CRYPTREC
Cryptographic Technology Guideline
(Lightweight Cryptography)

CRYPTREC
Lightweight Cryptography Working Group

March 2017

CRYPTREC Lightweight Cryptography WG Members

WG Chair	Naofumi Homma	Tohoku University
WG Member	Kazumaro Aoki	Nippon Telegraph and Telephone Corporation
WG Member	Tetsu Iwata	Nagoya University
WG Member	Kazuto Ogawa	Japan Broadcasting Corporation
WG Member	Hisashi Oguma	Toyota InfoTechnology Center
WG Member	Kazuo Sakiyama	The University of Electro-Communications
WG Member	Kyoji Shibutani	Sony Global Manufacturing & Operations Corporation
WG Member	Daisuke Suzuki	Mitsubishi Electric Corporation
WG Member	Yuichiro Nariyoshi	Renesas Electronics Corporation
WG Member	Kazuhiko Minematsu	NEC Corporation
WG Member	Hideyuki Miyake	Toshiba Corporation
WG Member	Dai Watanabe	Hitachi, Ltd.

Contents

Chapter 1

Introduction

Research and development of compact high-speed cryptographic technology have been conducted in response to the constant need for secure high-performance cryptography in restricted implementation environments. These days, with the evolution of the Internet of Things (IoT), devices with limited resources, such as sensors and actuators, are connected to the Internet, raising security and privacy concerns. This is why a wider variety of implementations are required of cryptographic technology. Lightweight cryptography, which can be implemented at lower costs and with lower power consumption compared to the conventional cryptography, is expected to have many applications including on-vehicle devices and medical equipment. It is often not easy for engineers without expertise to exploit this technology by choosing a proper lightweight cryptographic algorithm and applying it with appropriate caution. CRYPTREC mainly reviews the cryptographic technology used for e-government. It also studies the cryptographic technology that has potential applications in various fields, and publishes the result of the study for be benefit of the society. In particular, Lightweight Cryptography Working Group (WG) was established under the CRYPTREC Cryptographic Technology Evaluation Committee in 2013. It aims to supporting the users of the products and services that require lightweight cryptographic technology to select appropriate algorithms with ease. This guideline has been compiled by the Lightweight Cryptography WG with the purpose of contributing to technical decisions on the selection and application of lightweight cryptographic algorithms and promoting the dissemination of this technology. The target readers are engineers exploiting cryptographic technology for designing, developing, and implementing security functions for information systems. However, the guideline is also useful for readers in general who are interested in the lightweight cryptographic technology.

Chapter 1 provides an overview of this guideline. Chapter 2 describes lightweight cryptography, starting with the description of the types of lightweight cryptography and their applications, followed by an implementation guide explaining the features of lightweight cryptography, typical use cases and methods, parameter selection, and notes on use. Chapter 3 shows the performance of typical lightweight cryptographic algorithms. In the fields of "block cipher" and "authenticated encryption," where many cryptographic algorithms have been proposed, some representative algorithms were selected and their performances compared. Comparisons were made in the same environment for both hardware and software implementation. In the case of hardware implementation, circuit size, power consumption, and latency are compared, while software comparisons deal with the required amount of memory. In Chapter 4, the basic information on typical lightweight cryptographic algorithms is represented by technical field: block cipher, stream cipher, hush functions, message authentication codes, and authenticated encryption.

This guideline was written and edited by the members of the Lightweight Cryptography WG and the Cryptography Research and Evaluation Committee (CRYPTREC) Secretariat listed below. The organization to which each member belongs is as of October 2016. Moreover, Mitsuru Matsui, Takeshi Sugawara, Yumiko Murakami, and Shoei Nashimoto, Mitsubishi Electric Corporation made a great contribution to the implementations and performance comparison of lightweight cryptography for this guideline.

WG Chair	Naofumi Homma	Tohoku University
WG Member	Kazumaro Aoki	Nippon Telegraph and Telephone Corporation
WG Member	Tetsu Iwata	Nagoya University
WG Member	Kazuto Ogawa	Japan Broadcasting Corporation
WG Member	Hisashi Oguma	Toyota InfoTechnology Center
WG Member	Kazuo Sakiyama	The University of Electro-Communications
WG Member	Kyoji Shibutani	Sony Global Manufacturing & Operations Corporation
WG Member	Daisuke Suzuki	Mitsubishi Electric Corporation
WG Member	Yuichiro Nariyoshi	Renesas Electronics Corporation
WG Member	Kazuhiko Minematsu	NEC Corporation
WG Member	Hideyuki Miyake	Toshiba Corporation
WG Member	Dai Watanabe	Hitachi, Ltd.
Secretariat	Shiho Moriai	NICT
Secretariat	Miyako Ohkubo	NICT
Secretariat	Sachiko Kanamori	NICT

Chapter 2

Lightweight Cryptography and Its Applications

2.1 What is Lightweight Cryptography?

The research and development of lightweight cryptography for implementation on devices with limited resources has been on the rise in recent years, and many systems have been proposed in academic meetings. In Europe, this technology was chosen as the theme of the European Commission's 6th and 7th Framework Programmes, ECRYPT I and ECRYPT II, as early as 2004. Japan has made great advances in this field with cryptographic technology suitable for compact implementation. Standardization activities are ongoing and include ISO/IEC 29192, which defines the lightweight algorithms for each technical field, and ISO/IEC 29167, which describes cryptographic technology for radio-frequency identification (RFID) devices. The National Institute of Standards and Technology in the U.S. has also started a review of the standardization of lightweight cryptography in 2015.

Low-cost, low power-consumption, lightweight cryptographic technology will be used in devices including on-vehicle equipment and medical equipment. It is expected to become one of the security technologies useful for establishing next-generation network services such as the IoT and the cyber-physical system.

Various methods have been proposed for lightweight cryptographic technology. Some seek light weight in terms of hardware implementation size and power consumption, while others seek it in terms of the required memory size of embedded software. Each method is optimized according to a different performance metric. There is no generally agreed upon definition of "lightweight cryptography." In addition, there is a trade-off between performance and security. Actual performance is multifaceted. In consideration of these circumstances, this guideline covers cryptographic technology designed to have a weight advantage over conventional cryptographic technology in certain performance metrics, with the trade-off between implementation performance and security taken into account; it primarily targets those methods that insist their superiority in the typical performance metrics for which applications are assumed. Since there are almost no widely accepted lightweight cryptosystems in public-key cryptography at this time, this guideline deals only with lightweight cryptosystems in symmetric-key cryptography.

This guideline is based on the following representative performance metrics for lightweight cryptography:

- Hardware implementation
  - Circuit size
  - Energy
  - Latency
- Embedded software implementation
  - Memory size (ROM/RAM)

Circuit Size The size of the circuit in hardware implementation is directly connected to the cost. It is also known to serve as a metric for power consumption. Circuit downsizing is an important factor in applications such as RFID with severe circuit-mounting requirements. It is also crucial for devices such as non-contact IC cards that operate on the power supplied through electromagnetic induction instead of batteries or via electrical connections to external power sources.

Energy Saving energy is required by all battery-powered devices, including medical devices implanted in human bodies and those worn in close contact with the body.

Latency In this guideline, latency is the time required for a single process of encryption (or decryption). Low latency is required for applications including memory encryption and encryption in on-vehicle devices that demand real-time operation.

Memory Size In embedded software implementation, the cryptographic function is often implemented as part of various applications realized on the CPU. The CPU used for an embedded system has limited sizes of ROM and RAM, and the smaller the implementation size of the cryptography, the wider the selection of available CPUs and the lower the cost. Embedded CPUs are used in a variety of devices including electrical home appliances, sensors, and on-vehicle equipment where a small memory requirement (ROM/RAM) is important.

Performance Metrics	Applications
Circuit size (power consumption and cost)	RFID and low-cost sensors
Energy	Medical devices and battery-powered devices
Latency (real-time performance)	Memory encryption, on-vehicle device, industrial I/O device control
Memory size (ROM/RAM)	Electrical home appliances, sensors, and on-vehicle devices

This guideline shows lightweight cryptographic algorithms and schemes in the classification of block ciphers, stream ciphers, hash functions, message authentication codes, and authenticated encryption. The functions enabled by lightweight cryptography such as confidentiality or integrity are basically the same as those enabled by existing cryptographic techniques.

The lightweight cryptographic algorithms and schemes shown in this guideline are selected as of early 2016 according to some criteria such that 1) it is presented at a major academic workshop or conference, 2)severe weakness has not been found, 3) it has implementation efficiency or usefulness in resource-constrained environments. The basic information of the lightweight cryptographic algorithms and schemes shown below is provided in Chapter 4.

Block Ciphers	CLEFIA, LED, Midori, PiccoloPRESENT, PRINCE, SIMON, SPECK, TWINE
Stream Ciphers	ChaCha20, Enocoro, Grain v1, MICKEY 2.0, Trivium
Hash Functions	Keccak, PHOTON, QUARK, SPONGENT
Message Authentication Codes	SipHash
Authenticated Encryption	ACORN, Ascon, AES-JAMBU, AES-OTR, CLOC and SILC, Deoxys, Joltik, Ketje, Minalpher, OCB, PRIMATEs

Section 2.2 shows some examples of the target applications where lightweight cryptography is useful, and Section 2.3 shows how to select lightweight cryptographic algorithms and their parameters. Chapter 3 shows performance comparisons of many lightweight block ciphers and authenticated encryption schemes on the same platform, which would be useful in selecting appropriate lightweight cryptographic algorithms for real applications.

2.2 Target Applications of Lightweight Cryptography

A ubiquitous network that allows people to "connect with anyone or anything at anytime from anywhere" is represented by the IoT. Under the concept of IoT, not only conventional types of

Information and Communication Technology (ICT) terminals including PCs, smartphones, and tablet PCs, but also automobiles, electrical home appliances, robots, and even facilities themselves will be connected to the Internet [4]. However, presently, it is not possible to clearly state which devices will be connected to the Internet, and it is also uncertain which processes will be performed on those devices. In the age of IoT, we must be prepared for unexpected situations.

IoT terminals are spreading throughout our living spaces and contributing to building the ubiquitous network described above. Under these circumstances, there is a definite need for security functions to protect personal information and privacy and guarantee the integrity of information exchanged on the network. However, since providing such functions is not the main purpose of IoT services, the functions should not hinder the operations for the users.

Moreover, it is unlikely that all IoT devices utilize high-performance CPUs. It should be assumed that some devices have poor computing resources with throughput and memory capacity inferior to conventional ICT terminals and battery-power restrictions on operating time. As an example, automobile IoT devices requirements are shown in Figure 2.1.

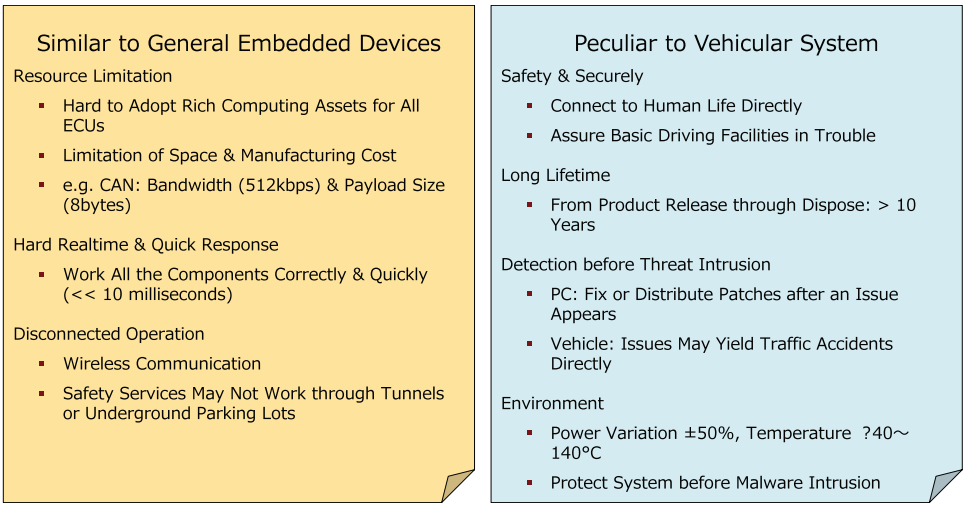

Figure 2.1: Requirements of IoT Terminals on Automobiles

Lightweight cryptography is a promising candidate for applications requiring less load on the CPU, less memory, and lower latency. It is suitable for IoT devices with relatively poor computing resources and is expected to serve various applications that exploit its features.

When the CPU and memory must be shared by many applications (hereafter referred to as "apps"), cryptography with less CPU cost and memory consumption is sometimes demanded. One example is the cryptography used in smartphones, tablet PCs, and smart TVs (high-performance television with an Internet connection).

Furthermore, some devices that operate on batteries expect a cryptographic algorithm that consumes less power. For example, environment-measuring devices are often installed at locations where no utility power is available. Medical implant devices rely only on battery power and are required to be as small as possible to lessen their effects on the human body. Lightweight cryptography is expected to serve these applications.

Some applications demand immediacy and low latency. To keep battery consumption as low as possible, some devices are turned on just before sending data, operate only during data transmission, and are then turned off. Lightweight cryptography featuring low latency is useful for this purpose. It may also be suitable for controlling automobiles where slow response times could compromise security.

The following sections describe the uses of lightweight cryptography by providing examples in the following applications: smart TV, RFID, environmental measuring devices on farmland, medical devices, industrial control equipment, and automobiles.

2.2.1 Electrical Home Appliances and Smart TV

Various types of CPUs, from basic to high-end, are used in smartphones, tablet PCs, and smart TVs. Powerful CPUs can perform a variety of functions; however, low-end CPUs have restrictions.

In a TV set, the hardware unscrambles and decompresses the broadcasting signal and reproduces audio and video content, while other processing is performed by a built-in CPU. The CPU carries out many tasks almost constantly under full (100%) load. Manufacturers are trying to evolve these CPUs so they can run applications that connect to the Internet. To lower the cost, most TV sets use low-end CPUs. On such devices, there is competition for the CPU and memory resources between the TV functions and applications that must run seamlessly. When adding a cryptographic feature on top of these functions, a cryptographic method that requires only a small amount of memory (ROM and RAM) and puts a small load on the CPU is preferable. As mentioned previously, the cryptographic function is implemented with software rather than a dedicated chip. Therefore, a cryptographic algorithm that uses less CPU time and smaller amounts of ROM has a higher practical value.

In the coming age of the IoT, many electrical home appliances will be coming online. Some data exchanged on the Internet are sensitive and must be concealed. Suppose air-conditioners and cooking stoves come online and there is a service that processes their control signals. They are exposed to the risks of unauthorized access that may tamper with the control signals or issue illegal commands that would lead to abnormal operations. When a household owns a home robot, the robot will collect personal data that needs to be concealed to protect one's privacy; and these devices can be used for more than ten years. These home devices mainly depend on software, which is updatable, to process information, and will carry low-priced CPUs with restricted functions, and it may be difficult to build special hardware into small cheap devices. In some devices, resource contention could occur similar to that of smart TVs. To protect data in devices with such low-end CPUs, cryptography software should consume only a small amount of the CPU processing time.

2.2.2 RFID Tag Applications (Logistics Control etc.)

RFID is a system for identifying various objects using radio waves. It is used for various purposes, including controlling warehouse inventory, managing physical distribution, preventing shoplifting at CD/DVD shops, managing item history, operating electronic money, and train IC card and employee ID card systems. Extraordinary examples include an application for tracking wild animals to study their ecology. Similar technology can be used to locate persons. The IoT technology will also be used to identity things in the home, for example, a refrigerator will be able to recognize its contents.

Since RFID uses relatively weak radio waves, it is used only for short distances. Therefore, third-persons outside a warehouse cannot eavesdrop on the RFID signals and detect items inside. Therefore, in the management of warehouse inventory, there is not much need for implementing cryptography to conceal data.

In contrast, when RFID is used to prevent shoplifting, to inhibit the counterfeiting of electronic money, train IC, and employee ID cards, to protect privacy when tracing people and peoplefs belongings, and to conceal the information on the tracking of wild animals from poachers, the data must be encrypted.

An RFID tag is a chip used for an RFID system. Its size varies from several μm per side to several cm per side. A processor, memory, and other components are squeezed into these spaces. Fig.2.2 shows the structure of an RFID chip.

The part labeled "transmitter" is an antenna; its size varies depending on the operating frequency (kHz to MHz). The other parts can be miniaturized; however, actual sizes are determined according to the data storage capacity and the manufacturing process.

In an RFID tag, there is a limit to the circuit area available for the cryptographic function. In the case of passive RFID tags or tags without batteries, there are rigid constraints because the power is obtained from radio waves. When protecting data under such restrictions, a cryptographic algorithm that runs on limited hardware and a small power supply is required.

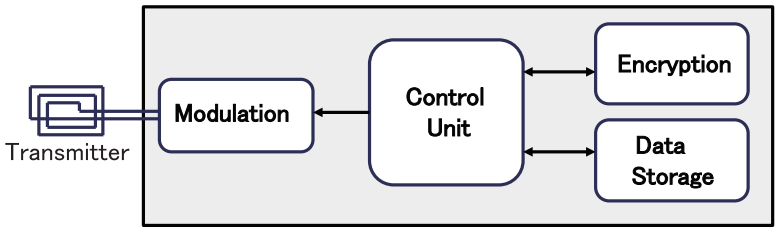

Figure 2.2: Structure of an RFID Device

2.2.3 Smart Agriculture Sensors

It is said that, when raising crops, taking appropriate measures according to changes in the weather stabilizes the yield, which increases productivity and improves the quality of the product. To do this, many farmers depend on their experience and intuition. If this process can be digitized by monitoring the weather conditions (temperature, humidity, hours, and quantity of sunlight, soil moisture, wind direction and speed, and precipitation), inexperienced farmers can enjoy stable harvests similar to those of skilled farmers. The data obtained by monitoring can be used to control the timing and amount of watering and to automatically open and close greenhouse windows. For veteran farmers, monitoring enhances response to weather changes and facilitates planning, work scheduling, and pest control. It also provides guidelines for evaluating and improving farmland to help stabilize the yield.

The precision of analyzing data increases with the resolution of monitoring. Therefore, it is preferable to divide a large farmland into sections and to collect data from each section. Farmers can expect monitoring equipment that can be operated with a small amount of labor at low costs for a long time. To meet this demand, the use of environmental sensor networks is spreading gradually.

The sensors of such networks have the following requirements:

- autonomously driven,
- small-sized,
- low power consumption, and
- used in large numbers.

For fine data acquisition, a large number of sensors are required; low-cost lightweight cryptography is suitable for this purpose.

In addition to monitoring weather, the observation of crustal movements could contribute to the timely prediction of earthquakes, volcanic eruptions, and landslides. Sensors serving these purposes are often installed at locations not easily accessible by people or where no power supply is available. Moreover, since disaster prevention data are crucial for the security of citizens, tampering, which is much more serious than eavesdropping, must be prevented. If data are tampered with, false warnings may be issued. Lightweight cryptography with authentication is suitable for preventing sensitive data from being tampered with.

2.2.4 Medical Sensors

When a person is admitted to a hospital, several types of sensors may be attached to a patient for measuring electrocardiogram, blood pressure, pulse, blood sugar, and blood-oxygen concentration. These sensors are generally connected to main units with cables to carry data and receive power, which inhibits the motion of patients. Patients who are allowed to leave the hospital may need to have their conditions monitored at home and at work and need to have measuring instruments implanted in their bodies. These external and internal devices are used to regularly measure data and determine medication times. An implanted device is usually left inside the body for a long time; therefore, it must be able to transmit data without wires and run on batteries for an extended

period of time. Even in the case of externally attached devices, a wireless connection that provides freedom of movement requires sensors that run for a long time on battery power.

These sensors collect very personal data that must be concealed from the perspective of privacy. Currently, particularly in the field of implant sensors, research and development of sensor miniaturization is ongoing, and nanometer-sized devices are being developed. It follows that the size of the cryptographic processing section must also be reduced.

Furthermore, with the recent advancement of wearable devices, a concept called "mHealth" has emerged. It is sometimes described as an abbreviation of "Mobile Health," but no established definition exists. The term implies applications in which a person wears a device to collect biological data such as cardiopulmonary functions to help maintain the personfs health. Data are accumulated daily and may be submitted during a physical checkup or medical examination at a clinic or hospital. Not only biological data but also activities conducted to improve health are also recorded. Today, conventional health promotion devices such as pedometers can detect such personal information as a personfs location from the built-in GPS.

Some mHealth devices have mechanisms for generating power by the movement of the person wearing the device. However, many wearable devices, which are measuring instruments, operate on batteries.

Depending on its use, wireless transmission may be required for a device that constantly collects data that are received, compiled, and analyzed by other equipment. Since such data are very personal, they must be completely concealed to protect one's privacy.

2.2.5 Industrial Systems

In factories, the transportation, processing, and assembly operations have been automated to improve operational efficiency. Several machine tools and robots can be connected by a network to share manufacturing information and to manage the processes based on the data collected by sensors. Through a network, it is also possible to store information at a single place and to manage the equipment from a central location.

Germany is leading the efforts for this type of automation. Under a national project, a large budget has been spent to realize a smart factory ("thinking factory") according to the Industry 4.0 concept, in which each machine or device has sensors and determines its operation based on the data collected by sensors.

For the central management of manufacturing data, a network standard called "EtherCAT" was proposed. In factory data management, the integrity of real-time data is important. In the case of the conventional TCP/IP Internet protocol suite, collisions that occur on the network result in processing delay and loss of data; therefore, it is not suitable for managing factory data. In EtherCAT, each device is a slave and is connected in series with other devices.

Sensors are located at various places in a factory. Some are at locations difficult for humans to access. Since they are attached to equipment, they have a stable power supply; however, it is sometimes difficult to connect them serially with cables. Therefore, wireless data transmission is considered. When sending data by radio waves, signals need to be transmitted for certain distances and must be encrypted.

2.2.6 Automobiles

Currently, automobiles support not only in-vehicle communication but also communicating with other entities. To support the driving assistance system that enables safe driving through mutual communication between vehicles and one-way communication with infrastructures such as traffic signals and road signs (Car2X communication), a large amount of information needs to be processed. The Car2X system consists of an on-board information system connected to the cloud to provide contents and services and an on-board control system that collects data from external radar and sensors and exchanges information between electronic control units (ECUs) of the body, chassis, and other parts of the vehicle.

For the coordinated control of the on-board ECUs, several types of networks including the controller area network (CAN), local interconnect network (LIN), and Ethernet are installed. The CAN is the main on-board network widely used for the coordinated control of the power train, chassis, and body systems. For example, the distance to a vehicle ahead is measured with a millimeter-wave radar, and if this distance becomes too short, the information shared through the

CAN activates a collision detection system that activates a warning sign or sound, applies the brake assistant system, and tightens the seatbelts. There have been demonstrations of attacks on actual automobiles. If incorrect messages are sent through the CAN, serious situations including improper braking may occur. Therefore, it will be necessary to encrypt messages and check for tampering of messages by authentication. These networks do not demand the very high level of processing performance and low latency required for the communication between vehicles and roadside infrastructures; however, real-time processing and other levels of high-speed processing need to be realized.

In the driving assistance system, there are a significant number of devices that communicate with each other (vehicle-to-vehicle and road-to-vehicle), and cryptographic processing is required to prevent the leakage of personal information. Therefore, a low-latency cryptography system that can be implemented in a small circuit size must be developed. On-board information terminals connect to the cloud and provide various services including traffic information and traffic congestion prediction. This information needs to be protected to prevent tampering. For content protection, high-throughput cryptographic processing is required.

The worldwide automotive open system architecture (AUTOSAR) development partnership describes the necessity for implementing message authentication technology for in-vehicle communication. It specifies message authentication with a counter and a message authentication code (MAC) for secure on-board communication (SecOC) stipulated in R4.2.2 of the AUTOSAR standard [1]. For external communication, the Car 2 Car Communication Consortium (C2C-CC) in Europe[2] or any other organizations discuss on vehicle-to-vehicle communication infrastructures. Based on these situation we can expect to apply lightweight cryptography for such an infrastructure.

2.3 Selection of Lightweight Cryptographic Algorithms and Parameters

2.3.1 General Policy

As shown in Section 2.1, compared to conventional cryptographic algorithms, the implementation of lightweight cryptographic algorithms is superior in one or more of the following criteria:

- Circuit size
- Energy
- Latency
- Memory size

In other words, there may be lightweight cryptographic algorithms featuring a circuit size smaller than a conventional cryptographic algorithms that do not necessarily have lower power consumption. As such, there is no versatile cryptographic algorithm. In actual systems, it is sometimes difficult to specify the required performance indices for cryptography. However, it is important to clarify the system requirement to some extent first, although this may not be easy, before considering the use of lightweight cryptographic algorithms. Then, conventional cryptographic algorithms, in particular, those on the CRYPTREC Cipher List, should be reviewed, and if conventional algorithms cannot satisfy the above criteria, then the use of lightweight cryptography should be considered.

Lightweight cryptographic algorithms are used to protect the privacy of sensitive information and/or to authenticate data. It is important to choose an algorithm that matches the purpose. For example, when using a block cipher, if the mode of operation does not use the decryption of the block cipher, its implementation cost can be lowered. Hard-coding the secret key can also reduce costs. An appropriate method should be selected according to the purpose and the constraints that accompany the implementation.

2.3.2 Selection of a Key Length

The key length is an important parameter that forms the basis of security and must be selected with great care. In a block cipher, in theory, an exhaustive search is possible with a single or a small number of input/output sets. This attack scenario is possible in many cases described in the next section. Let us consider reducing the key length from 128 bits to 80 bits. It should be noted that Moore's law, in which the density of an integrated circuit is quadrupled ($= 2^2$) every three years, the lifetime of the key will be shortened by 72 years ($(128 - 80)/2 \times 3 = 72$).

2.3.3 Selection of a Block Length

The block length of a block cipher is also an important parameter directly affects the security. In particular, when using a block cipher with a short block for block cipher modes and/or authenticated encryption, severe restrictions are posed on the amount of data for which security is maintained, and measures must be taken.

As an example, we introduce the method for evaluating the security of the CTR mode. Assuming a block cipher of block length n bits and σ, the total number of times the block cipher is called under the same key, it has been shown that the probability that the output can be distinguished from a random bit sequence does not exceed $\sigma^2/2^{n+1}$. (See p. 47 of CRYPTREC Technical Report No.2012 (updated on March 4, 2011).)

Table 2.1: Maximum Data Length with which a Ciphertext of a Single User Can Be Distinguished from a Random Character Sequence

Block Length n (bit)	# of Users	Data Length $n\sigma$
64	10^3	1.4Gbyte
	10^6	46.3Mbyte
	10^9	1.4Mbyte
48	10^3	4.3Mbyte
	10^6	139.0Kbyte
	10^9	4.4Kbyte

Under this probability, assuming that the risk of at most one user's ciphertext being distinguished from a random bit sequence is acceptable, the maximum data length that can be processed with the same key can be obtained as shown in Table 2.1. It should be noted that the data size is rather limited.

In addition, there is a known method to construct a hash function using a block cipher as a primitive. The security of this method is maintained by using a block cipher with a sufficiently long block. Therefore, lightweight block ciphers that have shorter blocks are not suitable for this purpose.

2.3.4 Amount of Data Processed, Key Update, and Other Measures

In the use case where the secret key can be updated, frequent updating is a way to maintain the security. If the secret key is hard-coded, it is impossible to change the secret key. In such a case, one solution is to set a limit on the amount of data processed and discard the device before the limit is exceeded.

In general, the security of a cryptographic algorithm gradually degrades each time it processes data. Therefore, it is preferable to update the key while the success probability of an attacker is sufficiently low and acceptable. For example, as mentioned in [3], CMAC recommends that in general applications, if AES with a block length of 128 bits is used, the key should be updated before processing 2^{48} blocks (2^{22} Gbytes) of data, and if TDES with a block length of 64 bits is used, the key should be changed before processing 2^{21} blocks (16 Mbytes) of data. By posing these limits, the probability of a successful attack is expected to be only one in a billion for AES and one in a million for TDES. The acceptable probability of a successful attack depends on the application and should be chosen with care.

An appropriate key-update method should be selected according to the application. In the use case where a key exchange protocol can be executed, there is no problem in updating the key.

Another method is to generate a session key from a master key and to maintain synchronization while updating the key.

In any method, it is necessary to update or discard the key at some time interval. Several cryptographic algorithms that can reduce the update frequency are known, including SUM-ECBC [6] and PMAC Plus [7] for MAC, and CENC [5] for encryption. In these algorithms, assuming the block length of a block cipher is n bits and the number of times the block cipher is called is σ, the success probability of an attacker is approximately at most $\sigma^3/2^{2n}$. Compared to the previous method, the probability of a successful attack on a 64-bit block cipher is $\sigma/2^{64}$ times smaller, which implies that a larger amount of data can be processed before the probability threshold is reached.

2.3.5 Use Scenarios

Lightweight block ciphers are generally designed to be sufficiently secure against linear and differential attacks. This level of security is expected not only for lightweight block ciphers but also for general-purpose block ciphers. In typical block ciphers, the security is evaluated by considering significantly powerful attackers including related-key, known-key, and chosen-key attacks. From the view point of implementation efficiency, lightweight block ciphers often adopt a simple design that is not necessarily secure against these attacks or has undergone sufficient security evaluation. Some lightweight block ciphers are known to be vulnerable to related-key and/or chosen-key attacks; when using these algorithms, measures should be taken to prevent these attack scenarios.

2.3.6 Note on Side-channel Attacks

There are many uses for lightweight cryptographic algorithms, and it is important to choose a proper algorithm for each implementation environment. Some algorithms are suitable for software implementation, and some for hardware implementation. It is important to review not only cryptanalytic attacks but also to take measures against side-channel attacks. In general, there are many scenarios where the side-channel attacks are possible, and it is necessary to consider countermeasures at the implementation level.

2.3.7 Difference from the CRYPTREC Cipher List

The block ciphers on the CRYPTREC Cipher List have a block length of 64 or 128 bits, and a key length of 128 bits or more. For lightweight block ciphers, there are many proposals that have a block length of 32 bits or a key length of 80 bits, which are both shorter than the algorithms listed on the CRYPTREC Cipher List.

The block length and key length are the parameters directly related to the security. It is necessary to consider if the risks by reducing the block and key lengths are acceptable.

Even within the range where the ciphers on the CRYPTREC Cipher List with no annotations can be used securely, when using a lightweight cipher with small parameters, it is necessary to re-evaluate the amount of data that can be securely processed.

No cryptographic algorithm is unconditionally and permanently secure; however, with proper definition of the purpose and evaluation of risks, the use of lightweight cryptographic algorithms are recommended if cryptographic algorithms cannot be used.

2.4 Examples and Effects of Lightweight Cryptography

For the applications described in Section 2.2, this section provides some criteria for selecting lightweight cryptography, in particular, lightweight block ciphers or authenticated encryption schemes. The description is based on the performance comparison of lightweight cryptography indicated in Chapter 3 as an example.

2.4.1 Electrical Home Appliances and Smart TV

For smart TV, in which resource contention can occur, lightweight cryptographic algorithms like SPECK, SIMON, Piccolo, and TWINE may be suitable because its software consumes less CPU time and occupies a smaller ROM space, according to Figure 3.40 in Chapter 3.

When protecting data in electrical home appliances, the updatable software will play the main role, and the CPU on which the software runs is a low-end model. To realize data protection with this type of CPU, lightweight cryptographic algorithms like SPECK with its lower CPU usage will be suitable examples, according to Figure 3.34 in Chaphter 3.

2.4.2 RFID Tag Applications (Logistics Control etc.)

In RFID devices, the amount of circuit area available for the cryptographic function is limited, and allowable power consumption is seriously restricted. Therefore, suitable lightweight cryptographic algorithms examples include SIMON, SPECK, Piccolo, and PRINCE, because they can be implemented in a circuit size of several kgates smaller than AES and consumes less power, according to Figure 3.18 etc. in Chapter 3. The difference in circuit sizes from the other algorithms is critical, especially in legacy chip manufacturing processes with a size of 180 nm or larger. Even in the 40 nm generation process, the difference determines whether cryptographic algorithm is implementable in the very small 50μm class chips.

2.4.3 Smart Agriculture Sensors

To improve crop productivity, a large number of sensors are required to provide the necessary fine data acquisition. Low-cost lightweight cryptographic algorithms will be suitable for encrypting the data from these sensors. In terms of disaster prevention, it is necessary to encrypt the data and append a MAC. For this purpose, lightweight authenticated encryption schemes like JAMBU-SIMON, SILC-PRESENT, ACORN, Ascon, and Minalpher may fit because they require only a small amount of ROM and are suitable for compact implementation, according to Figures 3.45 and 3.47 in Chapter 3. If a lightweight block cipher is implemented, SPECK, SIMON, PRESENT, TWINE, and Midori will be good candidates, because they run with the number of processing cycles smaller than AES, according to Figure 3.42 in Chaphter 3, which means lower power consumption and longer battery life. If messages are short and do not need encryption, the size can be reduced by using the SipHash lightweight MAC shown in Section 4.4 or by applying a lightweight block cipher with the CMAC mode.

2.4.4 Medical Sensors

Medical sensors collect very personal data that must be concealed to ensure privacy. Currently, particularly in the field of implant sensors, miniaturization research and development are ongoing, and nanometer-sized devices are being developed. Thus, the size reserved for cryptographic processing must also be reduced. Lightweight cryptographic algorithms SIMON, SPECK, Picollo, PRESENT etc. satisfy these requirements and can be implemented in hardware with less power consumption, according to Figure 3.14 etc. in Chapter 3.

For mHealth, assuming an able-bodied person is wearing the terminal, miniaturization and extended battery life are not serious issues; however, use of lightweight cryptography is preferable to reduce the size and cost of the CPUs.

2.4.5 Industrial Systems

In industrial automation, open-field networks are emerging, and the EtherCAT ultra-high-speed industrial open network is drawing attention. At a node connected to such a network, a 1,000-point digital I/O must be completed in 30 μs. Each Ethernet frame can exchange up to 1,486 bytes of data. To encrypt the communication channel and prevent tampering with AES, it is necessary to call the encryption circuit four times per block for MAC verification, decryption, (interpretation, rewriting), encryption, and MAC generation. Assuming it takes 100 ns to encrypt a single AES block, a total of 37.2 μs is required; therefore, the conditions are severe in AES. In contrast, according to Figure 3.7 in Chapter 3, the lightweight block ciphers Midori and PRINCE achieve the throughput of 3.9 Gbps and 3.6 Gbps, respectively, with circuit size smaller than AES in the Unrolled implementation. These lightweight cryptographic algorithms are expected to satisfy real-time processing requirements.

2.4.6 Automobiles

When implementing encryption and authentication functions in on-board hardware, the lightweight cryptographic algorithms Midori, PRINCE, PRESENT, and SIMON will be possible candidates because they can reduce the circuit size while maintaining the throughput of AES, according to Figures 3.14 and 3.15 in Chapter 3.

For the cryptographic algorithms suitable for an automatic driving assistance system, a process of several rounds, rather than a single round, is implemented in hardware to reduce latency. Therefore, Midori, PRINCE, PRESENT, and SIMON are promising because they have lower latency and can be implemented with a smaller circuit size compared to AES, according to Figures 3.6 and 3.7 in Chaphter 3.

To protect the information of on-board information terminals, high-throughput encryption/ decryption processing is required for content protection; therefore, for example, the lightweight block cipher Midori is a good candidate, according to Figure 3.15 in Chapter 3.

Bibliography

[1] AUTOSAR Specification of Module Secure Onboard Communication, https://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/safety-and-security/standard/AUTOSAR_SWS_SecureOnboardCommunication.pdf

[2] CAR 2 CAR Communication Consortium, https://www.car-2-car.org/index.php?id=5

[3] Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B (2005), National Institute of Standards and Technology.

[4] White paper 2015, Information and Communications in Japan (2015), Ministry of Internal Affairs and Communications, Japan, http://www.soumu.go.jp/johotsusintokei/whitepaper/eng/WP2015/2015-index.html

[5] Iwata, T.: New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In: FSE. Lecture Notes in Computer Science, vol. 4047, pp. 310–327. Springer (2006)

[6] Yasuda, K.: The Sum of CBC MACs Is a Secure PRF. In: CT-RSA. Lecture Notes in Computer Science, vol. 5985, pp. 366–381. Springer (2010)

[7] Yasuda, K.: A New Variant of PMAC: Beyond the Birthday Bound. In: CRYPTO. Lecture Notes in Computer Science, vol. 6841, pp. 596–609. Springer (2011)

Chapter 3

Comparing the Performance of Lightweight Cryptography

This chapter shows the result of comparing the performance of different cryptographic algorithms implemented on the same platform in hardware and software by the same implementer or under a single implementation policy in a single evaluation environment. For hardware implementation, lightweight block ciphers, and for software implementation, lightweight block ciphers and authenticated encryptions, are evaluated. In the examination of the existing literature, it was difficult to compare cryptographic algorithms because of the differences in the evaluating environments and the implementer.

Twelve lightweight block ciphers listed in Table 3.1 and ten authenticated encryptions listed in Table 3.2 were evaluated.

Table 3.1: Evaluated Lightweight Block Ciphers

Evaluated Block Cipher	Evaluated Block/Key Length	Reference Specifications
AES	128/128	[8]
Camellia	128/128	[9]
CLEFIA	128/128	[25]
TDES	64/168	[11]
LED	64/128	[17]
PRINCE	64/128	[16]
PRESENT	64/80	[19]
Piccolo	64/80	[24]
TWINE	64/80	[26]
SIMON	32/64, 64/96, 64/128, 128/128	[15]
SPECK	32/64, 64/96, 64/128, 128/128	[15]
Midori	64/128, 128/128	[14]

3.1 Block Cipher

3.1.1 Evaluation of Hardware Implementations

Various ways for implementing an encryption circuit are available depending on the purpose. This guideline considers three basic architectures systems: unrolled, round, and serial as indicated in Figure 3.1. In Figure 3.1, the box with Round Function represents a combinational circuit that performs the calculations for the basic functions specified in each encryption algorithm. In the case of the hardware implementation of 12 algorithms, two types of implementation were evaluated: one performing only the encryption operation and the other performing both encryption and decryption with the same module in which the operations are switched by a control signal. In this guideline, the former implementation is described as Enc and the later as Enc/Dec.

Table 3.2: Evaluated Authenticated Encryption

Evaluated Authenticated Encryption	Reference Specifications
ACORN	[27]
AES-GCM	[10]
AES-OTR	[2]
Ascon	[1]
CLOC	[18]
SILC	[12]
JAMBU	[7]
Ketje	[3]
Minalpher	[4]
OCB	[5]

In general, the block cipher algorithm can be divided into the key scheduling function and the encryption/decryption function. For the evaluation in this guideline, an implementation having both functions was built. The key scheduling function and the encryption/decryption function were controlled by the same clock. Algorithms that required no registers for key scheduling were implemented without registers.

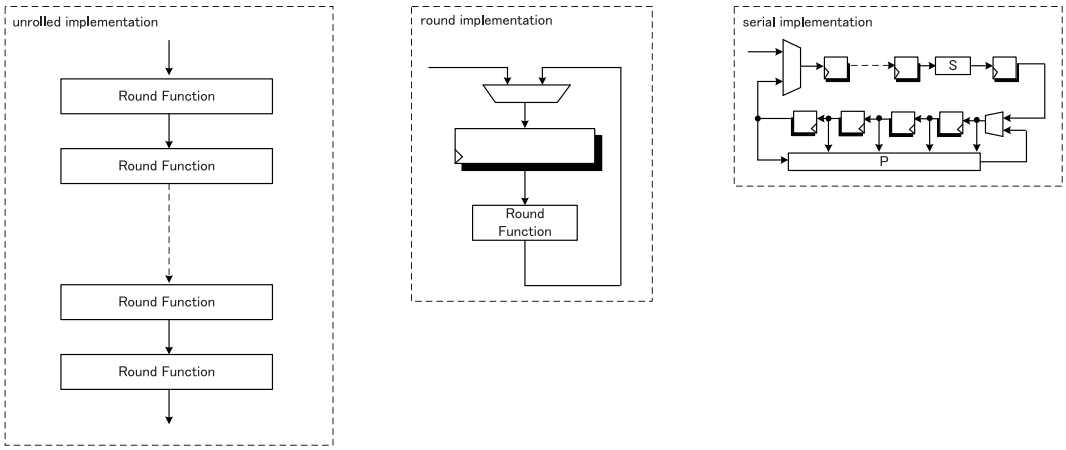

Figure 3.1: Basic Implementation Methods

3.1.1.1 Performance Comparison

Tables 3.3 to 3.5 show the implementation evaluation results. Table 3.6 compares the implemented circuit sizes excluding the interface circuits. Figures 3.2 to 3.25 graphically compare the circuit sizes, processing speeds, peak currents, and leakage currents of the target implementations.

In the tables, "(comp)" means that the S-box is implemented using composite field arithmetic, and "(table)" means that the S-box is implemented using Table lookups.

In Table 3.3 unrolled implementation, the CRYPTREC block ciphers including AES were much larger than those of the lightweight cryptography/low-latency cryptography (except LED). As Table 3.7 indicates, the difference in the performance of the S-box was the dominant factor. In the table implementation optimized for speed, the 8-bit S-box was 100 times larger than the 4-bit S-box of PRESENT and PRINCE, and the delay was five times longer. To achieve a delay performance similar to that of the 4-bit S-box with an 8-bit S-box without compromising efficiency, the number of rounds must be cut to a quarter compared to that of the 4-bit S-box. Even if that could be achieved, the S-box circuit size would be nearly 100 times larger.

From this viewpoint, since PRINCE has almost the same number of rounds as the AES, the difference in the performance of the S-box is directly related to the entire performance. The number of rounds of PRINCE is approximately one third of PRESENT. However, as Table 3.8 indicates,

PRESENT had the faster P-layer, which resulted in a speed difference of less than three times that of PRINCE. Therefore, PRINCE was approximately twice as superior to PRESENT in both circuit size and delay.

The area advantage of PRINCE, TWINE and Piccolo over PRESENT and LED becomes even bigger in implementations with decryption because of the influences of Generalized Feistel Network (GFN) and α-reflection properties. SIMON and Midori have similar throughput as that of PRINCE. Furthermore, the unrolled implementation cases of AES and LED, which need to generate a decryption key before starting decryption, and Camellia and CLEFIA, which generate an intermediate key, had the disadvantage of an additional delay in the critical path. For Piccolo and PRINCE, the maximum operating frequencies of the encryption circuit and the encryption/decryption circuit could be configured to be almost identical.

The next performance comparison is between the round implementation and serial implementation. In AES, around nine kgates could be reduced, but in the cases of PRESENT and PRINCE, the amount of reduction was only several hundred gates to one kgate. As Table 3.7 and Table 3.8 indicate, reducing the number of functional units for the S-box and P-layer has limited effects. However, since the processing speed drops to 1/10 or lower, the serial implementations are inefficient in terms of throughput/area. In addition, serial implementaions are more complicated and thus their source codes become less maintainable. Therefore, if there are no extreme restrictions in implementation, using the serial implementation for lightweight cryptography has no advantages.

Finally, the serial implementation will be discussed. In literature [22], the AES was implemented with 2.4 kgates, but our serial implementation is 1 kgate lager. The reason for the difference is that the number of gates per flip-flop in this implementation is approximately one gate more than the number in literature [22]. Other possible reasons include the buffers inserted during synthesis and the configuration of the control circuit. As described in literature [22], the lack of an optimization using Scan-FF also generates a difference. For PRESENT and PRINCE, the results in this guideline also indicate circuit sizes 1 to 2 kgates smaller than that of AES. Although there is no difference in the PRESENT and PRINCE circuit sizes, PRINCE can be implemented with half-number cycles compared to PRESENT.

Table 3.3: Evaluation of Unrolled Implementation

Algorithm	Block size [bit]	Key size [bit]	Cycles /block	Frequency [MHz]	Throughput [Gbps]	Area [kgate]	Peak power [mW]	Leak power [uW]
Unrolled, Enc								
AES(table)(128/128)	128	128	1	25.7	3.3	112.4	-	-
AES(comp)(128/128)	128	128	1	13.4	1.7	78.8	175.6	939.6
Camellia(comp)(128/128)	128	128	1	7.8	1.0	60.2	136.5	706.7
CLEFIA(128/128)	128	128	1	5.7	0.7	74.6	195.5	891.0
SIMON(128/128)	128	128	1	24.7	3.2	63.2	172.2	685.9
SPECK(128/128)	128	128	1	3.2	0.4	44.4	73.0	417.0
Midori(128/128)	128	128	1	38.5	4.9	34.6	118.2	446.1
TDES(64/168)	64	168	1	10.0	0.6	55.4	111.9	652.2
LED(64/128)	64	128	1	6.9	0.4	74.5	99.1	824.0
PRINCE(64/128)	64	128	1	57.1	3.7	9.8	28.1	107.4
SIMON(64/128)	64	128	1	27.8	1.8	23.8	71.5	260.4
SPECK(64/128)	64	128	1	7.3	0.5	19.5	35.6	183.0
Midori(64/128)	64	128	1	46.5	3.0	12.3	34.9	149.0
SIMON(64/96)	64	96	1	41.3	2.6	20.3	56.7	218.1
SPECK(64/96)	64	96	1	7.6	0.5	18.6	35.4	174.7
PRESENT(64/80)	64	80	1	34.3	2.2	23.9	57.8	259.6
Piccolo(64/80)	64	80	1	18.0	1.2	19.1	61.0	224.8
TWINE(64/80)	64	80	1	24.8	1.6	19.5	43.8	221.2
SIMON(32/64)	32	64	1	39.4	1.3	9.0	30.5	97.4
SPECK(32/64)	32	64	1	15.3	0.5	8.2	17.3	78.0
Unrolled, Enc/Dec								
AES(table)(128/128)	128	128	1	11.4	1.5	208.4	337.2	2612.0
AES(comp)(128/128)	128	128	1	6.4	0.8	144.2	294.3	1734.3
Camellia(comp)(128/128)	128	128	1	7.7	1.0	63.4	133.8	754.9
CLEFIA(128/128)	128	128	1	5.7	0.7	74.3	195.5	891.0
SIMON(128/128)	128	128	1	13.0	1.7	74.1	187.0	803.7
SPECK(128/128)	128	128	1	1.1	0.1	69.1	127.1	672.5
Midori(128/128)	128	128	1	30.7	3.9	55.6	123.7	720.2
TDES(64/168)	64	168	1	9.6	0.6	56.5	112.9	673.3
LED(64/128)	64	128	1	3.1	0.2	215.4	103.1	815.6
PRINCE(64/128)	64	128	1	56.1	3.6	10.1	29.1	108.2
SIMON(64/128)	64	128	1	16.8	1.1	27.5	83.2	299.1
SPECK(64/128)	64	128	1	2.7	0.2	29.9	62.3	290.8
Midori(64/128)	64	128	1	37.7	2.4	20.6	37.1	256.4
SIMON(64/96)	64	96	1	21.5	1.4	23.8	62.9	255.3
SPECK(64/96)	64	96	1	2.9	0.2	28.6	57.8	278.0
PRESENT(64/80)	64	80	1	26.8	1.7	43.8	127.8	505.4
Piccolo(64/80)	64	80	1	16.3	1.0	22.8	64.8	264.0
TWINE(64/80)	64	80	1	13.1	0.8	25.6	50.9	292.2
SIMON(32/64)	32	64	1	23.6	0.8	10.4	30.9	111.8
SPECK(32/64)	32	64	1	6.9	0.2	12.4	27.5	121.7

Table 3.4: Evaluation of Round Implementation

Algorithm	Block size [bit]	Key size [bit]	Cycles /block	Frequency [MHz]	Throughput [Gbps]	Area [kgate]	Peak power [mW]	Leak power [uW]
Round, Enc								
AES（comp)(128/128)	128	128	11	108.2	1.259	15.4	36.1	152.6
Camellia(comp)(128/128)	128	128	23	103.0	0.573	10.8	46.6	107.7
CLEFIA(128/128)	128	128	19	145.8	0.982	10.1	39.8	99.6
SIMON(128/128)	128	128	68	371.7	0.700	7.0	17.4	69.9
SPECK(128/128)	128	128	32	50.3	0.201	7.2	11.4	66.2
Midori(128/128)	128	128	20	386.1	2.471	7.1	11.9	79.7
TDES(64/168)	64	168	48	164.2	0.219	7.9	13.9	76.2
LED(64/128)	64	128	48	208.3	0.278	6.3	5.3	52.5
PRINCE(64/128)	64	128	13	234.2	1.153	5.1	16.4	47.1
SIMON(64/128)	64	128	44	371.7	0.541	5.3	12.4	51.1
SPECK(64/128)	64	128	27	95.8	0.227	5.3	10.5	48.3
Midori(64/128)	64	128	16	340.1	1.361	4.7	11.4	49.1
SIMON(64/96)	64	96	42	392.2	0.598	4.5	11.8	44.1
SPECK(64/96)	64	96	26	95.8	0.236	4.6	10.0	42.4
PRESENT(64/80)	64	80	33	326.8	0.634	4.1	4.7	33.4
Piccolo(64/80)	64	80	27	262.5	0.622	3.5	3.4	34.2
TWINE(64/80)	64	80	36	311.5	0.554	4.4	4.6	40.0
SIMON(32/64)	32	64	32	369.0	0.369	2.9	9.8	28.0
SPECK(32/64)	32	64	22	175.1	0.255	2.9	8.4	26.8
Round, Enc/Dec								
AES（comp)(128/128)	128	128	11	107.0	1.245	18.7	44.1	193.6
Camellia(comp)(128/128)	128	128	23	103.0	0.573	11.8	44.6	121.9
CLEFIA(128/128)	128	128	19	143.1	0.964	9.9	38.1	99.0
SIMON(128/128)	128	128	68	310.6	0.585	7.8	17.2	78.4
SPECK(128/128)	128	128	32	49.9	0.200	9.6	11.2	92.7
Midori(128/128)	128	128	20	271.0	1.734	8.4	11.9	96.9
TDES(64/168)	64	168	48	161.6	0.215	10.6	13.9	114.0
LED(64/128)	64	128	48	188.7	0.252	7.2	6.5	66.6
PRINCE(64/80)	64	128	13	224.7	1.106	5.3	18.7	50.3
SIMON(64/128)	64	128	44	342.5	0.498	6.0	12.4	58.2
SPECK(64/128)	64	128	27	93.5	0.222	6.7	10.6	63.2
Midori(64/128)	64	128	16	266.7	1.067	5.3	11.4	57.5
SIMON(64/96)	64	96	42	342.5	0.522	5.1	11.6	49.9
SPECK(64/96)	64	96	26	93.5	0.230	5.9	9.9	55.7
PRESENT(64/80)	64	80	33	280.9	0.545	4.7	4.9	44.8
Piccolo(64/80)	64	80	27	261.8	0.621	3.8	3.3	38.5
TWINE(64/80)	64	80	36	302.1	0.537	4.7	4.5	42.8
SIMON(32/64)	32	64	32	359.7	0.360	3.3	9.9	31.8
SPECK(32/64)	32	64	22	167.5	0.244	3.6	8.7	34.1

Table 3.5: Evaluation of Serial Implementation

Algorithm	Block size [bit]	Key size [bit]	Cycles /block	Frequency [MHz]	Throughput [Gbps]	Area [kgate]	Peak power [mW]	Leak power [uW]
Serial, Enc								
AES（comp)(128/128)	128	128	226	112.2	63.6	6.3	18.5	76.8
Camellia(comp)(128/128)	128	128	360	109.5	38.9	6.6	14.4	66.1
CLEFIA(128/128)	128	128	175	114.2	83.5	6.2	13.1	61.3
SIMON(128/128)	128	128	4481	269.5	7.7	4.8	8.2	47.1
SPECK(128/128)	128	128	2177	291.5	17.1	5.0	8.2	48.4
Midori(128/128)	128	128	489	254.5	66.6	4.9	11.9	49.2
LED(64/128)	64	128	1872	344.8	11.8	5.6	2.2	50.0
PRINCE(64/128)	64	128	247	246.3	63.8	3.9	8.7	40.0
SIMON(64/128)	64	128	1537	309.6	12.9	3.7	4.8	36.2
SPECK(64/128)	64	128	993	339.0	21.8	3.9	5.4	37.4
Midori(64/128)	64	128	393	253.2	41.2	3.5	11.4	35.3
SIMON(64/96)	64	96	1441	328.9	14.6	3.3	4.5	31.7
SPECK(64/96)	64	96	929	314.5	21.7	3.4	5.1	33.1
PRESENT(64/80)	64	80	563	186.9	21.2	3.9	3.4	36.4
Piccolo(64/80)	64	80	433	300.3	44.4	3.5	2.0	28.5
TWINE(64/80)	64	80	324	277.8	54.9	4.1	2.8	29.6
SIMON(32/64)	32	64	577	389.1	21.6	2.2	3.7	20.8
SPECK(32/64)	32	64	417	390.6	30.0	2.3	5.5	21.9
Serial, Enc/Dec								
AES（comp)(128/128)	128	128	226	108.6	61.5	7.2	14.5	61.2
Camellia(comp)(128/128)	128	128	360	108.3	38.5	7.3	14.8	63.1
CLEFIA(128/128)	128	128	175	113.1	82.7	6.8	12.5	59.3
SIMON(128/128)	128	128	4481	277.8	7.9	5.6	9.7	57.4
SPECK(128/128)	128	128	2177	316.5	18.6	5.9	8.3	57.2
Midori(128/128)	128	128	489	204.1	53.4	5.3	11.9	53.9
LED(64/128)	64	128	1872	303.0	10.4	6.9	1.4	34.5
PRINCE(64/128)	64	128	247	245.7	63.7	3.8	8.4	36.2
SIMON(64/128)	64	128	1537	277.0	11.5	4.5	5.6	45.3
SPECK(64/128)	64	128	993	317.5	20.5	4.8	7.6	46.2
Midori(64/128)	64	128	393	220.3	35.9	3.8	11.4	37.7
SIMON(64/96)	64	96	1441	298.5	13.3	3.9	5.1	39.0
SPECK(64/96)	64	96	929	280.1	19.3	4.1	7.6	40.1
PRESENT(64/80)	64	80	563	170.9	19.4	4.5	2.4	25.8
Piccolo(64/80)	64	80	433	292.4	43.2	3.7	2.0	23.4
TWINE(64/80)	64	80	324	270.3	53.4	4.2	2.6	28.4
SIMON(32/64)	32	64	577	299.4	16.6	2.6	4.1	25.7
SPECK(32/64)	32	64	417	295.9	22.7	2.8	6.3	27.3

Table 3.6: Circuit Size of Each Implementation

	Encryption Circuit Size without Interface [kgate]					
Algorithm	Unrolled, Enc	Unrolled, Enc/Dec	Round, Enc	Round, Enc/Dec	Serial, Enc	Serial, Enc/Dec
AES(table)(128/128)	109.7	205.6	—	—	—	—
AES(comp)(128/128)	76.1	141.5	12.4	15.6	3.2	4.1
Camellia(comp)(128/128)	57.4	60.6	8.0	9.0	4.1	4.3
CLEFIA(128/128)	71.5	71.5	7.3	7.1	3.6	3.8
SIMON(128/128)	60.4	71.3	4.3	5.0	2.1	2.9
SPECK(128/128)	41.6	66.4	4.4	6.8	2.2	3.1
Midori(128/128)	31.8	52.9	4.3	5.6	2.2	2.6
TDES(64/168)	52.8	53.8	5.3	7.9	—	—
LED(64/128)	71.9	212.9	3.8	4.7	3.0	4.3
PRINCE(64/128)	7.8	8.1	2.7	3.0	1.6	1.8
SIMON(64/128)	21.8	25.4	3.2	3.9	1.7	2.5
SPECK(64/128)	17.4	27.8	3.2	4.6	1.8	2.7
Midori(64/128)	10.2	18.5	2.6	3.2	1.5	1.7
SIMON(64/96)	18.4	21.9	2.7	3.2	1.4	2.0
SPECK(64/96)	16.8	26.8	2.8	4.1	1.6	2.3
PRESENT(64/80)	22.0	42.1	2.2	2.9	2.0	2.8
Piccolo(64/80)	17.4	21.1	1.6	1.9	1.1	1.3
TWINE(64/80)	17.8	23.9	2.7	2.9	2.4	2.5
SIMON(32/64)	7.8	9.2	1.7	2.1	1.0	1.4
SPECK(32/64)	7.0	11.2	1.7	2.4	1.1	1.6

Table 3.7: Comparison of S-boxes

Module	Area [gate]	Path delay [ns]
AES 8bit S-box (Table)	3,194	2.43
AES 8bit S-box (Composite)	315	5.75
PRESENT 4bit S-box (Table)	26	0.57
PRINCE 4bit S-box (Table)	18	0.48

Table 3.8: Comparison of P-Layers

Module	Area [gate]	Path delay [ns]
AES 128bit permutation	864	0.89
PRESENT 64bit permutation	0	0
PRINCE 64bit permutation	192	0.51

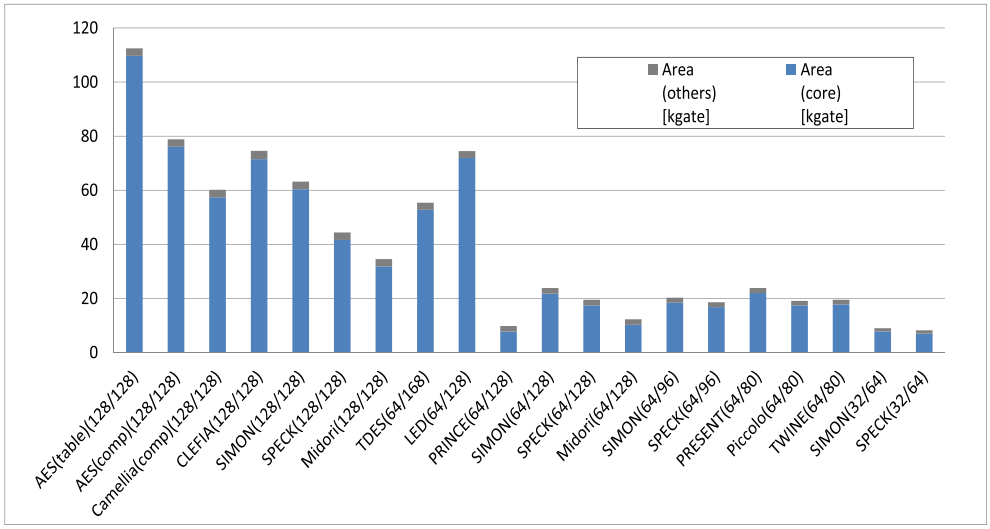

Figure 3.2: Area of Enc, Unrolled Implementation

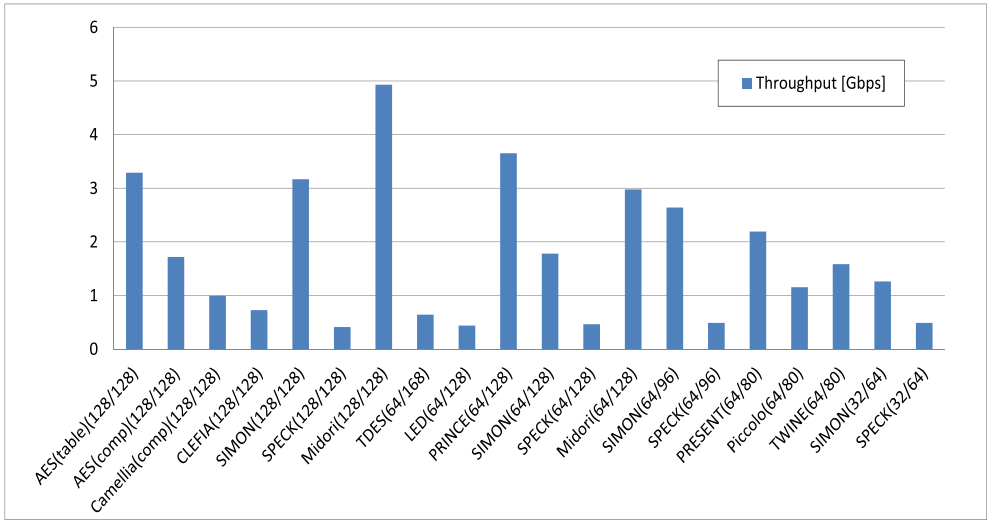

Figure 3.3: Throughput of Enc, Unrolled Implementation

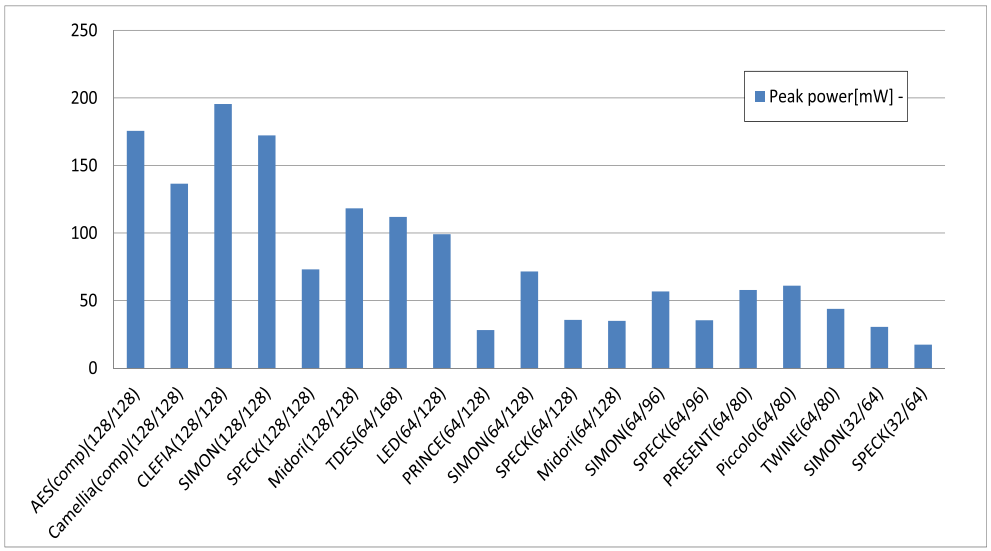

Figure 3.4: Peak power of Enc, Unrolled Implementation

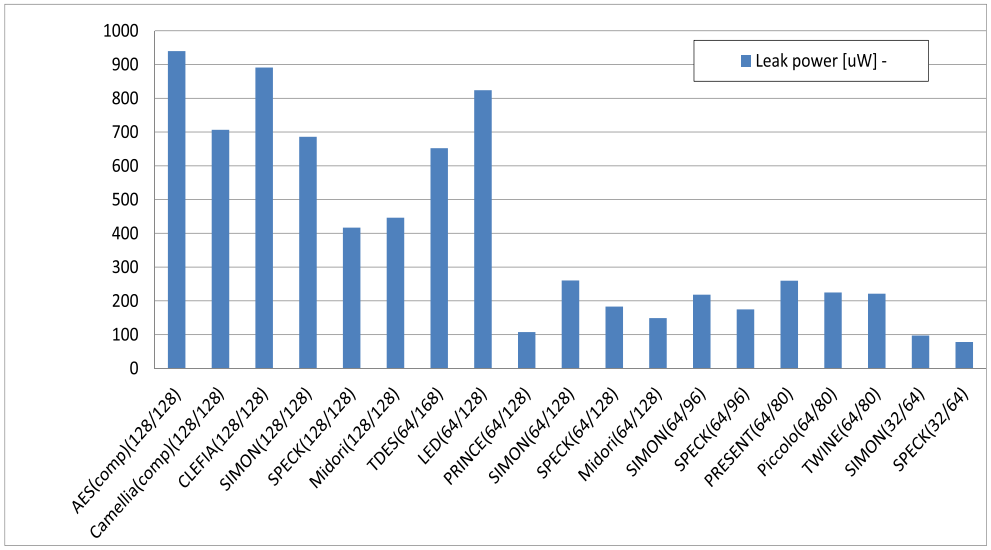

Figure 3.5: Leak power of Enc, Unrolled Implementation

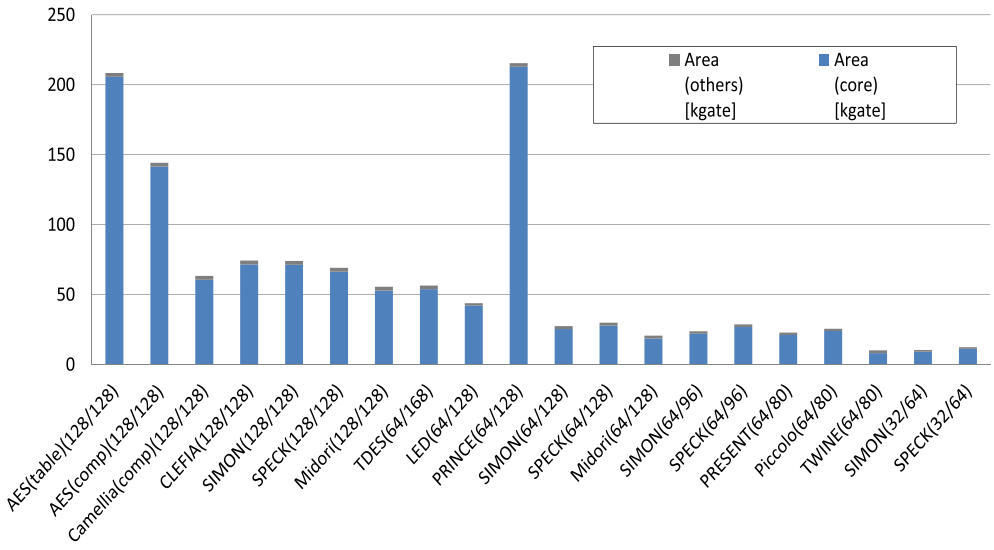

Figure 3.6: Area of Enc/Dec, Unrolled Implementation

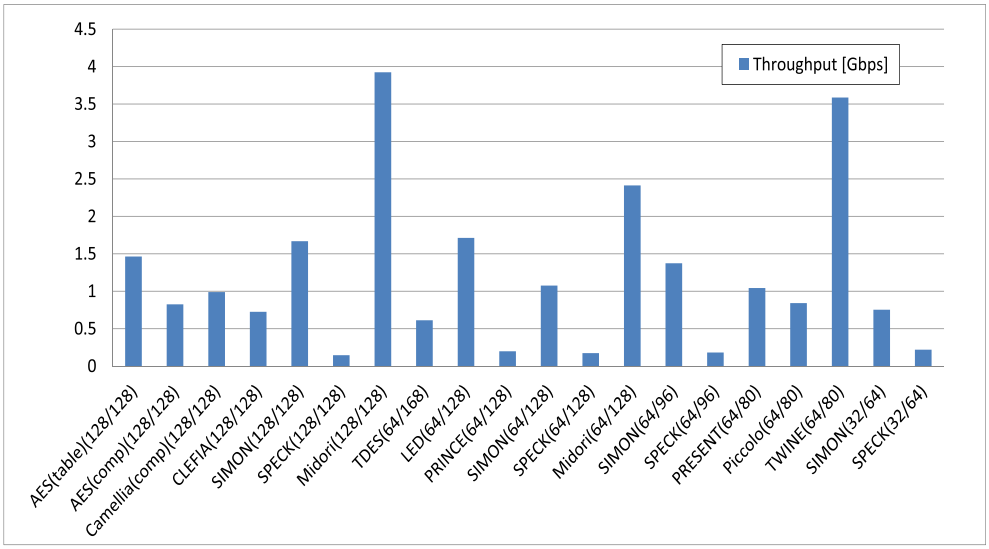

Figure 3.7: Throughput of Enc/Dec, Unrolled Implementation

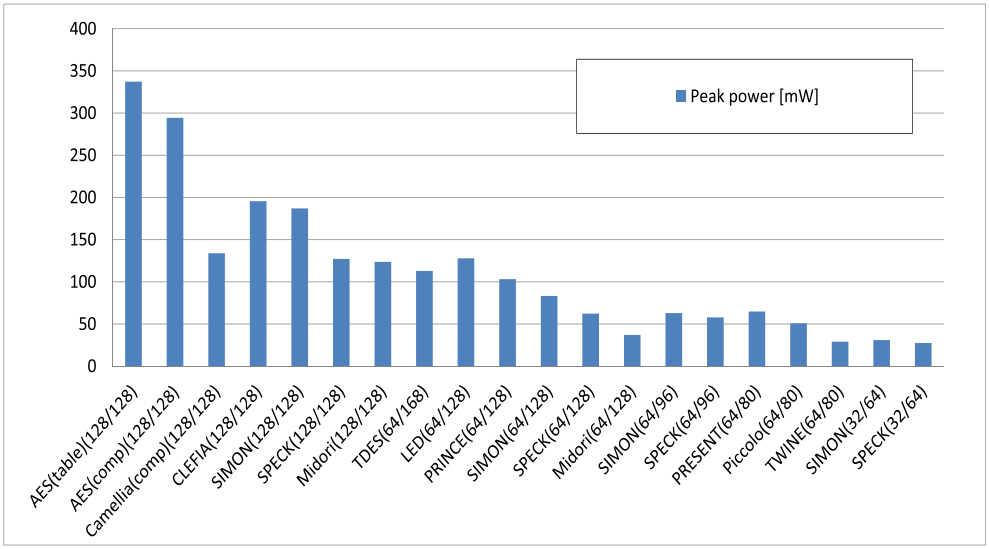

Figure 3.8: Peak power of Enc/Dec, Unrolled Implementation

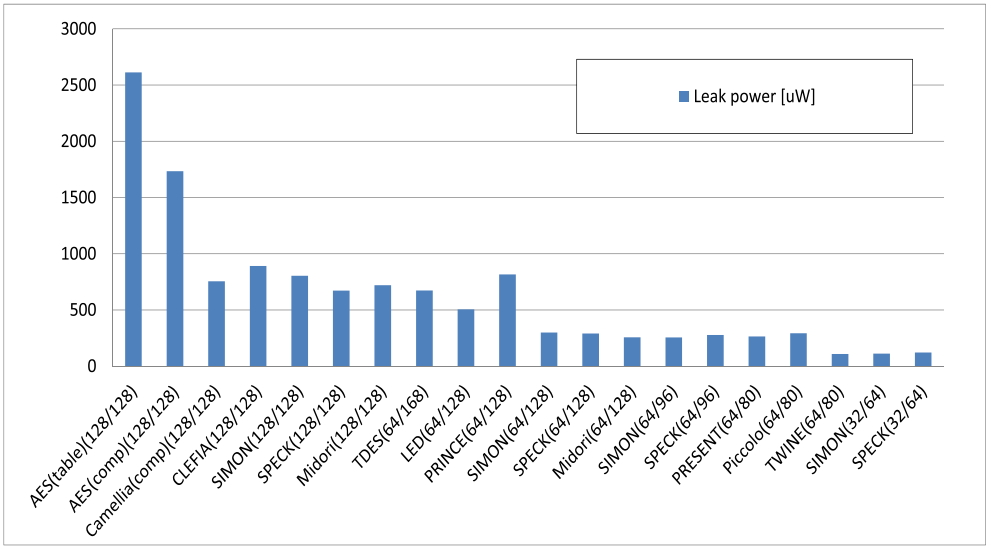

Figure 3.9: Leak power of Enc/Dec, Unrolled Implementation

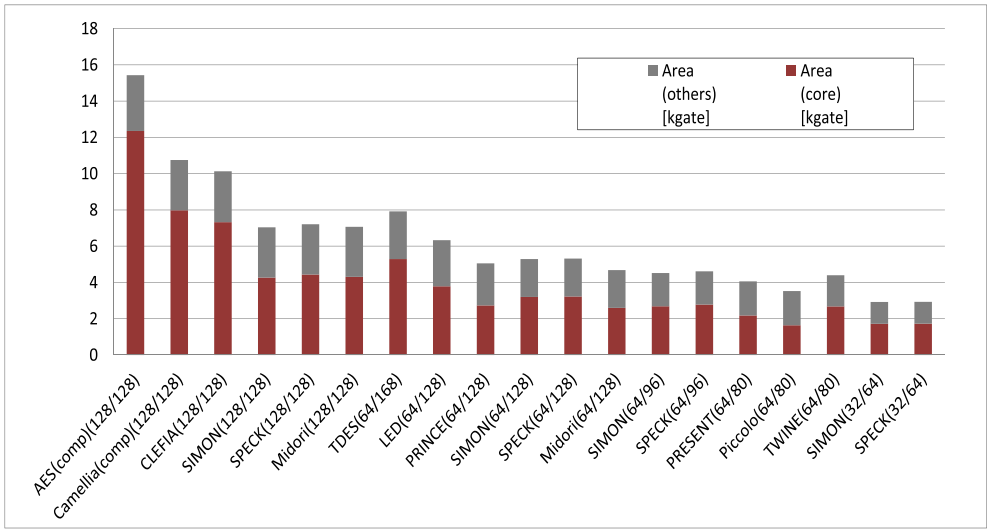

Figure 3.10: Area of Enc, Round Implementation

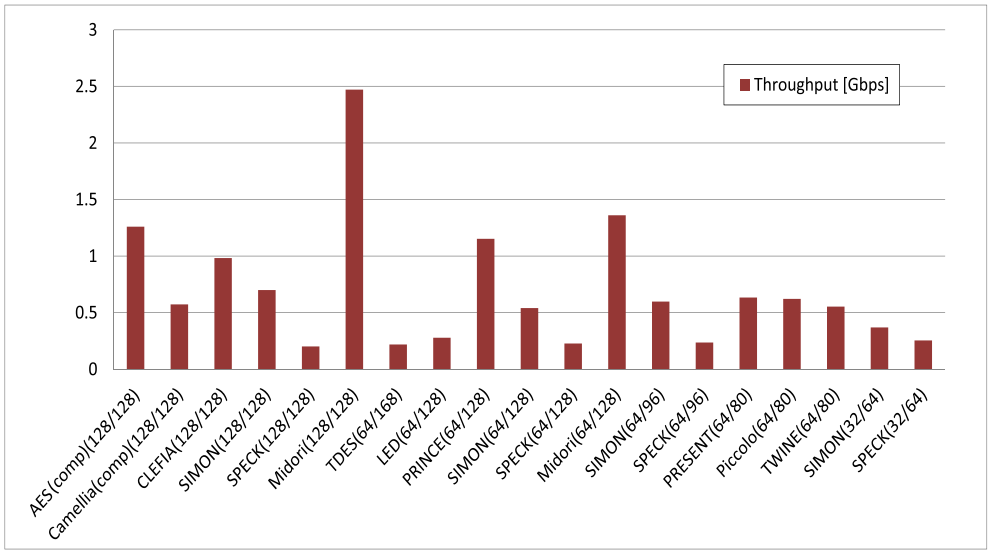

Figure 3.11: Throughput of Enc, Round Implementation

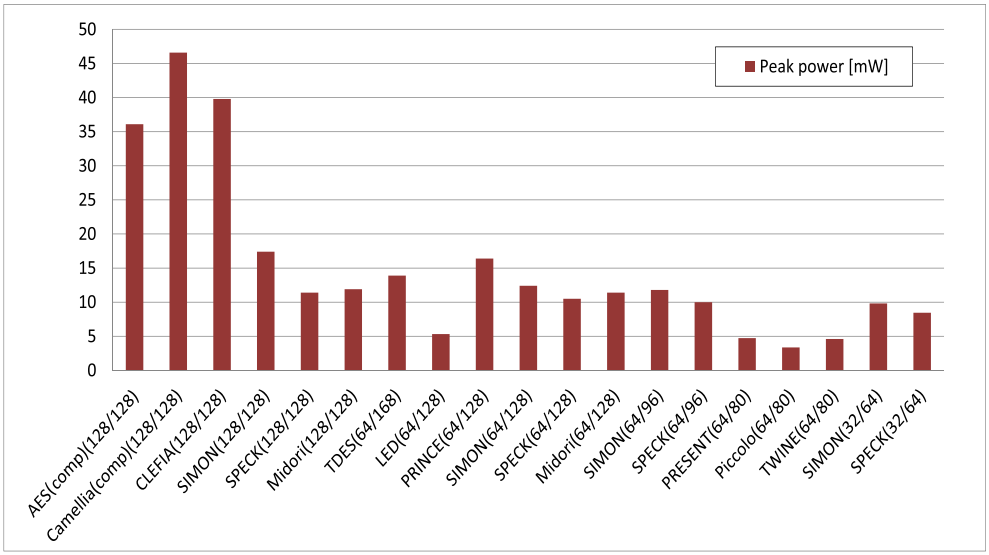

Figure 3.12: Peak power of Enc, Round Implementation

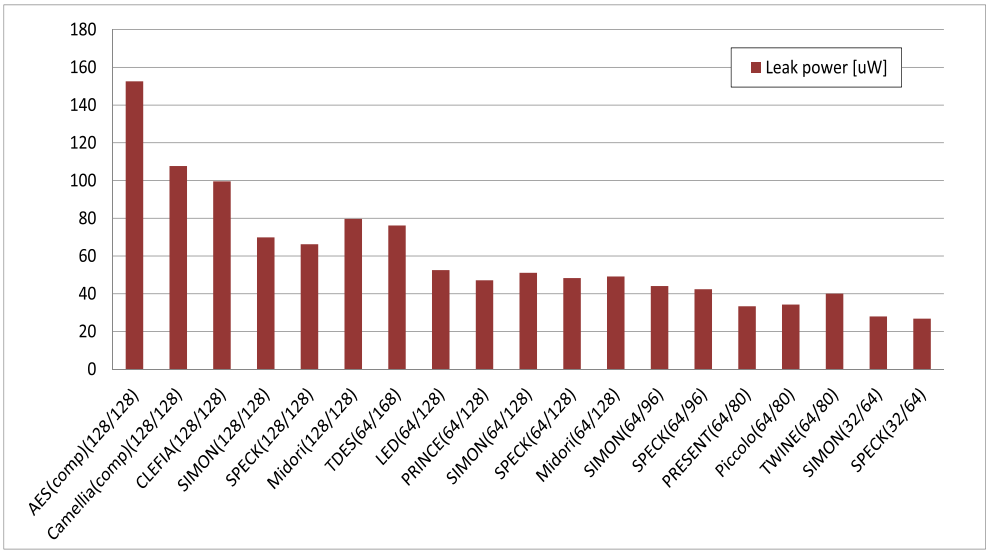

Figure 3.13: Leak power of Enc, Round Implementaiton

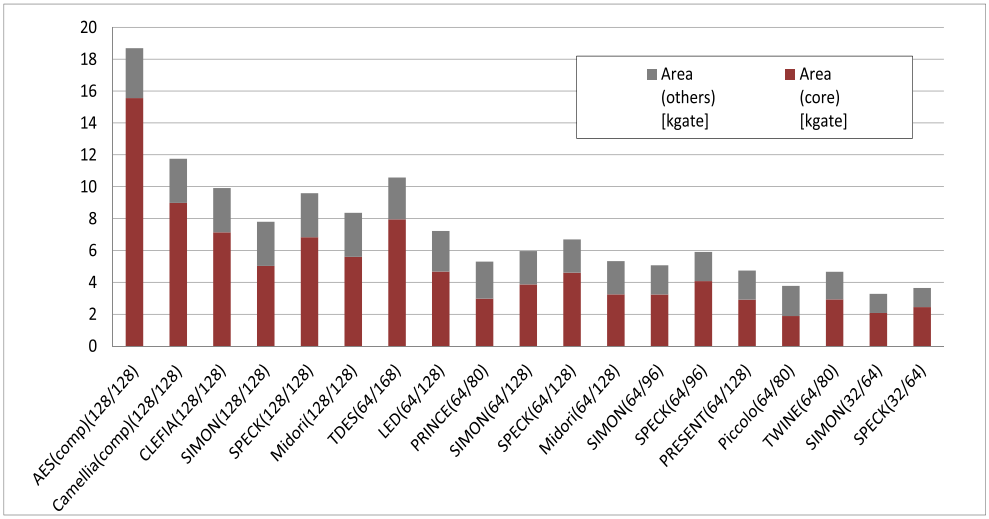

Figure 3.14: Area of Enc/Dec, Round Implementation

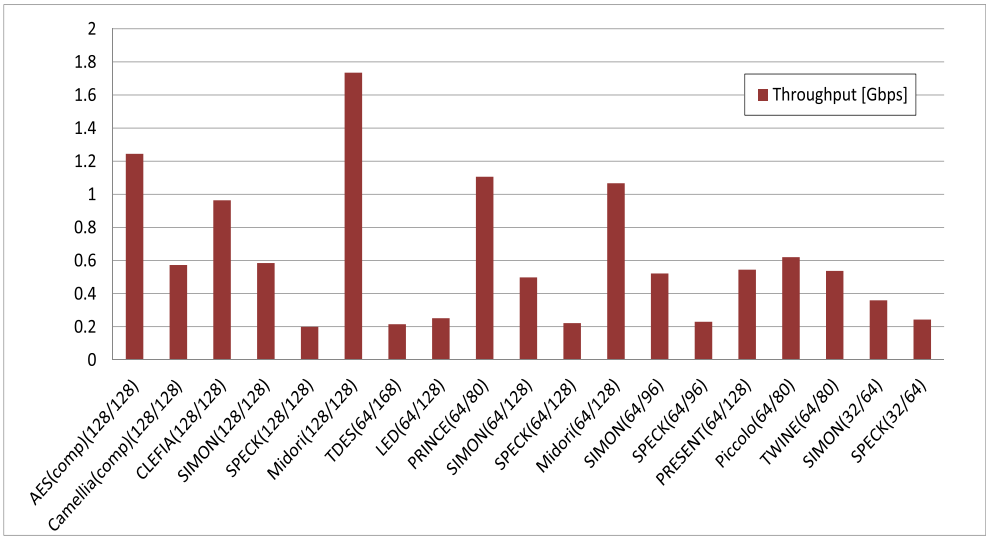

Figure 3.15: Throughput of Enc/Dec, Round Implementation

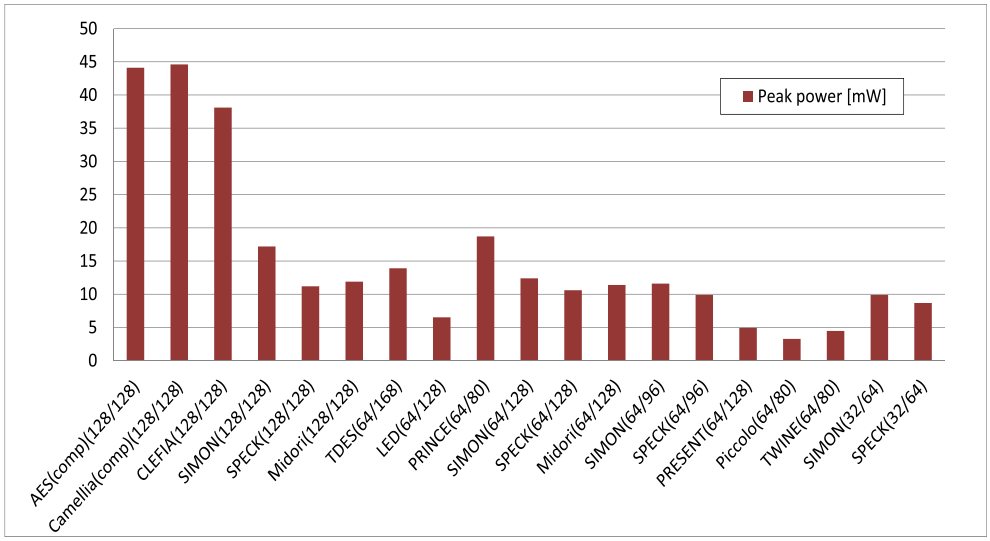

Figure 3.16: Peak power of Enc/Dec, Round Implementation

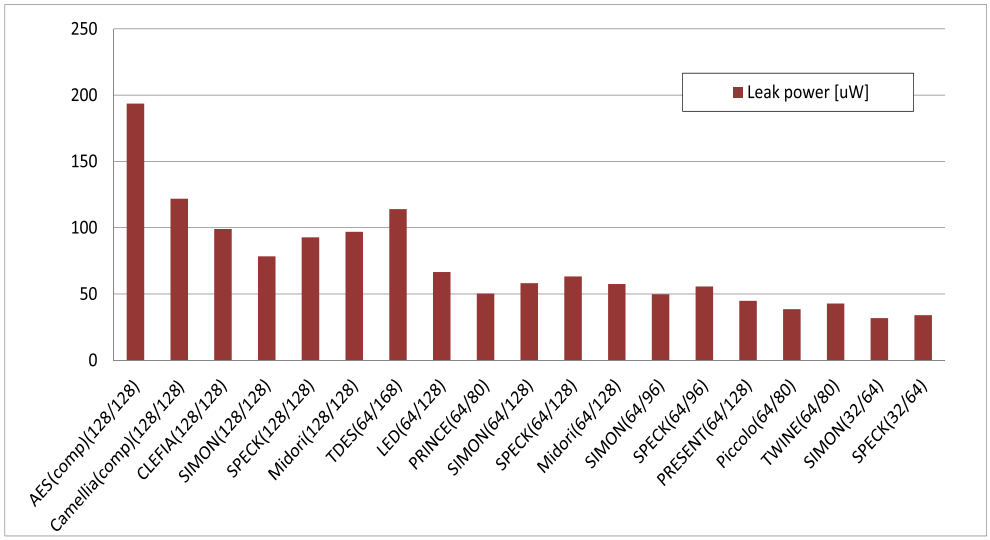

Figure 3.17: Leak power of Enc/Dec, Round Implementation

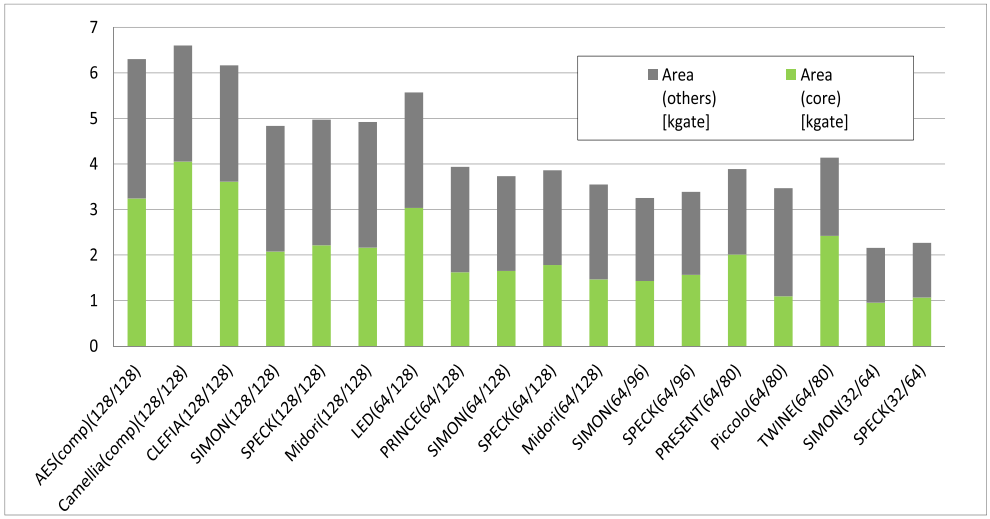

Figure 3.18: Area of Enc, Serial Implementation

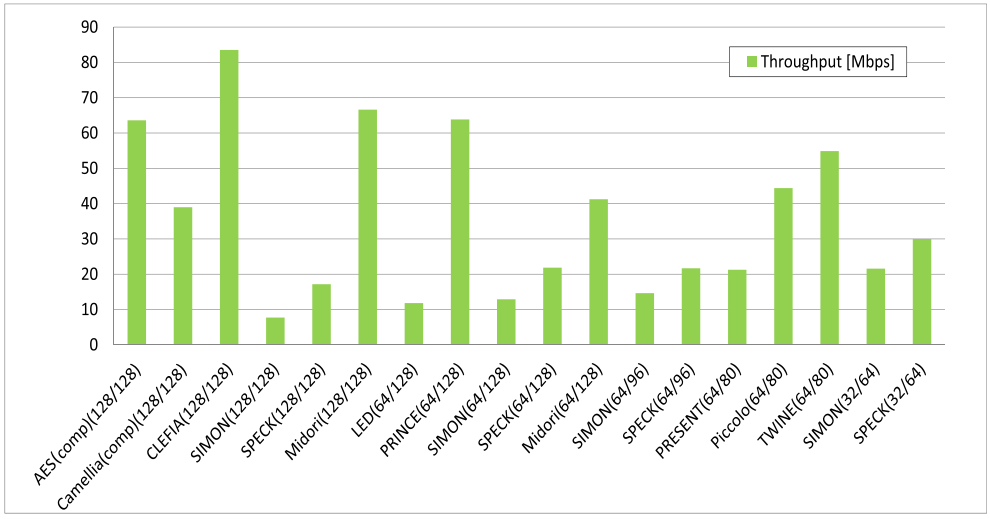

Figure 3.19: Throughput of Enc, Serial Implementation

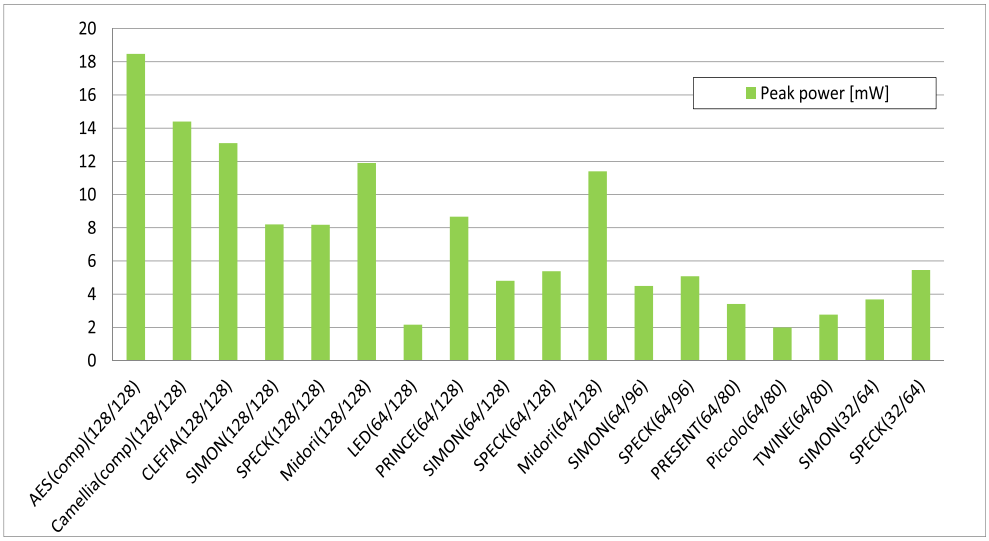

Figure 3.20: Peak power of Enc, Serial Implementation

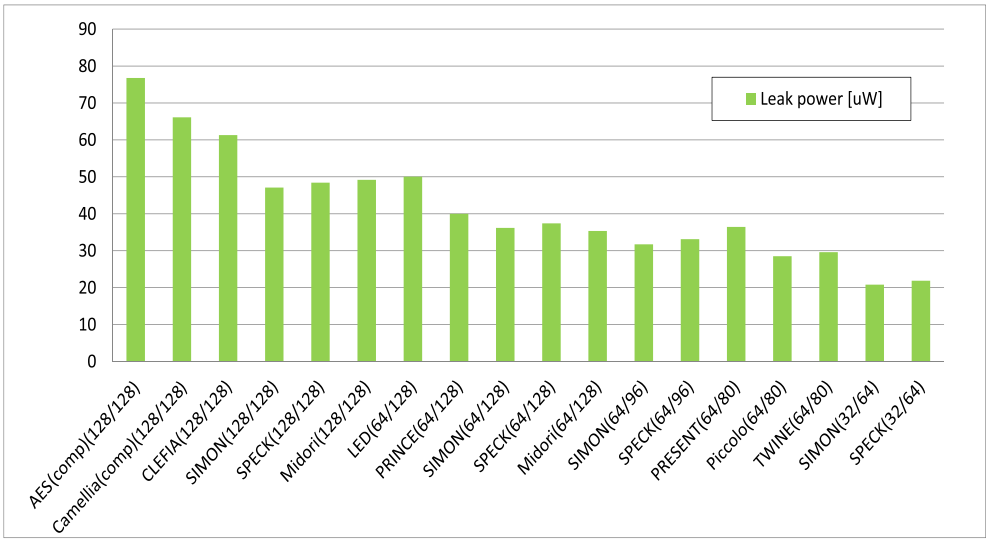

Figure 3.21: Leak power of Enc, Serial Implementation

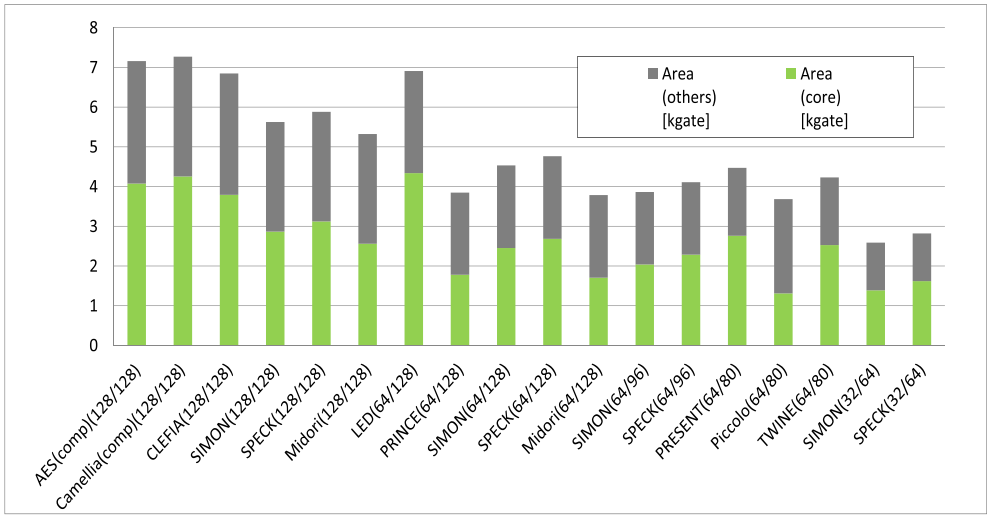

Figure 3.22: Area of Enc/Dec, Serial Implementation

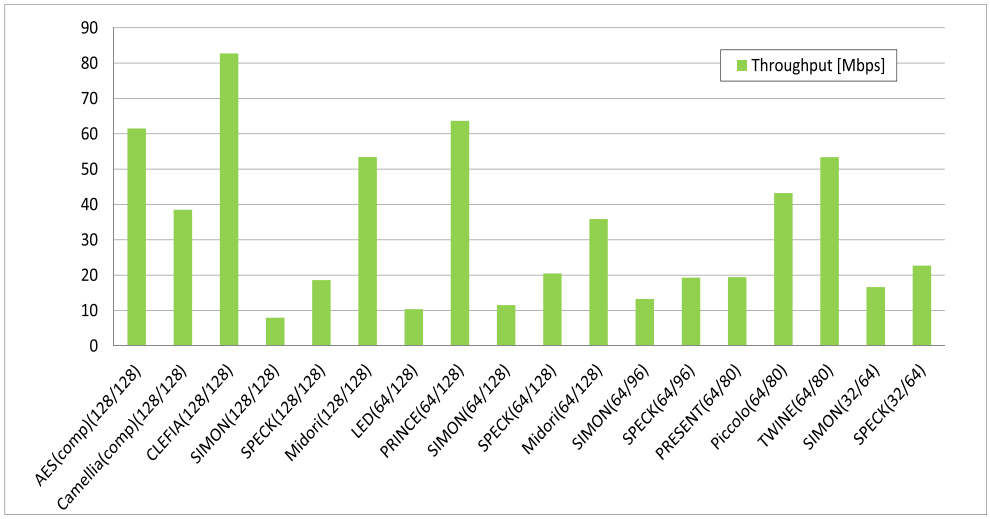

Figure 3.23: Throughput of Enc/Dec, Serial Implementation

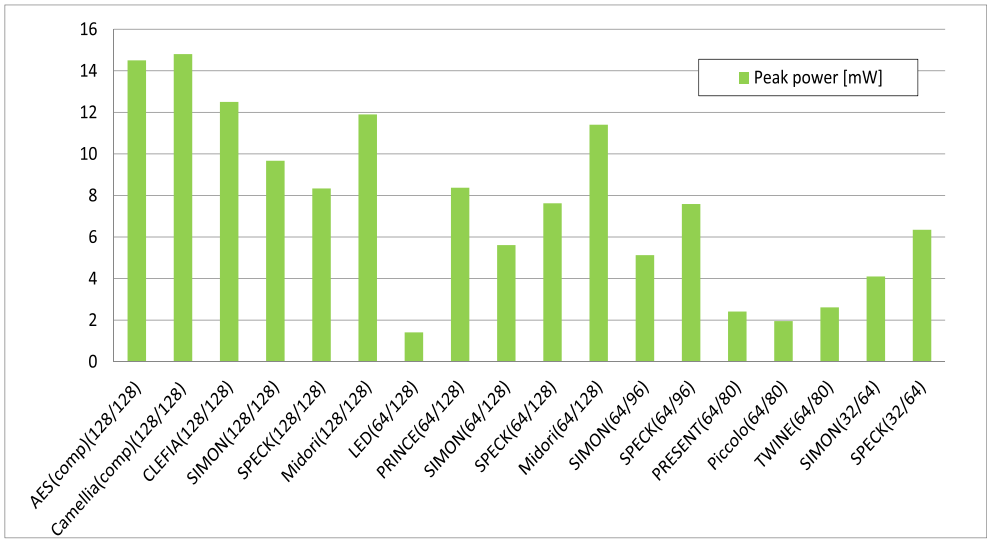

Figure 3.24: Peak power of Enc/Dec, Serial Implementation

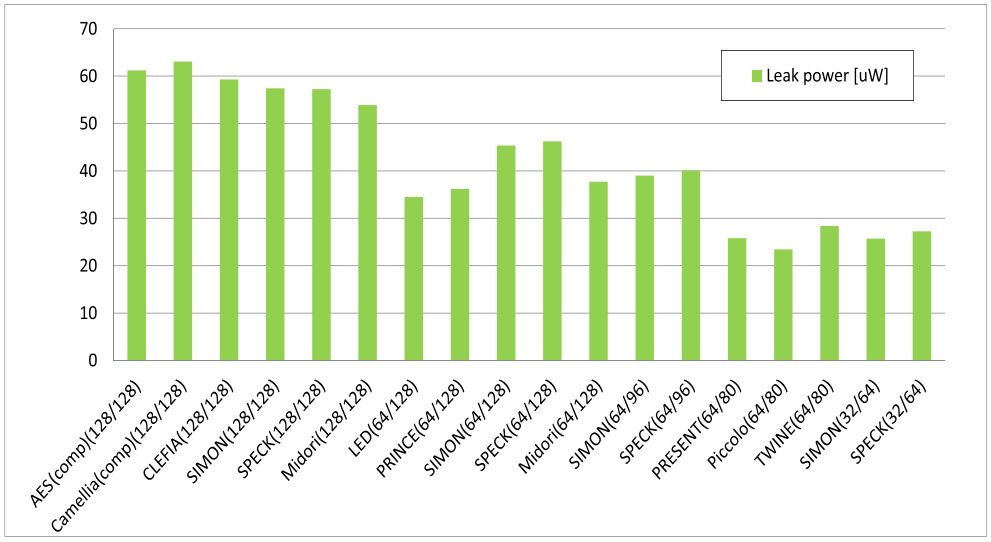

Figure 3.25: Leak power of Enc/Dec, Serial Implementation

3.1.1.2 Outline of Evaluation Method

This section outlines the evaluation method. The evaluation measured the amounts of circuit resource usage and the maximum operating frequencies of various types of lightweight cryptography using an open-source CMOS cell library. The implementation environment is shown in Table 3.9.

Table 3.9: Implementation Environment

Synthesis Tool	Design Compiler (Version G-2012.06-SP5)
Power Analysis Tool	PrimeTime PX (Version G-2012.06-SP3-2)
Optimization	Minimal Area
Library	NANGATE Open Cell Library (45-nm CMOS) `http://www.nangate.com/`
Delay Condition	NangateOpenCellLibrary_slow (Logical delay in worst conditions)

The functional properties of the logical circuit to be implemented is as follows:

F1. The evaluation is performed with a key length of 80 bits or more with minimum parameters. However, LED is evaluated with a key length of 128 bits for which a test vector is provided.
F2. An encryption circuit and an encryption/decryption circuit are implemented.
F3. provides compact and low-power APB bus interface [13] for the use as coprocessor.

Figure 3.26 is a block diagram of the APB bus and the encryption circuit. The meaning of each signal in the figure is listed in the legend below.

* PCLK:Bus clock signal
* PRESETn:Asynchronous reset signal
* PADDR[31:0]:Address signal
* PSEL:IP selection signal
* PENABLE:Enable signal
* PWRITE:Write signal (1: Write: 0: Read)
* PWDATA[31:0]:Write data
* PRDATA[31:0]:Read data

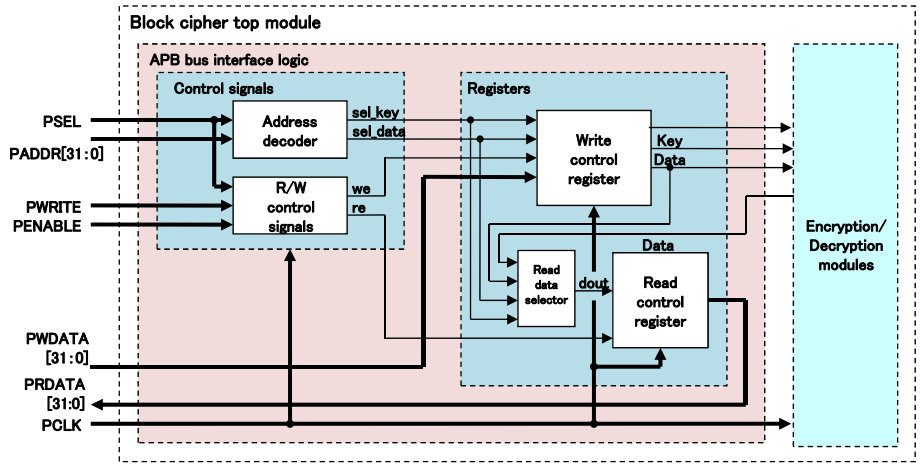

Figure 3.26: APB Bus and Encryption Circuit

APB signal specifications not used in this evaluation include PSTRB[3:0] (write strobe signal) and PREADY (APB extended transfer signal).

The design policy will be discussed in the following:

P1. There are three types of implementation for each algorithm: (i) the typical round-based implementation, (ii) the unrolled implementation in which the process is completed in a single cycle, and (iii) the serial implementation where the size of the data path is the size of the S-box.
P2. Key scheduling is implemented on-the-fly.
P3. An optimization that directly instantiates the CMOS cell library is not performed; a synthesizable description independent of the library is used.

For the logical circuits designed based on the above policy, the number of cycles, maximum operating frequencies (maximum delays), throughputs, number of gates, peak power, and leakage power were evaluated.

3.1.2 Evaluation of Software Implementations

This section compares the results of measuring the speeds of software implementation of the lightweight block cipher under a restricted memory resource on an embedded microprocessor.

3.1.2.1 Performance Evaluation

In this subsection, the results of implementing nine block ciphers on RL78, a 16-bit microprocessor from Renesas Electronics, will be presented. In the category column of the table below, ROM-Min indicates the implementation with the minimum ROM size and (n, m) the implementation with a ROM size of n bytes (n = 512 or 1,024) and a RAM size of m bytes (m = 64 or 128). Fast is an implementation that aims a higher processing speed with a ROM size of around 2 KB or less. All implementation types performed key scheduling in parallel with encryption/decryption (on-the-fly key scheduling).

AES

Table 3.10 shows the result of implementing AES on RL78. Implementation with a RAM size of 128 bytes was omitted because 64 bytes of RAM is sufficient when only encryption is implemented. In the implementation with both encryption and decryption, the S-box itself consumed 512 bytes of ROM; therefore, only the 1,024-byte category was implemented.

In general implementation, the more severe the memory size restrictions, the greater the number of loops in a round. The rightmost column shows the MixColumns implementation that greatly affects the code size for reference. M4 had an independent code for each of the four matrix multiplications of MixColumns. M1 had a code for only a single matrix multiplications that is looped four times in each round to execute MixColumns. MQ had a code for a single row of the matrix and it 16 times in a nested loop to perform one round of MixColumns calculation.

For the minimization of ROM [20], a ROM size of 430 bytes was achieved for encryption, which is the smallest as far as we know. When implementing AES encryption/decryption, a ROM size of 1,024 bytes was a great restriction, and the performance degradation because of a heavy use of loops was inevitable. This is why in most existing implementations with ATtiny, the size of the ROM is 1,500 bytes or greater. To speed up the processing time, almost all loops can be unrolled with 2 KB of ROM. Therefore, the maximum performance of AES on the RL78 processor may be approximately 3,500 cycles/block.

Table 3.10: AES Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed	Method
AES (E)	ROM-Min	430	66 + 14	$8753n$	-	MQ
AES (E)	(512,64)	510	48 + 10	$5302n$	-	M1
AES (E)	(1024,64)	926	48 + 8	$3554n$	-	M4
AES (ED)	(1024,64)	1020	48 + 14	$8193n$	$9719n$	M1
AES (ED)	(1024,128)	1020	66 + 14	$6946n$	$1380+8490n$	M1
AES (ED)	Fast	2044	50 + 10	$3554n$	$753+5527n$	M4

Camellia

Table 3.11 shows the result of implementing Camellia on RL78. Camellia includes many 128-bit rotating and shifting operations. However, since there are no rules for the number of rotations, the ROM size increases. In addition, the sizes of the FL function and Σ constant are not small, and with only a single S-box, the minimum ROM size was 749 bytes. To implement both encryption and decryption, the subroutines required to shrink the code size resulted in an increase in the number of stacks used, and thus it was unable to satisfy a RAM size of 64 bytes.

However, when only encryption was implemented under a 2-KB ROM size condition, a speed similar to AES was obtained. Therefore, if more ROM is available, Camellia, which is a Feistel cipher, is expected to be faster than AES in decryption. Thus, with a relatively large amount of memory, Camellia exhibits high performance.

In the implementation, the code implementing only encryption other than ROM-Min had a dedicated group of the required rotating and shifting subroutines inside. The code implementing both encryption and decryption included only 8-bit rotating and shifting routines and 1-bit rotating and shifting routines and realized a necessary number of bits of rotation and shifting online. The ROM-Min implementation had only a 1-bit rotating and shifting routine and similarly realized a necessary number of bits of rotation and shifting online. The rightmost column indicates the number of bits of rotation and shifting performed by the routine in each type of implementation for reference.

Table 3.11: Camellia RL78 Implementation

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed	Method
Camellia (E)	ROM-Min	749	56 + 16	$58382n$	-	1
Camellia (E)	(1024,64)	1024	54 + 10	$889+4709n$	-	17,15
Camellia (E)	(1024,128)	1018	56 + 14	$884+4520n$	-	17,15
Camellia (E)	Fast	1995	66 + 12	$740+3638n$	-	34,30,17,15
Camellia (ED)	(1024,128)	1021	58 + 22	$3034+25470n$	$3907+25498n$	8,1

CLEFIA

Table 3.12 shows the implementation of CLEFIA on RL78. Since CLEFIA consumes a large amount of memory to store an intermediate key in the key scheduling part, it cannot be realized with a RAM size of 64 bytes. In addition to the fact that there are two pieces of S-box and MixColumns, the constant consumes 384 bytes, which necessitates a large ROM size. Therefore, it is difficult to implement this method with a ROM size of 512 bytes.

However, one of the two pieces of the S-box and the constants can be dynamically generated during execution. In the cases of implementing ROM-Min and implementing both encryption and decryption, these items are actually created dynamically. When only encryption is implemented with (1,024, 128), only the constants were dynamically generated and the two S-boxes were stored in ROM. Therefore, 800 bytes of ROM were required in the minimum configuration. The rightmost column shows which implementation method was used as a reference. S indicates that S-box was generated dynamically, and C indicates that a constants were generated dynamically.

Table 3.12: CLEFIA Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed	Method
CLEFIA (E)	ROM-Min	800	58 + 22	$23854n$	-	SC
CLEFIA (E)	(1024,128)	1021	58 + 16	$12351n$	-	C
CLEFIA (E)	Fast	1681	74 + 14	$3010+5899n$	-	
CLEFIA (ED)	(1024,128)	1018	90 + 26	$19879n$	$20797n$	SC

TDES

Table 3.13 shows the implementation of TDES on RL78. TDES can be implemented with a RAM size of 64 bytes. However, since the algorithm mainly executes irregular bit operations, the tables for the S-box and bit indices occupy 400 bytes or more of ROM. Therefore, the entire program cannot be implemented in a ROM size of 512 bytes.

In the implementation with 1,832 bytes of ROM, since most speed-related parts of the algorithm are unrolled, the almost maximum speed achievable was obtained, and the speed of TDES was approximately 1/15 that of AES.

Table 3.13: TDES Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed
TDES (E)	ROM-Min	958	50 + 14	$111183n$	-
TDES (E)	(1024,64)	1024	50 + 14	$77708n$	-
TDES (E)	Fast	1832	50 + 8	$26697n$	-
TDES (ED)	(1024,64)	1019	50 + 14	$87879n$	$87543n$

LED

Table 3.14 shows the result of implementing the LED on RL78. The LED has a similar structure to AES with a 4-bit word. In the case of ciphers having such a structure, it is necessary to divide an 8-bit plaintext or key into two pieces of 4-bit data at some point. A trade-off exists between memory size and speed depending on the stage (before encryption or during execution) in which the division is performed. Transferring the 4-bit S-box from ROM to RAM in advance increases the speed in return for an increase in code and RAM sizes. This is because, on RL78, it takes four cycles to read data from the ROM, but only one cycle to read data from the RAM. Another trade-off is possible by performing a doubling operation on GF(16) during execution or on a table in RAM.

Under these trade-offs, it is a complicated task to determine the fastest solution under the given memory-size conditions. The rightmost column of the table lists symbols indicating which types of implementation were used for reference. S means that the S-box was transferred to the RAM, G means that the table used for the doubling operation on GF(16) was transferred to the RAM, T indicates that the plaintext (ciphertext) was initially divided into 4-bit units, and K that the key was initially divided into 4-bit units.

In the implementation of encryption only with (1,024, 128), since all major parts were unrolled, the maximum speed for RL78 was almost obtained.

Table 3.14: LED Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed	Method
LED (E)	ROM-Min	298	54 + 12	$36779n$	-	T
LED (E)	(512,64)	510	54 + 10	$18055n$	-	S
LED (E)	(512,128)	504	100 + 12	$17207n$	-	SGTK
LED (E)	(1024,64)	956	54 + 10	$15899n$	-	S
LED (E)	(1024,128)	1023	100 + 8	$14478n$	-	SGTK
LED (ED)	(512,64)	508	54 + 10	$35726n$	$32219n$	T
LED (ED)	(512,128)	508	54 + 14	$33950n$	$31787n$	T
LED (ED)	(1024,64)	1007	54 + 10	$17717n$	$17788n$	S
LED (ED)	(1024,128)	1023	100 + 8	$16753n$	$17472n$	SGT

PRINCE

Table 3.15 shows the result of implementing PRINCE on RL78. The implementation with a RAM size of 128 bytes aimed to speed up processing by transferring a total of 32 bytes for the two pieces of the S-box to the RAM, which is indicated by the symbol S in the rightmost column of the table. The high-speed implementation aimed to increase the processing speed by using two 256-byte tables to parallelize two pieces of S-box, indicated by the symbol S8.

PRINCE does very little key schedule processing, and both encryption and decryption can be realized in similar processes. However, since the number of constants is large and overhead exists in the matrix operation code, the size of the ROM in the minimum configuration is larger than the other lightweight 64-bit block ciphers.

Table 3.15: PRINCE Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed	Method
PRINCE (E)	ROM-Min	424	42 + 22	$9905n$	-	
PRINCE (E)	(512,64)	512	42 + 12	$7611n$	-	
PRINCE (E)	(512,128)	511	74 + 12	$7320n$	-	S
PRINCE (E)	(1024,64)	1019	42 + 12	$4928n$	-	
PRINCE (E)	(1024,128)	1020	74 + 12	$4541n$	-	S
PRINCE (E)	Fast	1789	42 + 8	$3307n$	-	S8
PRINCE (ED)	(512,64)	511	44 + 20	$9925n$	$10050n$	
PRINCE (ED)	(512,128)	511	76 + 24	$9541n$	$9810n$	S
PRINCE (ED)	(1024,64)	1007	42 + 12	$5117n$	$5214n$	
PRINCE (ED)	(1024,128)	1017	74 + 12	$4745n$	$4832n$	S

PRESENT

Table 3.16 shows the result of implementing PRESENT on RL78. Since PRESENT has a regular structure, it can be implemented in a very compact code. A minimum ROM implementation size of 164 was achieved for encryption. This is much smaller than and superior in speed to [23] and [20].

In any method, PRESENT is basically implemented by shifting the data in an input register by one bit, moving one carry bit into an output register and repeating this simple process. Using the 16-bit instruction of RL78, the movement of the carry bit can be performed with a single instruction, which contributes to the reduction in code size.

In PRESENT, size and speed trade-offs exist between how to create tables and whether to transfer their contents to RAM. In the rightmost column of the table, Sn-m means that the implementation had n 16-bit tables in ROM and transferred m out of n tables to RAM. S8 indicates the use of two parallel 256-byte tables.

The (1,024, 64) implementation of encryption completely unrolled a single stage and was assumed to indicate the speed limit of RL78. The PRESENT algorithm was slow but had the advantage of small memory size.

Table 3.16: PRESENT Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed	Method
PRESENT (E)	ROM-Min	164	38 + 22	$93412n$	-	S1-0
PRESENT (E)	(512,64)	491	44 + 16	$11344n$	-	S2-1
PRESENT (E)	(512,128)	499	60 + 16	$10560n$	-	S2-2
PRESENT (E)	(1024,64)	952	28 + 10	$9007n$	-	S8
PRESENT (ED)	(512,64)	512	42 + 18	$16924n$	$3736+19131n$	S2-0
PRESENT (ED)	(512,128)	509	74 + 18	$16407n$	$3643+18614n$	S2-2
PRESENT (ED)	(1024,64)	989	38 + 18	$12048n$	$1996+12367n$	S4-0
PRESENT (ED)	(1024,128)	1003	102 + 18	$10691n$	$1903+11010n$	S4-4

Piccolo

Table 3.17 shows the result of implementing Piccolo on RL78. The meaning of the symbols in the rightmost column of the table is the same as PRESENT. Since Piccolo can be implemented with a small amount of RAM, 64 bytes of RAM was sufficient in all categories.

Falling short of the level of PRESENT in the minimum implementation size, Piccolo was generally a fast algorithm.

Table 3.17: Piccolo Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed	Method
Piccolo (E)	ROM-Min	275	24 + 18	$12220n$	-	S1-0
Piccolo (E)	(512,64)	498	52 + 8	$5779n$	-	S2-2
Piccolo (E)	(1024,64)	1018	40 + 8	$4961n$	-	S8
Piccolo (E)	Fast	1172	40 + 8	$4636n$	-	S8
Piccolo (ED)	(512,64)	512	54 + 8	$6186n$	$6084n$	S2-2
Piccolo (ED)	(1024,64)	966	52 + 8	$5779n$	$5779n$	S2-2

TWINE

Table 3.18 shows the result of implementing TWINE on RL78. The meaning of the symbols in the rightmost column of the table is the same as PRESENT and Piccolo. The TWINE algorithm had little software overhead, which accommodated an extremely small code size. Similar to Piccolo, the implementation consumed little RAM, and a RAM size of 64 bytes served all categories.

For encryption only, the fastest implementation was realized in almost all categories with 512 bytes of ROM and 64 bytes of RAM. In terms of speed, TWINE achieved a similar level to Piccolo.

Table 3.18: TWINE Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed	Method
TWINE (E)	ROM-Min	232	52 + 8	$11043n$	-	S1-0
TWINE (E)	(512,64)	468	52 + 6	$4957n$	-	S1-1
TWINE (ED)	(512,64)	510	54 + 10	$6132n$	$2463+5570n$	S1-1
TWINE (ED)	(1024,64)	972	54 + 6	$4957n$	$1727+4892n$	S1-1

SIMON

Table 3.19 shows the result of implementing SIMON on RL78. The category "ROM-min" aims to minimize ROM size by introducing loops or subroutines for common modules. Because SIMON has the Feistel structure, the data randomization module is commonly used for encryption and decryption in the program supporting both encryption and decryption. The category "One" in Table 3.19 indicates the one-round implementation where the round function is unrolled and it is repeated the number of rounds. The data randomization and the key schedule modules are not shared, but the modules for the initialization and the key schedule are shared between encryption and decryption in the program supporting both encryption and decryption.

The category "Fast" aims a faster implementation by unrolling plural rounds and repeating it necessary times and by preventing common modules or subroutines. In the fast implementation of SIMON, the optimal number of rounds for unrolling is LCM$(2, m)$, where m = (key size) / (word size), because the data randomization of SIMON has a 2-round cycle, and the key schedule has an m-round cycle.

SPECK

Table 3.20 shows the result of implementing SPECK on RL78. The category "ROM-min" aims to minimize ROM size by introducing loops or subroutines for common modules. The data randomization and the key schedule modules share some common modules. The category "One" in Table 3.20 indicates the one-round implementation where the round function is unrolled and it is repeated the number of rounds. The data randomization and the key schedule modules are not shared, but the modules for the initialization and the key schedule are shared between encryption and decryption in the program supporting both encryption and decryption. The category "Fast" aims a faster implementation by unrolling plural rounds and repeating it necessary times and by preventing common modules or subroutines. In the fast implementation of SPECK, the optimal

Table 3.19: SIMON Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed
SIMON(32/64)(E)	ROM-Min	127	20 + 8	$3706n$	-
SIMON(32/64)(E)	One	171	20 + 6	$2480n$	-
SIMON(32/64)(E)	Fast	413	20 + 6	$1872n$	-
SIMON(64/96)(E)	ROM-Min	112	32 + 8	$7354n$	-
SIMON(64/96)(E)	One	243	32 + 6	$4598n$	-
SIMON(64/96)(E)	Fast	859	32 + 6	$3450n$	-
SIMON(64/128)(E)	ROM-Min	128	40 + 8	$9094n$	-
SIMON(64/128)(E)	One	303	40 + 6	$6404n$	-
SIMON(64/128)(E)	Fast	753	40 + 6	$4688n$	-
SIMON(128/128)(E)	ROM-Min	111	48 + 8	$21050n$	-
SIMON(128/128)(E)	One	415	48 + 6	$13148n$	-
SIMON(128/128)(E)	Fast	629	48 + 6	$10836n$	-
SIMON(32/64)(ED)	ROM-Min	273	20 + 14	$4227n$	$6586n$
SIMON(32/64)(ED)	One	310	30 + 10	$2777n$	$4473n$
SIMON(32/64)(ED)	Fast	1035	20 + 6	$1872n$	$3069n$
SIMON(64/96)(ED)	ROM-Min	244	32 + 14	$8035n$	$12063n$
SIMON(64/96)(ED)	One	436	32 + 10	$4985n$	$7559n$
SIMON(64/96)(ED)	Fast	1888	32 + 6	$3450n$	$5217n$
SIMON(64/128)(ED)	ROM-Min	277	40 + 14	$9807n$	$15408n$
SIMON(64/128)(ED)	One	546	40 + 10	$6809n$	$11057n$
SIMON(64/128)(ED)	Fast	1883	40 + 6	$4688n$	$7551n$
SIMON(128/128)(ED)	ROM-Min	203	48 + 14	$22147n$	$34005n$
SIMON(128/128)(ED)	One	506	48 + 10	$13767n$	$21023n$
SIMON(128/128)(ED)	Fast	1457	48 + 6	$10836n$	$16116n$

number of rounds for unrolling is $m - 1$, where m = (key size) / (word size), because the data randomization of SPECK has no cycle, and the key schedule has an $(m - 1)$-round cycle.

Table 3.20: SPECK Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed
SPECK(32/64)(E)	ROM-Min	96	24 + 8	$1817n$	-
SPECK(32/64)(E)	One	115	20 + 6	$1249n$	-
SPECK(32/64)(E)	ROM-Min	261	20 + 6	$1006n$	-
SPECK(64/96)(E)	ROM-Min	90	44 + 8	$6645n$	-
SPECK(64/96)(E)	One	185	32 + 6	$2335n$	-
SPECK(64/96)(E)	Fast	308	32 + 6	$2062n$	-
SPECK(64/128)(E)	ROM-Minl	89	52 + 8	$7448n$	-
SPECK(64/128)(E)	One	205	40 + 6	$2644n$	-
SPECK(64/128)(E)	Fast	451	40 + 6	$2122n$	-
SPECK(128/128)(E)	ROM-Min	71	67 + 8	$11432n$	-
SPECK(128/128)(E)	One	205	64 + 6	$5662n$	-
SPECK(128/128)(E)	Fast	309	48 + 6	$4793n$	-
SPECK(32/64)(ED)	ROM-Min	211	24 + 10	$2308n$	$3684n$
SPECK(32/64)(ED)	One	283	20 + 6	$1249n$	$1918n$
SPECK(32/64)(ED)	Fast	623	20 + 6	$1006n$	$1392n$
SPECK(64/96)(ED)	ROM-Min	211	44 + 10	$6600n$	$10837n$
SPECK(64/96)(ED)	One	447	32 + 6	$2335n$	$3585n$
SPECK(64/96)(ED)	Fast	742	32 + 6	$2062n$	$3088n$
SPECK(64/128)(ED)	ROM-Min	210	52 + 10	$7078n$	$11690n$
SPECK(64/128)(ED)	One	499	40 + 6	$2644n$	$4152n$
SPECK(64/128)(ED)	Fast	1087	40 + 6	$2122n$	$3165n$
SPECK(128/128)(ED)	ROM-Min	157	67 + 10	$11471n$	$18074n$
SPECK(128/128)(ED)	One	391	64 + 10	$5702n$	$8726n$
SPECK(128/128)(ED)	Fast	746	48 + 6	$4793n$	$7343n$

Midori

Table 3.21 shows the result of implementing Midori on RL78. Midori consists of 4-bit S-boxes, and it is implemented similarly to PRESENT. Therefore, the legend shown in the rightmost column of the table ("Sn-m" in the Method column) is the same as that in the tables for PRESENT, Piccolo, and TWINE. The fast implementation of Midori128 employs 8-bit S-box lookup tables and loop unrolling. In the ROM-Min implementation of Midori128, 8-bit S-boxes are calculated using 4-bit S-boxes, and loops and subroutines for common modules are employed.

Table 3.21: Midori Implementation on RL78

Algorithm	Category	ROM	RAM static+stack	Enc Speed	Dec Speed	Method
Midori64(E)	Fast	871	64 + 8	$6768n$	-	S2-0
Midori64(E)	ROM-Min	232	96 + 8	$16979n$	-	S2-0
Midori64(ED)	Fast	1576	64 + 10	$6768n$	$8360n$	S2-0
Midori64(ED)	ROM-Min	374	96 + 6	$17867n$	$27966n$	S2-0
Midori128(E)	Fast	1346	64 + 8	$9217n$	-	-
Midori128(E)	ROM-Min	560	64 + 8	$31794n$	-	-
Midori128(ED)	Fast	1745	64 + 10	$9217n$	$10166n$	-
Midori128(ED)	ROM-Min	605	64 + 6	$32495n$	$45586n$	-

3.1.2.2 Performance Comparison

Based on the implementation results, this subsection compares several criteria of the target algorithms.

Implementation with restricted memory sizes (Encryption only)

Figure 3.27 compares the speeds of implementations for encryption only using a ROM size of 1,024 bytes or less and a RAM size of 128 bytes or less. Figure 3.28 is similar to Figure 3.27 with TDES excluded to improve legibility. With these amounts of memory, AES was the fastest, followed by SPECK.

Figure 3.29 compares the speeds of implementations for encryption only using a ROM size of 1,024 bytes or less and a RAM size of 64 bytes or less. Figure 3.30 is similar to Figure 3.29 with TDES excluded to improve legibility. Except for CLEFIA, the results were similar to the implementation with ROM of 1,024 bytes or less and 128-byte RAM. CLEFIA cannot be implemented with 64 bytes of RAM and is indicated by a value of 0 in the figure.

Figure 3.31 compares the speeds of implementation for encryption only using a ROM size of 512 bytes or less and a RAM size of 128 bytes or less. Figure 3.32 compares the speeds of the same implementation with a ROM size of 512 bytes or less and a RAM size of 64 bytes or less. When the size of the ROM was reduced to 512 bytes or less, it was impossible to implement Camellia, TDES, and CLEFIA. Among the other algorithms, AES and SPECK achieve fast speed.

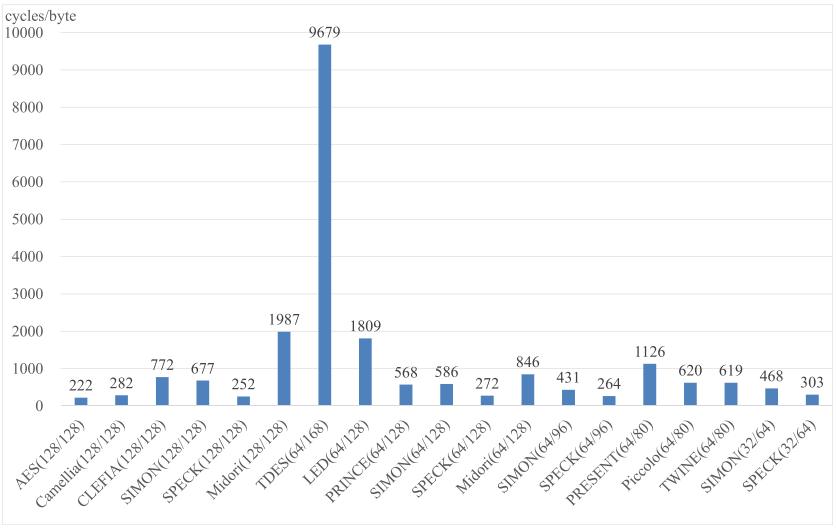

Figure 3.27: Speed with 1,024-byte ROM and 128-byte RAM

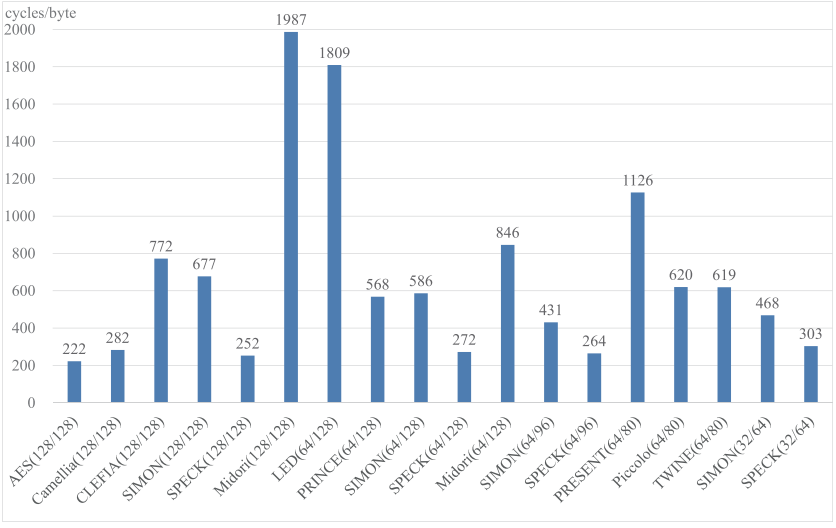

Figure 3.28: Speed with 1,024-byte ROM and 128-byte RAM (with TDES excluded)

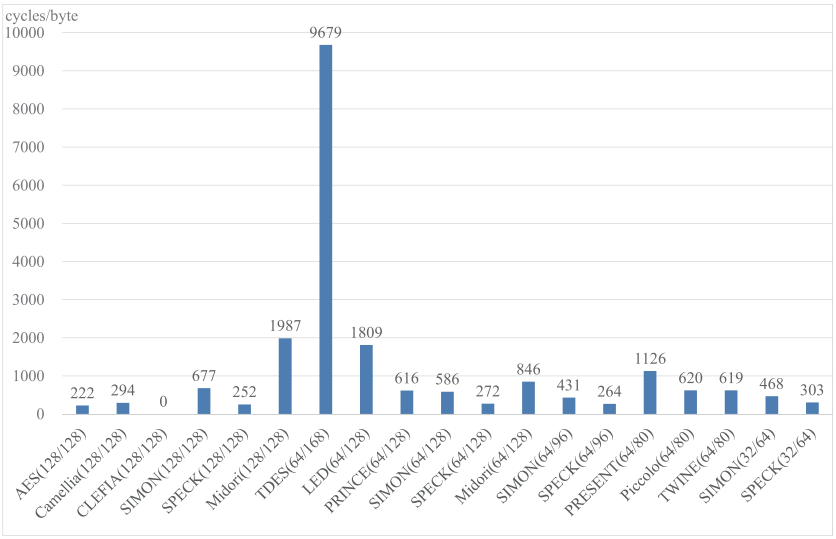

Figure 3.29: Speed with 1,024-byte ROM and 64-byte RAM

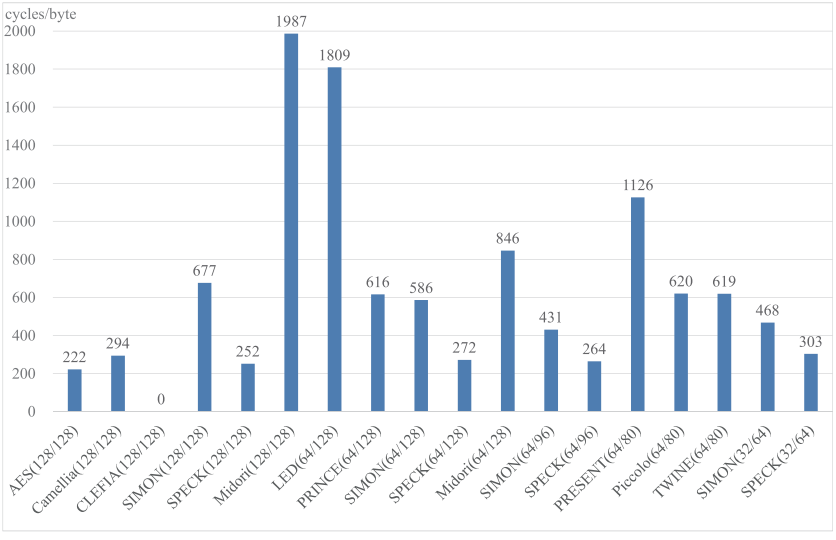

Figure 3.30: Speed with 1,024-byte ROM and 64-byte RAM (with TDES excluded)

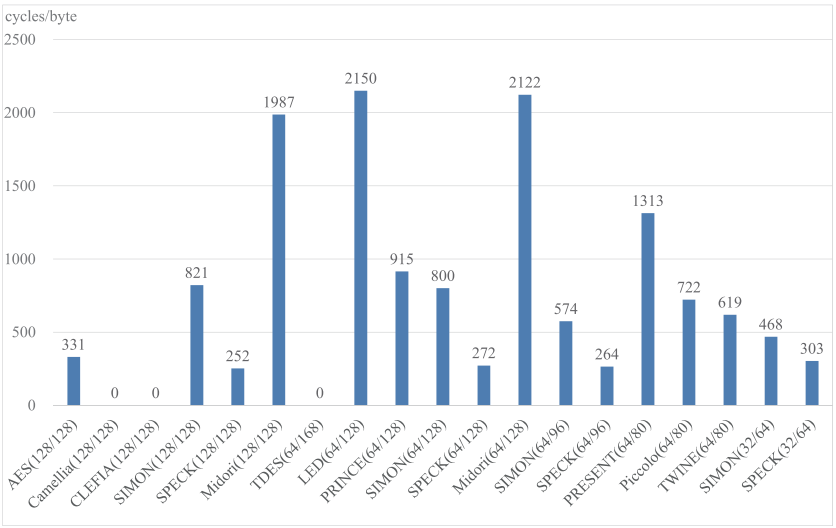

Figure 3.31: Speed with 512-byte ROM and 128-byte RAM

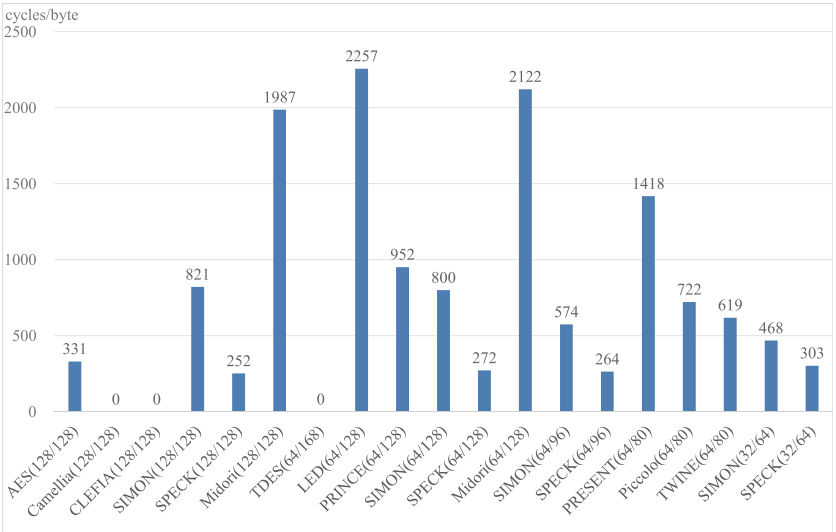

Figure 3.32: Speed with 512-byte ROM and 64-byte RAM

Implementation with restricted memory sizes (Encryption/Decryption)

Figure 3.33 compares the speeds of the implementation for both encryption and decryption using a ROM size of 1,024 bytes or less and a RAM size of 128 bytes or less. Figure 3.34 is similar to Figure 3.33 with TDES excluded to improve legibility. Also in this category, SPECK is the fastest, but the difference in speed from Piccolo, PRINCE, and TWINE is smaller than that in the category of implementation of encryption only.

Figure 3.35 compares the speeds of the implementation for both encryption and decryption using a ROM size of 1,024 bytes or less and a RAM size of 64 bytes or less. Figure 3.36 is similar to Figure 3.35 with TDES excluded to improve legibility. In this category, CLEFIA, Camellia cannot be implemented. The difference between the speed of AES and that of Piccolo, PRINCE, and TWINE was reduced, indicating that the speed of AES decreased because of insufficient memory resources. In this category, CLEFIA, Camellia, and Midori128 can not be implemented. The difference in speed between AES and Piccolo, PRINCE, and TWINE is small, because the speed of AES decreases due to lack of memory. Also in this category, SPECK is the fastest.

Figure 3.37 compares the speeds of the implementation for both encryption and decryption using a ROM size of 512 bytes or less and a RAM size of 128 bytes or less. Figure 3.38 compares the speeds of the same implementation with a ROM size of 512 bytes or less and a RAM size of 64 bytes or less. In this case, it was impossible to implement AES. As a result, only five algorithms survived. Among others, Piccolo and TWINE indicated similar speeds. In particular, SPECK achieves fast speeds which are not affected so much by restricted ROM sizes.

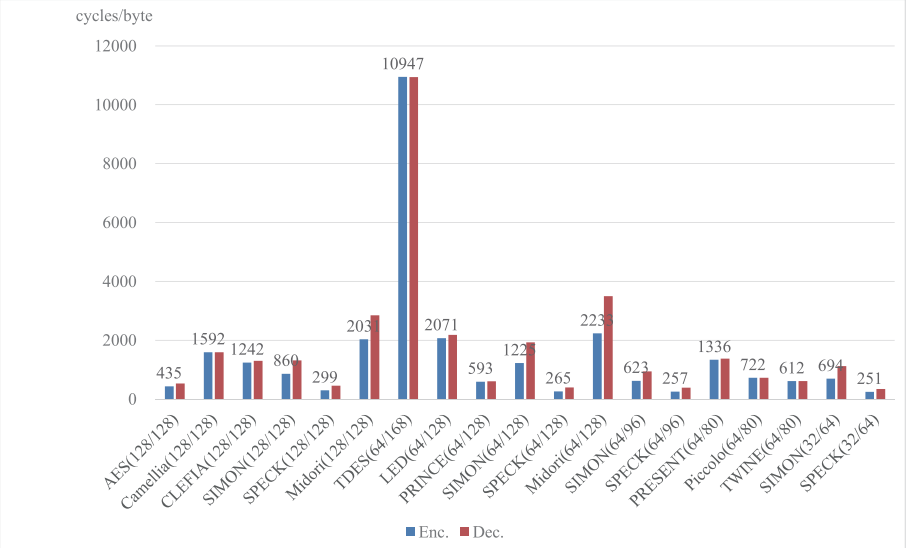

Figure 3.33: Speed with 1,024-byte ROM and 128-byte RAM

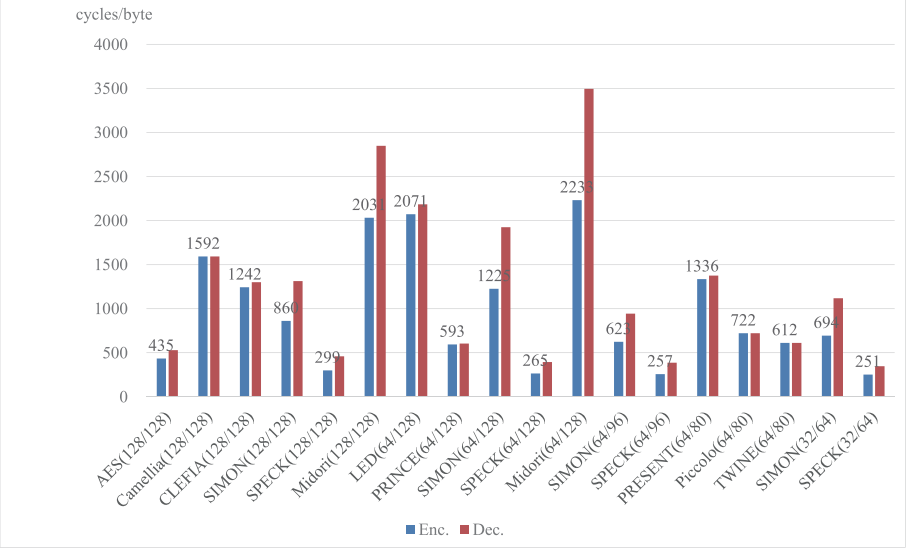

Figure 3.34: Speed with 1,024-byte ROM and 128-byte RAM (with TDES excluded)

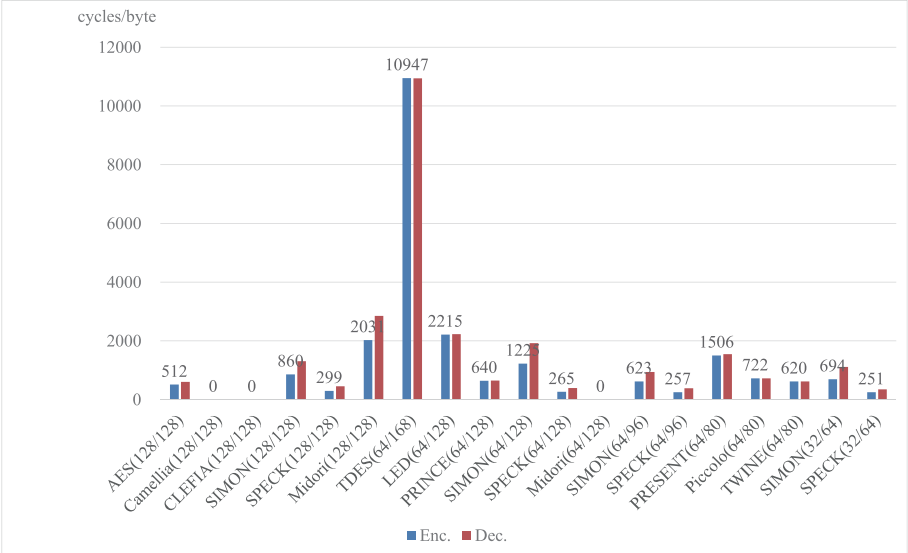

Figure 3.35: Speed with 1,024-byte 64-byte RAM

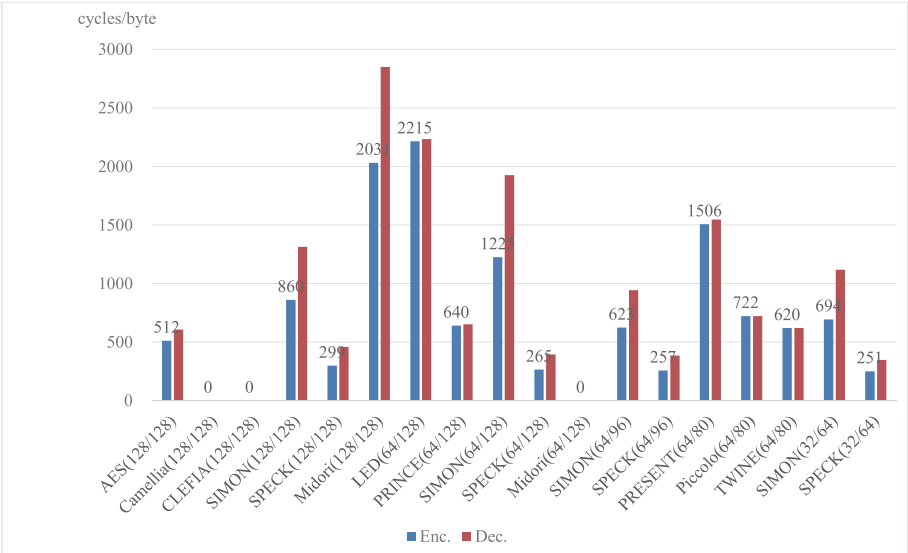

Figure 3.36: Speed with 1,024-byte ROM and 64-byte RAM (with TDES excluded)

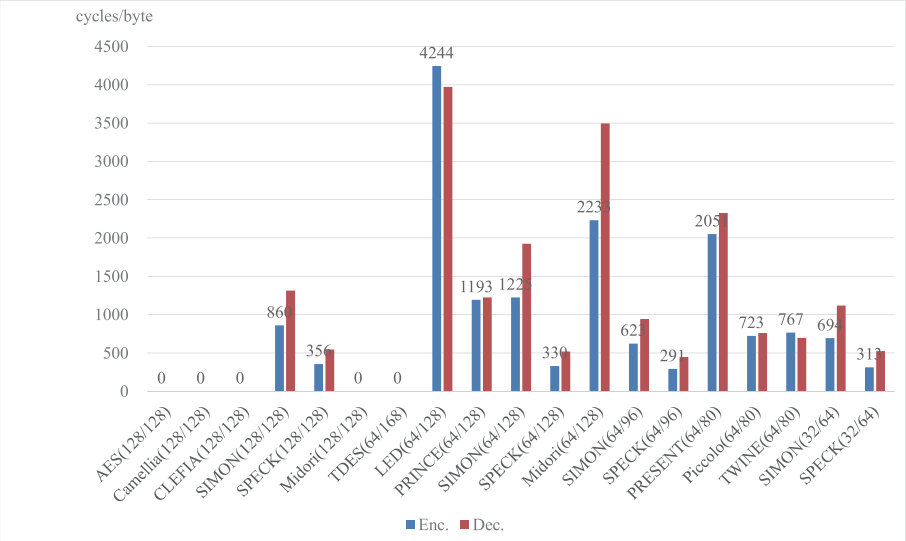

Figure 3.37: Speed with 512-byte ROM and 128-byte RAM

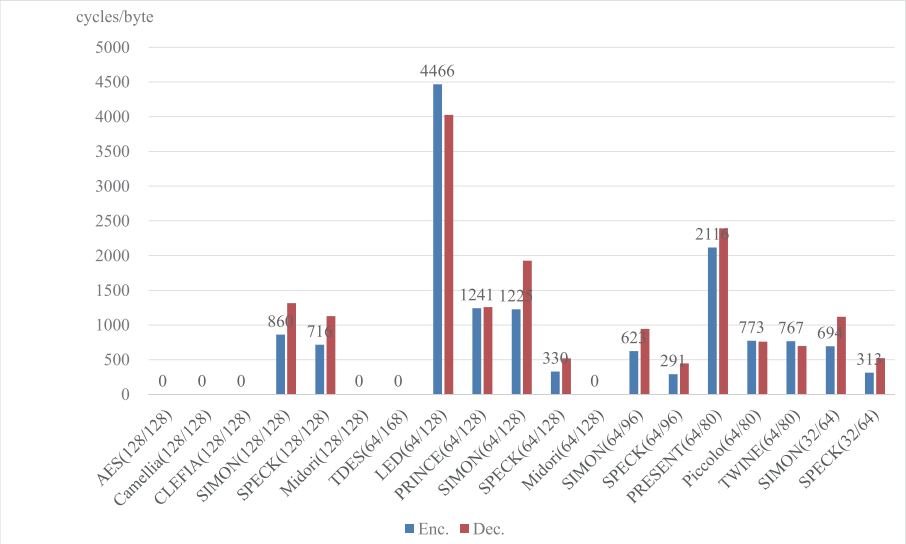

Figure 3.38: Speed with 512-byte ROM and 64-byte RAM

Implementation with restricted memory sizes (Summary)

Figure 3.39 and Figure 3.40 summarize the above results. Figure 3.40 is similar to Figure 3.39 with TDES excluded. Generally speaking, the more severe the memory-size restrictions (further right on the table), the lower the performance. SIMON, SPECK, Piccolo and TWINE don't show this performance reduction, which means that they can afford more memory-size reduction.

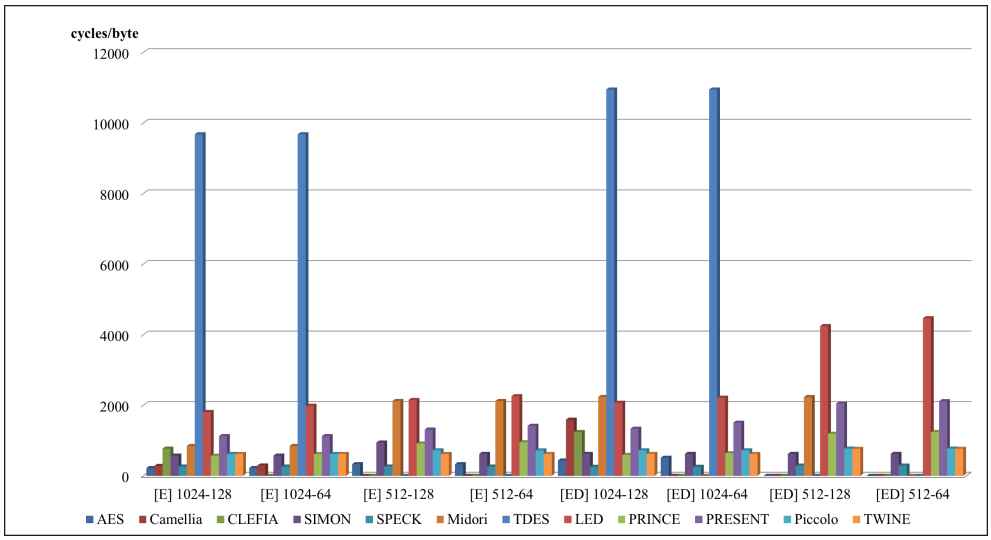

Figure 3.39: List of Speeds with Restricted Memory Sizes

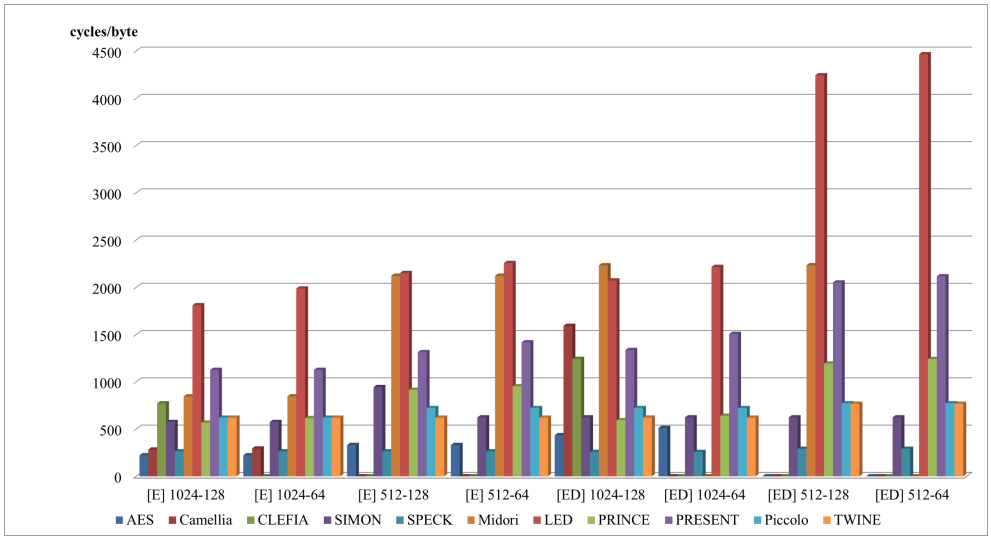

Figure 3.40: List of Speeds with Restricted Memory Sizes (with TDES excluded)

High-speed implementation

Figure 3.41 compares the speeds of the implementations designed to achieve high speeds with ROM of around 2 KB. The results indicate that the highest performance of each algorithm was attainable on the RL78 processor. In the evaluation, AES, Camellia and SPECK indicated similar performance.

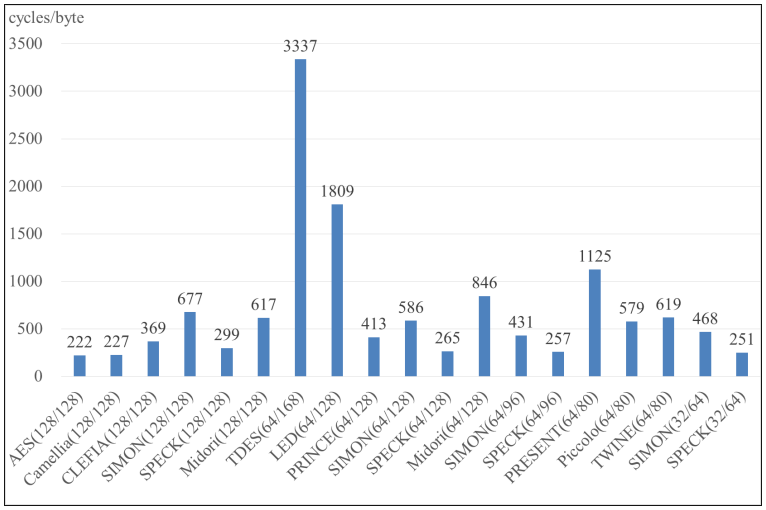

Figure 3.41: List of Speeds with Constrained Implementations

Minimum implementation

Figure 3.42 evaluates the implementation for encryption only with the minimum possible ROM sizes. The comparison clarifies the difference between recent lightweight cryptographic algorithms and other algorithms.

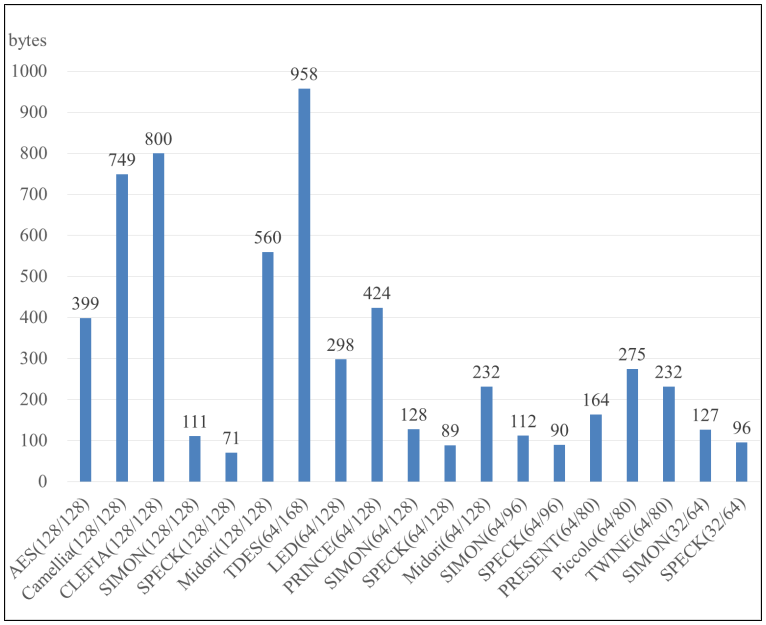

Figure 3.42: List of Speeds with Minimum Implementations

Other considerations

Figure 3.43 plots all implementations for encryption only with a ROM size of 512 bytes or less. The horizontal axis shows the ROM size and the vertical axis the speed. For AES, as long as S-box tables exist, the minimum ROM size was around 400 bytes; below this level, only the algorithms having a smaller S-box or having no S-box were feasible. As shown in the figure, the area of a ROM size of 200 bytes or less and 2,000 cycles/byte or less, could be attained only with SIMON and SPECK, thereby indicating a future direction for lightweight cryptography.

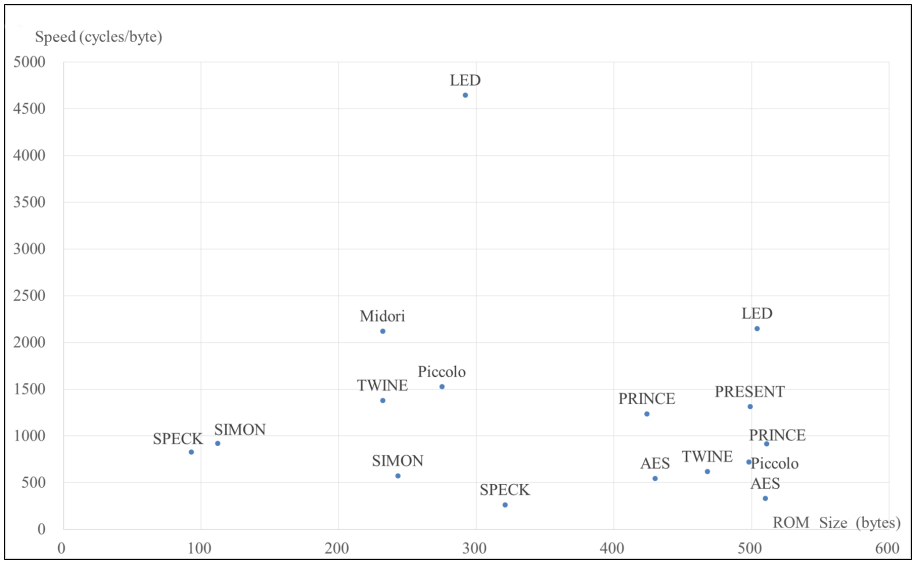

Figure 3.43: Trade-off between Memory Size and Speed

From an algorithmic structure viewpoint, an entire algorithm needs to have a simple structure with a small number of task repetitions to reduce the implementation size. In compact implementation, simple data movement and constant storage create significant overhead. If there are restrictions in the RAM size, it is necessary to perform key scheduling on-the-fly. However, key scheduling often causes a bottleneck in some algorithms.

Moreover, in terms of the structure of the processor, the efficiencies of the rotating and shifting instructions heavily depend on the processor types. Since most modern processors are now little endian, algorithms that assume big endian could cause overhead of changing the byte order, which should be considered when designing lightweight cryptography.

3.1.2.3 Outline of Evaluation Method

This subsection describes the environment in which the software implementation performance of lightweight cryptography was evaluated and the conditions of implementation.

RL78 embedded microprocessor and evaluation environment

In this evaluation, the target block ciphers were implemented on RL78, an embedded microprocessor from Renesas Electronics [6]. RL78 is an accumulator-based 16-bit CISC processor. Its instruction set contains many single-byte instructions, and load-modify instructions are supported. Therefore, this processor is suitable for reducing the ROM size. However, not all RL78 instructions can handle 16-bit data. For example, for the logical operations and rotating and shifting operations often used in block ciphers, only 8-bit instructions are supported.

RL78 has several series. The general-purpose low-end products of the G1x series have a ROM size of 1 KB and a RAM size of 128 bytes. The low-end products in the F1x series developed for automobiles have 8 KB of ROM and 512 bytes of RAM.

The RL78 series have a common instruction set (except for multiplication instructions which are not supported by some models), and their instruction lengths are the same. Therefore, as long as general-purpose instructions other than multiplication instructions are used, RL78 codes operate on all models and occupy the same amount of memory. There are differences in execution speed depending on the type of hardware core (S1, S2, or S3 [6]) used in some codes. In this evaluation, the speeds were measured on the S2, which is used in most models.

CubeSuite+ provided by Renesas Electronics was used for the development environment.

Conditions for implementation

To evaluate the performance of various memory resources, the speeds were measured in the implementation for both encryption only and encryption and decryption under four memory restrictions: 512 bytes and 1,024 bytes of ROM, and 64 bytes and 128 bytes of RAM. In addition, the implementation with the minimum ROM size without considering speed and the speed enhancement by allowing up to around 2 KB of ROM were evaluated.

Depending on the target block cipher algorithm, it was impossible to implement some algorithms under certain memory restrictions, and sufficient performance could be obtained with a limited amount of resources, but adding memory did nothing to enhance performance. In such cases, the corresponding implementation was omitted. The evaluation was conducted with a maximum RAM size of 128 bytes such that all target algorithms were implemented with on-the-fly key scheduling.

The program interface of the library and the method for calculating ROM and RAM sizes were similar to those used in literature [20]. Therefore, the programs were written as subroutines in assembly language to encrypt and decrypt a single block of data, which were with callable by C-language programs. Each subroutine had only one argument, and plaintext, ciphertext, key, and temporal data were stored at the address pointed to by the argument. To minimize the RAM size and achieve practical advantages, the algorithms were implemented under the following conditions:

- plaintext and ciphertext share the same area,
- the contents of the key area do not change before or after calling a subroutine, but may temporarily change during execution,
- the zero-page area (area with an absolute address less than 256) should not be used (reserved for the system),
- programs should be relocatable (no absolute addresses should be hard-coded), and
- no system-dependent coding (e.g., register-bank switching and writing to special-purpose registers directly) is allowed.

The calculation of the ROM and RAM sizes included all resources required to execute each subroutine. Specifically, the ROM size included the code and constant, and the RAM size included the plaintext (which shares the same area with the ciphertext), key, and temporary data stacks.

Therefore, in the case of a 128-bit block cipher and a 128-bit key, an area of 32 bits was occupied by only the plaintext and key. The size of the stack frame required for function calling was 6 bytes (4 bytes for the call instruction and 2 bytes for storing the callee- save register). It follows that, in the implementation with a RAM size of 64 bytes or less, only 28 bytes of RAM including stacks remains, and this is a severe constraint.

3.2 Authenticated Encryption

3.2.1 Evaluation of Software Implementations

This section shows the results of software implementations of authenticated encryption schemes on RL78, an embedded microprocessor with limited memory resources. The target schemes are shown in Figure 3.44.

	Security Level	Key Size	Block Size	Nonce Size	Tag Size
CLOC-TWINE	32	80	64	48	32
SILC-PRESENT	32	80	64	48	32
JAMBU-SIMON	48	96	96	48	48
CLOC-AES	64	128	128	96	64
SILC-AES	64	128	128	96	64
AES-OTR	64	128	128	96	128
AES-OCB	64	128	128	96	128
JAMBU-AES	64	128	128	96	128
AES-GCM	64	128	128	96	128
ACORN	128	128	128	128	128
Minalpher	128	128	256	128	128
Ketje-SR	128	128	32	128	128
Ascon	128	128	128	128	128

Figure 3.44: Evaluated Authenticated Encryption Schemes and their Parameters

Since there is a ongoing project on hardware implementations of authenticated encryption schemes using a unified platform by a research team at George Mason University, it is not mentioned in this guideline. For details of the hardware implementation, refer to the following URL:
Authenticated Encryption FPGA Ranking
`https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/rankings_view`

3.2.1.1 Performance Comparison

Result of evaluating the implementation of authenticated encryption

Figure 3.45 shows the result of evaluating the compact implementation of authenticated encryptions. In the table above, the columns indicate, from left to right: algorithm name, security level (bit), ROM size (bytes), RAM size (bytes), speed of each function for processing relevant data (red: cycles/byte; black: cycles), names of the Core Functions used inside, ROM size of the Core Functions (bytes), and speeds of the Core Functions (cycles/byte). The table below lists the speed of each function for processing encryption and decryption.

In CLOC-TWINE, the speeds of ENC NULL and DEC NULL are 11,160 and 11,099 cycles, respectively, and in CLOC-AES, the speeds of ENC NULL and DEC NULL are 8,895 and 8,790 cycles, respectively.

Figure 3.46 shows the result of evaluating the high-speed implementation of authenticated encryptions. The meaning of each column is the same as that in Figure 3.45.

The speeds of ENC NULL and DEC NULL in CLOC-TWINE are 5,074 and 5,013 cycles, respectively, and in CLOC-AES, the speeds of ENC NULL and DEC NULL are 3,748 and 3,643 cycles, respectively.

Above are the evaluation results of the programs that include the modules for encryption (and tag generation) and decryption (and tag verification). Removing the decryption functional group (DEC 0, DEC 1, DEC 2, DEC 3, DEC 4) from these programs left only the encryption function (and tag generation) in which the performance of other functions did not change and only the ROM size decreased as shown below (in bytes). The ROM sizes were constant and independent of the type of Core Function and compactness or speed of the implementation (and they were designed so).

It should be noted that the encryption/decryption functions for block ciphers used in Core Functions are different from the encryption/decryption functions for authenticated encryptions.

	Sec Level	Size		Associate Data Processing					Core Function		
		ROM	RAM	AD_0	AD_1	AD_2	AD_3	AD_4		ROM	Speed
CLOC-TWINE	32	811	108	-	11199	1409		365	TWINE	234	1380
SILC-PRESENT		537	98	93413	11706			312	PRESENT	164	11677
		600	96	38357	4796			312	PRESENT	227	4768
JAMBU-SIMON	48	522	90	12407	1050			12602	Simon (96/96)	109	1030
CLOC-AES	64	963	150	-	8966	570		637	AES	399	544
SILC-AES		772	144	9071	569			544			
AES-OTR		1202	178	142/9638	594		10134	-			
JAMBU-AES		803	128	8885	563		9010	-			
AES-OCB		1705	224	9252	564		8921		AES	1020	512/607
AES-GCM		760	172	8943	2557			-	Mul128	64	2534
ACORN	128	589	129	109075	489			15522	State	327	478
Minalpher		665	193	607/41361	1276		41220	-	P	295	1241
Ketje-SR		724	114	46827	988		3955	-	F	385	969
Ascon		617	132	40332	2491		19919	11	p^6	299	2475

	Plaintext Data Processing					Ciphertext Data Processing				
	ENC_0	ENC_1	ENC_2	ENC_3	ENC_4	DEC_0	DEC_1	DEC_2	DEC_3	DEC_4
CLOC-TWINE	11160	22365	2799	22593	-	11160	22365	2799	22532	64
SILC-PRESENT	93816	187276	23424		93786	93816	187276	23424		93778
	38343	76730	9606		38513	38543	76730	9606		38505
JAMBU-SIMON	-	1057		12619	25048	-	1057		12633	25099
CLOC-AES	9373	18211	1151	18096	-	9373	18211	1151	17991	108
SILC-AES	9443	18210	1151		9353	9443	18210	1151		9345
AES-OTR	9708	616		19839	10166	9708	609		19841	10158
JAMBU-AES	-	573		9035	17952	-	573		9042	17960
AES-OCB	9736	585		8804	8861	9736	680		8804	8885
AES-GCM	182	3124			49823	182	3124			50008
ACORN		490			62053		493			62085
Minalpher	40729	2560		81942	-	40729	2549		81525	196
Ketje-SR	-	992		23298	11724	-	993		23302	11747
Ascon		2503		206	40220		2517		300	40173

Figure 3.45: Implementation of Authenticated Encryption (Compact Implementation)

Table 3.22: Amount of ROM Reduction by Implementing only Encryption Module of Authenticated Encryption Schemes

Algorithm	AES-GCM	CLOC	SILC	AES-OTR	Ketje	Minalpher
ROM Reduction (bytes)	47	102	59	255	108	93

All of the target authenticated encryption schemes of this evaluation, except Minalpher and AES-OCB, do not require the inverse functions of the Core Functions for decrypting authenticated encryption (or for a decrypting block cipher if the Core Function was a block cipher). The Core Function of Minalpher and its inverse are not precisely the same, but very similar because of an involution structure.

The interface in this evaluation was created so that the functions for encryption and the functions for decryption for authenticated encryption were clearly separated. Therefore, the size of an authenticated encryption module that has only encryption (and tag generation) could be calculated simply by subtracting the size of the functions for decryption from the total size, and the speed was kept constant.

Figure 3.47 is a chart for comparing the ROM sizes in compact implementation. Figure 3.48 divides the ROM sizes into those of the Core Functions (below) and those of the other parts that correspond to the mode of operation (above). Figure 3.49 is a chart for comparing the speeds (more precisely the asymptotic speeds for a sufficiently large plain text) of the high-speed implementation. Figure 3.50 divides the speeds into those of the Core Functions (below) and those of the other parts that correspond to the mode of operation (above).

In these charts, the security levels (32-bit, 64-bit, and 128-bit) of the authenticated encryption schemes are shown in different colors.

	Sec Level	Size		Associate Data Processing					Core Function		
		ROM	RAM	AD_0	AD_1	AD_2	AD_3	AD_4		ROM	Speed
CLOC-TWINE	32	1049	106	-	5113	649		365	TWINE	470	620
SILC-PRESENT		896	118	10777	1349			262	PRESENT	499	1320
JAMBU-SIMON	48	1709	80	6342	531		6435	-	Simon (96/96)	477	527
CLOC-AES	64	1521	128	-	3819	248		637	AES	928	222
SILC-AES		1369	124	3924	248			438			
AES-OTR		1958	156	142/4081	239		4163	-			
JAMBU-AES		2466	102	3589	227		3710	-			
AES-OCB		2962	216	3809	238		3837		AES	2007	222/391
AES-GCM		1489	150	3669	1034			-	Mul128	239	1011
ACORN	128	871	116	55296	246			7752	State	446	237
Minalpher		1926	169	49/9580	308		9898	-	P	1455	289
		1977	425	866/8669	254		8170	-	P	1455	235
Ketje-SR		1482	114	15699	340		1361	-	f	927	321
Ascon		1966	116	11173	696		5639	11	p^6	1015	691

	Plaintext Data Processing					Ciphertext Data Processing				
	ENC_0	ENC_1	ENC_2	ENC_3	ENC_4	DEC_0	DEC_1	DEC_2	DEC_3	DEC_4
CLOC-TWINE	5074	10193	1278	10421	-	5074	10193	1278	10307	64
SILC-PRESENT	10657	21485	2689		10883	10657	21485	2689		10875
JAMBU-SIMON	-	532		6455	12699	-	532		6460	12767
CLOC-AES	3728	7585	476	7971	-	3728	7585	476	7866	108
SILC-AES	3694	7751	487		4100	3694	7751	487		4092
AES-OTR	4151	245		8140	4147	4151	243		8142	4139
JAMBU-AES	-	228		3738	7203	-	228		3745	7293
AES-OCB	4923	243		3893	3737	4923	412		3893	3888
AES-GCM	173	1271			20172	173	1271			20357
ACORN		247			31034		250			31082
Minalpher	9713	616		19739	-	9713	602		19554	196
	7985	508		16283	-	7985	494		16098	196
Ketje-SR	-	343		7734	3942	-	344		7738	3965
Ascon		698		188	11184		699		273	11312

Figure 3.46: Implementation of Authenticated Encryption (Fast Implementation)

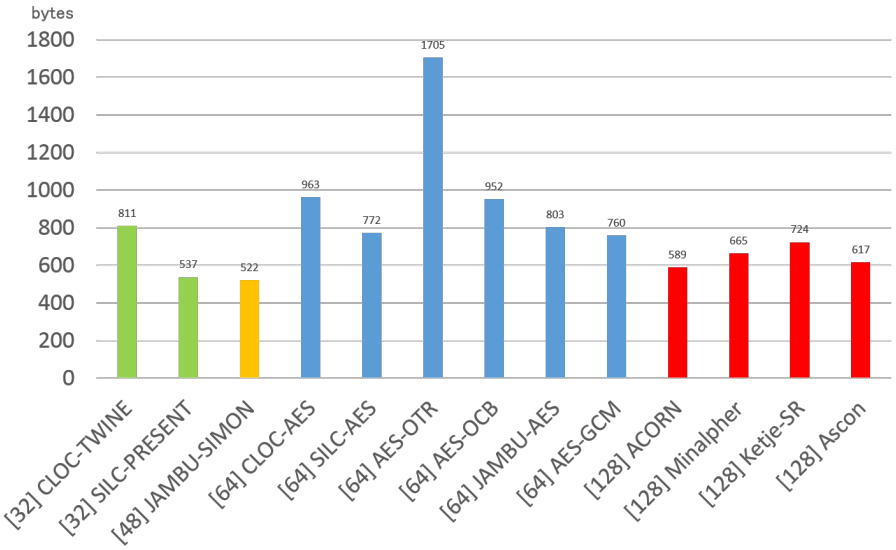

Figure 3.47: Comparison of the ROM Sizes in Compact Implementations

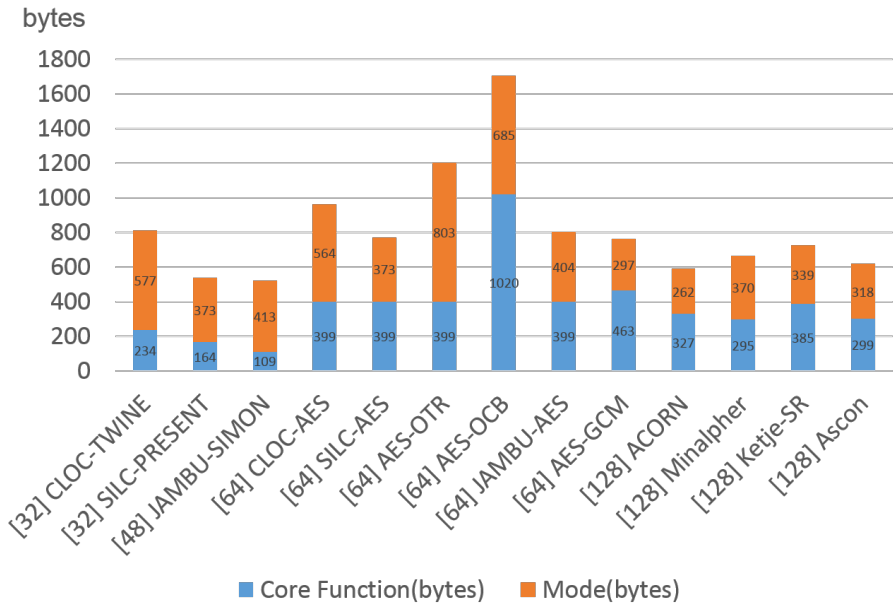

Figure 3.48: Comparison of the ROM Sizes in Compact Implementations (with Core Functions and Modes Separated)

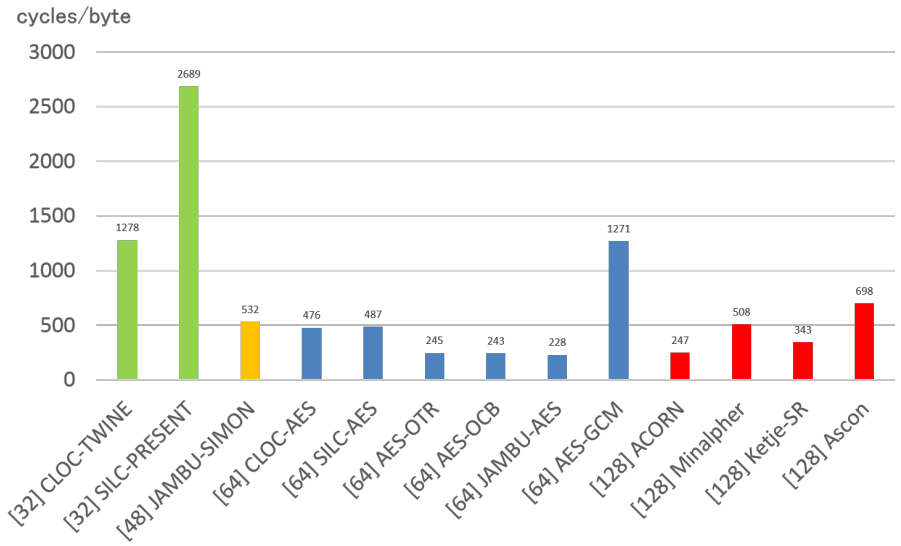

Figure 3.49: Comparison of Encryption Speed (Asymptotical Speeds) in Fast Implementations

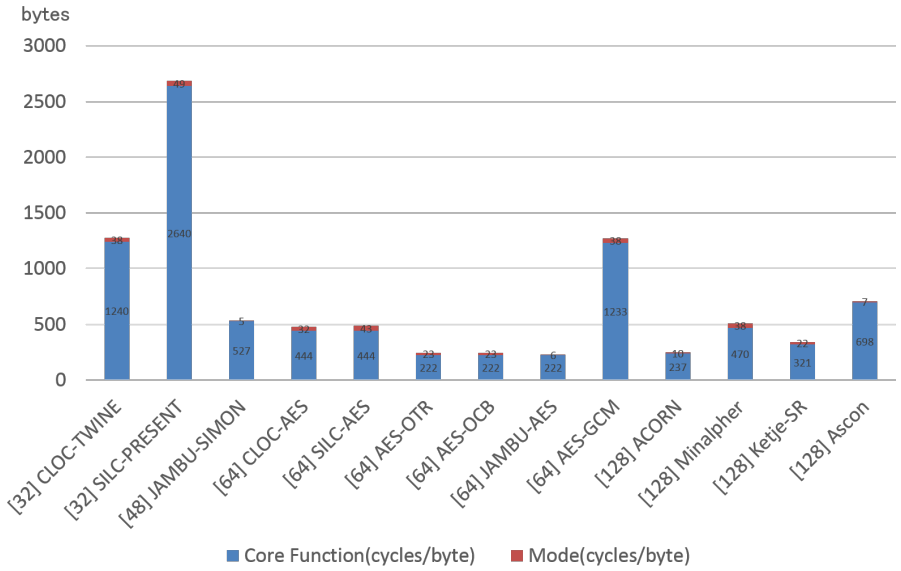

Figure 3.50: Comparison of Encryption Speed (Asymptotical Speeds) in Fast Implementations (with Core Functions and Modes Separated)

3.2.1.2 Outline of Evaluation Method

Coding policy and interface specifications

Since an authenticated encryption has many input/output parameters including plaintext, ciphertext, key, associated data, and nonce, it is difficult to perform a one-dimensional evaluation of speed as in the case of lightweight cryptography. Therefore, this evaluation divides each authenticated encryption scheme into several units and measures the speed of each unit while conforming to the coding policy indicated in literature [21]. Given the lengths of the plaintext and associated data, anyone can obtain the number of calculation cycles using the table created in this report.

However, if the scheme is divided into small parts, a high-layer program for controlling the overall flow is required to process the entire authenticated encryption, and if the program has large overhead, it is impossible to correctly estimate the total speed by accumulating the performance data of each part.

A method for dividing the scheme into blocks while keeping the overhead of the high-order program as small as possible is proposed here. Concretely, an authenticated encryption scheme is divided into an associated data processing part (AD), an encryption part (ENC), and a decryption part (DEC), and each of the three parts is sub-divided into five functions as listed below:

1 Initialization (process before feeding associated data, plaintext, and ciphertext)

2 Calculation of the first block

3 Calculation of intermediate blocks (each block from the second to the penultimate block)

4 Calculation of the last block

5 Finalization (process after feeding associated data, plaintext, and ciphertext)

Each function was coded as a single program function. However, since there were some unnecessary functions and duplicated functions, it was not necessary to implement each of a total of 15 functions for all authenticated encryption schemes. In addition, the boundary between two functions could not be determined precisely; therefore, the presumably most reasonable boundary was set for each scheme.

Notation used for the functions are exemplified as follows:

AD_0 Initialization of the associated data processing part

ENC_123 Calculation of the first, intermediate, and last blocks of the encryption part, which means ENC_1, ENC_2, and ENC_3 can be merged.

As an example of the AES-GCM encryption mode, a high-layer program that describes the entire authenticated encryption using the functions above is shown in Figure 3.51. Note that alen and mlen indicate the lengths of the associated data and plaintext, respectively. BLEN is the block length, which is 16 in this example. The functions defined above are shown in red.

```
int crypto_aead_encrypt(int mlen, int alen)
{
        int   clen=0,size;

        // Associate Data Handling

        AESGCM_AD_0();                          // Initialization

        while(alen > 0){
                size = (alen >= BLEN) ? BLEN : alen;
                AESGCM_AD_123(size);            // one block processing
                aadr += BLEN;                   // aadr = address of AD
                alen -= BLEN;
        }

        // Message Handling

        AESGCM_ENC_0();                         // Initialization

        while(mlen > 0){
                size = (mlen >= BLEN) ? BLEN : mlen;
                clen += AESGCM_ENC_123(size);   // one block processing
                madr += BLEN;                   // madr = address of MSG
                cadr += BLEN;                   // cadr = address of OUT
                mlen -= BLEN;
        }

        AESGCM_ENC_4(alen, clen);               // Tag Generation

        return clen;
}
```

Figure 3.51: Sample Code of AES-GCM Encryption Mode

AES-GCM

In this evaluation, the AES-GCM algorithm described in [10] was implemented using the following common parameters.

Table 3.23: AES-GCM Parameters Used in Evaluation

Algorithm	Key Size	Block Size	Nonce Size	Tag Size
AES-GCM	128 bits	128 bits	96 bits	64 bits

The AES-GCM algorithm was further divided into the functions shown in Figure 3.52.

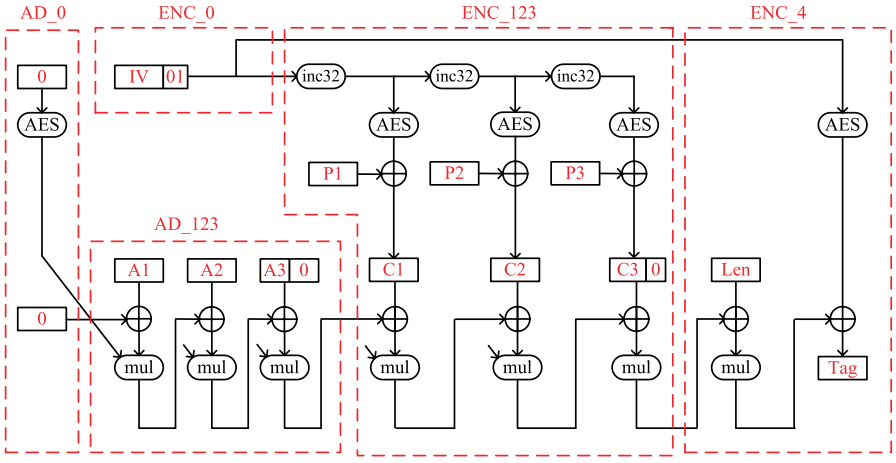

Figure 3.52: Functional Division of AES-GCM for Evaluation

CLOC

For the CLOC algorithm v2 [18], three parameters were recommended. In this evaluation, two were implemented; one used TWINE-64-80 for the Core Function, and the other used AES-128-128 for the Core Function.

Table 3.24: CLOC Parameters Used in Evaluation

Algorithm	Key Size	Block Size	Nonce Size	Tag Size
CLOC-TWINE	80 bits	64 bits	48 bits	32 bits
CLOC-AES	128 bits	128 bits	96 bits	64 bits

The CLOC algorithm was further divided into the functions shown in Figure 3.53. When the size of the plaintext was 0, CLOC performed a special process called ENC_NULL.

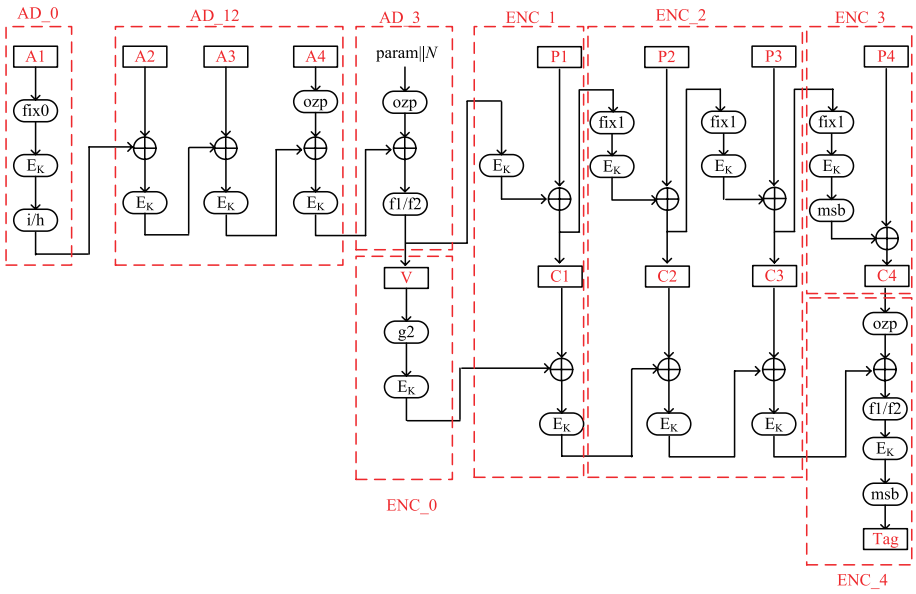

Figure 3.53: Functional Division of CLOC for Evaluation

SILC

For the SILC algorithm v2 [12], four parameters were recommended. In this evaluation, two were implemented; one used PRESENT-64-80 for the Core Function, and the other used AES-128-128 for the Core Function.

Table 3.25: SILC Parameters Used in Evaluation

Algorithm	Key Size	Block Size	Nonce Size	Tag Size
SILC-PRESENT	80 bits	64 bits	48 bits	32 bits
SILC-AES	128 bits	128 bits	96 bits	64 bits

The SILC algorithm was further divided into the functions shown in Figure 3.54.

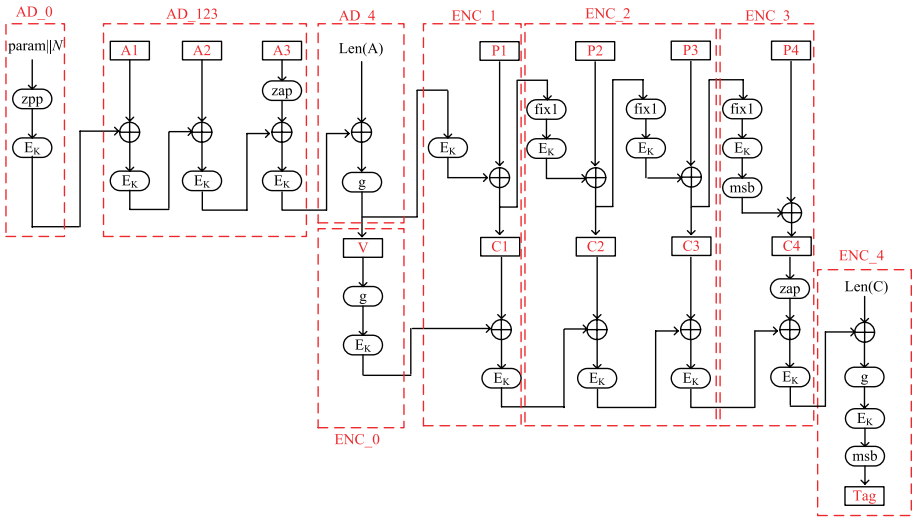

Figure 3.54: Functional Division of SILC for Evaluation

Minalpher

For the Minalpher algorithm [4], substitution, called Minalpher-P, was used as the Core Function. It had one parameter, indicated in the following.

The Minalpher algorithm was further divided into the functions shown in Figure 3.55.

Table 3.26: Minalpher Parameters Used in Evaluation

Algorithm	Key Size	Block Size	Nonce Size	Tag Size
Minalpher	128 bits	256 bits	104 bits	128 bits

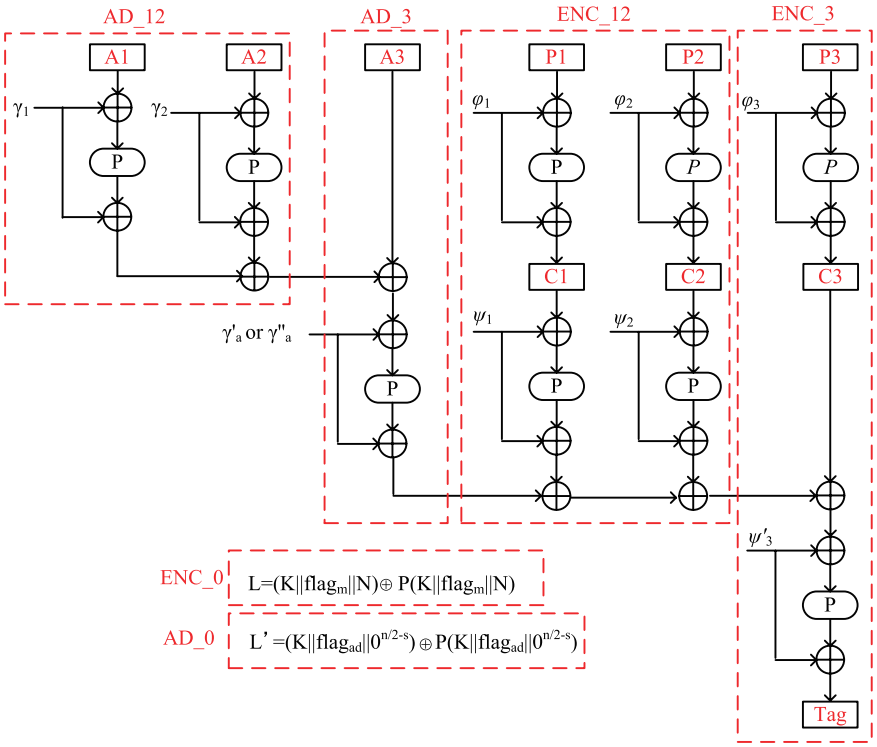

Figure 3.55: Functional Division of Minalpher for Evaluation

AES-OTR

The AES-OTR algorithm [2] uses AES-128-128 or AES-128-256 as the Core Function. Several parameters are defined for AES-OTR. The implementation for this evaluation used the one called primary parameter and AES-128-128 as the Core Function.

Table 3.27: AES-OTR Parameters Used in Evaluation

Algorithm	Key Size	Block Size	Nonce Size	Tag Size
AES-OTR	128 bits	128 bits	96 bits	128 bits

The AES-OTR algorithm was further divided into the functions shown in Figure 3.56.

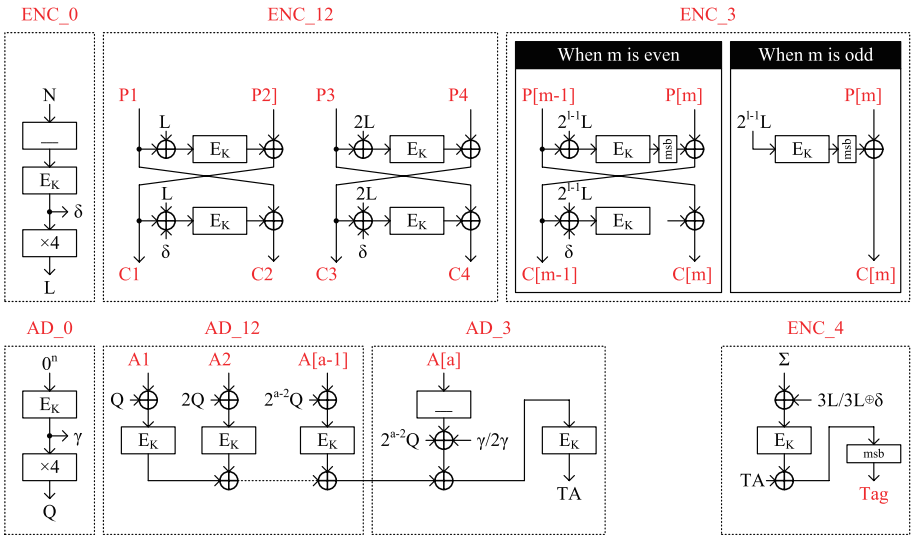

Figure 3.56: Functional Division of AES-OTR for Evaluation

Ketje

The Ketje algorithm [3] is an authenticated encryption scheme based on the sponge function, and employs a dedicated function f as the Core Function There are two parameters for Ketje: Ketje-SR and Ketje-JR. In this evaluation we implemented the former (primary recommendation) with 50-byte state size.

Table 3.28: Ketje Parameters Used in Evaluation

Algorithm	Key Size	Block Size	Nonce Size	Tag Size
Ketje-SR	128 bits	32 bits	128 bits	128 bits

The Ketje algorithm was further divided into the functions shown in Figure 3.57.

AD_12 and AD_3 are the same function except that the constants are different. The number below f indicates the number of repetitions inside the function f.

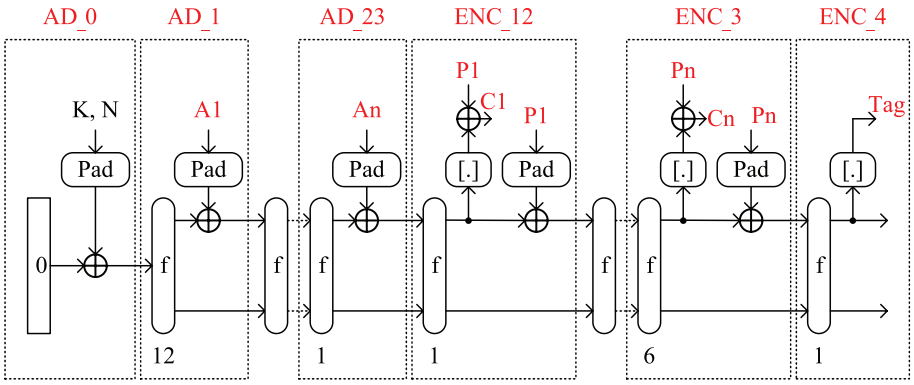

Figure 3.57: Functional Division of Ketje for Evaluation

ACORN

ACORN is a stream-cipher-based authenticated encryption scheme. In ACORN, the 293-bit state is updated by repeating the Core Function StateUpdate with changing the control bits and input bits.

The implementation for this evaluation used the parameters shown in Table 3.58.

Table 3.29: ACORN Parameters Used in Evaluation

Algorithm	Key Size	Block Size	Nonce Size	Tag Size
ACORN	128 bits	128 bits	128 bits	128 bits

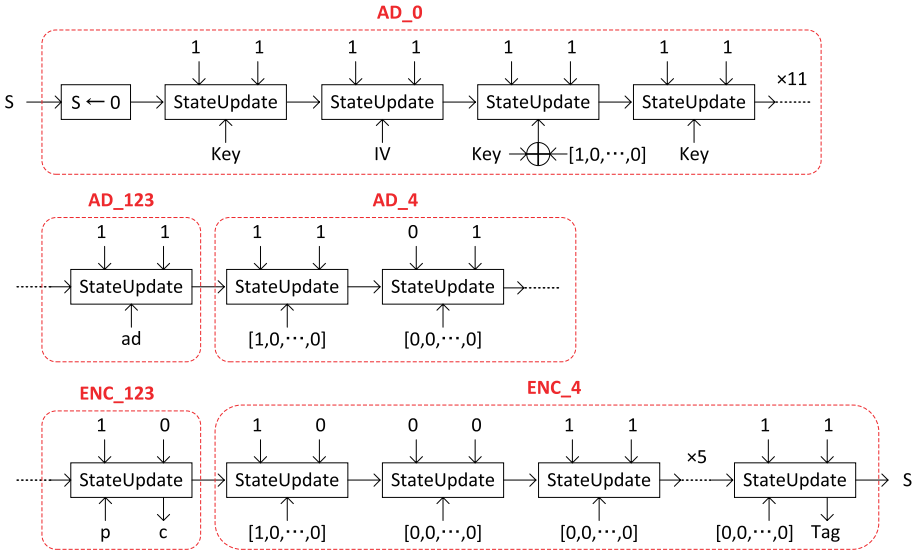

Figure 3.58: Functional Division of ACORN for Evaluation

AES-OCB

AES-OCB [5] uses AES-128, AES-192 or AES-256, as the Core Function. The implementation for this evaluation used the parameter considered as the primary recommendation, where AES-128 is used as the Core Function.

Note that AES-OCB requires the AES decryption in the decryption of authenticated encryption scheme, which is only one scheme among the AES-based authenticated encryption schemes in this evaluation.

Table 3.30: AES-OCB Parameters Used in Evaluation

Algorithm	Key Size	Block Size	Nonce Size	Tag Size
AES-OCB	128 bits	128 bits	96 bits	128 bits

The AES-OCB algorithm was divided into the functions shown in Figure 3.59

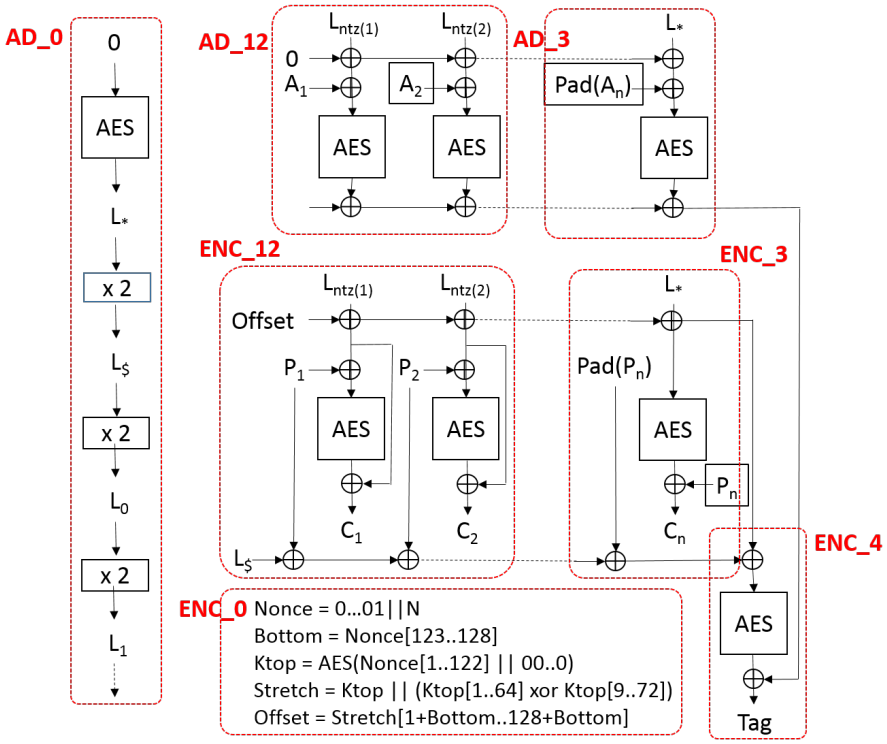

Figure 3.59: Functional Division of AES-OCB for Evaluation

JAMBU

The Jambu algorithm [7] uses Simon-96-96, Simon-64-96, Simon-128-128, or AES-128-128 as the Core Function. The implementation for this evaluation used two parameter sets shown in Table 3.31: one using Simon-96-96 (primary recommendation), and the others using AES-128-128.

Table 3.31: JAMBU Parameters Used in Evaluation

Algorithm	Key Size	Block Size	Nonce Size	Tag Size
JAMBU-SIMON	96 bits	96 bits	48 bits	48 bits
JAMBU-AES	128 bits	128 bits	64 bits	64 bits

The JAMBU algorithm was divided into the functions shown in Figure 3.60.

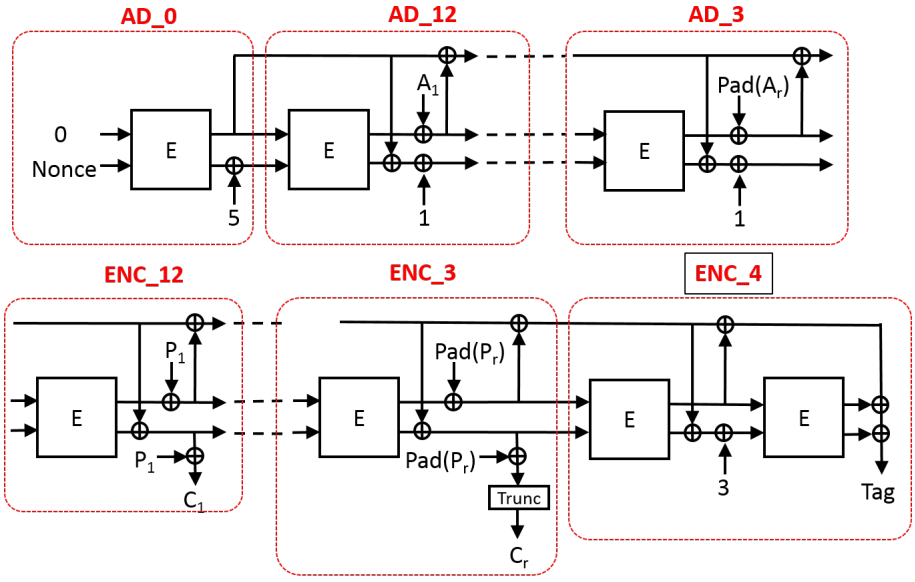

Figure 3.60: Functional Division of JAMB for Evaluation

Ascon

The Ascon [1] is an authenticated encryption scheme based on the sponge function, likely to Ketje, and employs a dedicated function p as the Core Function. The implementation for this evaluation used the parameter shown in Table 3.32 considered as the primary recommendation, where p has 40-byte input/output.

Table 3.32: Ascon Parameters Used in Evaluation

Algorithm	Key Size	Block Size	Nonce Size	Tag Size
Ascon	128 bits	128 bits	96 bits	128 bits

The Ascon algorithm was divided into the functions shown in Figure 3.61. In this figure, p^{12} and p^6 denote 12 and 6 times repetitions of the function p, respectively.

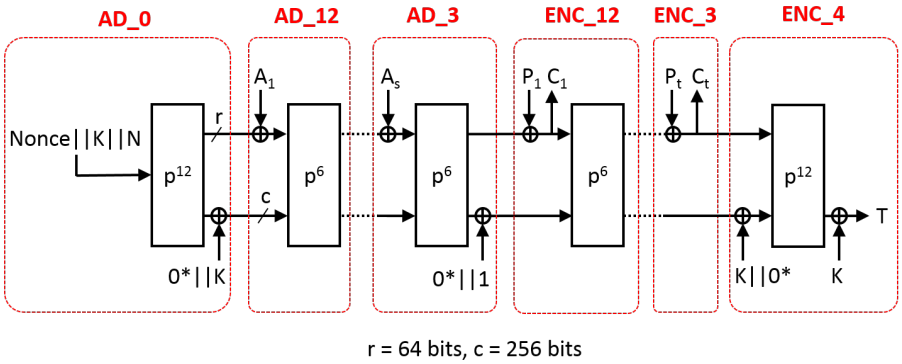

Figure 3.61: Functional Division of Ascon for Evaluation

Bibliography

[1] ACORN v2. https://competitions.cr.yp.to/round3/asconv12.pdf.

[2] AES-OTR v1. http://competitions.cr.yp.to/round1/aesotrv1.pdf.

[3] Ketje. http://ketje.noekeon.org/.

[4] Minalpher Homepage. http://info.isl.ntt.co.jp/crypt/minalpher/index.html.

[5] OCB (v1.1).

[6] RL78 Family. http://japan.renesas.com/products/mpumcu/rl78/index.jsp?campaign=tb_prod.

[7] The JAMBU Lightweight Authentication Encryption Mode. https://competitions.cr.yp.to/round3/jambuv21.pdf.

[8] Specification for the Advanced Encryption Standard (AES), 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[9] Specification of Camellia - a 128-bit Block Cipher, 2001. http://info.isl.ntt.co.jp/crypt/camellia/dl/01espec.pdf.

[10] NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, 2007. http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

[11] Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, 2012. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-67r1.pdf.

[12] CLOC and SILC—Authenticated Encryption Schemes for Constrained Devices, 2014. http://www.nuee.nagoya-u.ac.jp/labs/tiwata/AE/.

[13] ARM. AMBA 3 APB Protocol Specification v2.0, 2008. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0024c/index.html.

[14] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, pages 411–436, 2015.

[15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. SIMON and SPECK: block ciphers for the internet of things. *IACR Cryptology ePrint Archive*, 2015:585, 2015.

[16] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 208–225, 2012.

[17] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 326–341, 2011.

[18] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. CLOC: authenticated encryption for short input. In *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, pages 149–167, 2014.

[19] Lars R. Knudsen and Gregor Leander. PRESENT - block cipher. In *Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 953–955. 2011.

[20] Mitsuru Matsui and Yumiko Murakami. Minimalism of software implementation - extensive performance analysis of symmetric primitives on the RL78 microcontroller. In *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, pages 393–409, 2013.

[21] Mitsuru Matsui and Yumiko Murakami. AES smaller than s-box - minimalism in software design on low end microcontrollers. In *Lightweight Cryptography for Security and Privacy - Third International Workshop, LightSec 2014, Istanbul, Turkey, September 1-2, 2014, Revised Selected Papers*, pages 51–66, 2014.

[22] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of AES. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pages 69–88, 2011.

[23] Konstantinos Papagiannopoulos and Aram Verstegen. Speed and size-optimized implementations of the PRESENT cipher for tiny AVR devices. In *Radio Frequency Identification - Security and Privacy Issues 9th International Workshop, RFIDsec 2013, Graz, Austria, July 9-11, 2013, Revised Selected Papers*, pages 161–175, 2013.

[24] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 342–357, 2011.

[25] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, pages 181–195, 2007.

[26] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE : A lightweight block cipher for multiple platforms. In *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, pages 339–354, 2012.

[27] Hongjun Wu. ACORN v2. `http://competitions.cr.yp.to/caesar-submissions.html/`.

Chapter 4

Lightweight Cryptographic Algorithms and Schemes

4.1 Block Ciphers

This chapter shows the summary of the major lightweight block cipher algorithms presented in major cryptographic conferences/workshops: CLEFIA, LED, Midori, Piccolo, PRESENT, PRINCE, SIMON, SPECK, and TWINE. No effective attack against these algorithms has been found at this time, and they are thought to have sufficient levels of implementation performance. In addition to the basic input/output information including the block length and key length, the overall structure and number of rounds are described for each algorithm. The names of algorithms having unique names according to their key length and block length are also indicated. The features as described by the designers and authors of the proposal paper are shown as they are, as much as possible. The Security Analysis column describes the security analysis on the full round algorithms. For algorithms for which there are no known efficient attacks according to the specifications, the security analysis is based on the simplified algorithm for reference. The general security of block ciphers is determined by those key length and block length. However, it should be noted that the general security of PRINCE was claimed to be lower than the basic security setting. Regarding a designers' assumed attacker capacity, some algorithms were designed to resist more powerful attacks, including related-key and weak-key attacks, as well as attacks under the single-key setting; these algorithms are indicated in the feature column. The research on hardware implementation performance targets ASIC implementation, for which a sufficient number of evaluations have been conducted, and describes the gate equivalent (GE), number of cycles required for the operation of a single block (cycles/block), and throughput at 100 kHz. For the software implementation performance in high-end CPUs, the number of cycles required to process a single byte (cycles/byte) is indicated. The results of the implementation in low-end CPUs include the ROM and RAM usage as well as cycles/byte.

<table>
<tr><td></td><td>Block cipher</td></tr>
<tr><td>Name</td><td>CLEFIA</td></tr>
<tr><td>Designers</td><td>Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai (Sony Corporation, Japan), Tetsu Iwata (Nagoya University, Japan)</td></tr>
<tr><td>Publication</td><td>2007 (FSE 2007 [31])</td></tr>
<tr><td>Specifications</td><td><code>http://www.sony.net/Products/cryptography/clefia/</code>[31]</td></tr>
<tr><td>Features</td><td>The designers claim that the algorithms have high performance in both hardware and software implementation while maintaining high levels of security. They also feature an interface identical to the AES.
<table>
<tr><td>Overall Structure</td><td colspan="3">4-line type-II generalized Feistel type</td></tr>
<tr><td>Block Length [bits]</td><td colspan="3">128</td></tr>
<tr><td>Key Length [bits]</td><td>128
(CLEFIA-128)</td><td>192
(CLEFIA-192)</td><td>256
(CLEFIA-256)</td></tr>
<tr><td>No. of Rounds [rounds]</td><td>18</td><td>22</td><td>26</td></tr>
</table></td></tr>
<tr><td>Security Analysis</td><td>Many papers on security analysis have been published, but no attack more efficient than exhaustive search have been found for the specified number of rounds.
For CLEFIA-128 reduced to 13 rounds, CLEFIA-192 reduced to 14 rounds, and CLEFIA-256 reduced to 15 rounds, attacks more efficient than exhaustive search have been found [22, 9].</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation evaluation results
<table>
<tr><th>Algorithm</th><th>Area [GE]</th><th>Cycles/block</th><th>Throughput @100kHz [kbps]</th><th>Ref.</th></tr>
<tr><td>CLEFIA-128 (Enc)</td><td>2,488</td><td>328</td><td>39.0</td><td>[2]</td></tr>
<tr><td>CLEFIA-128 (Enc/Dec)</td><td>2,604</td><td>328/320</td><td>39.0/40.0</td><td>[2]</td></tr>
<tr><td>CLEFIA-128 (Enc/Dec)</td><td>5,979</td><td>18</td><td>711.1</td><td>[31]</td></tr>
</table>
Software implementation (low-end CPU) evaluation results
<table>
<tr><th>Algorithm</th><th>ROM [byte]</th><th>RAM [byte]</th><th>Cycles/byte [Enc/Dec]</th><th>Platform</th><th>Ref.</th></tr>
<tr><td>CLEFIA-128</td><td>1,309</td><td>78</td><td>39,357/152,023</td><td>RL78</td><td>[24]</td></tr>
<tr><td>CLEFIA-128</td><td>2,026</td><td>64</td><td>4,337/4,477</td><td>RL78</td><td>[24]</td></tr>
</table></td></tr>
<tr><td>Standardization</td><td>ISO/IEC 29192-2</td></tr>
</table>

<table>
<tr><td></td><td>Block cipher</td></tr>
<tr><td>Name</td><td>LED</td></tr>
<tr><td>Designers</td><td>Jian Guo (Institute for Infocomm Research, Singapore), Thomas Peyrin, Axel Poschmann (Nanyang Technological University, Singapore), Matt Robshaw (Orange Labs, France)</td></tr>
<tr><td>Publication</td><td>2011 (CHES 2011 [19])</td></tr>
<tr><td>Specifications</td><td><code>http://led.crypto.sg/home-1</code> [19]</td></tr>
<tr><td>Features</td><td>The designers claim that the algorithm has sufficient software implementation performance and eliminates key scheduling, resists against related-key attacks, and is tuned for lightweight hardware implementation. Similar to lightweight hash function PHOTON, it has a serialized MDS as an internal structure.
<table>
<tr><td>Overall Structure</td><td colspan="2">SPN type</td></tr>
<tr><td>Block Length [bits]</td><td colspan="2">64</td></tr>
<tr><td>Key Length [bits]</td><td>64 (LED-64)</td><td>128 (LED-128)</td></tr>
<tr><td>Number of Rounds [rounds]</td><td>32 (8 steps)</td><td>48 (12 steps)</td></tr>
</table></td></tr>
<tr><td>Security Analysis</td><td>Some papers on security analysis have been published, but no attack more efficient than exhaustive search has been found for the specified number of rounds.
For LED-64 reduced to 12 rounds and LED-128 reduced to 32 rounds, attacks more efficient than exhaustive search have been found [17].</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation evaluation results
<table>
<tr><th>Algorithm</th><th>Area [GE]</th><th>Cycles/block</th><th>Throughput @100kHz [kbps]</th><th>Ref.</th></tr>
<tr><td>LED-64 (Enc)</td><td>966</td><td>1,248</td><td>5.1</td><td>[19]</td></tr>
<tr><td>LED-64 (Enc)</td><td>2,695</td><td>32</td><td>200.0</td><td>[1]</td></tr>
<tr><td>LED-128 (Enc)</td><td>1,265</td><td>1,872</td><td>3.4</td><td>[19]</td></tr>
<tr><td>LED-128 (Enc)</td><td>3,036</td><td>48</td><td>133.3</td><td>[1]</td></tr>
</table>
Software implementation evaluation results
<table>
<tr><th>Algorithm</th><th>Type</th><th>Cycles/byte</th><th>Platform</th><th>Ref.</th></tr>
<tr><td>LED-64</td><td>Table/VPI/Bitslice</td><td>76.0/48.1/13.1</td><td>Core i3 2367M</td><td>[6]</td></tr>
<tr><td>LED-128</td><td>Table/VPI/Bitslice</td><td>113.3/54.6/17.6</td><td>Core i3 2367M</td><td>[6]</td></tr>
</table></td></tr>
<tr><td>Open Source Information</td><td>The implementations available on the LED website.
<code>http://led.crypto.sg/software</code>
lightweight-crypto-lib
<code>https://github.com/rb-anssi/lightweight-crypto-lib</code></td></tr>
</table>

<table>
<tr><td></td><td>Block cipher</td></tr>
<tr><td>Name</td><td>Midori</td></tr>
<tr><td>Designers</td><td>Subhadeep Banik, Andrey Bogdanov (Technical University of Denmark, Denmark), Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita (Sony Corporation, Japan), Francesco Regazzoni (University of Lugano, Switzerland)</td></tr>
<tr><td>Publication</td><td>2015 (ASIACRYPT 2015 [3])</td></tr>
<tr><td>Specifications</td><td>[3]</td></tr>
<tr><td>Features</td><td>The designers claim that the algorithm has low energy consumption, has compact hardware implementation, and is low latency.
<table>
<tr><td>Overall Structure</td><td colspan="2">SPN type</td></tr>
<tr><td>Block Length [bits]</td><td>64 (Midori64)</td><td>128 (Midori128)</td></tr>
<tr><td>Key Length [bits]</td><td colspan="2">128</td></tr>
<tr><td>Number of Rounds [rounds]</td><td>16</td><td>20</td></tr>
</table></td></tr>
<tr><td>Security Analysis</td><td>Some papers on security analysis have been published, but no attack more efficient than exhaustive search have been found for the specified number of rounds.
Weak keys exist for Midori64. If those keys are used, efficient attacks can be performed even on the specified numbers of rounds [18]. In addition, for Midori64 reduced to 12 rounds, more efficient attacks have been found [21].</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation evaluation results
<table>
<tr><th>Algorithm</th><th>Area [GE]</th><th>Cycles/block</th><th>Throughput @100kHz [kbps]</th><th>Ref.</th></tr>
<tr><td>Midori64 (Enc)</td><td>1,542</td><td>16</td><td>400.0</td><td>[3]</td></tr>
<tr><td>Midori64 (Enc/Dec)</td><td>2,450</td><td>16</td><td>400.0</td><td>[3]</td></tr>
<tr><td>Midori128 (Enc)</td><td>2,522</td><td>20</td><td>640.0</td><td>[3]</td></tr>
<tr><td>Midori128 (Enc/Dec)</td><td>3,661</td><td>20</td><td>640.0</td><td>[3]</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>Block cipher</td></tr>
<tr><td>Name</td><td>Piccolo</td></tr>
<tr><td>Designers</td><td>Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, Taizo Shirai (Sony Corporation, Japan)</td></tr>
<tr><td>Publication</td><td>2011 (CHES 2011 [30])</td></tr>
<tr><td>Specifications</td><td>[30]</td></tr>
<tr><td>Features</td><td>The designers claim that the algorithm is sufficiently secure against related-key attacks and meet-in-the-middle attacks, as well as conventional attacks, has particularly high performance in hardware implementation, is structured so that a decryption function can be implemented without causing serious overhead, and is superior in lightweightness and energy efficiency.
<table>
<tr><td>Overall Structure</td><td colspan="2">4-line type-2 generalized Feistel type</td></tr>
<tr><td>Block Length [bits]</td><td colspan="2">64</td></tr>
<tr><td>Key Length [bits]</td><td>80 (Piccolo-80)</td><td>128 (Piccolo-128)</td></tr>
<tr><td>Number of Rounds [rounds]</td><td>25</td><td>31</td></tr>
</table></td></tr>
<tr><td>Security Analysis</td><td>Some papers on security analysis have been published, but no attack more efficient than exhaustive search has been found for the specified number of rounds.
For Piccolo-80 reduced to 14 rounds and Piccolo-128 reduced to 21 rounds, more efficient attacks [30], and related-key attacks [25] have been found.</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation evaluation results
<table>
<tr><th>Algorithm</th><th>Area [GE]</th><th>Cycles/block</th><th>Throughput @100kHz [kbps]</th><th>Ref.</th></tr>
<tr><td>Piccolo-80 (Enc)</td><td>1,048</td><td>432</td><td>14.8</td><td>[30]</td></tr>
<tr><td>Piccolo-80 (Enc)</td><td>1,499</td><td>27</td><td>237.0</td><td>[30]</td></tr>
<tr><td>Piccolo-80 (Enc/Dec)</td><td>1,109</td><td>432</td><td>14.8</td><td>[30]</td></tr>
<tr><td>Piccolo-128 (Enc)</td><td>1,338</td><td>528</td><td>12.1</td><td>[30]</td></tr>
<tr><td>Piccolo-128 (Enc)</td><td>1,776</td><td>33</td><td>193.9</td><td>[30]</td></tr>
<tr><td>Piccolo-128 (Enc/Dec)</td><td>1,397</td><td>528</td><td>12.1</td><td>[30]</td></tr>
</table>
Software implementation evaluation results
<table>
<tr><th>Algorithm</th><th>Type</th><th>Cycles/byte</th><th>Platform</th><th>Ref.</th></tr>
<tr><td>Piccolo-80</td><td>Bitslice</td><td>4.57</td><td>Core i7 870</td><td>[23]</td></tr>
<tr><td>Piccolo-128</td><td>Bitslice</td><td>5.52</td><td>Core i7 870</td><td>[23]</td></tr>
<tr><td>Piccolo-80</td><td>Table/VPI/Bitslice</td><td>89.3/33.3/9.2</td><td>Core i3 2367M</td><td>[6]</td></tr>
<tr><td>Piccolo-128</td><td>Table/VPI/Bitslice</td><td>103.6/41.6/10.9</td><td>Core i3 2367M</td><td>[6]</td></tr>
</table></td></tr>
<tr><td>Open Source Information</td><td>lightweight-crypto-lib
<code>https://github.com/rb-anssi/lightweight-crypto-lib</code></td></tr>
</table>

<table>
<tr><td></td><td>Block cipher</td></tr>
<tr><td>Name</td><td>PRESENT</td></tr>
<tr><td>Designers</td><td>Andrey Bogdanov[1], Lars R. Knudsen[2], Gregor Leander[1], Christof Paar[1], Axel Poschmann[1], Matthew J. B. Robshaw[3], Yannick Seurin[3], C. Vikkelsoe[2] (1: Ruhr-University Bochum, Germany, 2: Technical University of Denmark, Denmark, 3: France Telecom, France)</td></tr>
<tr><td>Publication</td><td>2007 (CHES 2007 [10])</td></tr>
<tr><td>Specifications</td><td>[10]</td></tr>
<tr><td>Features</td><td>This is a pioneering algorithm for a lightweight block cipher. It has particularly high performance in compact hardware implementation.
<table>
<tr><td>Overall Structure</td><td colspan="2">SPN type</td></tr>
<tr><td>Block Length [bits]</td><td colspan="2">64</td></tr>
<tr><td>Key Length [bits]</td><td>80 (PRESENT-80)</td><td>128 (PRESENT-128)</td></tr>
<tr><td>Number of Rounds [rounds]</td><td colspan="2">31</td></tr>
</table></td></tr>
<tr><td>Security Analysis</td><td>Many papers on security analysis have been published, but no attack more efficient than exhaustive search has been found for the specified number of rounds. For PRESENT-80/128 reduced to 26 rounds, more efficient attacks have been found [14, 8].</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation evaluation results
<table>
<tr><th>Algorithm</th><th>Area [GE]</th><th>Cycles/block</th><th>Throughput @100kHz [kbps]</th><th>Ref.</th></tr>
<tr><td>PRESENT-80 (Enc)</td><td>1,000</td><td>563</td><td>11.4</td><td>[29]</td></tr>
<tr><td>PRESENT-80 (Enc)</td><td>1,570</td><td>32</td><td>200.0</td><td>[10]</td></tr>
<tr><td>PRESENT-128 (Enc)</td><td>1,391</td><td>559</td><td>11.4</td><td>[27]</td></tr>
<tr><td>PRESENT-128 (Enc)</td><td>1,886</td><td>32</td><td>200.0</td><td>[10]</td></tr>
</table>
Software implementation (high-end CPU) evaluation results
<table>
<tr><th>Algorithm</th><th>Type</th><th>Cycles/byte</th><th>Platform</th><th>Ref.</th></tr>
<tr><td>PRESENT-80/128</td><td>Bitslice</td><td>5.79</td><td>Core i7 870</td><td>[23]</td></tr>
<tr><td>PRESENT-80</td><td>Table/VPI/Bitslice</td><td>72.6/35.0/17.4</td><td>Core i3 2367M</td><td>[6]</td></tr>
<tr><td>PRESENT-128</td><td>Table/VPI/Bitslice</td><td>72.5/35.0/18.9</td><td>Core i3 2367M</td><td>[6]</td></tr>
</table>
Software implementation (low-end CPU) evaluation results
<table>
<tr><th>Algorithm</th><th>ROM [byte]</th><th>RAM [byte]</th><th>Cycles/byte [Enc/Dec]</th><th>Platform</th><th>Ref.</th></tr>
<tr><td>PRESENT-80</td><td>512</td><td>62</td><td>61,634/60,834</td><td>RL78</td><td>[24]</td></tr>
<tr><td>PRESENT-80</td><td>1,855</td><td>48</td><td>9,007/8,920</td><td>RL78</td><td>[24]</td></tr>
</table></td></tr>
<tr><td>Standardization</td><td>ISO/IEC 29192-2</td></tr>
<tr><td>Open Source Information</td><td>lightweight-crypto-lib
https://github.com/rb-anssi/lightweight-crypto-lib</td></tr>
</table>

<table>
<tr><td></td><td>Block cipher</td></tr>
<tr><td>Name</td><td>PRINCE</td></tr>
<tr><td>Designers</td><td>Julia Borghoff[1], Anne Canteaut[1,2], Tim Guneysu[3], Elif Bilge Kavun[3], Miroslav Knezevic[4], Lars R. Knudsen[1], Gregor Leander[1], Ventzislav Nikov[4], Christof Paar[3], Christian Rechberger[1], Peter Rombouts[4], Soren S. Thomsen[1], Tolga Yalcin[3] (1: Technical University of Denmark, Denmark, 2: INRIA, France, 3: Ruhr-University Bochum, Germany, 4: NXP Semiconductors, Netherlands)</td></tr>
<tr><td>Publication</td><td>2012 (ASIACRYPT 2012 [11])</td></tr>
<tr><td>Specifications</td><td>[11]</td></tr>
<tr><td>Features</td><td>The designers claim that the algorithm has low latency and is compact in hardware implementation.
Since the algorithm has symmetry called α-reflection, which is different from normal block ciphers, if the attacker has 2^n pairs of plain text and cipher text, only $(127-n)$-bit security can be claimed although a 128-bit key is used.
<table>
<tr><td>Overall Structure</td><td>SPN type</td></tr>
<tr><td>Block Length [bits]</td><td>64</td></tr>
<tr><td>Key Length [bits]</td><td>128</td></tr>
<tr><td>Number of Rounds [rounds]</td><td>12</td></tr>
</table></td></tr>
<tr><td>Security Analysis</td><td>Some papers on security analysis have been published, but no attack more efficient than exhaustive search have been found for the specified number of rounds.
For PRINCE reduced to 10 rounds, more effective attacks have been found [12].</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation evaluation results
<table>
<tr><td>Algorithm</td><td>Area [GE]</td><td>Cycles/block</td><td>Throughput @100kHz[kbps]</td><td>Ref.</td></tr>
<tr><td>PRINCE (Enc/Dec)</td><td>2,953</td><td>12</td><td>533.3</td><td>[4]</td></tr>
<tr><td>PRINCE (Enc/Dec)</td><td>8,577</td><td>1</td><td>6400.0</td><td>[4]</td></tr>
</table>
Software implementation (low-end CPU) evaluation results
<table>
<tr><td>Algorithm</td><td>ROM [byte]</td><td>RAM [byte]</td><td>Cycles/byte [Enc/Dec]</td><td>Platform</td><td>Ref.</td></tr>
<tr><td>PRINCE</td><td>2,382</td><td>220</td><td>225.4</td><td>ATtiny85</td><td>[26]</td></tr>
</table></td></tr>
<tr><td>Open Source Information</td><td>The source codes are not provided on an official PRINCE web page (this is something we should do at some point) or some personal web page. However, the designers will provide hardware (Verilog-HDL) and software (C) source codes when requested.</td></tr>
</table>

<table>
<tr><td></td><td>Block cipher</td></tr>
<tr><td>Name</td><td>SIMON</td></tr>
<tr><td>Designers</td><td>Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, Louis Wingers (National Security Agency, USA)</td></tr>
<tr><td>Publication</td><td>2013 (Cryptology ePrint Archive [5])</td></tr>
<tr><td>Specifications</td><td>[5]</td></tr>
<tr><td>Features</td><td>The designers claim that the algorithm supports various lengths of blocks and keys and has superior lightweightness in both hardware and software implementation, particularly in hardware implementation.
SIMON with a $2n$-bit block cipher and a m-word key is indicated as SIMON$2n/mn$. For example, SIMON64/128 means SIMON with a block length of 64 bits and a key length of 128 bits.
<table>
<tr><td>Overall Structure</td><td colspan="11">Feistel type</td></tr>
<tr><td>Block Length [bits]</td><td>32</td><td colspan="2">48</td><td colspan="2">64</td><td colspan="2">96</td><td colspan="3">128</td></tr>
<tr><td>Key Length [bits]</td><td>64</td><td>72</td><td>96</td><td>96</td><td>128</td><td>96</td><td>144</td><td>128</td><td>192</td><td>256</td></tr>
<tr><td>No. of Rounds [rounds]</td><td>32</td><td colspan="2">36</td><td>42</td><td>44</td><td>52</td><td>54</td><td>68</td><td>69</td><td>72</td></tr>
</table></td></tr>
<tr><td>Security Analysis</td><td>Many papers on security analysis have been published, but no attack more efficient than exhaustive search has been found for the specified number of rounds.
For SIMON with block lengths of 32, 48, 64, 96, and 128 bits reduced to 23, 25, 31, 38, and 53 rounds, more efficient attacks have been found [13].</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation evaluation results
<table>
<tr><th>Algorithm</th><th>Area [GE]</th><th>Cycles/block</th><th>Throughput@100kHz [kbps]</th><th>Ref.</th></tr>
<tr><td>SIMON64/96</td><td>809</td><td>1,455</td><td>4.4</td><td>[5]</td></tr>
<tr><td>SIMON64/128</td><td>958</td><td>1,524</td><td>4.2</td><td>[5]</td></tr>
<tr><td>SIMON128/128</td><td>1,234</td><td>4,414</td><td>2.9</td><td>[5]</td></tr>
<tr><td>SIMON128/256</td><td>1,782</td><td>4,923</td><td>2.6</td><td>[5]</td></tr>
</table>
Software implementation (low-end CPU) evaluation results
<table>
<tr><th>Algorithm</th><th>ROM [byte]</th><th>RAM [byte]</th><th>Cycles/byte [Enc/Dec]</th><th>Platform</th><th>Ref.</th></tr>
<tr><td>SIMON64/96</td><td>274</td><td>0</td><td>239</td><td>ATtiny45</td><td>[5]</td></tr>
<tr><td>SIMON64/128</td><td>282</td><td>0</td><td>250</td><td>ATtiny45</td><td>[5]</td></tr>
<tr><td>SIMON128/128</td><td>732</td><td>0</td><td>376</td><td>ATtiny45</td><td>[5]</td></tr>
<tr><td>SIMON128/256</td><td>764</td><td>0</td><td>398</td><td>ATtiny45</td><td>[5]</td></tr>
</table></td></tr>
<tr><td>Open Source Information</td><td>The SIMON and SPECK team has released public domain software implementing several versions of SIMON and SPECK for different platforms [1]. The contributions to the FELICS benchmarking framework [2] provide implementations of the 64-bit block sizes of SIMON and SPECK optimized for the AVR, MSP430, and ARM Cortex M microcontrollers. The contributions to the SUPERCOP benchmarking framework [3] provide implementations of the 64 and 128-bit block sizes of SIMON and SPECK optimized for higher-end CPUs such as X86 and ARM Cortex A.
An Internet search uncovers a number of other small open source software projects implementing SIMON or SPECK [4, 5, etc.]. However, they are not included in broadly used cryptographic libraries such as OpenSSL.
[1]<code>https://iadgov.github.io/simon-speck/implementations/</code>
[2]<code>https://www.cryptolux.org/index.php/FELICS</code>
[3]<code>https://bench.cr.yp.to/supercop.html</code>
[4]<code>https://rweather.github.io/arduinolibs/crypto.html</code>
[5]<code>http://cppcrypto.sourceforge.net/</code></td></tr>
</table>

<table>
<tr><td></td><td>Block cipher</td></tr>
<tr><td>Name</td><td>SPECK</td></tr>
<tr><td>Designers</td><td>Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, Louis Wingers (NSA, USA)</td></tr>
<tr><td>Publication</td><td>2013 (Cryptology ePrint Archive [5])</td></tr>
<tr><td>Specifications</td><td>[5]</td></tr>
<tr><td>Features</td><td>The designers claim that the algorithm supports various lengths of blocks and keys, and has superior lightweight for both hardware and software implementation, particularly software implementation.
Similar to SIMON, SPECK with a $2n$-bit block cipher and a m-word key is indicated as SPECK$2n/mn$. For example, SPECK64/128 means SPECK with a block length of 64 bits and a key length of 128 bits.
<table>
<tr><td>Overall Structure</td><td colspan="10">Feistel variant</td></tr>
<tr><td>Block Length [bits]</td><td>32</td><td colspan="2">48</td><td colspan="2">64</td><td colspan="2">96</td><td colspan="3">128</td></tr>
<tr><td>Key Length [bits]</td><td>64</td><td>72</td><td>96</td><td>96</td><td>128</td><td>96</td><td>144</td><td>128</td><td>192</td><td>256</td></tr>
<tr><td>No. of Rounds [rounds]</td><td colspan="2">22</td><td>23</td><td>26</td><td>27</td><td>28</td><td>29</td><td>32</td><td>33</td><td>34</td></tr>
</table></td></tr>
<tr><td>Security Analysis</td><td>Many papers on security analysis have been published, but no attack more efficient than exhaustive search has been found for the specified number of rounds.
For SPECK with block lengths of 32, 48, 64, 96, and 128 bits reduced to 14, 15, 19, 17, and 19 rounds, more efficient attacks have been found [7, 16].</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation evaluation results
<table>
<tr><th>Algorithm</th><th>Area [GE]</th><th>Cycles/block</th><th>Throughput@100kHz [kbps]</th><th>Ref.</th></tr>
<tr><td>SPECK64/96</td><td>860</td><td>1,778</td><td>3.6</td><td>[5]</td></tr>
<tr><td>SPECK64/128</td><td>996</td><td>1,882</td><td>3.4</td><td>[5]</td></tr>
<tr><td>SPECK128/128</td><td>1,280</td><td>4,267</td><td>3.0</td><td>[5]</td></tr>
<tr><td>SPECK128/256</td><td>1,840</td><td>4,571</td><td>2.8</td><td>[5]</td></tr>
</table>
Software implementation (low-end CPU) evaluation results
<table>
<tr><th>Algorithm</th><th>ROM [byte]</th><th>RAM [byte]</th><th>Cycles/byte [Enc/Dec]</th><th>Platform</th><th>Ref.</th></tr>
<tr><td>SPECK64/96</td><td>182</td><td>0</td><td>144</td><td>ATtiny45</td><td>[5]</td></tr>
<tr><td>SPECK64/128</td><td>186</td><td>0</td><td>150</td><td>ATtiny45</td><td>[5]</td></tr>
<tr><td>SPECK128/128</td><td>396</td><td>0</td><td>167</td><td>ATtiny45</td><td>[5]</td></tr>
<tr><td>SPECK128/256</td><td>412</td><td>0</td><td>177</td><td>ATtiny45</td><td>[5]</td></tr>
</table></td></tr>
<tr><td>Open Source Information</td><td>The SIMON and SPECK team has released public domain software implementing several versions of SIMON and SPECK for different platforms [1]. The contributions to the FELICS benchmarking framework [2] provide implementations of the 64-bit block sizes of SIMON and SPECK optimized for the AVR, MSP430, and ARM Cortex M microcontrollers. The contributions to the SUPERCOP benchmarking framework [3] provide implementations of the 64 and 128-bit block sizes of SIMON and SPECK optimized for higher-end CPUs such as X86 and ARM Cortex A.
An Internet search uncovers a number of other small open source software projects implementing SIMON or SPECK [4, 5, etc.]. However, they are not included in broadly used cryptographic libraries such as OpenSSL.
[1]<code>https://iadgov.github.io/simon-speck/implementations/</code>
[2]<code>https://www.cryptolux.org/index.php/FELICS</code>
[3]<code>https://bench.cr.yp.to/supercop.html</code>
[4]<code>https://rweather.github.io/arduinolibs/crypto.html</code>
[5]<code>http://cppcrypto.sourceforge.net/</code></td></tr>
</table>

<table>
<tr><td></td><td>Block cipher</td></tr>
<tr><td>Name</td><td>TWINE</td></tr>
<tr><td>Designers</td><td>Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, Eita Kobayashi (NEC, Japan)</td></tr>
<tr><td>Publication</td><td>2011 (ECRYPT Workshop on Lightweight Cryptography, SAC 2012 [32])</td></tr>
<tr><td>Specifications</td><td>http://jpn.nec.com/rd/crl/code/research/image/twine_SAC_full_v5.pdf, [32]</td></tr>
<tr><td>Features</td><td>The designers claim that the algorithm has superior lightweight in hardware implementation and software implementation in a wide range of CPUs from low-end to high-end. It uses an improved block shuffle proposed by the designers at FSE 2010 to enhance security.
<table>
<tr><td>Overall Structure</td><td colspan="2">16-line generalized Feistel type</td></tr>
<tr><td>Block Length [bits]</td><td colspan="2">64</td></tr>
<tr><td>Key Length [bits]</td><td>80 (TWINE-80)</td><td>128 (TWINE-128)</td></tr>
<tr><td>Number of Rounds [rounds]</td><td colspan="2">36</td></tr>
</table></td></tr>
<tr><td>Security Analysis</td><td>Some papers on security analysis have been published, but no attack more efficient than exhaustive search have been found for the specified number of rounds.</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation evaluation results
<table>
<tr><th>Algorithm</th><th>Area [GE]</th><th>Cycles/block</th><th>Throughput @100kHz [kbps]</th><th>Ref.</th></tr>
<tr><td>TWINE-80 (Enc)</td><td>1,503</td><td>36</td><td>177.8</td><td>[32]</td></tr>
<tr><td>TWINE-80 (Enc)</td><td>1,011</td><td>393</td><td>16.3</td><td>[32]</td></tr>
<tr><td>TWINE-80 (Enc/Dec)</td><td>1,799</td><td>36</td><td>177.8</td><td>[32]</td></tr>
<tr><td>TWINE-128 (Enc)</td><td>1,866</td><td>36</td><td>177.8</td><td>[32]</td></tr>
<tr><td>TWINE-128 (Enc/Dec)</td><td>2,285</td><td>36</td><td>177.8</td><td>[32]</td></tr>
</table>
Software implementation (high-end CPU) evaluation results
<table>
<tr><th>Algorithm</th><th>Type</th><th>Cycles/byte</th><th>Platform</th><th>Ref.</th></tr>
<tr><td>TWINE-80/128</td><td>Bitslice (Single/Double)</td><td>11.10/5.55</td><td>Core i7 2600S</td><td>[32]</td></tr>
</table>
Software implementation (low-end CPU) evaluation results
<table>
<tr><th>Algorithm</th><th>ROM [byte]</th><th>RAM [byte]</th><th>Cycles/byte [Enc/Dec]</th><th>Platform</th><th>Ref.</th></tr>
<tr><td>TWINE-80</td><td>2,294</td><td>386</td><td>163/163</td><td>ATmega163</td><td>[32]</td></tr>
<tr><td>TWINE-80</td><td>792</td><td>191</td><td>2,350/2,337</td><td>ATmega163</td><td>[32]</td></tr>
</table></td></tr>
<tr><td>Market Deployment</td><td>NEC Engineering Industrial IoT Solution: Secure Sensors
http://jpn.nec.com/engsl/pro/securesensor/index.html
http://www.nec-eng.co.jp/press/161024press.html
NEC Store Video Surveillance Cloud Service
http://jpn.nec.com/press/201604/20160421_02.html
NEC ExpEther 40G
http://jpn.nec.com/press/201511/20151109_01.html</td></tr>
</table>

Bibliography

[1] The LED block cipher, December 2013. Available from `https://sites.google.com/site/ledblockcipher/hardware`.

[2] Toru Akishita and Harunaga Hiwatari. Very compact hardware implementations of the block-cipher CLEFIA. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 278–292. Springer, 2011.

[3] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

[4] Lejla Batina, Amitabh Das, Baris Ege, Elif Bilge Kavun, Nele Mentens, Christof Paar, Ingrid Verbauwhede, and Tolga Yalçin. Dietary recommendations for lightweight block ciphers: Power, energy and area analysis of recently developed architectures. In Michael Hutter and Jörn-Marc Schmidt, editors, *Radio Frequency Identification - Security and Privacy Issues 9th International Workshop, RFIDsec 2013, Graz, Austria, July 9-11, 2013, Revised Selected Papers*, volume 8262 of *Lecture Notes in Computer Science*, pages 103–112. Springer, 2013.

[5] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. *IACR Cryptology ePrint Archive*, 2013:404, 2013.

[6] Ryad Benadjila, Jian Guo, Victor Lomné, and Thomas Peyrin. Implementing lightweight block ciphers on x86 architectures. In Lange et al. [20], pages 324–351.

[7] Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential analysis of block ciphers SIMON and SPECK. In Cid and Rechberger [15], pages 546–570.

[8] Céline Blondeau and Kaisa Nyberg. Links between truncated differential and multidimensional linear properties of block ciphers and underlying attack complexities. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 2014.

[9] Andrey Bogdanov, Huizheng Geng, Meiqin Wang, Long Wen, and Baudoin Collard. Zero-correlation linear cryptanalysis with FFT and improved attacks on ISO standards Camellia and CLEFIA. In Lange et al. [20], pages 306–323.

[10] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[11] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[12] Anne Canteaut, Thomas Fuhr, Henri Gilbert, María Naya-Plasencia, and Jean-René Reinhard. Multiple differential cryptanalysis of round-reduced PRINCE. In Cid and Rechberger [15], pages 591–610.

[13] Huaifeng Chen and Xiaoyun Wang. Improved linear hull attack on round-reduced simon with dynamic key-guessing techniques. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 428–449. Springer, 2016.

[14] Joo Yeon Cho. Linear cryptanalysis of reduced-round PRESENT. In Josef Pieprzyk, editor, *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings*, volume 5985 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2010.

[15] Carlos Cid and Christian Rechberger, editors. *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*. Springer, 2015.

[16] Itai Dinur. Improved differential cryptanalysis of round-reduced speck. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2014.

[17] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Key recovery attacks on 3-round even-mansour, 8-step LED-128, and full AES^2. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 337–356. Springer, 2013.

[18] Jian Guo, Jérémy Jean, Ivica Nikolic, Kexin Qiao, Yu Sasaki, and Siang Meng Sim. Invariant subspace attack against full midori64. *IACR Cryptology ePrint Archive*, 2015:1189, 2015.

[19] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Preneel and Takagi [28], pages 326–341.

[20] Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors. *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*. Springer, 2014.

[21] Li Lin and Wenling Wu. Meet-in-the-middle attacks on reduced-round midori-64. *IACR Cryptology ePrint Archive*, 2015:1165, 2015.

[22] Hamid Mala, Mohammad Dakhilalian, and Mohsen Shakiba. Impossible differential attacks on 13-round CLEFIA-128. *J. Comput. Sci. Technol.*, 26(4):744–750, 2011.

[23] Seiichi Matsuda and Shiho Moriai. Lightweight cryptography for the cloud: Exploit the power of bitslice implementation. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 408–425. Springer, 2012.

[24] Mitsuru Matsui and Yumiko Murakami. Minimalism of software implementation - extensive performance analysis of symmetric primitives on the RL78 microcontroller. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 393–409. Springer, 2013.

[25] Marine Minier. On the security of Piccolo lightweight block cipher against related-key impossible differentials. In Goutam Paul and Serge Vaudenay, editors, *Progress in Cryptology - INDOCRYPT 2013 - 14th International Conference on Cryptology in India, Mumbai, India, December 7-10, 2013. Proceedings*, volume 8250 of *Lecture Notes in Computer Science*, pages 308–318. Springer, 2013.

[26] Kostas Papapagiannopoulos. High throughput in slices: The case of present, PRINCE and KATAN64 ciphers. In Nitesh Saxena and Ahmad-Reza Sadeghi, editors, *Radio Frequency Identification: Security and Privacy Issues - 10th International Workshop, RFIDSec 2014, Oxford, UK, July 21-23, 2014, Revised Selected Papers*, volume 8651 of *Lecture Notes in Computer Science*, pages 137–155. Springer, 2014.

[27] Axel Poschmann. Lightweight cryptography - cryptographic engineering for a pervasive world. *IACR Cryptology ePrint Archive*, 2009:516, 2009.

[28] Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*. Springer, 2011.

[29] Carsten Rolfes, Axel Poschmann, Gregor Leander, and Christof Paar. Ultra-lightweight implementations for smart devices - security for 1000 gate equivalents. In Gilles Grimaud and François-Xavier Standaert, editors, *Smart Card Research and Advanced Applications, 8th IFIP WG 8.8/11.2 International Conference, CARDIS 2008, London, UK, September 8-11, 2008. Proceedings*, volume 5189 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2008.

[30] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In Preneel and Takagi [28], pages 342–357.

[31] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2007.

[32] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE : A lightweight block cipher for multiple platforms. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.

4.2 Stream Ciphers

This section shows the summary of the major lightweight stream ciphers. Stream ciphers mainly provide only confidentiality unlike block ciphers, which can provide other functionalities as well as confidentiality. In resource constrained environment where program size in software implementation or circuit size in hardware implementation are strongly limited, it may not be acceptable to implement various cryptographic algorithms. In this case using a cryptographic algorithm to achieve various functions is more desirable and a block cipher is the appropriate option. In addition, many stream ciphers take more time for initialization therefore a block cipher is often a better choice for processing short data. On the other hand, if only confidentiality is required and the process must be executed at a high speed with a small amount of resources, a stream cipher is more suitable.

Some stream ciphers proposed and evaluated earlier than other lightweight ciphers, so there are more mature algorithms. This section introduces several stream ciphers from eStream portfolio [24] and ISO/IEC 29192-3 [13], on which sufficient security evaluations have been conducted. The eStream project called for and evaluated algorithms in two categories: Profile 1 (software) and Profile 2 (hardware). This section describes three algorithms published in Profile 2 portfolio – Grain v1 [17], MICKEY 2.0 [4], and Trivium [9]. Four algorithms of Profile 1 portfolio – HC-128, Rabbit, Salsa20/12, and SOSEMANUK – are software oriented ciphers. Salsa20/12 is the fastest on CPUs for PCs and servers to process long data. This section describes ChaCha20 [7] instead of Salsa20/12, which was publised in RFC in 2015. ChaCha20, an improved version of Salsa20/12, is a stream cipher for software implementation. It is slightly slower than Salsa20/12 but has been adopted by various open source softwares. Therefore it is much easier to use ChaCha20 than Salsa20/12 in real applications while the performance difference is not so large. ISO/IEC 29192-3 includes two algorithms; Trivium and Enocoro. Therefore we also pick up Enocoro in this section.

The ISO/IEC 29167 standard specifies cryptographic suites for RFID. However, it includes a crypto suite "XOR," which is undesirable for security reasons, and its use is not generally recommended by CRYPTREC.

	Stream cipher
Name	ChaCha20
Designers	Daniel J. Bernstein (University of Illinois at Chicago, USA)
Publication	2008 (SASC 2008) [7]
Specifications	[6, 23]
Features	This is a stream cipher with a 256-bit key and a 96-bit initialization vector (IV). In addition to a secret key and an IV, it receives a 128-bit constant and a 32-bit block counter, and outputs a 512-bit pseudo-random number. The algorithm utilizes 32-bit word arithmetic additions, exclusive ORs (XORs), and cyclic shifts, and is suitable for software implementation. Since it requires little cost for initialization, it can process short messages quickly. In addition, since ChaCha20 uses no table lookups, straightforward implementation is secure against cache-timing attacks.
Security Analysis	As of 2016, no results compromising the security of ChaCha20 have been found. Aumasson et al. have demonstrated that 7 out of 20 rounds can be attacked using differential attacks more efficiently than exhaustive search [3]. Other analyses include that by Shi et al. [25] and that by Maitra [20]. Both have improved the result of Aumasson et al., but the number of attackable rounds was limited to 7.
Performance Analysis	According to the evaluation at eBASC [1] (as of June 2016), the algorithm achieved 1.2 cycles/byte on the Intel Core i5 processor. In addition, according to the evaluation of the Fair Evaluation of Lightweight Cryptographic Systems (FELICS) project [2] (as of June 2016), the algorithm requires 144 cycles for initialization on ARM Cortex-M3 processor and has a throughput of 54.3 cycles/byte.
Standardization	ChaCha20 has been standardized in IETF RFC 7539 [23].
Market Deployment	This cipher is mainly used in combination with a message authentication code, Poly-1305, to serve as an authenticated encryption. The cipher is used to protect the communication channels (https) for services provided by Google [8].
Open Source Information	OpenSSL, Google Chrome, Mozilla Firefox, OpenSSH. `https://ianix.com/pub/chacha-deployment.html`

	Stream cipher
Name	Enocoro
Designers	Hitachi, Ltd.
Publication	2008 (WAIS 2008), 2010 (ISITA 2010)
Specifications	`http://www.hitachi.co.jp/rd/yrl/crypto/enocoro/`
Features	This cipher consists of two algorithms, Enocoro-80 with 80-bit key and Enocoro-128v2 with 128-bit key. Both algorithms claim a level of security corresponding to the key length. However, the output data is limited to 2^{32} bytes and 2^{64} bytes, respectively, by fixing the key and IV. Uncommon for lightweight stream ciphers, this cipher processes in 8-bit units and can achieve processing speeds similar to AES even in software implementation.
Security Analysis	No vulnerability has been found in the single-key setting for either Enocoro-80 or Enocoro-128v2; however, their forerunner, Enocoro-128v1.1, has been found to be vulnerable to related-key attacks [12, 19, 18, 10]. For example, according to the evaluation by Ding et al. [10], the secret key of Enocoro-80 is weak with a probability of 2^{-8}, and it can be recovered with complexity of 2^{48} using 2^{17} chosen IVs. The effectiveness of related-key attacks on Enocoro-128v2 has not been confirmed.
Performance Analysis	Enocoro-80 [26]: With implementation on Pentium 4, 1,335 cycles is required for initialization, and the throughput is 27 cycles/byte. Regarding hardware implementation (ASIC), the circuit size is 2.7K GE, and the processing speed is 2197.6 Mbps (180nm process and maximum frequency of 274.7 MHz). Enocoro-128v2 [22]: With implementation on Intel Core2 Duo, 1,530 cycles is required for initialization, and the throughput is 14.8 cycles/byte. Regarding hardware implementation (ASIC), the circuit size is 2.4K GE, and processing speed is 6,250 Mbps (90-nm process and maximum frequency of 781.3 MHz).
Standardization	CRYPTREC Candidate Recommended Cipher (Enocoro-128v2), ISO 29192-3 (Enocoro-80, Enocoro-128v2)

	Stream cipher
Name	Grain v1
Designers	Martin Hell, Thomas Johansson (Lund University) Willi Meier (FH Aargau)
Publication	2005 (eSTREAM Project)
Specifications	`http://www.ecrypt.eu.org/stream/e2-grain.html`
Features	This is a hardware-oritented stream cipher and consists of two algorithms selected for the eSTREAM portfolio: one with an 80-bit key and a 64-bit IV and the other with a 128-bit key and an 80-bit IV. It comprises one bitwise linear feedback shift register and one non-linear feedback shift register. Among other lightweight stream ciphers, this stream cipher is superior for lightweight hardware implementation. It can achieve certain degrees of parallelism, and its software implementation performs sufficiently for practical uses.
Security Analysis	Berbain et al. evaluated Grain, which is the forerunner of Grain v1, and reported that an 80-bit secret key can be recovered with a key stream of 2^{38} bits and a complexity of 2^{43} [5]. Since Grain v1 is the version with an improved algorithm based on the above report, the attack attempted by Berbain et al. is not directly applicable. Dinur et al. have pointed out that Grain v1 with a 128-bit key has insufficient initialization and has a weak key space of 118 bits [11]. This weak key space is large, but the key recovery attack requires complexity of around 2^{103}.
Performance Analysis	Good et al. [16] provide a detailed evaluation of hardware implementation performance of Grain v1. This evaluation uses a 0.13μm standard cell library. 80-bit: The circuit size is 1,294 GE, the maximum operating frequency is 724.6 MHz, and the throughput is 724.6 Mbps. The cipher is designed to allow up to 16 rounds to be carried out in parallel. 128-bit: The circuit size was 1,857 GE, the maximum operating frequency was 925.9 MHz, and the throughput was 925.9 Mbps. The cipher is designed to allow up to 32 rounds to be carried out in parallel.
Standardization	Grain-128a, which is an authenticated encryption based on Grain v1, has been standardized in ISO/IEC 29167-13 [14].

	Stream cipher
Name	MICKEY 2.0
Designers	Steve Babbage (Vodafone), Matthew Dodd (Independent consultant)
Publication	2005 (eSTREAM Project)
Specifications	`http://www.ecrypt.eu.org/stream/mickeypf.html`
Features	This is a hardware-oriented stream cipher selected in the eSTREAM portfolio. It accepts an 80-bit key and an 80-bit IV. The number of available IVs for a fixed key is limited up to 2^{40}. In addition, the maximum length of a keystream for a pair of key and IV is 2^{40} bits. This srteam cipher comprises one linear feedback shift register and one non-linear feedback shift register, featuring irregular clock control. Parallel processing is difficult due to the clock control mechanism.
Security Analysis	For the security evaluation of MICKEY 2.0, only a self-evaluation exists and no vulnerability has been found.
Performance Analysis	Good et al. [16] provides a detailed evaluation of the MICKEY 2.0 hardware implementation performance. This evaluation uses a 0.13 μm standard cell library. MICKEY 2.0 has a circuit size of 3,188 GE, a maximum operating frequency of 454.5 MHz, and a throughput of 454.5 Mbps.
Open Source Information	The designers' code is freely available at `http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3source.zip`. It is also known that several researchers implemented MICKEY 2.0, but if open source code is made available is not known.

	Stream cipher
Name	Trivium
Designers	Christophe De Cannière, Bart Preneel (Katholieke Universiteit Leuven)
Publication	2005 (eSTREAM Project)
Specifications	`http://www.ecrypt.eu.org/stream/e2-trivium.html`
Features	This is a hardware-oriented stream cipher selected in the eSTREAM portfolio. It accepts an 80-bit key and an 80-bit IV. A keystream generated for each pair of key and IV is limited to 2^{64} bits. It has a unique design consisting of three serial non-linear feedback shift registers of different lengths. Based on bitwise operations, but having high levels of parallelism, this stream cipher features lightweight hardware implementation and high speeds in software implementation. However, it is not suitable for processing short data because of its long initialization time.
Security Analysis	As of 2016, no results compromising the security of Trivium have been found. For attacks recovering the internal state from the key stream, Maximov et al. reported that it was possible to recover the internal state from a keystream of $2^{61.5}$ bits with complexity similar to $2^{89.5}$ exhaustive search [21]. In addition, Fouque et al. conducted cube attacks on the initialization of Trivium, and reported that it was possible to recover the key up to 799 steps out of 1,152 steps with 2^{40} IVs (and a corresponding keystream) and 2^{62} exhaustive search [15].
Performance Analysis	Good et al. [16] evaluated in detail the hardware implementation performance of Trivium. This evaluation uses a 0.13 μm standard cell library. Trivium has a circuit size of 2,580 GE, a maximum operating frequency of 327.9 MHz, and a throughput of 327.9 Mbps. In addition, up to 64 state bits of Trivium can be generated in parallel. In this execution, the circuit size is 4,921 GE and the throughput is 22,299.6 Mbps. The software implementation has been evaluated at FELICS [2]. The throughput of the ARM Cortex-M3 is 49.4 cycles/byte, which is faster than that of ChaCha20. However, this stream cipher is not suitable for processing short data because of its 7,195-cycle initialization.
Standardization	ISO/IEC 29192-3 [13]

Bibliography

[1] ebacs: Ecrypt benchmarking of cryptographic systems. `http://bench.cr.yp.to/results-stream.html`.

[2] Felics stream ciphers brief results. `https://www.cryptolux.org/index.php/FELICS_Stream_Ciphers_Brief_Results`.

[3] Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, and Christian Rechberger. New features of latin dances: Analysis of salsa, chacha, and rumba. In Kaisa Nyberg, editor, *Fast Software Encryption FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 470–488. Springer Berlin Heidelberg, 2008.

[4] Steve Babbage and Matthew Dodd. The Mickey stream ciphers. In M. Robshaw and editors O. Billet, editors, *New Stream Cipher Designs: The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 191–209. Springer, 2008.

[5] C. Berbain, H. Gilbert, and A. Maximov. Cryptanalysis of Grain. In M. Robshaw, editor, *Fast Software Encryption, FSE'06*, volume 4047 of *Lecture Notes in Computer Science*, pages 15–29. Springer-Verlag, 2006.

[6] Daniel J. Bernstein. Chacha, a variant of Salsa20. `https://cr.yp.to/chacha/chacha-20080128.pdf`.

[7] Daniel J. Bernstein. Chacha, a variant of Salsa20. In *The State of the Art of Stream Ciphers, SASC 2008*. ECRYPT, 2008.

[8] Elie Bursztein. Google security blog: Speeding up and strengthening https connections for chrome on android (April 24, 2014), 2014. `https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html`.

[9] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, ISC 2006*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer Berlin Heidelberg, 2006.

[10] Lin Ding, Chenhui Jin, and Jie Guan. Slide attack on standard stream cipher Enocoro-80 in the related-key chosen IV setting. In *Pervasive and Mobile Computing, Special Issue on Secure Ubiquitous Computing*, volume 24, pages 224–230, 2015.

[11] I. Dinur and A. Shamir. Breaking Grain-128 with dynamic cube attack. In A. Joux, editor, *Fast Software Encryption, FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 167–187. Springer, 2011.

[12] D. Watanabe, T. Owada, K. Okamoto, Y. Igarashi, and T. Kaneko. Update on Enocoro stream cipher. In *2010 International Symposium on Information Theory and its Applications, ISITA 2010*, 2010.

[13] International Oragnization for Standards. *ISO/IEC 29192-3:2012 Information technology – Security techniques – Lightweight cryptography – Part 3: Stream ciphers.* October 2012.

[14] International Oragnization for Standards. *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.* May 2015.

[15] Pierre-Alain Fouque and Thomas Vannet. Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In Shiho Moriai, editor, *Fast Software Encryption, FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 502–517. Springer Berlin Heidelberg, 2013.

[16] Tim Good and Mohammed Benaissa. Hardware performance of eStream phase-III stream cipher candidates. In *The State of the Art of Stream Ciphers, SASC 2008*, 2008.

[17] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The Grain family of stream ciphers. In M. Robshaw and editors O. Billet, editors, *New Stream Cipher Designs: The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2008.

[18] Yasutaka Igarashi, Kazuto Okamoto, and Toshinobu Kaneko. On key recovery for Enocoro-128v1.1 with weak key in related-key attack scenario. In *Proceedings of the IEICE General Conference*, 2010.

[19] Yasutaka Igarashi, Kazuto Okamoto, and Toshinobu Kaneko. On key recovery for Enocoro with weak key in related-key attack scenario. In *Technical report of IEICE. ISEC 109(445)*, pages 275–280, 2010.

[20] Subhamoy Maitra. Chosen IV Cryptanalysis on Reduced Round ChaCha and Salsa, July 2015. `https://eprint.iacr.org/2015/698`.

[21] Alexander Maximov and Alex Biryukov. Two trivial attacks on Trivium. In Carlisle Adams and Ali Miri Michael Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007*, volume 4876 of *Lecture Notes in Computer Science*, pages 36–55. Springer Berlin Heidelberg, 2007.

[22] Shugo Mikami and Dai Watanabe. Software and hardware implementation and evaluation of stream cipher Enocoro-128v2. In *Computer Security Symposium, CSS2012*, pages 742–748, 2012.

[23] Y. Nir and A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*, volume RFC4493 of *Request For Comments*. May. `https://tools.ietf.org/html/rfc7539`.

[24] ECRYPT Network of Excellence. The eStream project. `http://www.ecrypt.eu.org/stream/`.

[25] Zhenqing Shi, Bin Zhang, Dengguo Feng, and Wenling Wu. Improved key recovery attacks on reduced-round Salsa20 and ChaCha. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *ICISC'12 Proceedings of the 15th international conference on Information Security and Cryptology*, volume 7839 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag Berlin Heidelberg, 2011.

[26] D. Watanabe, K. Ideguchi, J. Kitahara, K. Muto, H. Furuichi, and T. Kaneko. Enocoro-80: A hardware oriented stream cipher. In *Second International Workshop on Advances in Information Security*, 2008.

4.3 Hash Functions

This section describes lightweight hash functions. It covers four algorithms presented at major international conferences or workshops – Keccak, PHOTON, QUARK, and SPONGENT. In terms of lightweight, for Keccak, this section targets smaller permutations, which are not selected as SHA-3. In addition, it should be noted that the implementation evaluation results have been taken from the proposal papers and are not obtained in the same evaluation environment.

<table>
<tr><td></td><td>Hash function</td></tr>
<tr><td>Name</td><td>Keccak</td></tr>
<tr><td>Designers</td><td>Guido Bertoni (STMicroelectronics), Joan Daemen (STMicroelectronics), Michaël Peeters (NXP Semiconductors), Gilles Van Assche (STMicroelectronics)</td></tr>
<tr><td>Publication</td><td>2008 (NIST SHA-3 Competition)</td></tr>
<tr><td>Specifications</td><td>http://keccak.noekeon.org/</td></tr>
<tr><td>Features</td><td>Keccak is a family of sponge functions. Seven permutations are defined and indicated by Keccak-f[b] (b ∈ 25, 50, 100, 200, 400, 800, 1,600). From the viewpoint of lightweight cryptography, the schemes using Keccak-f[100], Keccak-f[200], and Keccak-f[400] will be described.
<table>
<tr><td>Keccak-f[b]</td><td>n</td><td>r</td><td>r’</td></tr>
<tr><td>Keccak-f[100]</td><td>80</td><td>20</td><td>20</td></tr>
<tr><td>Keccak-f[200]</td><td>64</td><td>72</td><td>72</td></tr>
<tr><td>Keccak-f[400]</td><td>128</td><td>144</td><td>144</td></tr>
</table>
∗ n: output length, r: input block length, r’ : output block length</td></tr>
<tr><td>Security Analysis</td><td>Many papers have analyzed Keccak, and no critical vulnerability has been reported.</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation[6](130nm process)
<table>
<tr><td></td><td>Area [GE]</td><td>Latency [clk]</td><td>Throughput [kbps]</td></tr>
<tr><td>Keccak-f[100]</td><td>1250</td><td>800</td><td>2.5</td></tr>
<tr><td>Keccak-f[200]</td><td>2520</td><td>900</td><td>8.00</td></tr>
<tr><td>Keccak-f[400]</td><td>5090</td><td>1000</td><td>14.40</td></tr>
</table></td></tr>
<tr><td>Standardization</td><td>The scheme using Keccak-f[1600] is standardized by NIST in FIPS202 [3]. For SHA-3 derived functions, a series of special publication is available by NIST SP800-185 [4].</td></tr>
<tr><td>Market Deployment</td><td>SHA-3 is being adopted in many different applications.
http://csrc.nist.gov/groups/STM/cavp/documents/sha3/sha3val.html,
http://www.3gpp.org/DynaReport/35-series.htm.</td></tr>
<tr><td>Open Source Information</td><td>http://keccak.noekeon.org/files.html,
https://github.com/gvanas/KeccakCodePackage</td></tr>
</table>

<table>
<tr><td></td><td>Hash function</td></tr>
<tr><td>Name</td><td>PHOTON</td></tr>
<tr><td>Designers</td><td>Jian Guo (Institute for Infocomm Research, Singapore), Thomas Peyrin (Nanyang Technological University, Singapore), Axel Poschmann (Nanyang Technological University, Singapore)</td></tr>
<tr><td>Publication</td><td>2011 (CRYPTO2011)</td></tr>
<tr><td>Specifications</td><td>https://sites.google.com/site/photonhashfunction/</td></tr>
<tr><td>Features</td><td>PHOTON has a sponge structure and a variety of configurations according to the internal parameters. ISO/IEC 29192 indicates five variations (see table below). An AES-like internal permutation repeats four steps, “AddConstants”, “SubCells”, “ShiftRows”, and “MixColumnsSerial” for 12 rounds. The 4-bit PRESENT S-box is used in “SubCells”.
<table>
<tr><td>PHOTON-n/r/r’</td><td>n</td><td>r</td><td>r’</td></tr>
<tr><td>PHOTON-80/20/16</td><td>80</td><td>20</td><td>16</td></tr>
<tr><td>PHOTON-128/16/16</td><td>128</td><td>16</td><td>16</td></tr>
<tr><td>PHOTON-160/36/36</td><td>160</td><td>36</td><td>36</td></tr>
<tr><td>PHOTON-224/32/32</td><td>224</td><td>32</td><td>32</td></tr>
<tr><td>PHOTON-256/32/32</td><td>256</td><td>32</td><td>32</td></tr>
</table>
∗ n: output length, r: input block length, r’ : output block length</td></tr>
<tr><td>Security Analysis</td><td>No critical vulnerability has been reported. However, since there have been only a few analytical results, there is a possibility that potential vulnerability exists.</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation[5](180nm process)
<table>
<tr><td></td><td>Area [GE]</td><td>Latency [clk]</td><td>Throughput [kbps]</td></tr>
<tr><td>PHOTON-80/20/16</td><td>865/1168</td><td>708/132</td><td>2.82/15.15</td></tr>
<tr><td>PHOTON-128/16/16</td><td>1122/1708</td><td>996/156</td><td>1.61/10.26</td></tr>
<tr><td>PHOTON-160/36/36</td><td>1396/2117</td><td>1332/180</td><td>2.70/20.00</td></tr>
<tr><td>PHOTON-224/32/32</td><td>1735/2786</td><td>1716/204</td><td>1.86/15.69</td></tr>
<tr><td>PHOTON-256/32/32</td><td>2177/4362</td><td>996/156</td><td>3.21/20.51</td></tr>
</table>
Software implementation[5] (Intel Core i7 @1.6GHz)
<table>
<tr><td></td><td>32bit optimized implementation [cycles/byte]</td></tr>
<tr><td>PHOTON-80/20/16</td><td>95</td></tr>
<tr><td>PHOTON-128/16/16</td><td>156</td></tr>
<tr><td>PHOTON-160/36/36</td><td>116</td></tr>
<tr><td>PHOTON-224/32/32</td><td>227</td></tr>
<tr><td>PHOTON-256/32/32</td><td>135</td></tr>
</table></td></tr>
<tr><td>Standardization</td><td>ISO/IEC 29192-5 (PHOTON-100, 140, 196, 256, 288)</td></tr>
<tr><td>Open Source Information</td><td>https://sites.google.com/site/photonhashfunction/downloads</td></tr>
</table>

<table>
<tr><td></td><td>Hash function</td></tr>
<tr><td>Name</td><td>QUARK</td></tr>
<tr><td>Designers</td><td>Jean-Philippe Aumasson (Nagravision SA, Cheseaux, Switzerland),
Luca Henzen (ETH Zurich, Switzerland), Willi Meier (FHNW, Windisch, Switzerland), Maria Naya-Plasencia (FHNW, Windisch, Switzerland)</td></tr>
<tr><td>Publication</td><td>2010 (CHES2010)</td></tr>
<tr><td>Specifications</td><td>http://aumasson.jp/quark/</td></tr>
<tr><td>Features</td><td>QUARK has a sponge structure and a variety of configurations according to the internal parameters. QUARK family is composed of the three instances U-QUARK, D-QUARK, and S-QUARK. The permutation is designed by combining the advantages of the stream cipher Grain and the block cipher KATAN. The numbers of rounds are 544, 704, and 1024, respectively.
<table>
<tr><td></td><td>n</td><td>r</td><td>r’</td></tr>
<tr><td>U-QUARK</td><td>128</td><td>8</td><td>8</td></tr>
<tr><td>D-QUARK</td><td>160</td><td>16</td><td>16</td></tr>
<tr><td>S-QUARK</td><td>224</td><td>32</td><td>32</td></tr>
</table>
∗ n: output length, r: input block length, r’ : output block length</td></tr>
<tr><td>Security Analysis</td><td>No critical vulnerability has been reported. However, since there have been only a few analytical results, there is a possibility that potential vulnerability exists.</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation[1](180nm process)
<table>
<tr><td></td><td>Area [GE]</td><td>Latency [clk]</td><td>Throughput [kbps]</td></tr>
<tr><td>U-QUARK</td><td>1379/2392</td><td>544/68</td><td>1.47/11.76</td></tr>
<tr><td>D-QUARK</td><td>1702/2819</td><td>704/88</td><td>2.27/18.18</td></tr>
<tr><td>S-QUARK</td><td>2296/4640</td><td>1024/64</td><td>3.13/50.00</td></tr>
</table></td></tr>
<tr><td>Market Deployment</td><td>QUARK or its variants are deployed in products of at least two companies.</td></tr>
<tr><td>Open Source Information</td><td>http://aumasson.jp/quark/
https://github.com/veorq/Quark</td></tr>
</table>

<table>
<tr><td></td><td>Hash function</td></tr>
<tr><td>Name</td><td>SPONGENT</td></tr>
<tr><td>Designers</td><td>Andrey Bogdanov (KU Leuven, Belgium), Miroslav Knežević (NXP Semiconductors, Belgium), Gregor Leander (Technical University of Denmark, Denmark), Deniz Toz (KU Leuven, Belgium), Kerem Varıcı(KU Leuven, Belgium), Ingrid Verbauwhede (KU Leuven, Belgium)</td></tr>
<tr><td>Publication</td><td>2011 (CHES2011)</td></tr>
<tr><td>Specifications</td><td>https://sites.google.com/site/spongenthash/</td></tr>
<tr><td>Features</td><td>SPONGENT has a sponge structure using a PRESENT-type permutation and a variety of configurations according to the internal parameters. The designers have indicated 13 variations (see table below), and five have been standardized as ISO/IEC 29192-5 ($\star$ in table).
<table>
<tr><th>SPONGENT-n/c/r</th><th>n</th><th>c</th><th>r</th></tr>
<tr><td>SPONGENT-88/80/8$\star$</td><td>88</td><td>80</td><td>8</td></tr>
<tr><td>SPONGENT-88/176/88</td><td>88</td><td>176</td><td>88</td></tr>
<tr><td>SPONGENT-128/128/8$\star$</td><td>128</td><td>128</td><td>8</td></tr>
<tr><td>SPONGENT-128/256/128</td><td>128</td><td>256</td><td>128</td></tr>
<tr><td>SPONGENT-160/160/16$\star$</td><td>160</td><td>160</td><td>16</td></tr>
<tr><td>SPONGENT-160/160/80</td><td>160</td><td>160</td><td>80</td></tr>
<tr><td>SPONGENT-160/320/160</td><td>160</td><td>320</td><td>160</td></tr>
<tr><td>SPONGENT-224/224/16$\star$</td><td>224</td><td>224</td><td>16</td></tr>
<tr><td>SPONGENT-224/224/112</td><td>224</td><td>224</td><td>112</td></tr>
<tr><td>SPONGENT-224/448/224</td><td>224</td><td>448</td><td>224</td></tr>
<tr><td>SPONGENT-256/256/16$\star$</td><td>256</td><td>256</td><td>16</td></tr>
<tr><td>SPONGENT-256/256/128</td><td>256</td><td>256</td><td>128</td></tr>
<tr><td>SPONGENT-256/512/256</td><td>256</td><td>512</td><td>256</td></tr>
</table>
$*$ n: output length, c: capacity, r: rate (input block length)</td></tr>
<tr><td>Security Analysis</td><td>No critical vulnerability has been reported. However, since there have been only a few analytical results, there is a possibility that potential vulnerability exists.</td></tr>
<tr><td>Performance Analysis</td><td>Hardware implementation[2](130nm process)
<table>
<tr><th></th><th>Area [GE]</th><th>Latency [clk]</th><th>Throughput [kbps]</th></tr>
<tr><td>SPONGENT-88/80/8</td><td>738/1127</td><td>990/45</td><td>0.81/17.78</td></tr>
<tr><td>SPONGENT-128/128/8</td><td>1060/1687</td><td>2380/70</td><td>0.34/11.43</td></tr>
<tr><td>SPONGENT-160/160/16</td><td>1329/2190</td><td>3960/90</td><td>0.40/17.78</td></tr>
<tr><td>SPONGENT-224/224/16</td><td>1728/2903</td><td>7200/120</td><td>0.22/13.33</td></tr>
<tr><td>SPONGENT-256/256/16</td><td>1950/3281</td><td>9520/140</td><td>0.17/11.43</td></tr>
</table></td></tr>
<tr><td>Standardization</td><td>ISO/IEC 29192-5:2016 (SPONGENT-88, 136, 176, 240, 272)
http://www.iso.org/iso/catalogue_detail.htm?csnumber=67173</td></tr>
<tr><td>Open Source Information</td><td>https://sites.google.com/site/spongenthash/downloads</td></tr>
</table>

Bibliography

[1] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A lightweight hash. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pages 1–15, 2010.

[2] Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. spongent: A lightweight hash function. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 312–325, 2011.

[3] NIST Computer Security Division. Sha-3 standard: Permutation-based hash and extendable-output functions. FIPS Publication 202, National Institute of Standards and Technology, U.S. Department of Commerce, Aug 2015.

[4] NIST Computer Security Division. Sha-3 derived functions: cshake, kmac, tuplehash, and parallelhash. NIST Special Publication 800-185, National Institute of Standards and Technology, U.S. Department of Commerce, Dec 2016.

[5] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 222–239, 2011.

[6] Elif Bilge Kavun and Tolga Yalçin. A lightweight implementation of keccak hash function for radio-frequency identification applications. In *Radio Frequency Identification: Security and Privacy Issues - 6th International Workshop, RFIDSec 2010, Istanbul, Turkey, June 8-9, 2010, Revised Selected Papers*, pages 258–269, 2010.

4.4 Message Authentication Codes

This section describes lightweight message authentication codes (MACs). General-purpose MACs include the mode of block ciphers (CMAC [9]) and the mode of hash functions (HMAC [10, 5]). Since the overheads of the CMAC and the HMAC are not high, it is possible to configure lightweight MACs by combining the lightweight block ciphers described in Section 4.1 with the lightweight hash functions described in Section 4.3. The HMAC mode calls a hash function twice; therefore, when processing a very short message, using the CMAC may be more efficient.

Focusing on software performance, the FELICS project (Fair Evaluation of Lightweight Cryptographic Systems) [4] which benchmarks lightweight cryptography is useful for selecting block ciphers and hash functions. The FELICS project compares many algorithms (block ciphers, stream ciphers, and hash functions) on the Atmel AVR ATmega128 (8-bit) microcontroller, the Texas Instruments MSP430F1611 (16-bit) microcontroller, and the Arduino Due ARM Cortex-M3 (32-bit) board.

Unlike block cipher modes and hash function modes, specially designed lightweight MACs are not well known. A pseudo-random function specific to processing short messages, SipHash [2, 1], is widely used as a MAC. However, since SipHash uses 64-bit addition for internal processing, it is lightweight and fast on high-end CPUs, but it is not suitable for use on 8- to 32-bit CPUs.

Chaskey [8, 6] lightweight MAC targets low-end CPUs. The FELICS project has classified Chaskey as a lightweight block cipher and it has recorded the highest scores in many categories. However, it has been reported that key recovery attack is possible to seven out of eight rounds [8, 6]. Therefore, it has a small security margin. To improve this, Chaskey-12 [7] has been proposed, which has increased the number of rounds from 8 to 12.

This section describes only SipHash, whose security is mature. The CMAC and the HMAC modes are not covered here. The Chaskey algorithm has not undergone a sufficient number of reviews and is not discussed.

	Message authentication code
Name	SipHash
Designers	Jean-Philippe Aumasson (Kudelski Security, Switzerland), Daniel J. Bernstein (University of Illinois at Chicago, USA)
Publication	2012 (INDOCRYPT 2012)
Specifications	[2, 1]
Features	SipHash is a keyed hash function with a key length of 128 bits and an output length of 64 bits developed for associative arrays. The upper limit of the length of an input message is 2,039 bytes, which is shorter than that of general-purpose hash functions. The SipHash algorithm consists of a compression phase of c rounds and a final process phase of d rounds, and indicated as SipHash-c-d. SipHash-2-4 with $c = 2$ and $d = 4$ is generally used. The algorithm consists of arithmetic additions, exclusive ORs, and cyclic shifts of 64-bit words and runs fast on a CPU that supports 64-bit operations. In addition, since the algorithm has no loop up table, straightforward implementation is secure against cache-timing attacks.
Security Analysis	As of 2016, no vulnerability has been found on SipHash. Dobraunig et al. [3] found a differential path with a differential characteristic probability of $2^{-236.3}$. However, since SipHash has a key length of 128 bits, brute-force key attacks are much more efficient. Therefore, the result does not compromise the security of SipHash.
Performance Analysis	According to the proposal paper [2], the throughput of SipHash is 1.5 to 3.0 cycles/byte on amd64 architecture. Because of relatively heavy finalization, the processing speed slows down for short data; For example 10 to 30 cycles/byte for 8-byte data.
Market Deployment	See `http://aumasson.jp/siphash/#us`, SipHash is used in many systems, including the Python and Rust languages, in the FreeBSD and OpenBSD operating systems, in the Wireguard VPN, and so on.
Open Source Information	Open-source reference code is available at `https://github.com/veorq/siphash`, and there are a multitude of third-party implementations, see `http://aumasson.jp/siphash/#sw`.

Bibliography

[1] Jean-Philippe Aumasson. Siphash: a fast short-input prf. https://131002.net/siphash/.

[2] Jean-Philippe Aumasson and Daniel J. Bernstein. Siphash: A fast short-input prf. In Steven Galbraith and Mridul Nandi, editors, *Progress in Cryptology – INDOCRYPT 2012*, volume 7668 of *Lecture Notes in Computer Science*, pages 489–508. Springer-Verlag Berlin Heidelberg, 2012.

[3] Christoph Dobraunig, Florian Mendel, and Martin Schläffer. Differential cryptanalysis of siphash. In Antoine Joux and Amr Youssef, editors, *Selected Areas in Cryptography – SAC 2014*, volume 8781 of *Lecture Notes in Computer Science*, pages 165–182. Springer International Publishing, 2014.

[4] FELICS. Fair evaluation of lightweight cryptographic systems. https://www.cryptolux.org/index.php/FELICS.

[5] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*, volume RFC2104 of *Request For Comments*. February. https://tools.ietf.org/html/rfc2104.

[6] Nicky Mouha. Chaskey. http://mouha.be/chaskey/.

[7] Nicky Mouha. Chaskey: a mac algorithm for microcontrollers – status update and proposal of chaskey-12 –. http://eprint.iacr.org/2015/1182.

[8] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, , and Ingrid Verbauwhede. Chaskey: An efficient mac algorithm for 32-bit microcontrollers. In Antoine Joux and Amr Youssef, editors, *Selected Areas in Cryptography – SAC 2014*, volume 8781 of *Lecture Notes in Computer Science*, pages 306–323. Springer, 2014.

[9] National Institute of Standards and Technology. *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. NIST Special Publication 800-38B. May 2005. http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf.

[10] National Institute of Standards and Technology. *The Keyed-Hash Message Authentication Code (HMAC)*. Federal Information Processing Standards Publication FIPS 198-1. July 2008. http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf.

4.5 Authenticated Encryption

This section focuses on the schemes proposed at the CAESAR competitions [6] featuring lightweight for which no security problems have been found (as of July 28, 2016) and describes the outline. We also include some recent results found after the above date for information. Many schemes use a block cipher or tweakable block cipher. For these schemes, a rate will be introduced as an index for measuring theoretical speeds. Rate indicates the number of input blocks that can be processed by a single call of a block cipher. The software implementation evaluation value is, unless otherwise specified, the result of processing a sufficiently long message on the Supercop Benchmark System [7] in eBACS. Similarly, the hardware implementation evaluation value is, unless otherwise specified, the result of the ATHENA Benchmark System [12]. The software evaluation scale is the number of processing cycles per byte (cycles/byte, or C/B) for a sufficiently long message. The hardware evaluation scales are the number of slices on an FPGA (slices) and the maximum operating frequency (fmax). In the case of ASIC hardware implementation, the gate equivalent (GE) is used as the scale for evaluating the size. In addition, some notable implementation cases, which are not targets of these official benchmarks, will be reported as required. In any case, it should be noted that the results could change greatly based on whether the implementation is optimized and the degree of optimization. The affiliations of the authors are at the time of proposal. On August 15, 2016, the 3rd round candidates of CAESAR were announced, and the number of candidates was narrowed to 15. The schemes belonging to the 3rd round candidates are noted.

	Authenticated encryption
Name	ACORN
Designers	Hongjun Wu (Nanyang Technological University, Singapore)
Publication	2014 (DIAC 2014 [41])
Specifications	CAESAR Website [6]
Features	This scheme uses LFSRs and simple non-linear operations. It has a simple structure similar to that of Grain and Trivium, hardware-oriented stream ciphers. The length of the key is 128 bits, and the state size is 293 bits. There are six LFSRs being concatenated in ACORN-128. Similar to Grain and Trivium, this scheme is suitable for hardware implementation. This scheme is a CAESAR third-round candidate.
Security Analysis	Salam et al. have reported an attack against a variant of ACORN with reduced preprocessing. They also reported an attack under nonce-misuse scenario, which is beyond the designer's security claim [36]. The latter has been reported to have a computational complexity of $2^{72.8}$. There are also other analytical results (e.g., [26]). However, they were structural analyses with known keys or attacks with computational complexities larger than those of brute-force attacks, thus they were not directly indicating the vulnerabilities of the scheme. In December 2016, Roy and Mukhopadhyay posted a practical initial-state-recovery attack against ACORN which requires computational complexity around 2^{40} on ePrint archive [35]. We remark that this result is yet to be verified.
Performance Analysis	(SW) 8.46 C/B on Intel Core i5-6600 (Skylake 3.31 GHz) (HW) 135 slices and fmax of 389 MHz on Virtex 6.

	Authenticated encryption
Name	Ascon
Designers	Christoph Dobraunig, Maria Eichlseder, Florian Mendel, Martin Schlaffer (Graz University of Technology, Austria)
Publication	2014 (DIAC 2014 [14])
Specifications	CAESAR Web site [6] and Official Website: `http://ascon.iaik.tugraz.at`
Features	This scheme has a sponge structure that was introduced to the SHA-3 hash function. The modes of operation are based on MonkeyDuplex [13]. The internal cryptographic permutation is 320-bit wide. It is based on a 5-bit S-box and has a structure intended for bitslice implementation. This scheme aims at lightweight in both software and hardware implementations. This scheme is a CAESAR third-round candidate.
Security Analysis	The designers evaluated the security when the internal permutation has a reduced number of rounds [15]. If the permutation is reduced to five rounds (from the full-spec, 12 rounds), then key recovery is possible with practical computational complexity of $O(2^{35})$. Similarly, if it is reduced to six rounds, the attack complexity is reported to be $O(2^{66})$. No successful attacks against the full-spec have been reported.
Performance Analysis	(SW) 11.51 C/B on Intel Core i5-6600 (Skylake 3.31 GHz). (HW) 413 slices and fmax of 347.0 MHz on Virtex 6. 7,950 GE at 5.5 Mbps on ASIC implementation including I/O circuits. [16].

	Authenticated encryption
Name	AES-JAMBU
Designers	Hongjun Wu, Tao Huang (Nanyang Technological University, Singapore)
Publication	2014 (DIAC 2014 [42])
Specifications	CAESAR Website [6]
Features	This is a block cipher mode of operation. It uses AES-128 or SIMON [4] for the internal block ciphers. For SIMON, three versions of block size/key length (bit), 64/96, 96/96, and 128/128 are specified. In addition to block cipher's I/O state, half of the block size is used as a state variable to perform processing, which is serial. In each block cipher invocation, half of the block size is encrypted. Since the size of the state variable is small, this scheme is suitable for small hardware applications. This scheme is a CAESAR third-round candidate.
Security Analysis	Unlike most of the general block cipher modes of operation, the designers did not present a security reduction based on computational security of the underlying block cipher, except the latest specification document showing integrity analysis JAMBUv2.1. The designers argue that for a $2n$-bit block cipher with a k-bit key, the security of encryption is k bits and the security of authentication is n bits. Peyrin et al. [31] have reported attacks by $2^{n/2}$ encryptions in the nonce-misuse scenario and attacks with computational complexity of $2^{3n/2}$ against the security of CCA2 [5] in the nonce-respecting scenario.
Performance Analysis	(SW) 5.71 C/B on Intel Core i5-6600 (Skylake 3.31 GHz) using AES-128. (HW) 453 slices and fmax of 209.8 MHz on Virtex 6.

	Authenticated encryption
Name	AES-OTR
Designers	Kazuhiko Minematsu (NEC, Japan)
Publication	2014 (Eurocrypt 2014 [28])
Specifications	CAESAR Website [6], Official Website: `http://jpn.nec.com/rd/crl/code/research/otr_en.html`
Features	This is a block cipher mode of operation. The CAESAR proposal uses AES. It has a structure similar to OCB. It uses two-round Feistel permutation and can perform processing with a computational complexity similar to that of encryption. Parallel processing is possible. Unlike OCB, decryption for authenticated encryption can be executed only with the AES encryption function without using AES decryption. This scheme is a CAESAR third-round candidate.
Security Analysis	The proposal paper indicated that the security of OTR can be reduced to the pseudorandomness of the block cipher. This algorithm has provable security of $n/2$ bits when a n-bit block cipher is used. Bost et al. [11] pointed out an inconsistency with the security proof for internal mask generation, and the designer has proposed a revised version.
Performance Analysis	(SW) 0.68 C/B on Intel Core i5-6600 (Skylake 3.31 GHz) using AES-128. (HW) 1,385 slices and fmax of 256.9 MHz on Virtex 6. ARM v7 Implementation [29]: 23.5 C/B (42.5 Mbyte/sec) on a 1-GHz Cortex-A8 microprocessor board. ASIC Implementation by Banik et al. [3]: On the order of 6,000 GE under special conditions including partial use of external memory and restrictions on the input length.
Market Deploy-ment	NEC Solution Innovators for Embedded Automotive Solutions (trial production, Japanese site) `http://www.nec-solutioninnovators.co.jp/sl/emb/automotive/`

	Authenticated encryption
Name	CLOC and SILC
Designers	Tetsu Iwata (Nagoya University, Japan), Kazuhiko Minematsu (NEC, Japan), Jian Guo (Nanyang Technological University, Singapore), Sumio Morioka (NEC Europe, United Kingdom), Eita Kobayashi (NEC, Japan)
Publication	2014 (FSE 2014 [18], DIAC 2014 [19])
Specifications	CAESAR Website [6], Official Website: `http://www.nuee.nagoya-u.ac.jp/labs/tiwata/AE/`
Features	These are block cipher modes of operation. They are rate 1/2 schemes based on CFB and CBC-MAC and have small memory requirements except for the key (approximately $2n$ bits when a n-bit block cipher is used). CLOC was developed to improve performance for short inputs by reducing process overhead and is suitable for embedded software. SILC is a variant of CLOC designed to be suitable for hardware implementation by simplifying the internal process of CLOC. This scheme is a CAESAR third-round candidate. Both CLOC and SILC use AES for the 128-bit block cipher. CLOC is also defined with a 64-bit block cipher TWINE [39]. SILC is also defined with 64-bit block ciphers, PRESENT [10] and LED [17].
Security Analysis	The proposal paper indicated that the security of CLOC and SILC can be reduced to the pseudorandomness of the block cipher used. They have a security of $n/2$ bits when a n-bit block cipher is used. Even if nonce is reused during encryption by mistake, security against ciphertext forgery is guaranteed.
Performance Analysis	(SW) 2.82 C/B for CLOC and 2.78 C/B for SILC on Intel Core i5-6600 (Skylake 3.31 GHz). (HW) 891 slices and fmax of 280.9 MHz for CLOC, and 989 slices and fmax of 280.7 MHz for SILC on Virtex 6. Implementation of CLOC on an 8-bit Microprocessor [18]: AVR ATmega128 (16 MHz). 2,000 cycles for initialization and 550 C/B for 32-byte encryption. ASIC Implementation by Banik et al. [3]: Approximately 3,100 GE for both CLOC-AES and SILC-AES with implementation under special conditions including partial use of external memory and restrictions on the input length.

	Authenticated encryption
Name	Deoxys
Designers	Jérémy Jean, Ivica Nikolić, Thomas Peyrin (Nanyang Technological University, Singapore)
Publication	2014 (DIAC 2014 [22], Asiacrypt 2014 [23])
Specifications	CAESAR Website [6] and Official Website: `http://www1.spms.ntu.edu.sg/~syllab/Deoxys/`
Features	This is a block cipher mode of operation using a dedicated tweakable block cipher, Deoxys-BC. Deoxys-BC is a 128-bit block cipher and has a 256-bit tweak+key. The round function of Deoxys-BC is identical to that of AES, and the number of rounds is 14 to 16. There are two types of block cipher modes of operation: TAE [27] and SCT [30]. When the TAE mode is used, the algorithm has 128-bit security. The TAE mode operates similar to OCB, and parallel processing is possible with rate 1. However, SCT is a 2-path rate 1/2 offline process. Since the SCT mode functions as a deterministic (or misuse-resistant) authenticated-encryption [34], it is secure against nonce reuse. This scheme is a CAESAR third-round candidate.
Security Analysis	The proposal [20] indicates that the security can be reducible to the security of Deoxys-BC, and the algorithm has security of 128 bits in the TAE mode. The security of the SCT mode has been proven in a recent paper [30].
Performance Analysis	(SW) 0.87 C/B on Intel Core i5-6600 (Skylake 3.31 GHz). (HW) 993 slices and fmax of 330 MHz on Virtex 6. 2,860 GE with ASIC implementation of Deoxys-BC alone [24].
Open Source Information	Hardware implementations on the ATHENA website.

	Authenticated encryption
Name	Joltik
Designers	Jérémy Jean, Ivica Nikolić, Thomas Peyrin (Nanyang Technological University, Singapore)
Publication	2014 (DIAC 2014 [22], Asiacrypt 2014 [23])
Specifications	CAESAR Website [6] and Official Website: `http://www1.spms.ntu.edu.sg/~syllab/Joltik/`
Features	This scheme uses a dedicated tweakable block cipher, Joltik-BC. Joltik-BC is a 64-bit block cipher and has a 128-bit tweak+key. The round function has the SPN structure using 4-bit S-boxes. The number of rounds is 24 to 32. Similar to Deoxys, there are two types of modes: TAE and SCT.
Security Analysis	The proposal [21] indicates that the security is reducible to the security of Joltik-BC, and the algorithm has security of 64 bits in the TAE mode.
Performance Analysis	(SW) 13.32 C/B on Intel Core i5-6600 (Skylake 3.31 GHz). (HW) 494 slices and fmax of 430 MHz on Virtex 6
Open Source Information	Hardware implementations on the ATHENA website.

	Authenticated encryption
Name	Ketje
Designers	Guido Bertoni[1], Joan Daemen[1], Michael Peeters[2], Gilles Van Assche[1], Ronny Van Keer[1] (1: STMicroelectronics2: NXP Semiconductors)
Publication	2014 (DIAC 2014 [9])
Specifications	CAESAR Website [6] and Official Website: `http://ketje.noekeon.org/`
Features	This scheme has a sponge structure. The mode of operation is based on MonkeyDuplex [13]. The internal cryptographic permutation, called Keccak-p, is based on the Keccak-f permutation [8] used in the SHA-3 function. The permutation using 200-bit width is called Ketje-JR, and the permutation with 400-bit width is called Ketje-SR. This scheme is designed to be lightweight in both software and hardware implementation because of its small memory size and low computational complexity. This scheme is a CAESAR third-round candidate.
Security Analysis	There is a security proof under the public random permutation model (i.e. assuming the internal permutation as a publicly accessible random permutation) [13]. The designers state that the Keccak-p permutation is secure because most of the results of security evaluation of Keccak-f can be inherited.
Performance Analysis	(SW) Ketje-SR, 42.57 C/B on Intel Core i5-6600 (Skylake 3.31 GHz). (HW) 456 slices and fmax of 229.5 MHz on Virtex 6.
Standardization	Ketje is submitted to the CAESAR competition and has been selected for the third round of the competition.
Open Source Information	`https://github.com/gvanas/KeccakCodePackage`

	Authenticated encryption
Name	Minalpher
Designers	Yu Sasaki[1], Yosuke todo[1], Kazumaro Aoki[1], Yusuke Naito[2], Takeshi Sugawara[2], Yumiko Murakami[2], Mitsuru Matsui[2], Shoichi Hirose[3] (1: NTT, Japan, 2: Mitsubishi Electric Corporation, Japan, 3: University of Fukui, Japan)
Publication	2014 (DIAC 2014 [38])
Specifications	CAESAR Website [6] and Official Website: `http://info.isl.ntt.co.jp/crypt/minalpher/index-j.html/`
Features	This scheme uses a dedicated 256-bit tweakable block cipher based on the Even-Mansour approach, called TEM. The TEM is based on a 256-bit cryptographic permutation, Minalpher-P. A new mode of operation has been developed for Minalpher. The internal cryptographic permutation for TEM has the SPN structure using 4-bit S-boxes that facilitates the integration of the encryption and decryption functions. It has security against nonce reuse, however, the protection is partial.
Security Analysis	It has been demonstrated that the security is reducible to the security of the TEM block cipher, and the scheme has 128-bit security. Minalpher-P has a total of 17.5 rounds. For security evaluation, the CAESAR proposal [37] indicates that when the number of rounds is reduced to 5.5, theoretical attacks are possible.
Performance Analysis	(SW) 5.81 C/B on Intel Core i5-6600 (Skylake 3.31 GHz). (HW) 1,104 slices and fmax of 280.9 MHz on Virtex 6. SIMD Implementation [37]: 5.6 C/B on Intel CPU Core i7 (Haswell) Implementation on an 8-bit RL78 microprocessor [37]: Approximately 2,800 C/B with 514-byte ROM and 214-byte RAM, and 514 C/B with 1,275-byte ROM and 470-byte RAM
Standardization	CAESAR: Competition for Authenticated Encryption `https://competitions.cr.yp.to/caesar-submissions.html`

Open Source Information	Reference code - Reference implementation in C for version 1.1.
	`https://info.isl.ntt.co.jp/crypt/minalpher/files/minalpherv11.tgz`
	- Reference implementation in C.
	`https://info.isl.ntt.co.jp/crypt/minalpher/files/minalpherv1.tar.gz`
	Hardware code - Hardware implementation for version 1.1.
	`https://info.isl.ntt.co.jp/crypt/minalpher/files/minalpherv11_MinalpherTeam.zip`

	Authenticated encryption
Name	OCB
Designers	Ted Krovetz (California State University, U.S.), Phillip Rogaway (Univ. of California, Davis, U.S.)
Publication	2001 (ACM CCS 2001 [33]), 2004 (ASIACRYPT 2004 [32]), 2011 (FSE 2011 [25])
Specifications	CAESAR Website [6] and Official Website: `http://web.cs.ucdavis.edu/~rogaway/ocb/`
Features	This is a block cipher mode of operation. The version using AES has been specified in RFC 7253. The CAESAR proposal is identical to RFC 7253. This scheme has a structure similar to the ECB mode. However, message authentication is realized by taking a checksum of the plain text block (obtaining an exclusive XOR), and encrypting the result. Therefore, the entire computational complexity is close to that of the algorithm with encryption-only modes. In addition, each block encryption can be parallelized. The basic structure was proposed in 2001, and several versions have been proposed with differences mainly in mask generations. In the case of using AES, the algorithm is significantly fast on CPUs that have AES-NI instructions. This scheme is a CAESAR third-round candidate.
Security Analysis	The designers [33, 32, 25] indicate that the security of OCB is reducible to the strong pseudorandomness of the block cipher. It has a proven security of $n/2$ bits when a n-bit block cipher is used.
Performance Analysis	(SW) 0.64 C/B on Intel Core i5-6600 (Skylake 3.31 GHz) using AES-128 (HW) 1,348 slices and fmax of 292.7 MHz on Virtex 6. The implementation results of many other CPUs have been reported. [25].
Standardization	RFC 7253

	Authenticated encryption
Name	PRIMATEs
Designers	Elena Andreeva†, Begul Bilgin†, Andrey Bogdanov (Technical University of Denmark, Denmark), Atul Luykx†, Florian Mendel (Graz University of Technology, Austria), Bart Mennink†, Nicky Mouha†, Qingju Wang†, Kan Yasuda (NTT, Japan) †: KU Leuven, Belgium
Publication	2014 (DIAC 2014 [1], FSE 2014 [2])
Specifications	CAESAR Website [6] and Official Website: `http://primates.ae/`
Features	This is essentially a family of three schemes having different modes of operation: HANUMAN, GIBBON, and APE. All schemes are sponge-based, and have 200-bit or 280-bit cryptographic permutation as an internal element. When public random permutation is used, 80-bit or 120-bit security is guaranteed. The internal permutation is SPN similar to AES or Rijndael, but it uses 5-bit S-box. HANUMAN and GIBBON are based on known modes of operation (SpongeWrap and MonkeyWrap). APE has a dedicated mode considering nonce reuse.
Security Analysis	Practical forgery attack on HANUMAN exploiting a flaw when there is no associated data, have been reported [40]. The designers have proposed a revision.
Performance Analysis	For GIBBON, (SW) 1,712 C/B on Intel Core i5-6600 (Skylake 3.31 GHz) (HW) 419 slices and fmax of 333.4 MHz on Virtex 6.

Bibliography

[1] Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mendel, F., Mennink, B., Mouha, N., Wang, Q., Yasuda, K.: CAESAR candidates PRIMATES. DIAC - Directions in Authenticated Ciphers (2014), http://2014.diac.cr.yp.to/

[2] Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography. In: FSE. Lecture Notes in Computer Science, vol. 8540, pp. 168–186. Springer (2014)

[3] Banik, S., Bogdanov, A., Minematsu, K.: Low-area hardware implementations of CLOC, SILC and AES-OTR. In: HOST. pp. 71–74. IEEE Computer Society (2016)

[4] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404 (2013), http://eprint.iacr.org/2013/404

[5] Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A Concrete Security Treatment of Symmetric Encryption. In: FOCS. pp. 394–403. IEEE Computer Society (1997)

[6] Bernstein, D.J.: CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness, http://competitions.cr.yp.to/caesar.html/

[7] Bernstein, D.J.: eBACS: ECRYPT Benchmarking of Cryptographic Systems, http://bench.cr.yp.to/results-caesar.html/

[8] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak reference (2011), http://keccak.noekeon.org/

[9] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V., Keer, R.V.: CAESAR candidates Ketje + Keyak. DIAC - Directions in Authenticated Ciphers (2014), http://2014.diac.cr.yp.to/

[10] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: CHES. Lecture Notes in Computer Science, vol. 4727, pp. 450–466. Springer (2007)

[11] Bost, R., Sanders, O.: Trick or Tweak: On the (In)security of OTRs Tweaks. Cryptology ePrint Archive, Report 2016/234 (2016), http://eprint.iacr.org/2016/234

[12] Cryptographic Engineering Research Group at George Mason University: ATHENa: Automated Tools for Hardware EvaluatioN, https://cryptography.gmu.edu/athena/

[13] Daemen, J.: Permutation-based encryption, authentication and authenticated encryption. DIAC - Directions in Authenticated Ciphers (2012), http://hyperelliptic.org/DIAC/

[14] Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: CAESAR candidates Ascon. DIAC - Directions in Authenticated Ciphers (2014), http://2014.diac.cr.yp.to/

[15] Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Cryptanalysis of Ascon. In: CT-RSA. Lecture Notes in Computer Science, vol. 9048, pp. 371–387. Springer (2015)

[16] Groß, H., Wenger, E., Dobraunig, C., Ehrenhofer, C.: Suit up! - Made-to-Measure Hardware Implementations of Ascon. In: DSD. pp. 645–652. IEEE Computer Society (2015)

[17] Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In: CHES. Lecture Notes in Computer Science, vol. 6917, pp. 326–341. Springer (2011)

[18] Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Authenticated Encryption for Short Input. In: FSE. Lecture Notes in Computer Science, vol. 8540, pp. 149–167. Springer (2014)

[19] Iwata, T., Minematsu, K., Guo, J., Morioka, S., Kobayashi, E.: CAESAR candidates SILC. DIAC - Directions in Authenticated Ciphers (2014), http://2014.diac.cr.yp.to/

[20] Jean, J., Nikolić, I., Peyrin, T.: Deoxyx v1.3, http://competitions.cr.yp.to/caesar-submissions.html/

[21] Jean, J., Nikolić, I., Peyrin, T.: Joltik v1.3, http://competitions.cr.yp.to/caesar-submissions.html/

[22] Jean, J., Nikolić, I., Peyrin, T.: CAESAR candidates DEOXYS + Joltik. DIAC - Directions in Authenticated Ciphers (2014), http://2014.diac.cr.yp.to/

[23] Jean, J., Nikolic, I., Peyrin, T.: Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 8874, pp. 274–288. Springer (2014)

[24] Jean, J., Nikolić, I., Peyrin, T.: Deoxys and Joltik. DIAC - Directions in Authenticated Ciphers (2015), http://www1.spms.ntu.edu.sg/~diac2015/

[25] Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. In: FSE. Lecture Notes in Computer Science, vol. 6733, pp. 306–327. Springer (2011)

[26] Lafitte, F., Lerman, L., Markowitch, O., Heule, D.V.: SAT-based cryptanalysis of ACORN. Cryptology ePrint Archive, Report 2016/521 (2016), http://eprint.iacr.org/2016/521

[27] Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. In: Yung, M. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 2442, pp. 31–46. Springer (2002)

[28] Minematsu, K.: Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 8441, pp. 275–292. Springer (2014)

[29] Minematsu, K., Shigeri, M., Kubo, H.: AES-OTR v2. DIAC - Directions in Authenticated Ciphers (2015), http://www1.spms.ntu.edu.sg/~diac2015/

[30] Peyrin, T., Seurin, Y.: Counter-in-Tweak: Authenticated Encryption Modes for Tweakable Block Ciphers. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 33–63. Springer (2016), http://dx.doi.org/10.1007/978-3-662-53018-4_2

[31] Peyrin, T., Sim, S.M., Wang, L., Zhang, G.: Cryptanalysis of JAMBU. In: FSE. Lecture Notes in Computer Science, vol. 9054, pp. 264–281. Springer (2015)

[32] Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 3329, pp. 16–31. Springer (2004)

[33] Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: ACM Conference on Computer and Communications Security. pp. 196–205. ACM (2001)

[34] Rogaway, P., Shrimpton, T.: A Provable-Security Treatment of the Key-Wrap Problem. In: Vaudenay, S. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 4004, pp. 373–390. Springer (2006)

[35] Roy, D., Mukhopadhyay, S.: Some results on ACORN. Cryptology ePrint Archive, Report 2016/1132 (2016), http://eprint.iacr.org/2016/1132

[36] Salam, M.I., Bartlett, H., Dawson, E., Pieprzyk, J., Simpson, L., Wong, K.K.H.: Investigating Cube Attacks on the Authenticated Encryption Stream Cipher ACORN. Cryptology ePrint Archive, Report 2016/743 (2016), http://eprint.iacr.org/2016/743

[37] Sasaki, Y., Todo, Y., Aoki, K., Naito, Y., Sugawara, T., Murakami, Y., Matsui, M., Hirose, S.: Minalpher v1.1, http://competitions.cr.yp.to/caesar-submissions.html/

[38] Sasaki, Y., Todo, Y., Aoki, K., Naito, Y., Sugawara, T., Murakami, Y., Matsui, M., Hirose, S.: CAESAR candidates Minalpher. DIAC - Directions in Authenticated Ciphers (2014), http://2014.diac.cr.yp.to/

[39] Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE : A Lightweight Block Cipher for Multiple Platforms. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 7707, pp. 339–354. Springer (2012)

[40] Vizár, D.: Ciphertext Forgery on HANUMAN. Cryptology ePrint Archive, Report 2016/697 (2016), http://eprint.iacr.org/2016/697

[41] Wu, H.: CAESAR candidates Acorn + MORUS. DIAC - Directions in Authenticated Ciphers (2014), http://2014.diac.cr.yp.to/

[42] Wu, H., Huang, T.: CAESAR candidates AEGIS + Jambu. DIAC - Directions in Authenticated Ciphers (2014), http://2014.diac.cr.yp.to/

CRYPTREC
Cryptographic Technology Guideline (Lightweight Cryptography)
[CRYPTREC-GL-0001-2016-E]

Date of publication : June 30, 2017 (First edition)

Published by

· NATIONAL INSTITUTE OF INFORMATION AND COMMUNICATIONS TECHNOLOGY
(Cybersecurity Research Institute, Security Fundamentals Laboratory)
4-2-1 NUKUI-KITAMACHI, KOGANEI
TOKYO, 184-8795 JAPAN

· INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN
(Technology Headquaters, IT Security Center (ISEC), Cryptography Research Group)
2-28-8 HONKOMAGOME, BUNKYO-KU
TOKYO, 113-6591 JAPAN