigBee-Based Wireless Networks”. In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 816–829.

[Che+19] L. Chen et al. *Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters*. Technical report. Oct. 2019.

[CLG13] J. K.-T. Chang, C. Liu, and J.-L. Gaudiot. “Hardware Acceleration for Cryptography Algorithms by Hotspot Detection”. In: *Grid and Pervasive Computing*. Ed. by J. J. (H. Park et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 472–481.

[Cou03b] N. T. Courtois. “Fast Algebraic Attacks on Stream Ciphers with Linear Feedback”. In: *Advances in Cryptology - CRYPTO 2003*. Ed. by D. Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 176–194.

[Cou04] N. T. Courtois. “Algebraic attacks on combiners with memory and several outputs”. In: *International Conference on Information Security and Cryptology*. Springer. 2004, pp. 3–20.

[CW79] J. L. Carter and M. N. Wegman. “Universal Classes of Hash Functions”. In: *Journal of computer and system sciences* 18.2 (1979), pp. 143–154.

[DH76] W. Diffie and M. Hellman. “New directions in cryptography”. In: *IEEE transactions on Information Theory* 22.6 (1976), pp. 644–654.

[Dwo04] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*. Technical report. May 2004.

[Dwo07] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Technical report. Nov. 2007.

[EJ00] P. Ekdahl and T. Johansson. “SNOW-a new stream cipher”. In: *Proceedings of first open NESSIE workshop, KU-Leuven*. 2000, pp. 167–168.

[EJ02] P. Ekdahl and T. Johansson. “A new version of the stream cipher SNOW”. In: *International Workshop on Selected Areas in Cryptography*. Springer. 2002, pp. 47–61.

[Ekd+19] P. Ekdahl et al. “A new SNOW stream cipher called SNOW-V”. In: *IACR Transactions on Symmetric Cryptology* (2019), pp. 1–42.

[Gay+03] D. Gay et al. *nesC 1.1 language reference manual*. 2003.

[GC09] K. Gaj and P. Chodowiec. “FPGA and ASIC Implementations of AES”. In: *Cryptographic Engineering*. Ed. by Ç. K. Koç. Boston, MA: Springer US, 2009, pp. 235–294.

[GE19] C. Gidney and M. Ekerå. “How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits”. In: *arXiv preprint arXiv:1905.09749* (2019).

[GM84] S. Goldwasser and S. Micali. “Probabilistic encryption”. In: *Journal of computer and system sciences* 28.2 (1984), pp. 270–299.

[Gol+67] S. W. Golomb et al. *Shift register sequences*. Aegean Park Press, 1967.

[Gol09] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge university press, 2009.

[GP73] P. R. Geffe and G. PR. “How to protect data with ciphers that are really hard to break”. In: (1973).

[Gue06] S. Gueron. *White paper: Intel® Advanced Encryption Standard (AES) New Instructions Set*. Tech. rep. MSU-CSE-06-2. Mobility Group, Israel Development Center, Intel Corporation, Jan. 2006, p. 81.

[Gün+20] C. Gündoğan et al. “IoT Content Object Security with OSCORE and NDN: A First Experimental Comparison”. In: *arXiv preprint arXiv:2001.08023* (2020).

[HJ07] M. Hell and T. Johansson. “Cryptanalysis of achterbahn-128/80”. In: *IET Information Security* 1.2 (2007), pp. 47–52.

[IEE16] IEEE. “IEEE Standard for Low-Rate Wireless Networks”. In: *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (2016), pp. 1–709.

[Int08] International Organization for Standardization. *Programming languages — C — Extensions to support embedded processors*. Standard. Geneva, CH: International Organization for Standardization, June 2008.

[JA04] N. Jansma and B. Arrendondo. “Performance comparison of elliptic curve and rsa digital signatures”. In: *nicj. net/files* (2004).

[JMM06] T. Johansson, W. Meier, and F. Muller. “Cryptanalysis of Achterbahn”. In: *Fast Software Encryption*. Ed. by M. Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–14.

[Kah96] D. Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, Dec. 1996.

[KMS07] N. Kushalnagar, G. Montenegro, and C. Schumacher. *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*. RFC 4919. RFC Editor, Aug. 2007.

[KR11b] T. Krovetz and P. Rogaway. “The Software Performance of Authenticated-Encryption Modes”. In: *Fast Software Encryption*. Ed. by A. Joux. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 306–327.

[KY11] A. Kircanski and A. M. Youssef. “On the sliding property of SNOW 3 G and SNOW 2.0”. In: *IET Information Security* 5.4 (2011), pp. 199–206.

[Len+90] A. K. Lenstra et al. “The number field sieve”. In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 1990, pp. 564–572.

[LM10] M. Lochter and J. Merkle. *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*. RFC 5639. RFC Editor, Mar. 2010.

[LS87] R. L. Rudell and A. Sangiovanni-Vincentelli. “Multiple-Valued Minimization for PLA Optimization”. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 6 (Oct. 1987), pp. 727–750.

[Mah+09] H. Mahmoodi et al. “Ultra Low-Power Clocking Scheme Using Energy Recovery and Clock Gating”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17.1 (2009), pp. 33–44.

[Mas69] J. Massey. “Shift-register synthesis and BCH decoding”. In: *IEEE Transactions on Information Theory* 15.1 (1969), pp. 122–127.

[Mat20] J. Mattsson. *Overview and Analysis of Overhead Caused by TLS*. Internet-Draft. May 2020.

[Mor+16] K. Moriarty et al. *PKCS #1: RSA Cryptography Specifications Version 2.2*. RFC 8017. RFC Editor, Nov. 2016.

[Mor+19] B. Moran et al. *A Firmware Update Architecture for Internet of Things*. Internet-Draft. Nov. 2019.

[MS89] W. Meier and O. Staffelbach. “Fast correlation attacks on certain stream ciphers”. In: *Journal of cryptology* 1.3 (1989), pp. 159–176.

[MV04] D. McGrew and J. Viega. “The Galois/Counter Mode of Operation (GCM)”. In: *submission to NIST Modes of Operation Process* 20 (2004).

[MV05] D. A. McGrew and J. Viega. “The Security and Performance of the Galois/Counter Mode (GCM) of Operation”. In: *Progress in Cryptology - INDOCRYPT 2004*. Ed. by A. Canteaut and K. Viswanathan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 343–355.

[PH78] S. Pohlig and M. Hellman. “An improved algorithm for computing logarithms over GF (p) and its cryptographic significance (Corresp.)” In: *IEEE Transactions on information Theory* 24.1 (1978), pp. 106–110.

[Pro01] J. G. Proakis. *Digital signal processing: principles algorithms and applications*. Pearson Education India, 2001.

[RCN02] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic. *Digital integrated circuits*. Vol. 2. Prentice hall Englewood Cliffs, 2002.

[Res18] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC. RFC Editor, Aug. 2018.

[Rog02] P. Rogaway. “Authenticated-Encryption with Associated-Data”. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. CCS ’02. Washington, DC, USA: Association for Computing Machinery, 2002, pp. 98–107.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[Sel+19] G. Selander et al. *Object Security for Constrained RESTful Environments (OSCORE)*. RFC 7252. RFC Editor, July 2019.

[Sha49] C. E. Shannon. “Communication theory of secrecy systems”. In: *Bell system technical journal* 28.4 (1949), pp. 656–715.

[SHB14] Z. Shelby, K. Hartke, and C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. RFC Editor, June 2014.

[Shi13] K. Shirriff. *Reverse-engineering the Z-80: the silicon for two interesting gates explained*. 2013. URL: `http://www.righto.com/2013/09/understanding-z-80-processor-one-gate.html` (visited on 07/02/2019).

[Sho94] P. W. Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[SHS15] Y. Sheffer, R. Holz, and P. Saint-Andre. *Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. RFC. RFC Editor, May 2015.

[Sin+15] M. Singh et al. “Secure MQTT for Internet of Things (IoT)”. In: *2015 Fifth International Conference on Communication Systems and Network Technologies*. 2015, pp. 746–751.

[SM00] P. Sarkar and S. Maitra. “Construction of nonlinear Boolean functions with important cryptographic properties”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2000, pp. 485–506.

[Sol+99] J. A. Solinas et al. *Generalized mersenne numbers*. Citeseer, 1999.

[ST13] A. Stanford-Clark and H. L. Truong. “Mqtt for sensor networks (mqtt-sn) protocol specification”. In: *International business machines (IBM) Corporation version* 1 (2013), p. 2.

[Val+12] V. Valimaki et al. “Fifty Years of Artificial Reverberation”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.5 (2012), pp. 1421–1448.

[VW99] P. C. Van Oorschot and M. J. Wiener. “Parallel collision search with cryptanalytic applications”. In: *Journal of cryptology* 12.1 (1999), pp. 1–28.

[WC81] M. N. Wegman and J. L. Carter. “New hash functions and their use in authentication and set equality”. In: *Journal of computer and system sciences* 22.3 (1981), pp. 265–279.

[WFF94] J.-M. Wang, S.-C. Fang, and W.-S. Feng. “New efficient designs for XOR and XNOR functions on the transistor level”. In: *IEEE Journal of Solid-State Circuits* 29.7 (July 1994), pp. 780–786.

[WHF03] D. Whiting, R. Housley, and N. Ferguson. “Counter with cbc-mac (ccm)”. In: (2003).

[Zig15] Zigbee Alliance. *ZigBee Specification*. Tech. rep. Aug. 2015.

[Zil16] T. Zillner. “ZigBee Exploited: The good, the bad and the ugly”. In: *Magdeburger Journal zur Sicher* (2016).

Included Publications

An AEAD variant of the Grain stream cipher

Abstract

A new Grain stream cipher, denoted Grain-128AEAD is presented, with support for authenticated encryption with associated data. The cipher takes a 128-bit key and a 96-bit IV and produces a pseudo random sequence that is used for encryption and authentication of messages. The design is based on Grain-128a but introduces a few changes in order to increase the security and protect against recent cryptanalysis results. The MAC is 64 bits, as specified by the NIST requirements in their lightweight security standardization process.

1 Introduction

Due to widespread usage of Internet of Things (IoT) technology, the need of protection from security threats on resource-constrained devices has been continuously growing. Since 2003, the cryptography community has already recognized the importance of this need, and researchers and developers have focused on cryptography tailored to limited computation resources in hardware and software implementations. This has resulted in opening up a new subfield of cryptography, namely, lightweight cryptography, which led to the launch of the eSTREAM project. This project running from 2004 to 2008 can be viewed as the most important research activity in the area of lightweight stream ciphers. The eSTREAM portfolio contains four software-oriented ciphers and three hardware-oriented ciphers.

Martin Hell, Thomas Johansson, Willi Meier, Jonathan Sönnerup, and Hirotaka Yoshida. "An AEAD Variant of the Grain Stream Cipher". In *Codes, Cryptology and Information Security: Third International Conference, C2SI 2019, Rabat, Morocco*. Springer Nature Switzerland AG. pp. 55-71.

From an industrial point of view, it has been widely recognized that *maturity* is important regarding deployment of cryptographic mechanisms. In fact, the ISO/IEC 18033-1 [Int15a] standard states this property as one criteria for inclusion of cryptographic mechanisms. The concept behind this it that, if cryptographic mechanisms are standardized, they should be in the public domain for many years. In this way, security and performance analysis of them can be performed by third parties, which would give the users a significant amount of confidence in security. The above mentioned eSTREAM project activity affected industry: one of the eSTREAM portfolio cipher, Trivium [CP08], is standardized in the lightweight stream cipher standard, ISO/IEC 29192-3 [Int12] together with Enocoro [Wat+10]. Grain-128a, which is based on the eSTREAM portfolio cipher Grain v1, is standardized in ISO/IEC 29167-13 [Int15b] for the RFID application standard.

Despite of the above extensive academic and industry efforts, there is still an important gap to fill. There has been no authenticated encryption with associated data (AEAD) mechanism that meets very severe performance requirements in hardware and still offers 128-bit security, accompanied by serious evidence on cryptanalysis. In 2013, NIST initiated a lightweight cryptography project, followed by two workshops on the same subject. In 2017, NIST published a call for submissions for lightweight cryptographic mechanisms. One remarkable feature is that NIST requires each submission to implement the AEAD functionality. In [Ban+18a], it was shown that lightweight stream ciphers are typically more suitable than lightweight block ciphers for energy optimization when encrypting longer messages, in particular when the speed can be increased at the expense of moderate extra hardware. Thus, a lightweight stream cipher seems to be a good starting point for a lightweight AEAD design.

This paper presents Grain-128AEAD, an authenticated encryption algorithm with support for associated data. The specification is in line with the requirements given by NIST and is based on the Grain stream cipher family. More specifically, it is closely based on Grain-128a, introduced in 2011, which has, already for several years, been analyzed in the literature. To benefit from the maturity of the Grain family, our strategy in the design of Grain-128AEAD is to have the changes made to Grain-128a as small as possible. Grain-128a is in turn based on Grain v1 and Grain-128, which have both been extensively analyzed, providing much insight into the security of the design approach. All Grain stream ciphers also allow the throughput to be increased by adding additional copies of the Boolean functions involved.

Industrial relevance of the Grain family can be explained as follows: Grain-128a receives a lot of attention from industry. ISO/IEC 29167-13:2015 specifying Grain-128a has been adopted in industrial applications. For instance, the passive IT70 RFID tag [Hon17] that Honeywell has designed for automotive applications implements this security standard.

The outline of the paper is as follows. In Section 2 the specification of the

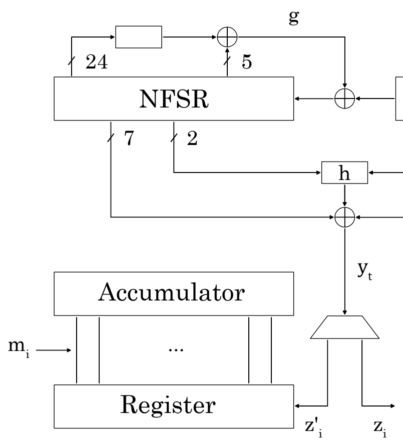


Figure 1: An overview of the building blocks in Grain-128AEAD.

new primitive is given. Then the overall design rationale, motivating the design choices, are given in Section 3. A security analysis, focusing on cryptanalysis of Grain-128a is then given in Section 4. The hardware implementation is described in Section 5 and the paper is concluded in Section 6.

2 Design Details

Grain-128AEAD consists of two main building blocks. The first is a pre-output generator, which is constructed using a Linear Feedback Shift Register (LFSR), a Non-linear Feedback Shift Register (NFSR) and a pre-output function, while the second is an authenticator generator consisting of a shift register and an accumulator. The design is very similar to Grain-128a, but has been modified to allow for larger authenticators and to support AEAD. Moreover, the modes of usage have been updated.

2.1 Building Blocks and Functions

The pre-output generator generates a stream of pseudo-random bits, which are used for encryption and the authentication tag. It is depicted in Fig. 1. The content of the 128-bit LFSR is denoted $S_t = [s_0^t, s_1^t, \dots, s_{127}^t]$ and the content of the 128-bit NFSR is similarly denoted $B_t = [b_0^t, b_1^t, \dots, b_{127}^t]$. These two shift registers represent the 256-bit state of the pre-output generator.

The primitive feedback polynomial of the LFSR, defined over GF(2) and denoted $f(x)$, is defined as

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128}.$$

The corresponding update function of the LFSR is given by

$$\begin{aligned} s_{127}^{t+1} &= s_0^t + s_7^t + s_{38}^t + s_{70}^t + s_{81}^t + s_{96}^t \\ &= \mathcal{L}(S_t). \end{aligned}$$

The nonlinear feedback polynomial of the NFSR, denoted $g(x)$ and also defined over GF(2), is defined as

$$\begin{aligned} g(x) = 1 &+ x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} \\ &+ x^{63}x^{67} + x^{69}x^{101} + x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117} \\ &+ x^{46}x^{50}x^{58} + x^{103}x^{104}x^{106} + x^{33}x^{35}x^{36}x^{40} \end{aligned}$$

and the corresponding update function is given by

$$\begin{aligned} b_{127}^{t+1} &= s_0^t + b_0^t + b_{26}^t + b_{56}^t + b_{91}^t + b_{96}^t + b_3^t b_{67}^t + b_{11}^t b_{13}^t \\ &+ b_{17}^t b_{18}^t + b_{27}^t b_{59}^t + b_{40}^t b_{48}^t + b_{61}^t b_{65}^t + b_{68}^t b_{84}^t \\ &+ b_{22}^t b_{24}^t b_{25}^t + b_{70}^t b_{78}^t b_{82}^t + b_{88}^t b_{92}^t b_{93}^t b_{95}^t \\ &= s_0^t + \mathcal{F}(B_t). \end{aligned}$$

Nine state variables are taken as input to a Boolean function $h(x)$. Two of these bits are taken from the NFSR and seven are taken from the LFSR. The function is defined as

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8,$$

where the variables $x_0, \ldots, x_8$ correspond to the state variables $b_{12}^t, s_8^t, s_{13}^t, s_{20}^t, b_{95}^t, s_{42}^t, s_{60}^t, s_{79}^t$ and s_{94}^t, respectively.

The output of the pre-output generator, is then given by the pre-output function

$$y_t = h(x) + s_{93}^t + \sum_{j \in \mathcal{A}} b_j^t,$$

where $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$.

The authenticator generator consists of a shift register, holding the most recent 64 odd bits from the pre-output, and an accumulator. Both are of size 64 bits. We denote the content of the accumulator at instance i as $A_i = [a_0^i, a_1^i, \ldots, a_{63}^i]$. Similarly, the content of the shift register is denoted $R_i = [r_0^i, r_1^i, \ldots, r_{63}^i]$.

2.2 Key and IV Initialization

Before the pre-output can be used as keystream and for authentication, the internal state of the pre-output generator and the authenticator generator registers are initialized with a key and IV. Denote the key bits as k_i, $0 \le i \le 127$ and the IV bits as IV_i, $0 \le i \le 95$. Then the state is initialized as follows. The 128 NFSR bits are loaded with the bits of the key $b_i^0 = k_i$, $0 \le i \le 127$ and the first 96 LFSR elements are loaded with the IV bits, $s_i^0 = IV_i$, $0 \le i \le 95$. The last 32 bits of the LFSR are filled with 31 ones and a zero, $s_i^0 = 1, 96 \le i \le 126,\ s_{127}^0 = 0$. Then, the cipher is clocked 256 times, feeding back the pre-output function and XORing it with the input to both the LFSR and the NFSR, i.e.,

$$\begin{aligned} s_{127}^{t+1} &= \mathcal{L}(S_t) + y_t, \quad 0 \le t \le 255, \\ b_{127}^{t+1} &= s_0^t + \mathcal{F}(B_t) + y_t, \quad 0 \le t \le 255. \end{aligned}$$

Once the pre-output generator has been initialized, the authenticator generator is initialized by loading the register and the accumulator with the pre-output keystream as

$$\begin{aligned} a_j^0 &= y_{256+j}, \qquad 0 \le j \le 63, \\ r_j^0 &= y_{320+j}, \qquad 0 \le j \le 63. \end{aligned}$$

When the register and the accumulator are initialized, the key is simultaneously shifted into the LFSR,

$$s_{127}^{t+1} = \mathcal{L}(S_t) + k_{t-256}, \quad 256 \le t \le 383,$$

while the NFSR is updated as

$$b_{127}^{t+1} = s_0^t + \mathcal{F}(B_t), \quad 256 \le t \le 383.$$

Thus, when the cipher has been fully initialized the LFSR and the NFSR states are given by S_{384} and B_{384}, respectively, and the register and accumulator are given by R_0 and A_0, respectively. The initialization procedure is summarized in Fig 2.

2.3 Operating Mode

For a message **m** of length L, denoted $m_0, m_1, \ldots, m_{L-1}$, set $m_L = 1$ as padding in order to ensure that **m** and **m**$||0$ have different tags.

After initializing the pre-output generator, the pre-output is used to generate keystream bits z_i for encryption and authentication bits z_i' to update the register in the accumulator generator. The keystream is generated as

$$z_i = y_{384+2i},$$

i.e., every even bit (counting from 0) from the pre-output generator is taken as a keystream bit. The authentication bits are generated as

$$z_i' = y_{384+2i+1},$$

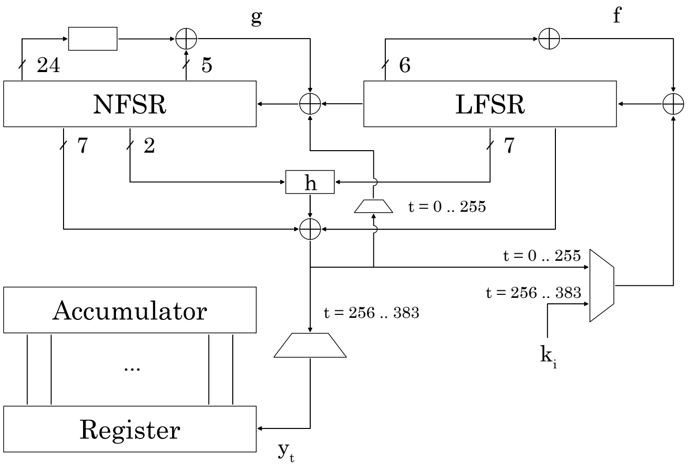


Figure 2: An overview of the initialization in Grain-128AEAD. Note that, in hardware, the accumulator initialization is realized by first loading 64 pre-output bits into the register, followed by moving them to the accumulator.

i.e., every odd bit from the pre-output generator is taken as an authentication bit. The message is encrypted as

$$c_i = m_i \oplus z_i, \quad 0 \leq i < L.$$

The accumulator is updated as

$$a_j^{i+1} = a_j^i + m_i r_j^i, \qquad 0 \leq j \leq 63, \quad 0 \leq i \leq L,$$

and the shift register is updated as

$$\begin{aligned} r_{63}^{i+1} &= z_i', \\ r_j^{i+1} &= r_{j+1}^i, \qquad 0 \leq j \leq 62. \end{aligned}$$

An AEAD scheme allows for data that is authenticated, but unencrypted. Grain-128AEAD achieves this simply by explicitly setting $y_{384+2i} = 0$ for bits that should not be encrypted, but should still be authenticated. This means that it is possible to control the associated data on bit level, and this data can appear anywhere in the message.

3 Design Rationale

This section presents a short overview of the Grain stream ciphers and how the design has evolved through the different versions. It also enumerates and discusses the differences between Grain-128a and the proposed Grain-128AEAD.

3.1 A Short History of the Grain Family of Stream Ciphers

The Grain family of stream ciphers are based on the idea behind the nonlinear filter generator. In a nonlinear filter, an LFSR is used to provide a sequence with large period, and a nonlinear function, taking parts of the LFSR sequence as input, is used to add nonlinearity to the keystream sequence. Much work has been put into analyzing the nonlinear filter generator and it is clear that it is very difficult to design a secure nonlinear filter generator with a reasonable hardware footprint [BL06]. In particular algebraic attacks have been shown to be very strong against this design, see e.g., [Cou03a; MPC04].

In order to better withstand algebraic attacks, and to make the relation between state/key and keystream more complex, Grain adds an NFSR to the nonlinear combiner. The initial submission to the ECRYPT eSTREAM project was analyzed in [KHK06; BGM06], showing that the nonlinear functions required higher resiliency and nonlinearity. The modified design was subsequently published as Grain v1 [HJM07] and was later selected for the final portfolio in eSTREAM. Grain v1 uses an 80-bit key, and a 128-bit key variant was proposed in [Hel+06b]. Based on previous results on the Grain construction, Grain-128 was more aggressively designed, making the nonlinear NFSR feedback function of degree 2, but with high nonlinearity and resiliency. The relatively small functions compensated for the fact that the shift registers were increased to 128 bits each, which increased the hardware footprint. The low degree functions were exploited in [Aum+09; Sta10] in order to cryptanalyze a significant number of initializations rounds. These results suggested that the nonlinear functions needed a higher security margin. Grain-128a was proposed in [Ågr+11b], and in addition to increasing the degree of the nonlinear feedback function, an optional authentication mode was added. Work on Grain-128 were subsequently improved [DS11a; Din+11; Fu+17; KHS18], emphasizing the need for more complex Boolean functions, and Grain-128 is considered broken and should not be used. The design proposed in this paper, Grain-128AEAD, is closely based on Grain-128a, using the same feedback and output functions. However, slight modifications have been made in order to add security and make it resistant to the attack proposed in [Tod+18].

3.2 Differences Between Grain-128AEAD and Grain-128a.

Grain-128AEAD takes Grain-128a as starting point, but introduces a number of slight modifications. The modifications are primarily motivated by the NIST Lightweight Cryptography Standardization Process, but inspiration also comes from recent results in [Tod+18; HK18].

Larger MACs

The register and the authenticator has been increased to 64 bits (instead of 32 bits) in order to allow for authentication tags (MACs) of size 64 bits.

No Encryption-only Mode

Grain-128a allowed for an operation mode with only encryption, where the authentication was removed. This mode resulted in smaller hardware footprint since the two additional registers, and their associated logic, could be left out from an implementation. The encryption-only mode was also more efficient since the initialization process does not include initializing the register and the accumulator, and every pre-output bit was used as keystream. The proposed Grain-128AEAD is a pure authenticated encryption algorithm, and authentication of data is always supported. Thus, there is only one mode of operation.

Initialization Hardening

Based on the ideas in [HK18] and used in Lizard [HKM17], Grain-128AEAD re-introduces the key into the internal state during the initialization clock cycles. More specifically, it is serially shifted into the LFSR in parallell to the initialization of the register and the accumulator. Several variants can be considered here, including where and when to add the key. The LFSR is chosen due to the fact that if the LFSR is recovered (e.g., in a fast correlation attack as in [Tod+18]), it is comparably easy to recover the NFSR state. Moreover, since the LFSR output is XORed with the NFSR input, the key bits will continue to affect also the NFSR during pre-output generation. As for when, we choose to re-introduce it during the last 128 clocks of the initialization. This provides maximum separation between its first introduction in the key loading part, where the key is loaded into the NFSR, and when it is re-introduced. Relations between keys are e.g., more difficult to exploit if the key is properly mixed into the state before the key is re-introduced.

By introducing the key as the last part of the initialization, we achieve the attractive effect that a state recovery attack does not immediately imply key recovery, as was the case for previous versions of Grain. While a state recovery would still render the cipher to be considered broken, the practical effect to deployed devices is highly limited. Recovering the state will only compromise the security of the current message, and not all messages using the same key.

Keystream Limitation

Grain stream ciphers have been designed to allow for encrypting large chunks of data using the same key/IV pair. Previously, the Grain ciphers have not had any explicit limitation on the keystream length. However, to rule out attacks that use very large keystream sequences, Grain-128AEAD restricts the number of keystream bits for each key/IV to 2^{80}. We believe that this is well above what will be needed in the foreseeable future. Restricting the number of keystream bits will also make attacks that use linear approximations more difficult, e.g., [Tod+18].

4 Security Analysis

The security of the Grain family of stream ciphers has been investigated by a large number of third party analysts, publishing various analysis results on the different variants of Grain. Since its first introduction in 2005, much have been learned about the construction and the design approach. There have also been several published ciphers inspired by the design, e.g., Sprout [AM15] and its successor Plantlet [MAM16]. Also Fruit [GHX16] and Fruit-80 [AH18] are based on the same design idea. These ciphers have in common that they attempt to realize extremely resource constrained encryption. To minimize the hardware footprint, the key is assumed to be stored in non-volatile memory (NVM) on a device, and this memory is made part of the cryptographic algorithm. Since the key needs to be stored on a device anyway, using the key directly from NVM in the algorithm does not impose additional hardware to the construction. This is not the case for Grain, as we allow the key to be updated in the device, and the key storage is not a part of the cipher. Still, the fact that the above mentioned ciphers use the Grain design idea shows that the design seems to be very suitable for lightweight cryptography.

4.1 General Security Analysis

A main class of attacks on stream ciphers is the Time/Memory/Data tradeoff (TMD-TO) attack, an efficient method of finding either the key or the state of ciphers by balancing between time, memory and keystream data. This can sometimes be much more efficient and more practically applicable than a simple exhaustive key search attack. Some stream ciphers are vulnerable to TMD-TO attacks and their effective key lengths could then be reduced. This typically happens if the state size is too small. A famous practical TMD-TO attack on A5/1 was given in [BSW01].

A TMD-TO attack consists of two parts. The first is a preprocessing phase, during which a table is constructed. The mapping of different keys or internal states to some keystream segment is computed and stored in the table. It is sorted on keystream segments and this process is assumed to use time complexity P and memory M. In the second (real-time) phase, the attacker has intercepted D keystream segments and search for a collision with the table with time complexity T. A collision will recover the corresponding input. By a trade-off between parameters P, D, M, and T, attackers can devise attacks according to available time, memory and data. Examples of tradeoffs are Babbage-Golic (BG) [Bab95; Gol97] and Biryukov-Shamir (BS) [BS00] with curves $TM = N$, $P = M$ with $T \leq D$;

and $MT^2D^2 = N^2$, $P = N/D$ with $T \geq D^2$, where N is the input space, respectively.

For Grain-128AEAD, attackers have no direct way to reconstruct the internal state, since the cipher has an internal state of size 256 bits (128-bit LFSR + 128-bit NFSR), i.e. $N = 2^{256}$. The best attack complexity achieved under BG tradeoff is with $T = M = D = N^{1/2} = 2^{128}$, which is not favourable compared to exhaustive key search. Also the BS tradeoff does not give complexity parameters of particular interest. Some improvements to TMD-TO attacks can be achieved through socalled BSW sampling [BSW01] and the performance of such an approach is characterized by the sampling resistance of the stream cipher. Various generalizations of the concept of sampling resistance can be considered, e.g. [JZW15], but it seems unlikely that this will lead to an attack with better performance than a standard Hellman-type time-memory tradeoff attack on the keyspace, a generic attack applicable to any cipher. Also, our limit on the length of keystreams affects such attacks.

Another class of general attacks are algebraic attacks, where the attacker derives a system of nonlinear equations in unknown key bits or unknown state bits and then solves the system. In general, solving a system of nonlinear equations is not known to be solvable in polynomial time, but there might be special cases that can be solved efficiently [Cou+00]. Due to the NFSR, the degree of the equations will gradually increase and it does not look promising to try to derive a system of nonlinear equations due to this property as well as the algebraic degree of the h function.

A general cryptanalytic technique is a guess-and-determine attack, where one guesses parts of the state and then from the keystream tries to determine other parts of the state. The goal is to guess as few positions as possible and determine as many as possible from equations involving the keystream. Again, since the dependence between a keystream symbol and the state includes many different positions in the state and some of them in nonlinear expressions, one has to guess a large portion of state variables in order to use an equation to determine a single state variable.

Being a binary additive stream cipher, Grain-128AEAD does not allow reuse of a key/IV pair since this will leak information about the corresponding plaintexts. Moreover, since Grain-128AEAD closely resembles Grain-128a, a key/IV pair used in one cipher may also not be reused in the other. Such cross-cipher key/IV reuse in a related cipher model is outside the security model of Grain-128AEAD.

In the subsequent subsections, we now describe the attacks that we consider as the main threat against lightweight stream ciphers in general and Grain-128AEAD in particular.

4.2 Correlation Attacks

Grain-128a was designed to resist conventional (fast) correlation attacks that exploit correlations between the state of the LFSR and the corresponding key stream.

There has been devised a fast correlation attack on small state Grain-like stream ciphers in [ZGM17]. Due a much bigger state, this attack does not apply to Grain-128a. On the other hand, a recent paper [Tod+18] reveals that there are multiple linear approximations in Grain-128a that together with a viewpoint based on a finite field allow a fast correlation attack on the raw encryption mode of Grain-128a (and on the other members of the Grain family), where every keystream bit is assumed to be accessible by an opponent. This attack recovers the state of Grain-128a with data and time complexity of about 2^{114}. The data needs to come from the same secret key and the same IV.

It should be noted that this fast correlation attack does not apply to Grain-128a in authentication mode, as then only every second key stream bit may be accessible to an opponent. Thus, it does not apply to Grain-128AEAD.

4.3 Chosen IV Attacks

A variety of chosen IV attacks on Grain have been proposed, in both fixed key scenario as well as in the related key setting, and either for distinguishing purpose or for key recovery. In a fixed key scenario, chosen IV attacks have been devised on reduced-round versions using conditional differentials and using cube attacks, or combinations of both [KMN10; LM12; GH17; MTQ16]. On Grain-128, a dynamic cube attack has been developed that succeeds in finding the secret key for the full 256-round initialization for a fraction of keys, [Din+11]. Dynamic cube attacks have not been successful on Grain-128a thus far. Most of these results are experimental in nature, and do work only if the computational effort is practically feasible.

More recently, division property has been developed to improve cube attacks. Division property is an iterated technique for integral distinguishers introduced by Todo, in [Tod15] and was applied initially to block ciphers. It turned out that it also applies to the initialization of stream ciphers, not only for distinguishers but also for key recovery. As opposed to conventional cube attacks, it can provide theoretical results. The latest result on Grain-128a in this direction is a key recovery on 184 initialization rounds, [Wan+18]. The data complexity is 2^{95}, and the computational complexity corresponds to about 2^{110} operations.

An attack that reaches the largest number of initialization rounds of Grain-128a in a fixed key scenario thus far is a conditional differential distinguishing attack and reaches 195 initialization rounds, but it works only for a fraction of all keys, [MTQ16].

The relevance of related key cryptanalysis of stream ciphers has been a subject of debate. A related key attack on Grain-128a in [DG13] recovers the secret key with a computational complexity 2^{96}, requiring 2^{96} chosen IVs and about 2^{104} keystream bits. It requires only 2 related keys. Another related key attack in [Ban+13] recovers the secret key using 2^{64} chosen IVs and 2^{32} related keys, where these figures need to be multiplied by some factor (about 2^{8}).

4.4 Fault Attacks

In the scenario of fault attacks on stream ciphers, the attacker is allowed to inject faults into the internal state, which means either flipping a binary value in memory or assigning a value to zero. By analyzing the difference in keystreams for the faulty and the fault-free case, one attempts to deduce the complete or some partial information about the internal state or the secret key. Fault attacks on stream ciphers have recently received some attention, starting with the work of Hoch and Shamir [HS04]. The most common methods of injecting faults is by using laser or through clock glitches. Fault attacks usually rely on assumptions that is beyond the model of cryptanalysis and for this reason one can often find rather efficient fault attacks on most ciphers. In some scenarios they are, however, not unrealistic and the exact complexity and the related requirements are of interest to study.

Fault attacks on the Grain family of stream ciphers were studied in [Cas+09] and [KR11a]. More recently, there was a number of papers providing improved attacks, [BMS12b; SBM15; BMS12a; BMS12c]. In [SBM15] the model is the most realistic one as it considers that the cipher has to be re-initialized only a few times and faults are injected to any random location and at any random clock cycle. No further assumptions are needed over location and timing for injections. In the attack one constructs algebraic equations based on the description of the cipher by introducing new variables so that the degrees of the equations do not increase. Following algebraic cryptanalysis, such equations based on both fault-free and faulty key-stream bits are collected. Then a solving phase using the SAT Solver recovers the state of any Grain member in minutes, For Grain v1, Grain-128 and Grain-128a, it uses only 10, 4 and 10 injected faults, respectively.

We stress that we are not claiming resistance against fault attacks for Grain-128AEAD. Rather, when fault attacks is a realistic threat, one has to implement protection mechanisms against fault injection.

5 Implementation

Lightweight ciphers are important in constrained devices. A minimal design is desirable, e.g., minimum area and very low power consumption since they often must operate for an extended period of time, without a battery change. In some cases, devices run without its own power supply, something that is often the case with RFID tags.

Grain-128AEAD can be constructed using primitive hardware building blocks, such as NAND gates, XOR gates and flip flops. In order to get an idea of the hardware footprint related to an implementation of the cipher, we implement the stream cipher using 65 nm library from ST Microelectronics, `stm065v536`. For synthesis and power simulation, the Synopsys Design Compiler 2013.12 is used. It can be noted that the result is highly dependent on what kind of gates are available and how the tool utilizes the standard cells. We define a 2-input NAND gate

Table 1: The gate count for different functions.

Function	Gate Count
NAND2	1.0
NAND3	1.5
NAND4	2.0
XOR2	2.5
XOR3	6.5
Flip flop	8.0

to have a gate count of 1 and other gate counts are given in relation to this NAND gate. An excerpt from the standard-cell library documentation is given in Table 1.

Table 2: Gate count for the different building blocks, for different levels of parallelization, s.

Building Block	Gate Count		
	$s = 1$	$s = 2$	$s = 32$
LFSR	1024	1024	1024
NFSR	1024	1024	1024
f	19	38	608
g	62.5	125	2000
h	41.5	83	1328
Control logic	219.5	475.5	942.5
Accumulator	512	512	512
Register	512	512	512
Accumulator logic	224	224	4160
Total	3638.5	4017.5	12110.5

We synthesize the design and extract the gate count for each building block. A summary of the gate count for each building block, and for different parallelization levels, is given in Table 2. The control logic and accumulator logic is extra circuitry and state machines for controlling the stream cipher, i.e., loading key and IV, multiplexing data, etc.

The gate count remains constant during synthesis, but the physical area, power and speed changes based on the optimization techniques employed. First, we synthesize the design at clock frequency 100 kHz. The design is synthesized for three levels of parallelization; 1, 2, and 32 times. The result is given in Table 3.

We also synthesize for the maximum possible speed, to achieve maximum throughput, without constraints on area. The results are given in Table 4.

Table 3: Implementation results running at 100 kHz, for different levels of parallelization.

Parallelization	Area	Power	Throughput
1	4934 μm^2	313 nW	50 kbit/s
2	5336 μm^2	368 nW	100 kbit/s
32	16853 μm^2	574 nW	1600 kbit/s

Table 4: Implementation results running at maximum possible speed, for different levels of parallelization.

Parallelization	Speed	Area	Power	Throughput
1	1.12 GHz	5258 μm^2	3.6 mW	560 Mbit/s
2	1.18 GHz	5629 μm^2	4.3 mW	1.18 Gbit/s
32	662 MHz	17632 μm^2	4.0 mW	10.59 Gbit/s

6 Conclusions

We have presented Grain-128AEAD, a new cipher in the Grain family. It is closely based on Grain-128a and takes advantage of the well-analyzed design principle behind the Grain stream ciphers. By making slight modifications to Grain-128a, the cipher meets the requirements in the NIST lightweight standardization process, providing 64-bit MAC, 128-bit key and 96-bit IV. The hardware footprint makes the cipher well suited for constrained environments, but the design is flexible enough to allow for also very high speed requirements at the expense of additional hardware.

References

[Ågr+11b] M. Ågren et al. “Grain-128a: a new version of Grain-128 with optional authentication”. In: *International Journal of Wireless and Mobile Computing* 5.1 (2011), pp. 48–59.

[AH18] V. Amin Ghafari and H. Hu. “Fruit-80: A Secure Ultra-Lightweight Stream Cipher for Constrained Environments”. In: *Entropy* 20.3 (2018), p. 180.

[AM15] F. Armknecht and V. Mikhalev. “On Lightweight Stream Ciphers with Shorter Internal States”. In: *Fast Software Encryption*. Ed. by G. Leander. Springer Berlin Heidelberg, 2015, pp. 451–470.

[Aum+09] J.-P. Aumasson et al. “Efficient FPGA implementations of high-dimensional cube testers on the stream cipher Grain-128”. In:

SHARCS'09 Special-purpose Hardware for Attacking Cryptographic Systems (2009), p. 147.

[Bab95] S. Babbage. "Improved "exhaustive search" attacks on stream ciphers". In: *IET Conference Proceedings* (Jan. 1995), 161–166(5).

[Ban+13] S. Banik et al. "A Chosen IV Related Key Attack on Grain-128a". In: *Information Security and Privacy*. Ed. by C. Boyd and L. Simpson. Springer Berlin Heidelberg, 2013, pp. 13–26.

[Ban+18a] S. Banik et al. "Towards Low Energy Stream Ciphers". In: *IACR Transactions on Symmetric Cryptology* 2018.2 (June 2018), pp. 1–19.

[BGM06] C. Berbain, H. Gilbert, and A. Maximov. "Cryptanalysis of Grain". In: *Fast Software Encryption*. Ed. by M. Robshaw. Springer Berlin Heidelberg, 2006, pp. 15–29.

[BL06] A. Braeken and J. Lano. "On the (Im)Possibility of Practical and Secure Nonlinear Filters and Combiners". In: *Selected Areas in Cryptography*. Ed. by B. Preneel and S. Tavares. Springer Berlin Heidelberg, 2006, pp. 159–174.

[BMS12a] S. Banik, S. Maitra, and S. Sarkar. "A Differential Fault Attack on Grain-128a Using MACs". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by A. Bogdanov and S. Sanadhya. Springer Berlin Heidelberg, 2012, pp. 111–125.

[BMS12b] S. Banik, S. Maitra, and S. Sarkar. "A Differential Fault Attack on the Grain Family of Stream Ciphers". In: *Cryptographic Hardware and Embedded Systems – CHES 2012*. Ed. by E. Prouff and P. Schaumont. Springer Berlin Heidelberg, 2012, pp. 122–139.

[BMS12c] S. Banik, S. Maitra, and S. Sarkar. "A Differential Fault Attack on the Grain Family under Reasonable Assumptions". In: *Progress in Cryptology - INDOCRYPT 2012*. Ed. by S. Galbraith and M. Nandi. Springer Berlin Heidelberg, 2012, pp. 191–208.

[BS00] A. Biryukov and A. Shamir. "Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers". In: *Advances in Cryptology — ASIACRYPT 2000*. Ed. by T. Okamoto. Springer Berlin Heidelberg, 2000, pp. 1–13.

[BSW01] A. Biryukov, A. Shamir, and D. Wagner. "Real Time Cryptanalysis of A5/1 on a PC". In: *Fast Software Encryption*. Ed. by G. Goos et al. Springer Berlin Heidelberg, 2001, pp. 1–18.

[Cas+09] G. Castagnos et al. "Fault analysis of GRAIN-128". In: *Hardware-Oriented Security and Trust, IEEE International Workshop on(HST)*. 2009, pp. 7–14.

[Cou+00] N. Courtois et al. “Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations”. In: *Advances in Cryptology — EUROCRYPT 2000*. Ed. by B. Preneel. 2000, pp. 392–407.

[Cou03a] N. T. Courtois. “Fast algebraic attacks on stream ciphers with linear feedback”. In: *Annual International Cryptology Conference, CRYPTO 2003*. Springer. 2003, pp. 176–194.

[CP08] C. D. Cannière and B. Preneel. “Trivium”. In: *New Stream Cipher Designs - The eSTREAM Finalists* (2008), pp. 244–266.

[DG13] L. Ding and J. Guan. “Related key chosen IV attack on Grain-128a stream cipher”. In: *IEEE Transactions on Information Forensics and Security* 8.5 (2013), pp. 803–809.

[Din+11] I. Dinur et al. “An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware”. In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by D. H. Lee and X. Wang. Springer Berlin Heidelberg, 2011, pp. 327–343.

[DS11a] I. Dinur and A. Shamir. “Breaking Grain-128 with Dynamic Cube Attacks”. In: *Fast Software Encryption*. Ed. by A. Joux. Springer Berlin Heidelberg, 2011, pp. 167–187.

[Fu+17] X. Fu et al. “Determining the Nonexistent Terms of Non-linear Multivariate Polynomials: How to Break Grain-128 More Efficiently.” In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 412.

[GH17] V. A. Ghafari and H. Hu. “A New Chosen IV Statistical Attack on Grain-128a Cipher”. In: *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2017 International Conference on*. IEEE. 2017, pp. 58–62.

[GHX16] V. A. Ghafari, H. Hu, and C. Xie. “Fruit: ultra-lightweight stream cipher with shorter internal state”. In: *eSTREAM, ECRYPT Stream Cipher Project* (2016).

[Gol97] J. D. Golić. “Cryptanalysis of Alleged A5 Stream Cipher”. In: *Advances in Cryptology — EUROCRYPT ’97*. Ed. by W. Fumy. Springer Berlin Heidelberg, 1997, pp. 239–255.

[Hel+06b] M. Hell et al. “A stream cipher proposal: Grain-128”. In: *Information Theory, 2006 IEEE International Symposium on*. IEEE. 2006, pp. 1614–1618.

[HJM07] M. Hell, T. Johansson, and W. Meier. “Grain: a stream cipher for constrained environments”. In: *International Journal of Wireless and Mobile Computing* 2.1 (2007), pp. 86–93.

[HK18] M. Hamann and M. Krause. “On stream ciphers with provable beyond-the-birthday-bound security against time-memory-data tradeoff attacks”. In: *Cryptography and Communications* 10.5 (2018), pp. 959–1012.

[HKM17] M. Hamann, M. Krause, and W. Meier. “LIZARD–A lightweight stream cipher for power-constrained devices”. In: *IACR Transactions on Symmetric Cryptology* 2017.1 (2017), pp. 45–79.

[Hon17] Honeywell. “IT70 Secure Passive RFID Tag”. In: *Technical Specifications* (2017).

[HS04] J. J. Hoch and A. Shamir. “Fault Analysis of Stream Ciphers”. In: *Cryptographic Hardware and Embedded Systems - CHES 2004*. Ed. by M. Joye and J.-J. Quisquater. Springer Berlin Heidelberg, 2004, pp. 240–253.

[Int12] International Organization for Standardization. *ISO/IEC 29192-3:2012 Information technology – Security techniques – Lightweight cryptography – Part 3: Stream ciphers*. 2012.

[Int15a] International Organization for Standardization. *ISO/IEC 18033-1:2015 Information technology – Security techniques – Encryption algorithms – Part 1: General*. 2015.

[Int15b] International Organization for Standardization. *ISO/IEC 29167-13:2015 Information technology — Automatic identification and data capture techniques — Part 13: Crypto suite Grain-128A security services for air interface communications*. 2015.

[JZW15] L. Jiao, B. Zhang, and M. Wang. “Two Generic Methods of Analyzing Stream Ciphers”. In: *Information Security*. Ed. by J. Lopez and C. J. Mitchell. Springer International Publishing, 2015, pp. 379–396.

[KHK06] S. Khazaei, M. M. Hasanzadeh, and M. S. Kiaei. “Linear Sequential Circuit Approximation of Grain and Trivium Stream Ciphers.” In: *IACR Cryptology ePrint Archive* 2006 (2006), p. 141.

[KHS18] L. Karlsson, M. Hell, and P. Stankovski. “Not So Greedy: Enhanced Subset Exploration for Nonrandomness Detectors”. In: *Information Systems Security and Privacy*. Ed. by P. Mori, S. Furnell, and O. Camp. Springer International Publishing, 2018, pp. 273–294.

[KMN10] S. Knellwolf, W. Meier, and M. Naya-Plasencia. “Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems”. In: *Advances in Cryptology - ASIACRYPT 2010*. Ed. by M. Abe. Springer Berlin Heidelberg, 2010, pp. 130–145.

[KR11a] S. Karmakar and D. Roy Chowdhury. “Fault Analysis of Grain-128 by Targeting NFSR”. In: *Progress in Cryptology – AFRICACRYPT 2011*. Ed. by A. Nitaj and D. Pointcheval. Springer Berlin Heidelberg, 2011, pp. 298–315.

[LM12] M. Lehmann and W. Meier. “Conditional Differential Cryptanalysis of Grain-128a”. In: *Cryptology and Network Security*. Ed. by J. Pieprzyk, A.-R. Sadeghi, and M. Manulis. Springer Berlin Heidelberg, 2012, pp. 1–11.

[MAM16] V. Mikhalev, F. Armknecht, and C. Müller. “On ciphers that continuously access the non-volatile key”. In: *IACR Transactions on Symmetric Cryptology* (2016), pp. 52–79.

[MPC04] W. Meier, E. Pasalic, and C. Carlet. “Algebraic attacks and decomposition of Boolean functions”. In: *International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2004*. Springer. 2004, pp. 474–491.

[MTQ16] Z. Ma, T. Tian, and W.-F. Qi. “Conditional differential attacks on Grain-128a stream cipher”. In: *IET Information Security* 11.3 (2016), pp. 139–145.

[SBM15] S. Sarkar, S. Banik, and S. Maitra. “Differential Fault Attack against Grain family with very few faults and minimal assumptions”. In: *IEEE Transactions on Computers* 64.6 (2015), pp. 1647–1657.

[Sta10] P. Stankovski. “Greedy Distinguishers and Nonrandomness Detectors”. In: *Progress in Cryptology - INDOCRYPT 2010*. Ed. by G. Gong and K. C. Gupta. Springer Berlin Heidelberg, 2010, pp. 210–226.

[Tod+18] Y. Todo et al. “Fast Correlation Attack Revisited”. In: *Advances in Cryptology – CRYPTO 2018*. Ed. by H. Shacham and A. Boldyreva. Springer International Publishing, 2018, pp. 129–159.

[Tod15] Y. Todo. “Structural Evaluation by Generalized Integral Property”. In: *Advances in Cryptology – EUROCRYPT 2015*. Ed. by E. Oswald and M. Fischlin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 287–314.

[Wan+18] Q. Wang et al. “Improved Division Property Based Cube Attacks Exploiting Algebraic Properties of Superpoly”. In: *Advances in Cryptology – CRYPTO 2018*. Ed. by H. Shacham and A. Boldyreva. Springer International Publishing, 2018, pp. 275–305.

[Wat+10] D. Watanabe et al. “Update on Enocoro stream cipher”. In: *2010 International Symposium On Information Theory Its Applications*. Oct. 2010, pp. 778–783.

[ZGM17] B. Zhang, X. Gong, and W. Meier. "Fast Correlation Attacks on Grain-like Small State Stream Ciphers". In: *IACR Trans. Symmetric Cryptol.* 2017.4 (2017), pp. 58–81.

Test Vectors

Here, we give some test vectors for different keys, IVs, and messages. The test vectors are given in hexadecimal, e.g., the key

```
0x01234FFFFFFFFFFFFFFFFFFFFFFFFFFF
```

corresponds to

$$(k_0, ..., k_{127}) = (0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, ..., 1).$$

The message stream is given with the padding included. A padding bit of 1 equals a padding byte of 0x80. Note that for an empty message, the message stream is just the padding.

```
Key:       0x00000000000000000000000000000000
IV:        0x000000000000000000000000
Keystream: 0xc800a52f948b89b85cee6cfd8571f90f
Message:   0x80
Tag:       0xaab555c073e67664

Key:       0x0123456789abcdef123456789abcdef0
IV:        0x0123456789abcdef12345678
Keystream: 0xc2b918c6baf6dea0865200d46858a37b
Message:   0xFF80
Tag:       0x782f4c4a8907ba7f
```

Efficient Hardware Implementations of Grain-128AEAD

Abstract

We implement the Grain-128AEAD stream cipher in hardware, using a 65 nm library. By exploring different optimization techniques, both at RTL level but also during synthesis, we first target high throughput, then low power. We reach over 33 Gb/s targeting a high-speed design, at expense of power and area. We also show that, when targeting low power, the design only requires 0.23 µW running at 100 kHz. By unrolling the design, the energy consumed when encrypting a fixed length message decreases, making the 64 parallelized version the most energy efficient implementation, requiring only 11.2 nJ when encrypting a 64 kbit message. At the same time, the best throughput/power ratio is achieved at a parallelization of 4.

1 Introduction

Due to the growth and widespread use of resource-constrained connected devices, e.g., in the Internet of Things (IoT), the need for protection against security threats has increased. RFID devices, smart cards, and sensor networks often require low power consumption as they are driven by batteries. The cost of manufacturing an IC chip is correlated to the area. Hence, area efficient designs are needed to

Jonathan Sönnerup, Martin Hell, Mattias Sönnerup, and Ripudaman Khattar. "Efficient Hardware Implementations of Grain-128AEAD". In *Progress in Cryptology – INDOCRYPT 2019 - 20th International Conference on Cryptology Proceedings, Hyderabad, India*. Springer Gabler. pp. 495-513.

reduce the cost when producing large quantities. This puts a demand, not only on the architectural design, but also on the implementation. The implementation may vary largely depending on what techniques are being utilized, both during programming (HDL), but also during synthesis, if aiming for an ASIC. At the same time, high-speed implementations are required for environments with much data, and where low latency is needed.

There are a large number of proposed cryptographic algorithms and several attempts have been made towards identifying suitable algorithms for widespread adoption, e.g., NESSIE, ECRYPT, CRYPTREC and the NIST AES contest. The successful standardization of AES has been followed by more NIST initiatives, most notably the SHA-3 competition, the Post-Quantum Cryptography Standardization Process and the recent Lightweight Cryptography Standardization Process. The latter particularly addresses the need for algorithms that are specifically targeting resource constrained environments [Nat18].

Grain-128AEAD is an instance of the Grain family of stream ciphers. Grain was first proposed in 2005, as an 80-bit stream cipher. The 128-bit variant Grain-128 was presented in 2006 [Hel+06a], and was successfully cryptanalyzed in [DS11b]. Building upon previous analysis results, a new 128-bit variant with authentication (MAC) support, Grain-128a, was proposed in 2011 [Ågr+11a]. It has also been adopted as an ISO standard [Int15b]. Most recently, Grain-128AEAD supporting Authenticated Encryption with Associated Data, was proposed in 2019 and also submitted to the above mentioned NIST Lightweight Cryptography Standardization Process [Hel+19a; Hel+19b]. It is built upon the Grain-128a cipher, but with some added features. There have been several implementations of Grain-128a, most notably the work in [MD13], where the authors utilize Galois transforms, pipelining and multiple clocks, targeting a high-throughput implementation. While this is important in high-speed applications such as 5G (and beyond) and in servers and gateways which handle multiple connections simultaneously, low energy consumption for certain packet sizes is essential for constrained devices. In [Ban+18b], the authors target low-energy implementations of stream ciphers, including Grain-128a, discussing multiple techniques for reducing power consumption. In this paper, we discuss several optimization techniques applied to Grain-128AEAD, targeting both high-speed implementations and low-power implementations. Optimizations are considered in both the RTL and at the synthesis level. A small area does not necessarily mean low energy consumption for encrypting a network packet. Adding some area to Grain will reduce the energy for encrypting a packet, even though the power consumption is slightly higher. Our results can be used to better understand the trade-offs between area, power, energy and throughput for the Grain-128AEAD stream cipher. They also provide new benchmark figures for its hardware performance, allowing better and more transparent comparison with other ciphers supporting AEAD. The code is made available at `https://github.com/Grain-128AEAD`.

The paper is outlined as follows. In Section 2, a high-level overview of the

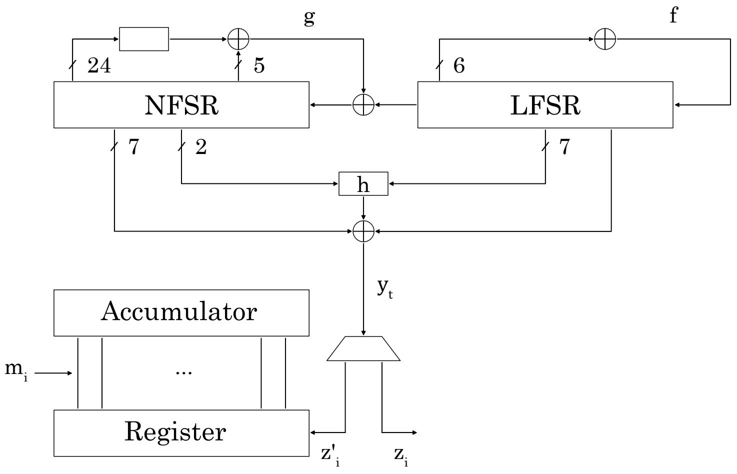


Figure 1: An architectural overview of Grain-128AEAD.

Grain-128AEAD design is presented. Section 3 presents a straightforward implementation providing results from where optimization strategies are derived. In Section 4, the utilized RTL optimizations are discussed, whereas in Section 5, different synthesis level optimizations are introduced. Finally, the results are presented in Section 6 and the conclusions are given in Section 7.

2 Grain-128AEAD

This section will provide a brief overview of the Grain-128AEAD design in order to support the optimization approaches discussed later. For a comprehensive design description, we refer to the specification [Hel+19a; Hel+19b].

Grain-128AEAD is a cipher in the Grain family. It supports Authenticated Encryption with Associated Data (AEAD) to simultaneously assure confidentiality and authenticity of the data. The overall design is similar to the other ciphers in the family, in particular Grain-128a. It consists of two main building blocks. The first is a pre-output generator consisting of a Linear Feedback Shift Register (LFSR) with feedback function f, a Non-linear Feedback Shift Register (NFSR) with feedback function g, and a pre-output function denoted h. The pre-output generator outputs a stream y_t. The second block is the authentication block consisting of a shift register and an accumulator. A multiplexer (MUX) is used to control if the pre-output stream y_t is used for authentication, z'_i, or for keystream, z_i. The architectural overview of Grain-128AEAD is depicted in Fig. 1.

2.1 Phases of Grain-128AEAD

For the hardware implementation, we logically divide the cipher into three phases. The first phase is the *loading phase*, in which the shift registers are loaded with the key and the nonce. Next, Grain-128AEAD enters the *initialization phase* in which the registers and the authentication module are initialized. Finally, the cipher enters the *running phase*, in which pre-output is generated both for encryption and authentication.

2.2 Pre-output Generation

The pre-output generator uses a 128-bit LFSR and a 128-bit NFSR. The content, at instance t, of the LFSR is denoted as $S_t = \left[s_0^t, s_1^t, \ldots, s_{127}^t\right]$, and similarly for the NFSR, $B_t = \left[b_0^t, b_1^t, \ldots, b_{127}^t\right]$. Together, the two FSRs form the 256-bit state of the generator. The feedback polynomial of the LFSR, f, may be written as the recurrence relation given by

$$\begin{aligned} s_{127}^{t+1} &= s_0^t + s_7^t + s_{38}^t + s_{70}^t + s_{81}^t + s_{96}^t \\ &= \mathcal{L}(S_t). \end{aligned}$$

The feedback polynomial of the NFSR, g, may be written as the recurrence relation given by

$$\begin{aligned} b_{127}^{t+1} &= s_0^t + b_0^t + b_{26}^t + b_{56}^t + b_{91}^t + b_{96}^t + b_3^t b_{67}^t + b_{11}^t b_{13}^t \\ &\quad + b_{17}^t b_{18}^t + b_{27}^t b_{59}^t + b_{40}^t b_{48}^t + b_{61}^t b_{65}^t + b_{68}^t b_{84}^t \\ &\quad + b_{22}^t b_{24}^t b_{25}^t + b_{70}^t b_{78}^t b_{82}^t + b_{88}^t b_{92}^t b_{93}^t b_{95}^t \\ &= s_0^t + \mathcal{F}(B_t). \end{aligned}$$

The Boolean function h_t uses bits from both the LFSR and the NFSR, and is defined as

$$h_t = b_{12}^t s_8^t + s_{13}^t s_{20}^t + b_{95}^t s_{42}^t + s_{60}^t s_{79}^t + b_{12}^t b_{95}^t s_{94}^t.$$

The output, y_t, from the pre-output generator is given by

$$y_t = h_t + s_{93}^t + \sum_{j \in \mathcal{A}} b_j^t,$$

where $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$.

After the initialization phase, the pre-output is used to generate keystream bits z_i for encryption and authentication bits z_i' to update the register in the accumulator generator. The keystream is generated as

$$z_i = y_{384+2i},$$

i.e., every even bit (counting from 0) from the pre-output generator is taken as a keystream bit. The authentication bits are generated as

$$z'_i = y_{384+2i+1},$$

i.e., every odd bit from the pre-output generator is taken as an authentication bit.

2.3 Authentication Module

The authenticator generator consists of a 64-bit shift register and a 64-bit accumulator. The content of the shift register, at instance i, is denoted $R_i = \left[r_0^i, r_1^i, \ldots, r_{63}^i\right]$, and similarly for the accumulator, the content is denoted $A_i = \left[a_0^i, a_1^i, \ldots, a_{63}^i\right]$. The accumulator is updated as

$$a_j^{i+1} = a_j^i + m_i r_j^i, \qquad 0 \le j \le 63, \quad 0 \le i \le L, \tag{1}$$

where m_i is the ith message bit, and the shift register is updated as

$$\begin{aligned} r_{63}^{i+1} &= z'_i, \\ r_j^{i+1} &= r_{j+1}^i, \qquad 0 \le j \le 62. \end{aligned}$$

2.4 Loading and Initialization

After reset, the cipher must be loaded and initialized. The loading is performed as follows. Let k_i be the key bits where $0 \le i \le 127$, and let IV_i be the nonce (IV) bits where $0 \le i \le 95$. The NFSR is loaded with the key, i.e., $b_i^0 = k_i,\ 0 \le i \le 127$. The first 96 bits of the LFSR is loaded with the nonce, i.e., $s_i^0 = IV_i,\ 0 \le i \le 95$, and the last 32 bits are filled with 31 ones and a zero, i.e., $s_i^0 = 1,\ 96 \le i \le 126,\ s_{127}^0 = 0$. Next, in the initialization phase, the cipher is clocked 256 times, feeding back the pre-output adding it with the input to the NFSR and LFSR, using the XOR operation, i.e.,

$$\begin{aligned} s_{127}^{t+1} &= \mathcal{L}(S_t) + y_t, \quad 0 \le t \le 255, \\ b_{127}^{t+1} &= s_0^t + \mathcal{F}(B_t) + y_t, \quad 0 \le t \le 255. \end{aligned}$$

Next, the shift register and accumulator in the authenticator are initialized with the pre-output stream as

$$\begin{aligned} a_j^0 &= y_{256+j}, \quad 0 \le j \le 63, \\ r_j^0 &= y_{320+j}, \quad 0 \le j \le 63. \end{aligned}$$

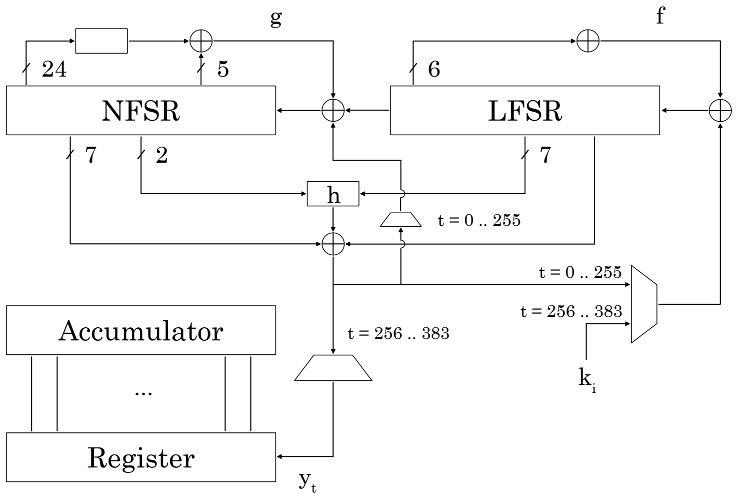


Figure 2: An architectural overview of the initialization in Grain-128AEAD.

At the same time, the key is added to the feedback of the LFSR as

$$s_{127}^{t+1} = \mathcal{L}(S_t) + k_{t-256}, \quad 256 \le t \le 383,$$

while the NFSR is updated as

$$b_{127}^{t+1} = s_0^t + \mathcal{F}(B_t), \quad 256 \le t \le 383.$$

The loading phase and the initialization phase are summarized in Fig. 2.

3 A Straightforward Approach

The stream cipher is implemented in hardware using RTL design in VHDL. For synthesis and power simulation, the Synopsys Design Compiler 2013.12 is used along with a 65 nm library from ST Microelectronics, `stm065v536`. The number of required gates, and the number of transistors in a gate depends on the library used and may vary by a large degree. In this paper, the area of the designs are given in gate equivalents (GE), which is the physical area divided by the area of a 2-input NAND gate for the given library.

A straightforward approach is taken when implementing the cipher, closely following the proposed architectural design in [Hel+19a; Hel+19b] - the FSRs are in Fibonacci configuration parallelized at most 32 times. The key and nonce are simultaneously loaded serially, and the accumulator is loaded by first loading the shift register, then moving the values to the accumulator. For the parallelized implementations, the loading phase is sped up by a factor n, where n is the parallelization level. A simple Finite State Machine (FSM) is used to keep track of the different phases, or states, in order to control the data paths. Finally, we let the synthesizing tool optimize for speed. This implementation and synthesis is used for benchmarking and comparison with our optimized implementations.

In order to improve the bottlenecks in the synthesized design, we must analyze the critical paths. Similar to [MD13], we define the following delays:

- D_n: the maximal delay from any NFSR or LFSR flip-flop to any other NFSR or LFSR flip-flop.
- D_y: the maximal delay from any NFSR or LFSR flip-flop to the output, via the y function.
- D_{ya}: the maximal delay from any NFSR or LFSR flip-flop to any accumulator flip-flip, via the y function.
- D_a: the maximal delay from any flip-flop in the authentication section to any accumulator flip-flop, or output.
- D_{yn}: the maximal delay from a flip-flop of the NFSR or LFSR through the y function to the first flip-flop of the NFSR. This path only exists during initialization of the cipher, via a MUX.

The critical paths are highlighted in Fig. 3. Note that y_{out}, after initialization, corresponds to z as in Fig. 1. Similarly, y_{accum}, after initialization, corresponds to z'.

Table 1: Clock periods and critical paths of the straightforward implementation, for different levels of parallelization.

	x1	x2	x4	x8	x16	x32
Period (ns)	490	610	640	690	770	840
Critical Path	D_{yn}	D_{yn}	D_{yn}	D_{yn}	D_{ya}	D_{ya}

Synthesizing the design yields the results shown in Table 1, where the propagation delay of the critical path is listed. D_{yn} is only available during initialization. In the running state of the cipher, it is instead D_n which is the critical path. These critical delays, together with D_{ya}, will be targeted in the next section.

4 RTL Level Optimizations

Here, we present the architectural optimization techniques utilized when targeting speed, area, and power. In particular, for speed improvements, we aim to lower the delay induced by the critical paths in Table 1. We start by presenting a general optimization technique, Galois transform, to reduce the D_n path. Then, we utilize a similar technique in order to reduce the path D_{yn}, discussed in Section 4.2.

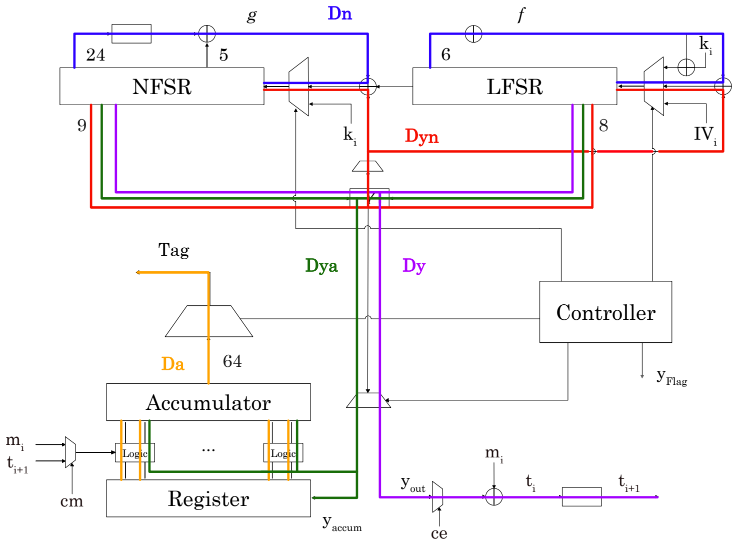


Figure 3: Architectural overview of Grain-128AEAD with the following potential critical paths highlighted: D_n (blue), D_y (purple), D_{ya} (green), D_a (yellow), and D_{yn} (red).

4.1 Galois Transformation

As described earlier, the D_n path lies between two flip-flops (any flip-flop to the right most flip-flop in Fig. 1) in the FSRs, for the 1, 2, 4, and 8 parallelized versions, causing a bottleneck in the running mode. The usual strategy would be to pipeline the FSRs by inserting flip-flops at well chosen positions. In order to pipeline a design, the delay elements can only be inserted in the feed-forward cutset of the corresponding graph [PM06]. This is not possible for the FSRs due to their intrinsic feedback property. In order to decrease the propagation delay in the D_n path, the cipher may be transformed from its normal Fibonacci configuration to a Galois configuration. In the Fibonacci configuration, the flip-flops are updated with the value of the previous flip-flop every clock cycle, i.e., $x_i = x_{i+1}$, except for x_{n-1} which gets updated with the result of the feedback polynomial, i.e., $x_{n-1} = f(x)$. In the Galois configuration, some of the flip-flops get updated with the result of a function of other flip-flops, i.e., $x_i = g(x)$. The Galois configuration leads to shorter propagation delays due to the feedback function in Fibonacci being split up and put in between flip-flops.

For an LFSR, the Fibonacci to Galois transform is a one-to-one mapping. For an NFSR, multiple Galois configurations exist for a given Fibonacci configuration [Dub09]. The Galois transform is identical to Grain-128a, hence we refer to [MD13] for details. Grain-128a can not be transformed to a Galois configuration for a parallelization level above 16. The same holds for Grain-128AEAD.

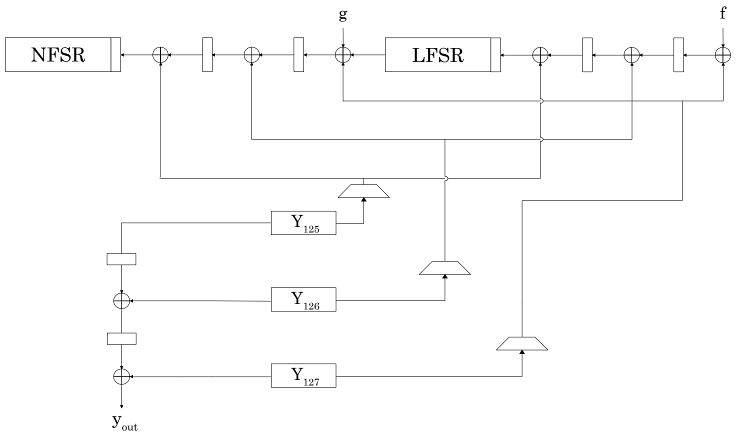


Figure 4: The transformation of the y function along with pipeline steps and control logic.

4.2 Transforming the y Function

During the initialization phase, the y function is being fed back to the shift registers, forming an FSR. As in the previous section, it is not possible to insert pipelines due to the lack of a feed-forward cutset. Instead, similar to the Galois transform, it is possible to transform the y function in such way that it is split up and fed back to different registers, which reduces the critical path, D_{yn}. The transformed functions are denoted as $Y_{125}, Y_{126},$ and Y_{127}. For a parallelization level of 8 and 16, only Y_{126} and Y_{127} may be used. As an example for the non-parallelized version, the functions are given by

$$\begin{aligned}
Y_{127} &= b_{12}s_8 + s_{13}s_{20} + b_{95}s_{42}, \\
Y_{126} &= b_{11}b_{94}s_{93} + b_{72} + b_1 + s_{50}s_{78}, \\
Y_{125} &= s_{91} + b_{87} + b_{13} + b_{34} + b_{43} + b_{62}.
\end{aligned}$$

After initialization, during the running state, the feedback loop is disconnected. This allows for insertion of pipelines steps. By combining both Galois-like transformation and pipelining, both D_{yn} and D_y may be reduced, see Fig. 4. The controller switches between the two methods when required.

4.3 Isolating the Authentication Module

The accumulator is updated using the values in the shift register as in Eq. (1). When parallelizing the design, the accumulator is updated with the corresponding shift register plus values from the shift registers with larger index as

$$a_j^{i+1} = a_j^i + \sum_{k=0}^{\frac{p}{2}-1} m_{i+k} \cdot r_{j+k}^i, \qquad p \geq 4,$$

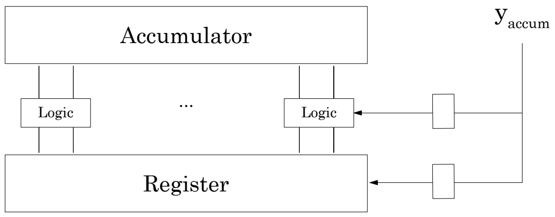


Figure 5: Isolating the authentication module using pipelining.

where p is the parallelization level. For $p = 2$, the update expression is equivalent to Eq. 1, since 1 bit is generated every clock cycle for authentication. Note that for a parallelization level of 4 and above, some values have not yet been shifted in to the shift register, e.g., for $p = 4$, the register a_{63} is updated as

$$a_{63}^{i+1} = a_{63}^{i} + \left(m_i \cdot r_{63}^{i}\right) + \left(m_{i+1} \cdot r_{64}^{i}\right),$$

where r_{64} has been generated but is not located in the shift register. This can be seen as a future value, and requires extra combinational logic to handle. This means that the path D_{ya} becomes longer and, for the higher levels of parallelization, affects the timing of the design, as seen in Table 1.

In order to make the D_{ya} path shorter, a pipeline step is inserted between the y function and the accumulator logic, as shown in Fig. 5. This allows for the throughput to increase due to a higher clock frequency, but adds a 1 clock cycle delay to the accumulator calculation. Note that this does not affect the security.

4.4 Optimizing the Controller

A controller is a unit responsible for managing the data flow and operations, such as the feedback loop, when to accumulate data, and when to encrypt the plaintext.

The straightforward implementation of the controller is a finite state machine (FSM) with states corresponding to loading, initialization, and the running phase. The controller can also be implemented using a LFSR with some combinational logic. Experiments with LFSRs gave roughly the same results as the FSM generated by the synthesizer. Here, we explore an alternative strategy to keep track of the state by using a clock divider and a shift register, which often, but not always, gave better performance.

The idea is to start from an empty shift register (all zeroes), and shifting in a 1 each clock cycle. This means that after n clock cycles, there is a 1 at index n in the shift register. We can use this to control the logic in the cipher via, e.g., MUXes. However, Grain requires 512 clock cycles before it produces the first bit, i.e., 128 for loading key and nonce plus 384 for the initialization rounds. This results in a shift register with 512 flip-flops, which is not desirable due to the huge increase in size. Instead, note that the resolution given by 512 registers is not fully utilized, since we ignore most of the intermediate values, hence we can reduce the size. With a reduced size, we must compensate with a lower clock frequency for controlling the design at the correct time instances. This is done using a clock divider to slow the shift register down by a factor 2^k. The value of $2^k p$ can not exceed 128, since the least amount of clock cycles that we need to keep track of are 128, for loading key and nonce. The number of registers required depends on the level of parallelization, p, and the k value as $512/(2^k p)$. Taking the clock divider registers into account gives an expression for the total number of registers required as

$$\frac{512}{2^k p} + k.$$

This design does not require much hardware, e.g., letting $p = 16, k = 3$ results in 7 flip-flops. In this paper, the largest value of k is selected, for every level of parallelization, i.e., $k = \log_2(128/p)$.

The index of the shift register corresponding to the different phases are calculated as

$$i_{\text{load}} = \frac{128}{kp}, \quad i_{\text{init}} = \frac{128+256}{kp}, \quad i_{\text{run}} = \frac{512}{kp}.$$

For the control MUXes that remain in their state after being activated may be directly connected to the controller. For the control MUXes that are only activated during a single state, an inverter together with an AND-gate is required.

4.5 Unrolling

Grain natively supports parallelization up to 32 times by using multiple feedback and output functions, f, g and y. However, there is nothing preventing us to go further, at RTL level. Consider AES, where multiple rounds of the AES round function is executed one after the other, in a loop structure. The block is fed back via MUXes to realize successive iterations. Unrolling the AES rounds to a level L means that we put L AES round functions serially in a single combinational block as done in [ZNC04].

When unrolling Grain, we do not utilize multiple instances of the FSRs, but rather the feedback functions. When reaching a parallelization level above 32, the feedback functions start to interact. Revising the feedback function f of the LFSR, we can extend this to include multiple copies of f, denoted as f_k. The index k is related to the level of parallelization, where the highest k value plus 1 ($\max(k)+1$) equals the parallelization level. Expressing the bits as a sequence, we can write the expression as

$$s_{i+128+k} = s_{i+0+k} + s_{i+7+k} + s_{i+38+k} + s_{i+70+k} + s_{i+81+k} + s_{i+96+k}.$$

With $k = 31$, the bit with the highest index in f_{31} is $s_{i+96+31} = s_{i+127}$, by design. However, $k = 32$ includes bit index s_{i+128} in f_{32}, which is not a reg-

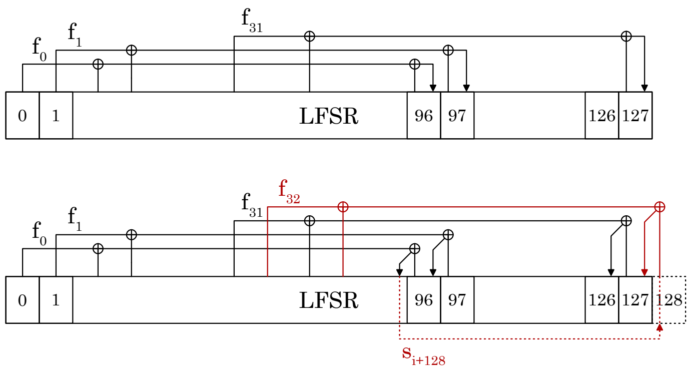


Figure 6: Example of unrolling above the specified level of 32. The bottom picture shows the structure when using a parallelization level of 33. The last feedback function, f_{32}, needs to read the value from a flip flop that does not exist, the register index 128. Instead, this value is the output from f_0, which would have been stored in register index 128 if it existed. We can therefore take the output from f_0 as an input to f_{32}. From this it is clear that the propagation delay increases when exceeding the specified level of parallelization.

ister index, but rather the output from the function f_0. Thus, the two feedback functions are connected, as seen in Fig. 6. Increasing k leads to more interconnection between the feedback functions, thus increasing the propagation delay. Parallelization of higher degrees for the authentication module continues the approach described in Section 4.3.

Apart from an increase in throughput, unrolling also allows for energy saving as more data is processed in a single clock cycle which reduces the total switching activity and the number of clock cycles it takes to complete a computation [BBR16].

5 Synthesis Level Optimization

Synthesis is the process where high-level RTL code, like VHDL and Verilog, is used to generate a gate-level netlist. There are 3 steps involved during synthesis:

1. Translation: The RTL code is converted to a technology-independent representation of Boolean expressions.

2. Optimization: The Boolean expressions are minimized, with respect to gates, using a minimization algorithm.

3. Technology mapping: The Boolean expressions are mapped to a library, based on the used technology, in order to produce a gate-level netlist.

Design Vision requires the RTL code, design constraints, and a standard cell library, in order to generate a netlist. Design Vision offers two commands used for compiling - `compile` and `compile_ultra`. The `compile_ultra` command is used for designs with tight timing constraints, and produces better quality of results compared to `compile`. Hence, only `compile_ultra` is used during synthesis.

There are several compiler options to be utilized during synthesis. Here, we highlight some of the most commonly used features:

- **Structuring** - The process where intermediate variables are added to the design, in order to reduce area. The synthesis tool factors out common sub functions that mostly reduces the area and turn them into intermediate variables.
- **Flattening** - Here, the tool converts combinational logic paths into a sum-of-products representation. This often leads to a faster design due to the combinational logic requiring only two levels. Consequently, it may lead to an increase in area.
- **Ungrouping** - A common strategy when implementing a design is to group different parts of the code, to have a hierarchical design. This leads to well structured design and it is easy to analyze the synthesized design. By ungrouping, the tool is less constrained and may reorganize the design as it see fits, which may lead to a faster design, at the expense of area.
- **Clock Gating** - Insert control logic in order to regulate the clock signal, either to shut it down at time instances, or to modify the clock pulse. This may be used to save energy.

5.1 Transistor Types

In a complementary MOS (CMOS) design, both NMOS and PMOS are used. When one is conducting, the other is not, resulting in very small static power consumption, given by

$$P_s = V_{dd} \cdot I_{\text{leakage}},$$

where V_{dd} is the supply voltage. The leakage current depends on the threshold voltage, V_{th}, and

a transistor with low V_{th}, LVT, has higher leakage current than a transistor with a high V_{th}, HVT. To minimize leakage current, HVT transistors are most suitable for power efficient implementations. For the high-speed implementations, LVT transistors are most suitable since they allow to increase the switching speed.

6 Synthesis Results

Providing results for all possible combinations of implementations, synthesis options and transistors would become very verbose. Instead, to facilitate a more clear and concise presentation and basis for comparison, the implementations considered will be as follows.

- **Straightforward implementation**. This implementation will closely follows the architectural design, using no optimization techniques. The design is synthesized for high speed utilizing LVT transistors, and different synthesis flags to achieve the best result.
- **High speed implementation**. Here, we apply all viable optimizations at RTL and synthesis level and synthesize for high speed using LVT transistors.
- **Low Power implementation**. In the low power scenario, we cut back the clock frequency, employing only unrolling and the improved controller as optimization techniques. Both the LVT and HVT transistors are used for comparison of power consumption.

6.1 Straightforward Implementation

Results for the straightforward implementation are shown in Table 2, using no RTL optimizations, but synthesized for maximum speed. Synthesis options such as flattening, structuring, and ungrouping were utilized. Neither flattening nor structuring affected the result significantly. Only the grouping/ungrouping option made a difference. This difference was typically in the order of 0.02 ns for the period. In the result tables, the best result is presented, and we also highlight whether grouping (G) or ungrouping (U) yielded the result.

Similar to [Ban+18b], we also calculate the energy consumed when encrypting 1 block of data (64 bits) and 1000 blocks, shown in Table 3. For example, encrypting 1 block of data, in the non-parallelized ($n = 1$) version at 2.04 GHz, requires 128 (loading key and IV) + 384 (initialization) + 128 (64 keystream bits + 64 bits for authentication) = 640 clock cycles. The energy consumed results in $640 \times 0.49\,\text{ns} \times 170\,\mu\text{W} = 0.053\,\text{nJ}$. Note that the number of clock cycles required for encryption is inversely proportional to n.

The non-parallelized version has the highest clock frequency, but the lowest throughput (thrp). The clock period does not scale at the same rate as n, which allows the higher levels of parallelization to have higher throughput. Between $n = 1$ and $n = 32$, the throughput increases by a factor 19, whereas the area only increases by a factor 3.8. For $n = 32$, we achieve the highest throughput to area ratio. For $n = 4$, the highest throughput to power is reached along with the lowest energy consumption, making it the most power efficient version.

The ungrouping option seems to be worse for all versions except $n = 1$. Using the ungrouping option led to a higher clock frequency, but the tool reported fanout violations which it could not resolve. Choosing to only consider results without any violations, these results were omitted.

Table 2: Straightforward implementation synthesized for high speed. The throughput per area is given in kbit/s per GE. The throughput per power is given in Gb/s per mW. The synthesis optimization (Opt.) shows whether grouping (G) or ungrouping (U) gave the best result.

n	Period	Freq.	Thrp.	Area	Power	Thrp. / Area	Thrp. / Power	Opt.
	(ns)	(GHz)	(Gb/s)	(GE)	(mW)			
1	0.49	2.04	1.02	2689	0.17	182	5.99	U
2	0.61	1.64	1.64	2776	0.14	284	11.76	G
4	0.64	1.56	3.12	3333	0.21	450	14.93	G
8	0.69	1.44	5.76	4324	0.42	640	13.70	G
16	0.77	1.29	10.32	6265	0.92	792	11.24	G
32	0.84	1.19	19.04	10226	2.54	895	7.52	G

Table 3: This shows the energy consumption for the straightforward implementation, processing 1 and 1000 blocks of data. 1 block equals 64 bits of data.

Energy (nJ)	x1	x2	x4	x8	x16	x32
1 Block	0.053	0.027	0.022	0.023	0.028	0.042
1000 Blocks	10.70	5.48	4.31	4.65	5.69	8.57

6.2 High Speed Implementation

Here, we apply the techniques described earlier in order to increase the throughput of the design. For the parallelization levels 1, 2, 4, 8 and 16, Galois transform together with y transform, isolation of authentication module, and the optimized controller are utilized. For the 32 (parallelized) and 64 (unrolled) versions, only isolation of authentication and the optimized controller are possible. Transformation of the y function is not applicable due to similar constraints as for the Galois transformation of the shift registers.

The results for the optimized implementation are presented in Table 4. The energy consumption for a given message length is given in Table 5, where the highest speed at each level of parallelization from Table 4 is used. Table 4 shows an increase in throughput for every level of parallelization, at the expense of increased power consumption. However what is interesting is that the optimized controller actually reduces the power consumption while increasing the throughput for $n = 32$ and $n = 64$. For $n = 32$, the power consumption is lower than the straightforward implementation, while for $n = 64$, it is just 0.22 mW more than the straightforward, 32 parallelized version, but with a 76% increase in throughput. We can again note that a parallelization level of 4 yields the highest throughput

Table 4: Results for the high-speed implementation, with optimized controller on greyed background and regular controller on white. 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	Opt.
	(ns)	(GHz)	(Gb/s)	(GE)	(mW)			
1	0.43	2.3	1.15	2791	0.24	412	4.79	U
	0.40	2.5	1.25	2645	0.25	472	5.00	U
2	0.46	2.17	2.17	2800	0.21	776	10.33	G
	0.43	2.32	2.32	2695	0.23	861	10.09	G
4	0.47	2.13	4.26	3335	0.29	1277	14.69	G
	0.48	2.08	4.16	3199	0.29	1300	14.34	U
8	0.48	2.08	8.32	4537	0.67	1834	12.42	G
	0.46	2.17	8.68	4448	0.67	1951	12.96	G
16	0.50	2.00	16.00	6270	1.44	2552	11.11	G
	0.48	2.08	16.64	7118	1.55	2338	10.74	U
32	0.69	1.45	23.20	9148	2.66	2536	8.72	G
	0.64	1.56	24.96	9206	1.78	2710	14.02	U
64	1.00	1.00	32.00	16618	4.76	1926	6.72	G
	0.95	1.05	33.60	16958	2.76	1982	12.17	U

Table 5: This shows the energy consumption for the high-speed implementation, processing 1 and 1000 blocks of data. 1 block equals 64 bits of data.

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

per power along with the lowest energy consumption. The optimized controller also affects the throughput per power the most for $n = 32$ and $n = 64$.

As also seen in Table 4, ungrouping the design led to a higher throughput when using the optimized controller.

It is clear that the area increases with higher throughput, due to higher levels of parallelization. An important metric is the throughput per area, which measures area efficiency. From the table, we find that the most area efficient implementation occurs when $n = 32$, using the improved controller. This is not surprising since increasing parallelization should only require a "small" increase in area, by design. This is an important feature in the Grain family of stream ciphers.

6.3 Low Power Implementation

When targeting low power, a clock period must be specified. Many low-power devices run at frequencies around 10 MHz. The ISO standard for contactless smart cards, ISO/IEC 15693, defines the frequency to be 13.56 MHz. Older proximity cards operate at 125 kHz. Thus, for low power applications, we choose to synthesize the design at the clock frequencies 100 kHz and 10 MHz, shown in Table 6 and 7, respectively.

The synthesis script utilizes `compile_ultra` with clock gating and low power transistors (HVT). For comparison, we also synthesize the design using the high speed scripts and select the best result, for comparison. The RTL optimization implemented for low power is unrolling along with the optimized controller.

Table 6: Result for the low power implementation running at 100 kHz. Here, we compare the speed script (S_s), the power (P_s) script, and the power script using the optimized controller (P_{opt}). 1 block equals 64 bits of data.

n	**Area** (GE)			**Power** (µW)			**Energy** (nJ)	
	S_s	P_s	P_{opt}	S_s	P_s	P_{opt}	1 block	1k blocks
1	2509	2375	2337	2.29	0.23	0.26	1.47	296
	-	-5%	-7%	-	-89%	-88%	-	-
2	2592	2588	2511	2.33	0.28	0.30	0.90	180
	-	0%	-3%	-	-87%	-86%	-	-
4	2952	2950	2862	2.33	0.29	0.32	0.46	93.2
	-	0%	-3%	-	-87%	-86%	-	-
8	3695	3692	3594	2.76	0.31	0.35	0.25	49.8
	-	0%	-2%	-	-88%	-87%	-	-
16	5168	5158	5053	3.77	0.42	0.39	0.16	31.3
	-	0%	-2%	-	-89%	-90%	-	-
32	8168	8126	7950	5.93	0.62	0.46	0.09	18.5
	-	0%	-3%	-	-90%	-92%	-	-
64	14100	14093	13800	10.89	1.08	0.63	0.06	12.7
	-	0%	-2%	-	-90%	-94%	-	-

Overall, there was very little difference in area when synthesizing for high speed and low power using the standard controller. For such low frequencies, the timing is easily met and the tool optimizes for area in both cases, thus there is not much to improve. For the power however, there is a clear difference using HVT transistors compared to LVT. There is a 86 - 92% reduction in power consumption for all levels of parallelization running at 100 kHz, and a 19 - 37% power reduction for 10 MHz. In the design paper of Grain [Hel+19a; Hel+19b], the authors used HVT transistors when synthesizing for high speed. This led to a lower clock frequency and a higher power consumption than the figures in Table 4. Hence,

Table 7: Result for the low power implementation running at 10 MHz. Here, we compare the speed script (S_s), the power (P_s) script, and the power script using the optimized controller (P_{opt}). 1 block equals 64 bits of data.

n	**Area** (µm^2)			**Power** (µW)			**Energy** (nJ)	
	S_s	P_s	P_{opt}	S_s	P_s	P_{opt}	1 block	1k blocks
1	2510	2375	2337	33.66	22.07	25.21	1.41	283
	-	-5%	-6%	-	-34%	-25%	-	-
2	2592	2589	2511	33.96	26.93	29.13	0.86	173
	-	0%	-3%	-	-21%	-14%	-	-
4	2952	2951	2862	34.43	27.38	31.05	0.44	88.0
	-	0%	-3%	-	-20%	-10%	-	-
8	3695	3693	3595	36.83	29.38	33.59	0.24	47.2
	-	0%	-3%	-	-20%	-9%	-	-
16	5168	5162	5057	44.02	39.49	36.93	0.15	29.7
	-	0%	-2%	-	-10%	-16%	-	-
32	8172	8128	7951	66.02	57.08	41.66	0.08	16.7
	-	0%	-2%	-	-13%	-37%	-	-
64	14101	14093	13810	117.4	97.39	55.93	0.06	11.2
	-	0%	-2%	-	-17%	-52%	-	-

HVT should only be used for lower frequencies where power is the main concern, whereas LVT should be used for higher frequencies where the target is speed.

Even though the power consumption increases with increasing n, the energy cost decreases since the computation can be done in much shorter time. This leads to the unrolled 64-parallelized version being the most energy efficient implementation for a given message length.

The optimized controller reduces the area in all cases at expense of higher power consumption for $n = 1, 2, 4, 8$. For $n = 16, 32, 64$, the power consumption is reduced when using the optimized controller.

7 Conclusions

In this paper, we implemented Grain-128AEAD and investigated the impact of different implementation strategies, from RTL to synthesis-level design, to either achieve high throughput or low power consumption.

By utilizing different optimization techniques, we reduced the power by up to 94% compared to a straightforward implementation. By unrolling the design, the power consumption increases while the energy for encrypting a message of fixed size decreases. The 64-level parallelization implementation requires only 11.2 nJ when encrypting 64 kbits of data compared to 283 nJ for the non-parallelized version. For the high-speed implementation, the maximum throughput reached is 33.6 Gb/s. It is not obvious in which cases the (un)grouping option yields the best result, hence both options should be analyzed in order to find the best result. We notice that a parallelization level of 4 yields the most power efficient implementation, both for the straightforward implementation and the high-speed one. The experiments show that Grain is well suited both in high-speed applications as well as on constrained devices requiring low power consumption.

Acknowledgements

This paper was supported by the Swedish Foundation for Strategic Research, grant RIT17-0032.

References

[Ågr+11a] M. Ågren et al. "Grain-128 a: a new version of Grain-128 with optional authentication". In: *International Journal of Wireless and Mobile Computing* 5.1 (2011), pp. 48–59.

[Ban+18b] S. Banik et al. "Towards Low Energy Stream Ciphers". In: *IACR Transactions on Symmetric Cryptology* 2018.2 (June 2018), pp. 1–19.

[BBR16] S. Banik, A. Bogdanov, and F. Regazzoni. "Exploring Energy Efficiency of Lightweight Block Ciphers". In: *Selected Areas in Cryptography – SAC 2015*. Ed. by O. Dunkelman and L. Keliher. Cham: Springer International Publishing, 2016, pp. 178–194.

[DS11b] I. Dinur and A. Shamir. "Breaking Grain-128 with Dynamic Cube Attacks". In: *Fast Software Encryption*. Ed. by A. Joux. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 167–187.

[Dub09] E. Dubrova. "A Transformation From the Fibonacci to the Galois NLFSRs". In: *IEEE Transactions on Information Theory* 55.11 (Nov. 2009), pp. 5263–5271.

[Hel+06a] M. Hell et al. "A Stream Cipher Proposal: Grain-128". In: *2006 IEEE International Symposium on Information Theory*. July 2006, pp. 1614–1618.

[Hel+19a] M. Hell et al. "An AEAD Variant of the Grain Stream Cipher". In: *Codes, Cryptology and Information Security*. Ed. by C. Carlet et al. Cham: Springer International Publishing, 2019, pp. 55–71.

[Hel+19b] M. Hell et al. *Grain-128AEAD - A lightweight AEAD streamcipher*. NIST Lightweight Cryptography, Round 1 Submission. 2019.

[Int15b] International Organization for Standardization. *ISO/IEC 29167-13:2015 Information technology — Automatic identification and data capture techniques — Part 13: Crypto suite Grain-128A security services for air interface communications*. 2015.

[MD13] S. S. Mansouri and E. Dubrova. “An Improved Hardware Implementation of the Grain-128a Stream Cipher”. In: *Information Security and Cryptology – ICISC 2012*. Ed. by T. Kwon, M.-K. Lee, and D. Kwon. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 278–292.

[Nat18] National Institute of Standards and Technology. *Proposed submission requirements and evaluation criteria for the post-quantum cryptography standardization process*. 2018. URL: `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf` (visited on 07/02/2019).

[PM06] J. G. Proakis and D. K. Manolakis. *Digital Signal Processing (4th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[ZNC04] J. Zambreno, D. Nguyen, and A. Choudhary. “Exploring Area/Delay Tradeoffs in an AES FPGA Implementation”. In: *Field Programmable Logic and Application*. Ed. by J. Becker, M. Platzner, and S. Vernalde. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 575–585.

Energy Consumption for Securing Lightweight IoT Protocols

Abstract

In this paper we address the energy consumption of CoAP and MQTT and compare their overhead. We also pay attention to the use case of security in IoT and analyze the energy consumption when using TLS/DTLS for the two protocols. In our experiments we use ESP32 with libcoap, MQTT, and mbed TLS libraries and conduct real-world measurements using Otii, a high precision voltage and current measurement tool. While the particular numbers are implementations and hardware dependent, we can still make interesting observations. For data transfer, we find that aggregating data to larger packets can significantly reduce the energy consumption. We also find that AES-CCM8 seems slightly more efficient than other modes of operation. In comparison, the DTLS handshake for setting up the secure connection is very expensive, and also very dependent on security level and algorithm choices. For firmware updates, AES-CCM8 is again slightly better than the alternatives, but the differences between CoAP and MQTT are much more significant, favoring MQTT due to the use of the retransmission support in TCP. This is also evident in lossy networks, where MQTT saves up to 91% energy compared to CoAP at 20% loss rate. Finally, we find that energy consumption in CoAP can to some extent be reduced in lossy networks by modifying the retransmission timeout.

Pegah Nikbakht Bideh, Jonathan Sönnerup, and Martin Hell. "Energy Consumption for Securing Lightweight IoT Protocols". This paper has been submitted but not yet published.

1 Introduction

Devices connected to the Internet of Things (IoT) are seen as key enablers for e.g., the smart city, home automation, wearables, and asset tracking. Connected devices will supposedly also improve or even revolutionize, among other things, energy management, healthcare and the management of our infrastructure. The devices will be realized as e.g., sensors for detecting and monitoring physical characteristics of the environment, or actuators to control our environment, machines, systems or processes. Their interconnection with other devices, gateways and/or the cloud introduces new security challenges, but it also makes them more exposed to attacks, targeting e.g., unpatched vulnerabilities. The most common application level communication protocols for IoT are the Constraint Application Protocol (CoAP) and the Message Queue Telemetry Transport (MQTT). CoAP is a lightweight protocol, in many aspects similar to HTTP, while MQTT is a publish/subscribe protocol. Both protocols are widely supported and have gained widespread adoption, but neither include security functionality. Still, support for confidentiality and integrity of messages, as well as message authentication, is often needed, and the natural choice is then to use DTLS for CoAP and TLS for MQTT.

The ubiquitous nature of IoT devices often requires them to run on batteries, making energy efficiency a primary concern. The large number of devices make it costly to replace batteries, and it will also make the total energy consumption considerable, further motivating energy efficient data communication and processing. At the same time, adding security to the communication will add additional overhead. Thus, it is important to not only develop lightweight security protocols, but also to understand to which extent security affects the energy consumption of the devices. Such understanding will allow vendors and users to make informed decisions when choosing and implementing security in the devices and systems.

The main contribution of this paper is a thorough analysis of CoAP and MQTT and the investigation of their energy footprint in different scenarios, and how added security at the transport layer (TLS/DTLS) affects the energy consumption. Important design choices, such as cipher suite, PKI vs. PSK, and client authentication are also analyzed in order to better understand how such choices impact the energy consumption. In our real-world experiments, we use ESP32 to represent a device. For measurements, we use an Otii, which is a high precision voltage and current measurement unit. Note that the actual numbers given in this paper are implementation and hardware specific and might differ between libraries or IoT devices. Still, the main takeaways will apply to the general case.

The paper is organized as follows. In Section 2, we first discuss related work. In Section 3, we provide some background on the CoAP, MQTT, TLS and DTLS protocols. Our experimental setup including the hardware and software used are discussed in Section 4. Then, the methodology and the motivation for our different measurements are given in Section 5. The results from the measurements are given and discussed in Section 6 and the paper is concluded in Section 7.

2 Related Work

Efficiency and comparisons of IoT protocols have been considered in several previous works. An important thing to note is that the library or implementation used can heavily impact performance [IOU17]. Optimized implementations can be used to lower energy consumption. This was pursued for DTLS in [Cap+15], where the authors exploited ECC optimizations in order to minimize ROM occupancy, time and energy. They only considered ECC based operations with two different cipher suites, one with ECDH and ECDSA, and one with ECDHE and PSK. The importance of optimized implementations was also noted in [Suá+18] where it was shown that secp256r1 was more efficient than the secp224r1 curve due to a more optimized implementation. The authors compared ECDSA and RSA in TLS 1.2 and on ESP32, for different security levels. Their results showed that ECDSA performs better than RSA. This work was extended in [SFF18], where a complete fog and mist computing architecture and testbed was presented and evaluated.

The performance of security updates for IoT were measured and discussed in [TBK19]. Three different models, CoAP, MQTT, and encapsulating CoAP inside MQTT, was proposed for delivering Over-the-Air updates and software patches. While it was shown that MQTT is faster and more reliable than CoAP to send urgent updates, no energy measurements were made and cryptographic protection was not considered. Since power consumption differs over time, it is not possible to draw accurate conclusions for energy consumption by only measuring time. In this paper, we consider both energy and time for the software update case, while also comparing both CoAP and MQTT using different encryption algorithms.

In [Tha+14], the authors designed a common middleware for MQTT and CoAP and measured performance of these based on end-to-end delay of single packets and bandwidth consumption. Their results indicated CoAP has lower average delay in case of high packet loss, around 25%. No energy measurements were made. In this paper, we measure energy and looks at a sequence of packets instead of average time for individual packets.

Energy can also be reduced by instead making changes to the protocol. Lithe (Lightweight Secure CoAP for the Internet of Things) [Raz+13] is an integration of DTLS and CoAP for IoT, in which the authors proposed a header compression scheme which aims to reduce energy consumption by leveraging 6LoWPAN. In their evaluation, they demonstrated that CoAP overhead by using DTLS compression can be significantly reduced in terms of energy consumption and network response time. While such modifications are valuable, they require protocol modifications. In this paper we do not aim to make modifications to the protocols.

3 CoAP and MQTT

3.1 CoAP

CoAP [GMS15] is a web transfer protocol for constrained networks and was developed by IETF CoRE (Constrained RESTful Environments) Working Group. CoAP was designed for UDP communication over 6LoWPAN networks. It uses a Universal Resource Identifier (URI) to identify available resources on constrained devices. Messages are exchanged between endpoints using CoAP requests and responses.

A CoAP message has a 4-byte fixed header, consisting of 2 bits for Version field, 2 bits for message Type, 4 bits of Token Length, 8 bits of Code field and 16 bits of Message ID. The Message ID is utilized for duplicate detection. A token can optionally be used to match requests and responses. In our experiments, we do not include Token or Options, so the Token Length is zero.

The CoAP specification describes four security modes:

- **Nosec**: No security provided.
- **PreSharedKey**: Symmetric pre-shared keys are used in one of the PSK cipher suites in DTLS.
- **RawPublicKey**: DTLS is used with an asymmetric key pair that is preinstalled on the device, without a corresponding X.509 certificate.
- **Certificates**: x.509 certificates are used with DTLS.

For CoAP, we may choose to send either confirmable (CON) messages or nonconfirmable (NON). CON messages will be ACKed, or retransmitted in case of packet loss, whereas NON messages will not.

3.2 MQTT

The MQTT protocol is a publish/subscribe messaging protocol designed for low bandwidth environments, originally for communication over satellite links. It uses a server (broker), together with a set of clients. A client sends messages tagged with a topic, and other clients can subscribe to that topic, in which case the server routes the messages to the subscribing clients. In MQTT, the connections to the server can be specified with a Quality of Service (QoS). QoS can vary from 0 to 2 in which 0 has the least overhead. In our experiments, we used the default QoS, which is 0.

MQTT runs over TCP protocol. There is also a variant called MQTT-SN which can use UDP. It has however not received widespread adoption and is not considered in this paper.

3.3 Security on the Transport Layer

Both CoAP and MQTT can have security added at the application layer. OSCORE (Object Security for Constrained RESTful Environments) [Sel+19], is an example of object security protocol for CoAP. Still, the two protocols are commonly used with DTLS and TLS and both specifications explicitly discuss the use of these protocols. TLS and DTLS differ primarily by the fact that DTLS, being used for UDP, must e.g., handle packet loss and out-of-order packets in the handshake phase.

The two main protocols in TLS are the handshake protocol and the record protocol. In the handshake protocol, the peers are authenticated and encryption and message authentication keys are established. A Diffie-Hellman key exchange, or a pre-shared key, is used to agree on a premaster secret, from which the encryption keys are derived. With Diffie-Hellman, the messages are authenticated using a digital signature. It is also possible to combine Diffie-Hellman and PSK, in which case the two values are concatenated to form the premaster secret. For improved performance, Diffie-Hellman can be computed over elliptic curves (denoted ECDHE in the cipher suites). Also the digital signature can be computed over an elliptic curve (ECDSA), instead of using an RSA signature. Once keys have been established in the handshake protocol, data can be encrypted and authenticated in the record protocol.

The record layer in DTLS is very similar to TLS, but an explicit epoch (2 bytes) and sequence number (6 bytes) are added to the record. This results in DTLS 1.2 messages having 29 bytes overhead while TLS 1.2 has only 21 bytes overhead.

Other than normal message overheads, DTLS 1.2 handshake messages have 7 extra bytes header overhead. This overhead in the DTLS 1.2 handshake, in comparison to TLS 1.2, is due to the fact that DTLS handshake messages can be fragmented over several DTLS records.

4 Experimental setup

Here we present the architecture used during the experiments, i.e., the hardware and software components.

4.1 Selected Hardware

To select the development board, based on our requirements including low price, good documentation, support for CoAP/MQTT and DTLS/TLS and widespread use, we decided to use ESP32-WROOM-32D [Esp19]. It is relatively cheap, has support for the software libraries we target, and has good documentation and a large community. It can also be used in many applications ranging from low-power sensor networks to very demanding tasks. Moreover, AES hardware acceleration is supported (and was enabled). It has an Xtensa 32-bit dual-core microprocessor with 240 MHz clock speed and 512 KiB of RAM.

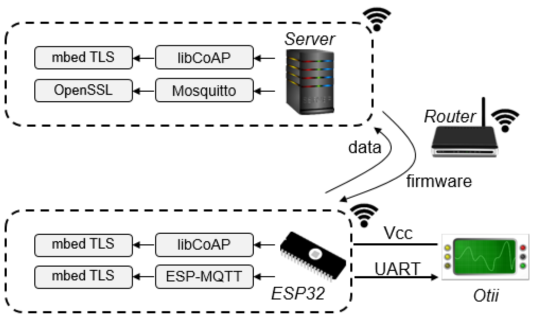


Figure 1: CoAP/MQTT testbed architectural overview

As WiFi router, an Asus RT-ACS1U was used. The CoAP server and the MQTT broker, were run on a 64-bit Linux system with an Intel Core i5-6200U at 2.4 GHz with 8 GB RAM.

To measure energy consumption, an Otii Arc was used. Otii is a portable power supply and data-acquisition module which can be used for very accurate voltage measuring. It is commonly used by developers in device and application designs to optimize energy consumption. Otii Arc has a desktop application available for Windows, Ubuntu and macOS. We used the application on Ubuntu.

4.2 Software

The official development framework for ESP32-IDF v3.3 [Esp20], denoted ESP-IDF, was used for development. The CoAP client was developed with libcoap 4.2.1, and the mbed TLS 2.16.2 library was used to setup the DTLS connection.

The libcoap 4.2.1 library was also used to setup the CoAP server on the Linux machine. For transport layer security, libcoap has support for GnuTLS, OpenSSL and Tinydtls, and can also be configured with mbed TLS. In our testbed libcoap with mbed TLS 2.8 was used.

The MQTT library used on the client side is the one included in the ESP-IDF, running MQTT version 3.1.1. The library utilized mbed TLS for TLS 1.2, supporting both PKI-based and PSK-based cipher suites. On the server side, the Mosquitto broker 1.6.7 is utilized using OpenSSL 1.1.0l.

To do the measurements with the Otii software, the Otii device is connected to a power supply via a USB port. Then, the Otii device powers the ESP32, allowing it to measure the power used by the ESP32. UART messages are sent from the ESP32 to the Otii in order to annotate the measurements. This allows us to accurately correlate the energy consumption to the different phases of the application under test.

The selected components are connected as shown in Fig. 1.

5 Methodology and Use Case

In this section, we discuss, and give the rationale behind, our chosen measurements. In the following, the cost will refer to the energy cost of first computing, and then sending information over the communication channel.

A typical use case would be that of a sensor communicating data back to a server for aggregation and further analysis. The sensors are developed using a mix of third party and in-house developed code. They will thus be subject to discovered security vulnerabilities, requiring new firmware with regular intervals. The sensors will typically be powered by a battery. The process of changing the battery is resource consuming, requiring energy efficient computation and communication.

In this paper, we focus on TLS 1.2 and DTLS 1.2. While TLS 1.3 was finalized in August 2018, DTLS 1.3 is still only in draft state. The versions are not compatible with each other, so to have a reasonable comparison between CoAP and MQTT, we do not consider TLS 1.3 for MQTT. Still, we only consider cipher suites that are compatible with TLS 1.3, i.e., we exclude RSA key exchange and we only consider AEAD modes of operation on the record layer.

The main goal is to better understand the following aspects:

- The cost of adding security to CoAP and MQTT and the difference between AES modes of operation and key sizes when encrypting bulk data.
- The handshake cost, using different cipher suites.
- The cost updating the device firmware in a secure manner.
- The influence of packet loss for bulk data transfer.

Adding TLS/DTLS at the transport layer will incur overhead for both initial handshake messages and encryption/decryption and authentication of messages in the record layer.

The choices for comparison will be detailed in the following subsections, while the results are given in Section 6.

5.1 Adding Encryption to Data

To understand the cost of encrypting data, and how algorithm choices affect the cost, we measure the energy used to encrypt messages of different sizes.

According to the CoAP specification [BZ16], CoAP messages should fit into a single IP datagram to avoid IP fragmentation. Thus, the payload size is bound to 1024 bytes. To transfer larger payloads, CoAP supports a block-wise transfer option. This option enables transferring multiple blocks of information represented as multiple request-response pairs. The block-wise option enables a server to be stateless, since the server handles each block separately and there is no need for any connection setup or memory on server side. In order to cover a wide range of use-cases, we measure the energy for message sizes between 16 and 8192 bytes, i.e., up to 8 blocks.

When measuring the encryption overhead, the TLS/DTLS handshake is not considered, only the encryption in the record layer. The handshake is measured separately, as detailed in the next section.

5.2 TLS/DTLS Handshake

The TLS/DTLS handshake can be based on asymmetric keys or a symmetric Pre-Shared Key (PSK). In the case of asymmetric keys, digital signatures are used to authenticate the key exchange message in the handshake, while in the PSK case it is possible to either directly agree on a PSK to use as pre-master secret, or to use Diffie-Hellman key exchange (for perfect forward secrecy), and then adding the PSK in order to construct the pre-master secret. Since asymmetric operations are computationally expensive, PSK can be favourable in constrained environments. Still, digital signatures can be preferred when it is not feasible to pre-share keys. The RawPublicKey variant is currently not supported in mbed TLS, the library used by our device. We analyse the energy consumption for the handshake in the other modes, PreSharedKey and Certificates. The comparison measures both the difference between the modes and how different algorithms and levels of security compare to each other. Moreover, we also compare how client authentication influences the energy and time. All measurements are made for CoAP.

Many cipher suites are available, but we take the NIST guidelines [MC19] into account for the selection of cipher suites. Apart from PSK, we only consider ephemeral keys and Diffie-Hellman key exchange (RSA for key exchange is deprecated). Since mode of operation and cipher algorithm only marginally effects the handshake, we fix these to AES in GCM mode.

We use two approximate security levels of 128 (moderate) and 256 (high) bits. For *moderate* security level we choose elliptic curves of size 224 and 256 bits. For RSA signatures, we then use RSA-2048. CoAP explicitly mandates the use of the curve secp256r1, which uses an elliptic curve with a 256-bit prime (also known as NIST P-256). The curve secp224r1 corresponds to the 112-bit security provided by RSA-2048. For the application data, we use AES-128 together with SHA-256. Since we use GCM/CCM, the hash algorithm is only used for the PRF in the handshake, not to compute a MAC on the record layer. For *high* security level, we instead use 384- and 512-bit curves, and RSA-4096. We adjust the encryption key to 256 bits and use SHA-384. See e.g., [Bar19] for more information on security levels.

Although the use of PSK is not recommended by NIST, they do list a set of PSK cipher suites that can be used. We compare both plain PSK, i.e., (PSK as premaster secret (`PSK`), and when Diffie-Hellman is used together with the PSK (`DHE_PSK`).

5.3 Firmware Update

A new firmware is often relatively large, at least in comparison to the typical data packets sent by devices. A firmware update process typically consists of:

1. Download the firmware from a server.
2. Store the firmware such that the bootloader can boot into the firmware.
3. Reboot the device, using the new firmware.

The ESP32 has builtin OTA (Over-The-Air) update functionality, combining steps 1 and 2 above. We measure the energy and time used to both download and store a firmware with size 870 KiB. Rebooting is not part of the measurement. We compare different encryption key sizes and modes of operation for AES, for both CoAP and MQTT, in order to analyze the difference for these larger data transmissions.

5.4 Packet Loss

Packet loss can be costly due to retransmission, resulting in more energy consumption. If a CoAP message is marked as CON, it will be retransmitted until the receiver sends an ACK. For MQTT, retransmission is handled by the TCP layer.

We simulate packet loss with loss rate varying from 0 to 20% using CoAP and MQTT, both with and without security. Very high ranges of packet loss may cause a session to be lost due to timeout and ranges of very low packet loss does not have much effect on energy consumption. Also, based on used loss rates defined in [Tha+14], we use packet loss rates up to 20%.

The CoAP messages were marked as CON in our experiments, thus when a packet loss occurs, the packet will be retransmitted until an ACK is received from the server. One might believe that NON messages are more efficient, if packet loss is not a concern, but this is not the case. According to RFC 7252 [SHB14], CoAP is a request and response protocol, this means that CON and NON messages both are followed by a response. In the case of a CON message, the ACK is usually piggybacked, resulting in no byte overhead.

In CoAP, the retransmission time is primarily controlled by the `ACK_TIMEOUT` parameter, i.e., the time after which a retransmission is made. According to RFC 7252, the default value is 2 seconds, and it is recommended to not be less than 1 second. Following this, we vary the timeout between 1 and 3 seconds, and analyze its effect on the energy consumption with different packet loss rates.

6 Results and Discussion

In this section we give the results of our measurements. In some measurements, we also measured the duration time since energy and time is not fully correlated. Energy consumption is not necessarily the same for doing the required computations and for sending the actual payload. All our measurement results are given as the average of 10 measurements. The power consumption in the idle state is 108 mW, which corresponds to 30 μWh during a second. For the PKI-based handshake, certificates are signed by a CA, so they have a certificate chain of length two.

It is important to note that we are only looking at a specific implementation of CoAP and MQTT, as well as the crypto library. Other hardware and implementations will likely give different results. Instead, focus should be on the general takeaways from our measurements. Such takeaways will be given at the end of each subsection below.

6.1 Analysis of Adding Encryption to Data

We analyze the energy consumption for sending data from the client to the server. CoAP and MQTT communication is analyzed, both without security, and with different variants of channel encryption, as negotiated by the TLS/DTLS handshake. We start the measurement after the client's finished message in the handshake and stop the measurement when all data has been transmitted.

The total energy consumption for CoAP is given in Fig. 2, while the results for MQTT can be found in Fig. 3. Looking at plain data (NULL), CoAP consumes slightly less energy than MQTT and is more efficient for packet sizes up to 1024 bytes. This is reasonable since UDP is more efficient than TCP over a reliable network. However for larger data chunks (> 1024 bytes), CoAP consumes much more energy, so here MQTT is more efficient. The reason for this can be found on application level. CoAP has a 1024 byte limitation on packets with no support for fragmentation. Thus, CoAP must send several consecutive chunks of 1024 bytes, while MQTT can send a continuous stream of data. Further, we can note that the energy cost of sending data is constant for up to around 1 KiB of data for both CoAP and MQTT. The energy needed to initiate and end the actual WiFi communication is then significantly larger than the energy cost of the actual data transmission. For CoAP (UDP), this overhead is around 5.7 μWh (with standard deviation 0.07) and for MQTT (TCP) it is around 6.1 μWh (with standard deviation 0.3).

From the result in Fig. 2, we can see that adding DTLS to CoAP messages (shown with different cipher suites in Fig. 2) adds a small amount of energy (compared to NULL), in the order of 0.3 μWh for small messages. For larger messages, the added energy is increasing. Adding TLS to MQTT gives a lower overhead compared to DTLS, likely since it has less byte overhead compared to DTLS as discussed earlier. For smaller messages the difference is not visible, and within one standard deviation of the measurements. For larger messages the difference is evident.

Looking at Fig. 2 and 3, with different cipher suites, it is clear that there is not much difference in the choice of keysize (128 vs. 256 bits) and modes of operation (GCM vs. CCM). Calculating a 95% confidence interval shows that there is no

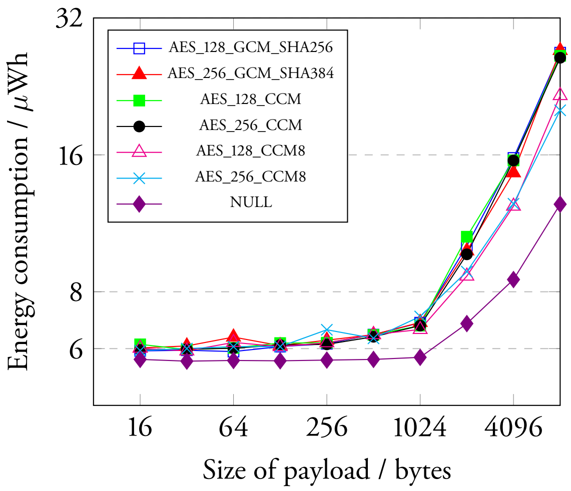


Figure 2: Energy consumption for CoAP/CoAPs protocols with different cipher suites and different payload sizes

statistical difference. One thing that we can notice is that for CoAP messages, the CCM8 variants require (statistically) less overhead compared to the other variants. For messages of size 8192 bytes the energy saving is around 6 μWh.

From Fig. 2 and 3 we also can conclude that aggregating as much data as possible is beneficial for energy consumption. For instance, sending 8 kiB in frames of 16 bytes is much more energy consuming than transferring 8 KiB into frames of 1024 bytes. This is more important if this also can reduce the number of handshakes, as will be seen in the next section.

Key takeaways:

- Though modes of operation and key sizes changes the cryptographic algorithms, for our used hardware and software libraries, this has very little impact on energy consumption. Still, CCM8 seems to be somewhat cheaper.
- Since CoAP can send packets of size up to 1024 bytes, the overhead of sending several packets makes MQTT less energy consuming for payload sizes larger than 1024.
- In both CoAP and MQTT, sending aggregated data is more beneficial than sending data in smaller packets.

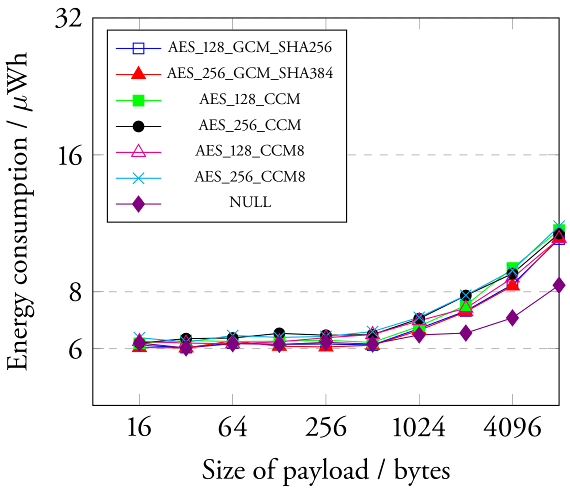


Figure 3: Energy consumption for MQTT/MQTTs protocols with different cipher suites and different payload sizes

6.2 Analysis of PSK-based and PKI-based DTLS Handshakes

To measure the handshake part of a secure connection in CoAP, we setup a secure connection from the ESP32 to the server, forcing the client to support only one specific cipher suite listed in Table 1. To establish the actual connection, the client sends 16 bytes of data to the server. For each cipher suite, we measure the energy consumption for the handshake, including transferring 16 bytes of data and closing the connection. This is performed both with and without client authentication. The measured values and duration times, are given in Table 1.

For the two PSK-based methods, the addition of Diffie-Hellman (i.e., PFS) will significantly increase the energy needed.

The remaining cipher suites can be compared from different perspectives. Comparing RSA and ECDSA shows that when only the server is authenticated, RSA performs better, but when also the client is authenticated, RSA becomes less efficient than ECDSA. This is due to the asymmetry in RSA signatures, where signing is much more costly than verification. For ECDSA, these costs are much more symmetric.

Comparing the elliptic curves, the increase in energy when increasing the size is relatively constant, except for the case of RSA signatures. This suggests that using 4096-bit signatures is much more costly than using 2048-bit signatures. For high security levels and mutual authentication, ECDSA is thus highly preferred. However, without client authentication, RSA can be considered.

DHE (2048-bit prime) with RSA signatures requires much energy. The main reason is that this (Diffie-Hellman) uses an exponentiation with a secret value. It is thus clear that ECDHE should always be preferred over DHE.

Looking at the time for the handshake, it is often very slow. Analyzing the communication using Wireshark, we find that the vast majority of the time is being spent while waiting for the client to respond with the Client Key Exchange message.

Comparing the handshake to sending data, it costs around 6 μWh to send up to 1 KiB data, while any (non-PSK) key exchange will cost in the order of 50-500 μWh depending on cipher suite. Thus, minimizing the number of handshakes, and to make them more efficient, should be prioritized.

Key takeaways:

- Due to the asymmetric cost for signing and verifying RSA signatures, RSA is a viable option when client authentication is not used.
- ECDHE should always be preferred over DHE.
- Client computations are responsible for most of time and energy. This is particularly evident when the client computes RSA signatures, as is the case when we have mutual authentication (and RSA).
- Plain PSK is significantly more efficient than any other alternative.

Table 1: The handshake energy consumption (with standard deviation from 10 measurements is given in parentheses)

	Mutual authentication		**One way authentication**	
Cipher suite	**CoAP energy consumption (μWh)**	**CoAP time (ms)**	**CoAP energy consumption (μWh)**	**CoAP time (ms)**
`TLS_DHE_PSK_WITH_AES_128_GCM_SHA256`	268.66 (5.13)	4186 (23.06)	-	-
`TLS_PSK_WITH_AES_128_GCM_SHA256`	18.53 (0.61)	163.39 (5.28)	-	-
`TLS_ECDHE(224r1)_RSA_AES_128_GCM_SHA256`	114.33 (2.88)	1606 (8.14)	48.90 (3.90)	578.96 (34.78)
`TLS_ECDHE(256r1)_RSA_AES_128_GCM_SHA256`	125.00 (2.00)	1771 (12.16)	57.36 (1.59)	720.66 (9.44)
`TLS_ECDHE(384r1)_RSA(4096)_AES_256_GCM_SHA384`	463.66 (19.29)	7572 (367.73)	118.37 (20.09)	1678 (326.05)
`TLS_ECDHE(521r1)_RSA(4096)_AES_256_GCM_SHA384`	473.00 (15.39)	7691 (205.14)	155.00 (4.35)	2209 (112.42)
`TLS_DHE_RSA_AES_128_GCM_SHA256`	339.00 (1.00)	5395 (30.31)	276.33 (3.51)	4328 (19.15)
`TLS_ECDHE_ECDSA(224r1)_AES_128_GCM_SHA256`	106.33 (12.22)	1556 (217.41)	75.86 (0.90)	1033 (18.71)
`TLS_ECDHE_ECDSA(256r1)_AES_128_GCM_SHA256`	135.66 (0.57)	2098 (26.57)	100.23 (1.53)	1475 (7.63)
`TLS_ECDHE_ECDSA(384r1)_AES_256_GCM_SHA384`	178.00 (8.18)	2721 (151.71)	143.00 (2.64)	2168 (36.17)
`TLS_ECDHE_ECDSA(521r1)_AES_256_GCM_SHA384`	276.66 (7.37)	4432 (148.67)	213.33 (0.57)	3327 (26.40)

6.3 Analysis of Firmware Update

In our experiments, we used the block-wise option in CoAP for transferring a firmware update. The firmware, which is around 870 KiB, is transferred in 870 consecutive blocks from the server to the client, which the client writes to flash. We used MQTT as-is, since it handles the data size without problems. Upon

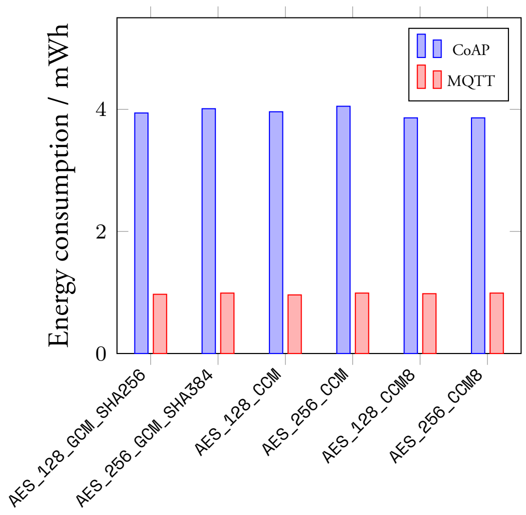


Figure 4: OTA firmware update energy consumption in CoAP and MQTT protocols with different cipher suites

completion, the client verifies the firmware. Once verified, the device reboots into the new firmware.

In the experiments, we measured the required energy between the client receiving the first and last block of the firmware. The energy required for rebooting into the new firmware was disregarded. The results of the energy consumption and also the duration time for receiving all the blocks are shown in Fig. 4 and 5. As illustrated in Fig. 4, there is not much difference between different cipher suites in case of energy consumption for both CoAP and MQTT. The small differences that we can see in Fig. 4 are consistent with the previous measurement in that CCM8 is slightly more efficient than the other options (the two rightmost bars). Calculating a 95% confidence interval supports this observation. The duration time to receive all the encrypted firmware blocks in CoAP using CCM8 (shown in the two rightmost bars) is also less than the required time for other ciphers, as illustrated in Fig. 5.

We again notice that CoAP requires more energy and time compared to MQTT. This is consistent with the previous measurement shown in Fig. 2 and 3.

Key takeaway:

- For transferring the OTA update, MQTT is more efficient with a factor of 4 for energy, and almost a factor of 2 for time. This is because in CoAP, the firmware is transferred in several small blocks.

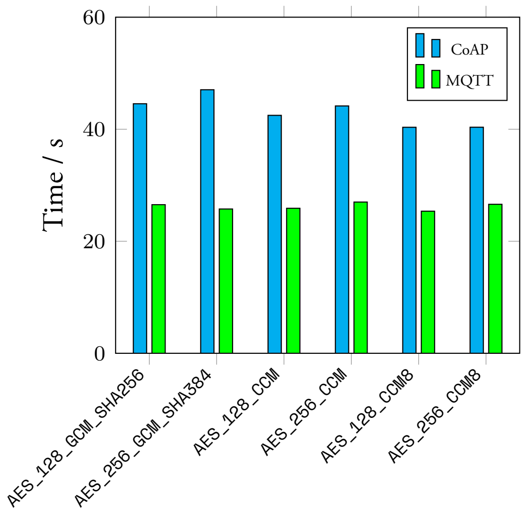


Figure 5: OTA firmware update duration time in CoAP and MQTT protocols with different cipher suites

6.4 Analysis of Packet Loss

We simulated packet loss using the traffic control utility `tc` along with the network emulator `NetEm` on the server side for both CoAP and MQTT with and without security. `NetEm` was configured to drop packets from the interface with a given probability. In our experiments, 500 packets with a fixed size of 512 bytes were sent from the client to the server (without any delay between sending the packets), with a loss rate up to 20 %. The energy consumption was measured to capture the effect of retransmission.

In bulk CoAP transmission, the packets will be sent consecutively until a packet loss occurs. When loss occurs, the client waits until the next retransmission time and no more packets will be sent before then. In the retransmission, the lost packet and also the remaining packets will be sent to the server.

Bulk transmission is handled differently in MQTT. Since MQTT utilizes TCP with its sender window, the sender might send the last packet while it is still waiting for previous packet acknowledgements. As a result, bulk transmission in MQTT with packet loss is much faster than CoAP since it does not have to wait for every packet to be acknowledged.

Since there was not much difference between different key sizes and different cipher suites as discussed in Section 6.1, in the packet loss measurements we only consider AES with key size of 128. The energy consumption for different packet loss rates for transferring 500 packets of size 512 bytes for CoAP and MQTT protocols are given in Fig. 6 and 7. The energy consumption will increase with increasing packet loss, but this increase in CoAP, as illustrated in Fig. 6, is much more than in MQTT (note the different scaling of the y-axis). This is due to the differences in handling bulk transmission as explained above.

In Fig. 6 and 7, there is not any significant difference between different modes of operation. NULL has the lowest energy consumption in both CoAP and MQTT protocols but in CoAP, the difference between NULL and other cipher suites decreases for packet loss rates higher than 10 percent. Because, in CoAP, when packet loss rate increases, the waiting time between retransmissions also increases and the actual number of retransmissions decreases. The required energy for each second of waiting time between retransmissions is around 30 μWh. Sending the same number of packets (500 packets) in CoAP and MQTT, the required time in CoAP for 20 percent packet loss is around 7 minutes while in MQTT this time is only around 1 minute. This results in the huge difference between energy consumption in CoAP and MQTT protocols for higher packet loss rates.

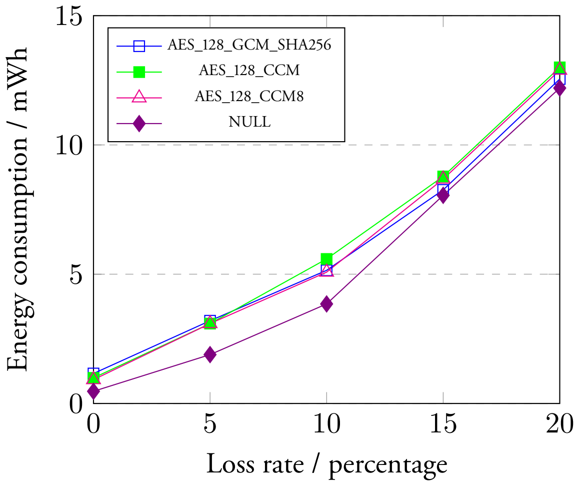


Figure 6: Energy consumption for CoAP/CoAPs protocols with different packet loss rates

The effect of the ACK_TIMEOUT parameter under different packet loss rates is shown in Table 2. Since much energy is consumed while waiting for the next packet, lowering this waiting time can reduce the energy. Comparing a 1 second ACK_TIMEOUT with the default 2 second value indicates that the control parameters can have significant impact on the energy consumption, in particular in environments with high packet loss rates. As seen in Table 2, the energy consumption for 15 and 20% packet loss rates with 2 seconds ACK_TIMEOUT is almost double the required energy for 1 second.

The experiments showed that by increasing the ACK_TIMEOUT, the energy consumption will be increased but this increase is more visible in higher packet loss rates. Therefore, it is important to optimize the retransmission parameters in CoAP protocol according to the environment.

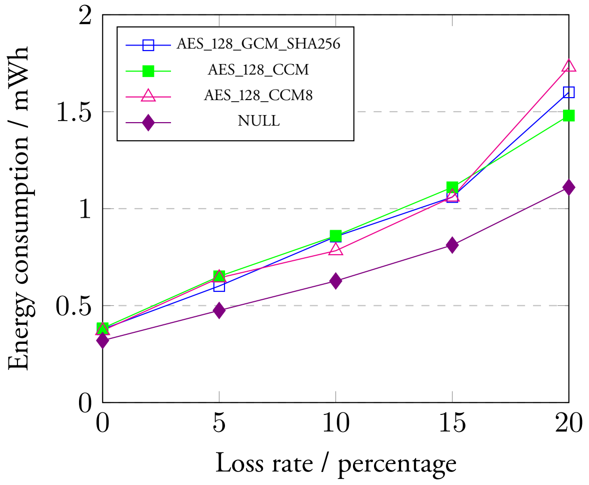


Figure 7: Energy consumption for MQTT/MQTTs protocols with different packet loss rates

Key takeaways:

- For CoAP on lossy networks, the encryption overhead decreases with the loss rate, since the waiting time between retransmissions also increases and the actual number of retransmissions decreases.
- MQTT performs significantly better in lossy networks due to the algorithms used in TCP.
- In CoAP, varying the retransmission control parameters such as `ACK_TIMEOUT` can help in reducing energy consumption quite much, if the network supports this.

7 Conclusions

In this paper we give a better understanding of the difference in energy overhead for two common IoT protocols, CoAP or MQTT. We have done real-world experiments to measure and analyze the energy overhead of adding security to CoAP and MQTT on an ESP32 IoT deviceDuring data transfer, we show that there is a constant energy overhead for up to around 1 KiB of data, which is the cost for setting up the WiFi connection. Above 1 KiB, CoAP has a higher penalty compared to MQTT, due to the block-wise transfer mechanism. Thus, for smaller packet sizes, CoAP and MQTT are comparable, while MQTT is favorable for larger packets. Due to the overhead of setting up the wireless connection, it is clear that data should be aggregated, whenever possible.

Table 2: The effect of ACK_TIMEOUT parameter on CoAP protocol energy consumption

Energy Consumption (mWh)					
ACK_TIMEOUT (s)	1	1.5	2	2.5	3
5% packet loss	1.39	1.55	1.88	2.38	2.50
10% packet loss	2.39	3.16	3.85	4.85	6.29
15% packet loss	3.98	6.00	8.05	9.86	10.73
20% packet loss	6.46	7.78	12.20	13.58	16.65

Looking at the DTLS handshake, ECDHE should always be preferred over DHE, while RSA as digital signature has a slight advantage when client authentication is not needed.

In most cases, MQTT outperforms CoAP, much due to the window based retransmission strategy in the underlying TCP protocol. The most suitable use case for CoAP is on a reliable network, transmitting small packets of data. For other cases, the strength of TCP becomes evident.

Note that the numbers provided are hardware and implementation specific. Also there are other factors that can affect the results, such as environmental conditions, network topology, integration of many sensors in IoT systems, device use case, etc. As a result all of above factors need to be considered in the energy consumption of IoT devices.

References

[Bar19] E. Barker. “Draft NIST Special Publication 800-57, Part 1, Revision 5”. In: *National Institute of Standards and Technology (NIST)* (2019).

[BZ16] C. Bormann and E. Z. Shelby. *Block-Wise Transfers in the Constrained Application Protocol (CoAP)*. RFC 7959. https://tools.ietf.org/html/rfc7959. RFC Editor, Aug. 2016.

[Cap+15] A. Capossele et al. “Security as a CoAP resource: an optimized DTLS implementation for the IoT”. In: *2015 IEEE international conference on communications (ICC)*. IEEE. 2015, pp. 549–554.

[Esp19] Espressif Systems. *ESP32-WROOM-32D & ESP32-WROOM-32U*. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf, Last accessed 2019-05-29. 2019.

[Esp20] Espressif. *Espressif IoT Development Framework*. `https://github.com/espressif/esp-idf`, Last accessed 2020-01-20. 2020.

[GMS15] J. Granjal, E. Monteiro, and J. S. Silva. “Security for the internet of things: a survey of existing protocols and open research issues”. In: *IEEE Communications Surveys & Tutorials* 17.3 (2015), pp. 1294–1312.

[IOU17] M. Iglesias-Urkia, A. Orive, and A. Urbieta. “Analysis of CoAP implementations for industrial Internet of Things: A survey”. In: *Procedia Computer Science* 109 (2017), pp. 188–195.

[MC19] K. McKay and D. Cooper. *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations*. Tech. rep. National Institute of Standards and Technology, 2019.

[Raz+13] S. Raza et al. “Lithe: Lightweight secure CoAP for the internet of things”. In: *IEEE Sensors Journal* 13.10 (2013), pp. 3711–3720.

[Sel+19] G. Selander et al. *Object Security for Constrained RESTful Environments (OSCORE)*. RFC 7252. RFC Editor, July 2019.

[SFF18] M. Suárez-Albela, P. Fraga-Lamas, and T. Fernández-Caramés. “A practical evaluation on RSA and ECC-based cipher suites for IoT high-security energy-efficient fog and mist computing devices”. In: *Sensors* 18.11 (2018), p. 3868.

[SHB14] Z. Shelby, K. Hartke, and C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. RFC Editor, June 2014.

[Suá+18] M. Suárez-Albela et al. “A Practical Performance Comparison of ECC and RSA for Resource-Constrained IoT Devices”. In: *2018 Global Internet of Things Summit (GIoTS)*. IEEE. 2018, pp. 1–6.

[TBK19] A. Thantharate, C. Beard, and P. Kankariya. “CoAP and MQTT Based Models to Deliver Software and Security Updates to IoT Devices over the Air”. In: *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2019, pp. 1065–1070.

[Tha+14] D. Thangavel et al. “Performance evaluation of MQTT and CoAP via a common middleware”. In: *2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)*. IEEE. 2014, pp. 1–6.