Lightweight Symmetric Cryptography

INAUGURALDISSERTATION
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
EINES DOKTORS DER NATURWISSENSCHAFTEN
DER UNIVERSITÄT MANNHEIM

vorgelegt von

M. Sc. Vasily Mikhalev

Mannheim, 2018

Dekan:	Dr. Bernd Lübcke, Universität Mannheim
Referent:	Prof. Dr. Frederik Armknecht, Universität Mannheim
Korreferent:	Prof. Dr. Tim Güneysu, Ruhr-Universität Bochum

ABSTRACT

The Internet of Things is one of the principal trends in information technology nowadays. The main idea behind this concept is that devices communicate autonomously with each other over the Internet. Some of these devices have extremely limited resources, such as power and energy, available time for computations, amount of silicon to produce the chip, computational power, etc. Classical cryptographic primitives are often infeasible for such constrained devices. The goal of lightweight cryptography is to introduce cryptographic solutions with reduced resource consumption, but with a sufficient security level. Although this research area was of great interest to academia during the last years and a large number of proposals for lightweight cryptographic primitives have been introduced, almost none of them are used in real-word. Probably one of the reasons is that, for academia, lightweight usually meant to design cryptographic primitives such that they require minimal resources among all existing solutions. This exciting research problem became an important driver which allowed the academic community to better understand many cryptographic design concepts and to develop new attacks. However, this criterion does not seem to be the most important one for industry, where lightweight may be considered as "rightweight". In other words, a given cryptographic solution just has to fit the constraints of the specific use cases rather than to be the smallest. Unfortunately, academic researchers tended to neglect vital properties of the particular types of devices, into which they intended to apply their primitives. That is, often solutions were proposed where the usage of some resources was reduced to a minimum. However, this was achieved by introducing new costs which were not appropriately taken into account or in such a way that the reduction of costs also led to a decrease in the security level. Hence, there is a clear gap between academia and industry in understanding what lightweight cryptography is. In this work, we are trying to fill some of these gaps. We carefully investigate a broad number of existing lightweight cryptographic primitives proposed by academia including authentication protocols, stream ciphers, and block ciphers and evaluate their applicability for real-world scenarios. We then look at how individual components of design of the primitives influence their cost and summarize the steps to be taken into account when designing primitives for concrete cost optimization, more precisely - for low energy consumption. Next, we propose new implementation techniques for existing designs making them more efficient or smaller in hardware without the necessity to pay any additional costs. After that, we introduce a new stream cipher design philosophy which enables secure stream ciphers with smaller area size than ever before and, at the same time, considerably higher throughput compared to any other encryption schemes of similar hardware cost. To demonstrate the feasibility of our findings we propose two ciphers with the smallest area size so far, namely Sprout and Plantlet, and the most energy efficient encryption scheme called Trivium-2. Finally, this thesis solves a concrete industrial problem. Based on standardized cryptographic solutions, we design an end-to-end data-protection scheme for low power networks. This scheme was deployed on the water distribution network in the City of Antibes, France.

ZUSAMMENFASSUNG

Das Internet der Dinge ist heutzutage einer der wichtigsten Trends in der Informationstechnologie. Die Hauptidee hinter diesem Konzept sind Geräte, die autonom über das Internet kommunizieren. Einige dieser Geräte haben extrem limitierte Ressourcen, wie zum Beispiel Strom und Energie, verfügbare Rechenzeit, verfügbare Siliziummenge zur Chip-Produktion, Rechenleistung usw. Klassische kryptographische Primitive sind auf derart eingeschränkten Geräten oft nicht realisierbar. Das Ziel der leichtgewichtigen Kryptographie besteht darin, kryptographische Lösungen mit reduziertem Ressourcenverbrauch, aber ausreichendem Sicherheitsniveau einzuführen. Obwohl auf diesem Forschungsgebiet reges akademisches Interesse bestand in den letzten Jahren und eine große Anzahl neuer Vorschläge für leichtgewichtige kryptographische Primitive vorgestellt wurden, werden nahezu keine davon in Praxisanwendungen eingesetzt. Wahrscheinlich liegt einer der Gründe darin, dass in der akademischen Welt normalerweise kryptographische Primitive so entworfen werden sollten, dass sie weniger Ressourcen als alle existierenden Lösungen benötigen. Dieses spannende Forschungsproblem entwickelte sich zu einer treibenden Kraft, welche der akademischen Gemeinschaft erlaubte, viele kryptographische Designkonzepte besser zu verstehen und neue Angriffe zu entwickeln. In der Industrie hingegen scheint dies keinesfalls das wichtigste Kriterium zu sein, wo leichtgewichtig eher als "rechtgewichtig" angesehen werden kann. Mit anderen Worten, eine gegebene kryptographische Lösung muss lediglich die durch den spezifischen Anwendugsfall vorgegebenen Beschränkungen erfüllen anstatt die kleinste zu sein. Leider hat die akademische Forschung wichtige Eigenschaften der konkreten Arten von Geräten vernachlässigt, auf denen die entwickelten Primitiven eingesetzt werden sollten. So wurden häufig Lösungen vorgeschlagen, die den Verbrauch bestimmter Ressourcen zwar auf ein Minimum reduzierten. Dies wurde allerdings dadurch erreicht, dass hieraus entstehende neue Kosten nicht ausreichend in Betracht gezogen worden sind oder eine Reduzierung der Kosten ein niedrigeres Sicherheitsniveau zur Folge hatte. Es besteht eine klare Kluft zwischen Wissenschaft und Industrie, wenn es darum geht zu verstehen, was leichtgewichtige Kryptographie ist. In dieser Arbeit schließen wir diese Lücke. Wir untersuchen sorgfältig eine breite Palette existierender kryptographischer Primitive, die von der Wissenschaft vorgeschlagen wurden, einschließlich Authentifizierungsprotokollen, Stromchiffren und Blockchiffren, und evaluieren deren Anwendbarkeit in realen Szenarien. Dann untersuchen wir die Auswirkungen einzelner Komponenten beim Chiffre-Design hinsichlich Kosten und fassen die einzelnen zu berücksichtigenden Schritte zusammen, wenn man Chiffren entwickelt, die für konkrete Kosten optimiert sein sollen, genauer gesagt für niedrigen Energieverbrauch. Als Nächstes schlagen wir neue Implementierungstechniken für existierende Designs vor, wodurch diese effizienter oder kleiner in Hardware werden, ohne dabei weitere Kosten zu induzieren. Hiernach stellen wir eine neue Chiffrendesign-Philosophie vor, die sichere Stromchiffren mit kleinerer Fläche als jemals zuvor und gleichzeitig bedeutend höherem Durchsatz verglichen mit anderen Verschlüsselungsverfahren mit ähnlichen Hardwarekosten ermöglicht. Um die Realisierbarkeit unserer Ergebnisse zu zeigen, schlagen wir zwei neue Chiffren mit geringstem Platzbedarf, nämlich Sprout und Plantlet, sowie das bislang energieeffizienteste Verschlüsselungsverfahren namens Trivium-2 vor. Am Ende dieser Arbeit lösen wir ein konkretes industrielles Problem. Basierend auf standardisierten kryptographischen Lösungen entwickeln wir ein Ende-zu-Ende-Datenschutzkonzept für Niedrigenergie-Netzwerke. Das System wurde auf dem Wasserleitungsnetz der französischen Stadt Antibes installiert.

Acknowledgements

Completing this thesis, I have the last, but not least and the most pleasant note to acknowledge contribution and express my gratitude to my advisors, allies, family and friends for their input and support.

First of all, through the entire journey my PhD advisor, Freredik Armknecht was always there supporting me in my research, helping me not only in the academic area, but also developing as a person and a professional scientist. Understanding my private needs and balancing load, Frederik was there to help me to go through rough and tough time gaining best from the academic environment. He also supported me taking interesting opportunities in parallel to the science carrier, inspiring me by his example. Without Frederik I would not be able to proceed that far with my work. Therefore, I will use formulation “our work” onwards to highlight his input.

I also would like to thank Tim Güneysu, who found time in his overbooked calendar to support us. I feel lucky that one of the best experts in our area of the research agreed to be my second supervisor.

Moreover, I want to extend the meaning of "our work" to acknowledge the input from my colleagues from University Mannheim: Matthias Hamann and Christian Müller who did an outstanding job helping to improve this thesis, providing their help with proofreading.

In Theoretical Computer Science and IT Security Group of the University of Mannheim we have not just great and inspiring environment from the academic perspective, but wonderful people and team, so I was always surrounded by allies sharing my eager to research, learn and teach. Frederik Armknecht, Matthias Krause, Gabi Atkinson, Karin Teynor, Matthias Hamann, Christian Müller, Christian Gorke, Angela Jäschke, Alexander Moch, Walter Müller, thank you all for the great time, lots of fun and many wonderful days I spent developing our work and enjoying this journey.

It was and it is wonderful to work together in our cryptographic community in the competitive though friendly environment as we are sharing same passion to discover and develop.

The last statements in this work I dedicate to my parents who supported me not just during the years of working on this thesis but made me prepared for such journey sparking my interest in discovering new things since I was a child. I felt great support and love from my family, which allowed me to enjoy the time working on this thesis and make the work done.

Dedication

This work is dedicated to my parents.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Introduction to Lightweight Cryptography

Nowadays, the Internet is already integrated into our daily life almost as much as electricity. The transmission of information has become so ordinary and simple that it ceases to be noticeable. Very often data is transmitted wireless over the air which makes it easy to access by anyone who has appropriate equipment; hence it has to be protected. Therefore today cryptography is probably more critical than ever.

The Internet of Things (IoT) is seen as one of the most groundbreaking and game-changing evolutions of information technology in modern times. This comprehensive term is characterized by many smart devices that use the Internet to communicate with each other without people being involved. In many cases these devices process sensitive personal or enterprise data, so care needs to be taken that the communication is secure. However, because of the tight cost constraints inherent in mass deployments and the need for mobility these devices often have very limited resources regarding memory, computing power or battery supply. Therefore, it is often impossible to use classical cryptographic primitives, as long as their implementation requires more resources than available.

Lightweight cryptography deals with cryptographic solutions which can be efficiently implemented on the targeted constrained devices. It implies that this discipline is located on the crossroads of security, mathematics, computer science, and engineering.

We would like to stress that based on our experience the understanding of lightweight cryptography is somewhat different for academia and industry. For academia, lightweight usually means to design cryptographic primitives in such a way that they require minimal resources among the existing solutions. In the best cases, the designers are trying to reduce some costs, making sure that the other ones are not increased and that the security level is still high. However, often solutions are proposed where the usage of some resources was reduced to minimal, but this is achieved by introducing new costs which are not taken into account or by decreasing the level of security.

For industry, the term lightweight is something different. It may be considered as ″right-weight″, or in other words, a given cryptographic solution just has to fit the constraints of specific use cases, being at the same time as cheap as possible [Ava18].

Academia usually aims to make sure that the security of a particular primitive is high enough so that it cannot be practically broken by any means, no matter how much effort is spent. For industry security is a business decision, where the goal is to make the attacker′s Return on Investment (ROI) measure negative, meaning that an attacker has to invest more as compared to what she gets if she manages to break the solution.

1.1.1 Applications of Lightweight Cryptography

To illustrate the topicality of lightweight cryptography nowadays, we discuss it's most important applications.

A typical application example are RFID tags. Radio-Frequency Identification (RFID) is an emerging technology which uses electromagnetic fields to automatically identify and track real-world objects. An RFID system consists of a reader device which communicates with a passive or an active RFID tag - a small microchip attached to an antenna. Passive tags are powered by the electromagnetic field of the nearby RFID reader. Active RFID tags have their own power source and do not depend on the reader's energy. As a result they may operate hundreds meters away from the reader. Many applications like supply-chain management, access control, and inventory management already use RFID technology. In many published articles, however, the missing privacy protection is an argument against the use of RFID tags. Therefore, integrating security into RFID systems is inevitable.

Another example are Body Area Networks (BAN). They consist of several miniaturized sensors that are connected to a human body. The sensors continuously monitor health conditions like heartbeat or blood pressure. The data is passed wirelessly to a central device, which in turn forwards it through the network to a doctor or hospital. This allows for an immediate reaction in the case of emergency (improving health care), while removing the need for permanent visits to a doctor. For the sake of mobility and comfort, the sensors should be as minimal as possible. These constraints become even more severe when the medical devices are to be implanted into the body of a patient in order to treat, monitor or improve the functioning of some body part. Here it is crucial to make sure that the energy consumption of such devices is small enough to guarantee that their life cycle is sufficiently long to avoid the necessity to replace them. Note that additional constraints may be relevant here. For example, the temperature level of the chip has to be within the specific ranges, which is mainly achieved by precise control of the device power consumption and through usage of particular materials. As the communication is wireless, it might be eavesdropped. Therefore, data protection measures are crucial.

One more example is Wireless Sensor Networks (WSNs), where spatially distributed autonomous devices equipped with sensors are used to collect data. Nowadays, various industries actively introduce such systems into their production process: they create a virtual copy of the physical world in order to monitor physical processes and make decentralized decisions. Often the nodes of WSNs are low-powered, meaning that the devices are restricted to consume only very little energy to operate. The technology called low-powered wide-area networks (LPWANs) or low-power networks (LPNs) is usually used to achieve connectivity in such scenarios. Nowadays, at the market there exist several LPN technology solutions, which offer economically viable options to physically deploy new sensors along with the necessary communications infrastructure in order to generate, transport and ingest data coming from any industrial assets. However, even if the connectivity is achieved, low security level of such systems is often a big issue.

1.2 Summary of Research Contributions and Outline

In most of the lightweight scenarios, the three main security requirements are: authenticity, meaning that we need to be sure that the data is sent by the valid device; integrity, which ensures that the message has not been modified while in transit; and confidentiality, which implies that the data cannot be eavesdropped. To obtain the first two properties, so-called authentication protocols are employed, while for the third we need to use encryption schemes (or ciphers). Therefore, in this work, we mainly consider these two kinds of cryptographic primitives. In the last one and a half decades, there was a huge number of proposals by academia which introduced many examples of the lightweight authentication protocols, e.g., HB [HB01], HB^+[JW05], HB^* [DK07], PUF-HB [HS08], Lapin [HKL+12], and lightweight ciphers, e.g. Midori [BBI+15], PRESENT [BKL+07], LED [GPPR11], PRINTCipher [KLPR10], CLEFIA [SSA+07], Grain [HJMM08], PRINCE [BCG+12], Trivium[CP08], Sprout [AM15], Plantlet, [MAM17], LIZARD [HKM17] to name a few.

Despite a few examples[1], most of these proposal were never used in real-world applications. As was previously mentioned, the reason for this situation is probably due to the fact that there are gaps between academia and industry in understanding of what lightweight means. In this work, we are trying to fill some of these gaps.

In a nutshell, the following goals were achieved while working on this thesis:

Understanding constraints: We specified and summarized the real-world limitations and conditions for different scenarios in which the lightweight protocols are employed. These results are covered by the publications [AHM14, MAM17].

Evaluation of primitives: We implemented and evaluated numerous proposed cryptographic primitives with respect to their real suitability for the targeted scenarios, as discussed in [AHM14, MAM17, BMA+18]

Implementation techniques: In [AM14, AM] we proposed a new technique which allowed more efficient implementations of the existing algorithms.

Design approaches: We developed a new design method, discussed in [AM15, MAM17], which led to secure cryptographic primitives (e.g., stream ciphers) that are more suitable for lightweight scenarios due to their reduced costs.

New designs: In [AM15, MAM17, MGAM17], we designed new cryptographic primitives and security solutions.

We now provide an overview of these results in further detail.

[1]We note that PRESENT and CLEFIA were included in the international standard for lightweight cryptographic methods by The International Organization for Standardization and the International Electrotechnical Commission [ISO12].

1.2.1 Investigating Real-World Scenarios and Evaluation of Existing Cryptographic Primitives

While there is quite a good understanding of the security of most of the lightweight schemes proposed by academia, often only very little is known about their applicability in practice. One of the main reasons is that without careful investigation of the real-world systems, it is difficult to understand what amount of resources is available for a given primitive when it needs to be used in a concrete situation and if there exist any other scenario-specific constraints which are to be taken into account.

To this end, we examined three different scenarios, where such analysis was missing or not complete. For each of these scenarios, we focused on understanding their limitations and on the evaluation of applicability of existing cryptographic primitives to each of them.

1.2.1.1 Evaluation of Authentication Protocols on low-cost RFID tags.

One of the primary use-cases for RFID tags are authentication solutions, e.g., access control for buildings or cars, electronic passports or human-implantable chips providing sensitive medical information about a person. For economical reasons low-cost RFID tags (e.g., in the production cost range of $0.05 to $0.10) are particularly interesting for industry. This cost pressure directly translates into severe hardware restrictions for the targeted devices. Consequently, the search for appropriate lightweight authentication protocols has become an essential topic in cryptography during the last years with high relevance for academia and industry, generating a significant number of different approaches and schemes. However, open literature rarely provides information on what conditions need to be met by a scheme in practice, hindering a sound development and analysis of schemes. In [AHM14] we investigated this scenario. Our contributions here were twofold.

First, we provided a collection of several conditions that should be met by lightweight authentication schemes if deployed in RFID systems using tags in the mentioned cost range like *Electronic Product Codes* (EPCs). These conditions were derived both from open literature and numerous discussions with various experts from industry and may be of independent interest. The results are summarized in Table 3.2 page 33 .

The second contribution was an analysis of relevant existing, allegedly lightweight authentication schemes with respect to these conditions. Here, we focused on the two most important approaches: (i) based on the hardness of the learning parity with noise (LPN) problem and (ii) using encryption schemes. Possibly surprising it turned out that none of the existing (unbroken) LPN-based protocols, which were implemented and evaluated for various reasonable parameters (see Subsubsection 3.2.4.2) for more details) complied to these conditions. This can be seen from the rightmost column of Table 3.27 on page 63, where the costs which exceed the constraints together with security issues of the protocols are indicated. We showed, however, that for the other approach, namely, using lightweight encryptions schemes, instantiations do exist which meet the derived conditions.

1.2.1.2 Reconsideration of Performance of Ciphers that Continuously Access Non-volatile Key

Due to the increased use of devices with restricted resources such as limited area size, power or energy, the community has developed various techniques for designing lightweight ciphers. One approach that is increasingly discussed is to use the cipher key that is stored on the device in non-volatile memory not only for the initialization of the registers but during the encryption/decryption process as well. Initially this idea was used in block ciphers, e.g. KTANTAN [CDK09], LED [GPPR11], PRINTcipher [KLPR10] and Midori [BBI+15]. Later, in [AM15, MAM17], we demonstrated the advantages of this approach for stream ciphers as well. This technique may on the one hand help to save resources, but also may allow for a stronger key involvement and hence higher security. However, only little is publicly known so far if and to what extent this approach is indeed practical. Thus, cryptographers without strong engineering background face the problem that they cannot evaluate whether certain designs are reasonable (from a practical point of view), which hinders the development of new designs.

We investigated this design principle from a practical point of view in [MAM17]. After a discussion on reasonable approaches for storing a key in non-volatile memory, motivated by several commercial products we focused on the case that the key is stored in Electrically Erasable Programmable Read-Only Memory (EEPROM) and . Here, we highlighted existing constraints and derived that some designs, based on the impact on their throughput, are better suited for the approach of continuously reading the key from all types of non-volatile memory.

1.2.1.3 On Energy Consumption of Stream Ciphers

As was mentioned in Subsection 1.1.1, there exist scenarios (for example, implantable medical devices), where efficient energy consumption is vital. However, since the time when lightweight cryptography became a hot topic in academia, most of the research effort was devoted to low-area or low-power designs. Note that though being closely related, energy and power are two different parameters. Energy, which is the integral of power over time is much more critical for battery-operated devices. Moreover, the designs for low energy and low power consumption can be completely different. Nevertheless, for some reason, the question of low-energy design was mostly ignored until the first energy-optimized block cipher Midori [BBI+15] was proposed in 2015. On the other hand, not much is known by now about the energy efficiency of stream ciphers.

We took a detailed look into energy consumption traits of stream ciphers in [BMA+18].

For several selected stream ciphers, namely, Trivium [CP08], Grain v1 [HJM07], Grain-128 [HJMM06], Lizard [HKM17], Plantlet [MAM17] and Kreyvium [CCF$^+$16], we examined all implementation level aspects that are likely to affect the energy consumption of stream ciphers and then drew necessary conclusions from our studies.

Our principal finding was that although block ciphers are more energy-efficient when encrypting short data streams, while for longer data streams multiple-round-unrolled stream ciphers perform better. In particular, stream ciphers with simple update functions were found to be more energy-efficient unrolling does not lead to a significant increase in circuit complexity and power consumption.

1.2.2 New Approaches and Techniques

To enhance the theory of lightweight cryptography we presented a new technique which allows to improve implementations of existing algorithms and a new design approach enabling new secure stream ciphers with low area size.

1.2.2.1 Technique for Efficient Implementations of Existing Cryptographic Primitives

A new implementation technique that allows for increasing the maximum throughput of stream ciphers without any (or only small) increase in the hardware size was introduced in [AM14] and further investigated in [AM]. Our technique can be used with stream ciphers that are composed of a very common building block - namely a feedback shift register (FSR), an external block (which is treated as a black box), and an output function which combines values from the FSR and the external block. This covers the majority of stream ciphers proposed so far.

The technique can be seen as a combination of two existing approaches: pipelining and FSR-transformation. The main idea is to reduce the circuit delay of the output function by integrating parts of it into several update functions of the FSR.

We provided a detailed technical description for sufficient conditions under which this technique is applicable and proved that the proposed transformation preserves the functionality of the cipher. Moreover, we demonstrated the practicability of our approach by applying it to the stream ciphers Grain-128 in [AM14] and Grain-128a in [AM].

The transformation allowed an increase of the throughput by about 20% in both cases, while the area size was almost unchanged. The improvements for both cases (Grain-128 and Grain-128a) are summarized in Table 4.4 on page 112.

1.2.2.2 Approach Enabling New Lightweight Stream Ciphers with Shorter Internal State

To be resistant against certain time-memory-data-tradeoff (TMDTO) attacks, a common rule of thumb says that the internal state size of a stream cipher should be at least twice the security parameter. As memory gates are usually the most area and power consuming components, this implies a severe limitation with respect to possible lightweight implementations. We investigated an extension in the common design which allows for realizing secure lightweight stream cipher with an area size beyond the trade-off attack bound mentioned above in [AM15] and [MAM17]. To achieve this goal, we suggested to involve the key into the update process of the internal state of the cipher. We argued that such a shift in the established design paradigm, namely to involve the fixed secret key not only in the initialization process but in the keystream generation phase as well, improves the resistance against the mentioned TMDTO attacks and allows to design secure stream ciphers with much smaller area size.

1.2.3 New Designs

We designed new lightweight cryptographic schemes in order to prove the feasibility of our findings and to solve real-life problems with lightweight cryptographic solutions.

1.2.3.1 Low-Area Stream Ciphers Sprout and Plantlet

Our new design principles, discussed in Subsection 4.3.3, were demonstrated by two concrete keystream generators with keyed update function, namely Sprout [AM15] and Plantlet [MAM17]. Both ciphers have a similar structure that has been inspired by Grain-128a [ÅHJM11]. The main differences are the following:

1. Sprout and Plantlet have shorter internal state size compared to any of the Grain family ciphers [HJMM08].
2. They use the round key function to make the state update key-depended.

As can be seen from Table 5.1 on page 122, Sprout and Plantlet use significantly less area than comparable existing lightweight stream ciphers. Note that Sprout was broken while, to the best of our knowledge, Plantlet remains secure.

1.2.3.2 Low-Energy Stream Cipher Trivium-2

We presented the stream cipher Trivium-2. It is based on the design of Trivium, but provides 128-bit security, whereas Trivium is designed for 80-bit security level. Trivium-2 is optimized for energy consumption. The design was developed together with Subhadeep Banik, Frederik Armknecht, Takanori Isobe, Willi Meier, Andrey Bogdanov, Yuhei Watanabe and Francesco Regazzoni while working on the paper [BMA+18] and is going to be published in the full version of the paper. Among stream ciphers that provide 128-bit security, for encryption of longer data streams the energy consumption of the cipher is around 2.5 times better than Grain-128 and approximately 15% better than Kreyvium (see Table 5.5 on page 134 for comparison).

We also argued the security of the cipher by performing extensive cryptanalysis on reduced round variants of the design.

1.2.3.3 End-To-End Data Protection in Low-Powered Networks

We solved a concrete industrial problem. As was discussed in Subsection 1.1.1, a significant, emerging trend in the context of the Internet of Things (IoT) are low-power networks (LPNs), referring to networks that target devices with very limited access to energy sources. While several approaches allow to comply to these novel power restrictions, none of them provide a sufficient level of security, in particular concerning data protection.

In [MGAM17] we proposed a data protection scheme that can be realized on top of the existing solutions and which ensures end-to-end security from low-power devices to back-end applications. It meets the technical constraints imposed by LPNs, while preserving data confidentiality and integrity. Our solution has been deployed on the water distribution network of the City of Antibes in France. The evaluation of the overhead introduced by the proposed data protection scheme shows promising results with respect to energy consumption. Moreover, the results of this work were used in the exhibit prepared for the German National Digital Summit (Digital-Gipfel) 2017.

1.2.4 Outline

The remainder of this work is organized as follows:

- In Chapter 2, we introduce notions and concepts which are essential for understanding the further parts of this thesis.
- Chapter 3 is devoted to the investigation of real-world scenarios where the lightweight cryptography is applicable. We discuss the limitations and conditions relevant to these scenarios and evaluate the applicability of the existing cryptographic primitives to each of them.
- Chapter 4 describes our new techniques and approaches which enhance the theory of lightweight cryptography.
- In Chapter 5, we use the results of our research to design the new cryptographic solutions.
- Chapter 6 provides the conclusions of this work.

1.2.5 Publications

This work is based on the following publications.

[BMA+18] S. Banik, V. Mikhalev, F. Armknecht, T. Isobe, W. Meier, A. Bogdanov, Y. Watanabe, and F. Regazzoni. Towards low energy stream ciphers. *IACR Transactions on Symmetric Cryptology*, 2018(2):1–19, Jun. 2018.

[MGAM17] V. Mikhalev, L. Gomez, F. Armknecht, and J. Márquez. Towards End-to-End Data Protection in Low-Power Networks. In: Computer Security. Springer, 2017, pp. 3–18.

[MAM17] V. Mikhalev, F. Armknecht, and C. Müller. On ciphers that continuously access the non-volatile key. *IACR Transactions on Symmetric Cryptology*, 2016(2):52-79, 2017.

[AM15] F. Armknecht and V. Mikhalev. On lightweight stream ciphers with shorter internal states. In G. Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 451-470. Springer, 2015.

[AHM14] F. Armknecht, M. Hamann, and V. Mikhalev. Lightweight authentication protocols on ultra-constrained RFIDs - myths and facts. In N. Saxena and A. 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 1–18. Springer, 2014.

[AM] F. Armknecht and V. Mikhalev. Revisiting a Recent Resource-Efficient Technique for Increasing the Throughput of Stream Ciphers. *International Conference on Security and Cryptography*, Vienna, Austria, 2014.

[AM14] F. Armknecht and V. Mikhalev. On increasing the throughput of stream ciphers. In *Topics in Cryptology-CT-RSA 2014*, pages 132–151. Springer, 2014.

CHAPTER 2

Preliminaries

2.1 Performance Metrics, Constraints, and Optimization Goals

When designing a lightweight cryptographic primitive, there is always a trade-off between security, performance, and costs. However, depending on the targeted platform and the concrete usage scenario, there may be completely different performance metrics which need to be considered in a given solution. In general, all of the designs can be classified into two main categories: hardware and software oriented ones. As the terms already indicate, in the first case, the primitives need to be compact and efficient when implemented in hardware. In the second case, the primitives are meant to be implemented in software for example when implemented on the low cost Microcontroller Units (MCUs).

2.1.1 Hardware-oriented Designs

Nowadays, there exist two main platforms for the hardware-oriented designs. These are *Application-specific Integrated Circuits* (ASIC) and Field-Programmable Gate Array (FPGA). Both platforms are different types of Integrated Circuits. ASICs are factory-customized for a particular task, meaning that after ASIC cheap is produced, it has a fixed functionality which can not be altered later. Whereas, an FPGA is intended for general-purpose usability which can be programmable at any time.

In this work, we mainly focus on the designs targeting ASIC which is the cheaper and more commonly used platform nowadays. Next, we discuss the hardware costs which are relevant to this platform.

Area size Area is the amount of silicon used for implementation of the design in ASIC. Considering that area requirements in μm^2 strongly depend on the used fabrication technological process, it has become common practice to resort to a more general metric called Gate Equivalents (GEs) instead. In short, one GE is equivalent to the area of a two-input drive-strength-one NAND gate used in a given technology library. This allows for a rough comparison of area requirements derived using different technologies.

Clock Frequency This is the clock rate at which the chip is running. This parameter is selected by the designer but is limited by the maximal possible value for a give design and technology.

Maximal Clock frequency Upon receiving an input, a circuit processes these values and eventually produces an output. The time period between getting the input and producing the output is called its *delay*. The operations between circuits are synchronized by clock pulses. Naturally, the time interval between clock pulses must be long enough so that all the logic gates have time to respond to the changes and their outputs "settle" to stable logic values before the next clock pulse occurs. As long as this condition is met, the circuit is guaranteed to be stable and reliable. Each of the connections between inputs, registers, and outputs of the implemented algorithm forms a *timing path*. The path which has the biggest delay is called *critical path*. It defines the maximum operating clock frequency of the circuit.

Throughput The throughput is the rate at which a new output is produced with respect to time. It is determined as the number of bits-per-cycle multiplied by the clock frequency [GB08] of the circuit.

Power In complementary metal–oxide–semiconductor (CMOS) technology, commonly used for constructing integrated circuits, the total power consumption depends on the static power (mainly depends on the area size) and a dynamic power (which depends on the probability of a switching event, so called switching activity).

Energy Low power consumption is desired for applications like battery-less/contact-less devices. However, for battery driven devices, energy rather than power may be a more relevant parameter to measure the efficiency of a design. It is a measure of the total electrical work done by the battery source during the execution of any operation and is equal to the integral of power over time. Hence, it is highly relevant for devices that run on tight battery budgets like hand-held devices or medical implants.

Non-volatile Memory While the cost of volatile memory is often implicitly included in the numbers for area in the form of flip-flops/latches (respectively the components needed to build those), non-volatile memory is commonly not considered, which determines the data size which can be stored on the device even when it is disconnected from the energy source. In the case of cryptographic primitives, the NVM is used to store the secrets (e.g., private key) on the devices.

2.1.2 Software-oriented Designs

For the software-oriented designs, the essential metrics are the following.

Code Size This is the amount of read-only memory (ROM) taken by the machine code of the implemented algorithm. This is important since embedded systems store all program code in on-chip ROM whose size directly determines the cost of a device.

Memory Usage This metric determines the maximal amount of Random Access Memory (RAM) which the program takes during the runtime. This often matters heavily in case of restricted devices since RAM is a precious resource in many of them, such as sensor nodes.

Throughput The throughput measures the amount of data which is processed during a fixed period of time.

Energy Similar to the case of hardware-oriented primitives, low energy consumption is of the greatest importance for software implementation at any battery-powered device.

2.2 Stream Ciphers

Ciphers (encryption schemes) are algorithms that allow encrypting data for ensuring it's confidentiality. The operation of modern ciphers depends on the key - piece of auxiliary information that decides the details of encryption in the specific case.

Depending on the way how the keys are used, there are two main types of cryptographic systems - symmetric (or secret-key) and asymmetric (or public key). The asymmetric security systems are designed in such a way that the key which is used to encrypt the information (enciphering key) is different from the one used for decryption (deciphering key). Moreover, the deciphering key cannot be easily obtained from the enciphering key. Such systems are called public key, since the enciphering key can be open. That is, everyone can encrypt the message, but only the one who knows the deciphering key can decrypt it. In symmetric systems the same key is used for both encryption and decryption. In practice, this information is shared between two or more communicating parties for providing secure exchange of messages over a public channel.

All symmetric ciphers known so far fall in one of the following two categories: block ciphers and stream ciphers. As the names already indicate, block ciphers are designed for encrypting data blocks. Stream ciphers are encryption schemes that are dedicatedly designed to efficiently encrypt data streams of arbitrary length. In opposite to block ciphers, stream ciphers operate on the single digits of plain text, one by one to produce the ciphertext.

2.2.1 Stream Ciphers Design Principles

Stream ciphers attempt to imitate the action of a proven unbreakable cipher the one-time pad (OTP) [Sha49], originally known as the Vernam cipher. The one-time pad uses a keystream of completely random digits. The keystream $Z = (z_0, z_1 \cdots z_{n-1})$ is XORed with the plaintext $M = (m_0, m_1 \cdots m_{n-1})$ digits one at a time to form the ciphertext $C = (c_0, c_1 \cdots c_{n-1})$:

$$c_t = m_t \oplus z_t, 0 \leq t \leq n-1$$

To decipher the message the ciphered text is XORed with the same keystream:

$$m_t = c_t \oplus z_t, 0 \leq t \leq n-1$$

This system was proved to be unbreakable by Claude Shannon in 1949 [Sha49]. However, the keystream, generated completely at random, must be the same length as the plaintext, which makes the system infeasible in the real life.

A stream cipher can be viewed as the approximation of OTP. Based on a secret key and an initialization vector, the stream cipher produces a sequence of bits that is called a keystream. Similar to OTP, encryption is achieved by bitwise XORing of a keystream with the plaintext. Hence, a cryptographically strong keystream generator is the primary element of any additive stream cipher.

2.2.1.1 Keystream Generators

In a nutshell, a keystream generator (KSG) is a finite state machine using an internal state, an update function, and an output function. At the beginning, the internal state is initialized based on a secret key and, optionally, an initial value (IV). Given this, the KSG regularly outputs keystream bits that are computed from the current internal state and updates the internal state. The majority of existing KSGs are covered by the following definition:

Definition 1 (Keystream Generator)**.** A keystream generator (KSG) comprises three sets, namely

- the key space $\mathcal{K} = \mathrm{GF}(2)^{\kappa}$,
- the IV space $\mathcal{IV} = \mathrm{GF}(2)^{\nu}$,
- the state space $S = \mathrm{GF}(2)^{\sigma}$,

and the following three functions

- an initialization function $\mathsf{Init} : \mathcal{IV} \times \mathcal{K} \to S$
- an update function $\mathsf{Upd} : S \to S$
- an output function $\mathsf{Out} : S \to \mathrm{GF}(2)$

A KSG operates in two phases. In the *initialization phase*, the KSG takes as input a secret key k and an IV iv and sets the internal state to an initial state $St_0 := \mathsf{Init}(iv, k) \in S$. Afterwards, the keystream generation phase executes the following operations repeatedly (for $t \geq 0$):

1. Output the next keystream bit $z_t = \mathsf{Out}(St_t)$
2. Update the internal state St_t to $St_{t+1} := \mathsf{Upd}(St_t)$

2.2.1.2 Feedback Shift Registers (FSRs).

A *Feedback Shift Registers (FSR)* is an established building block for designing stream ciphers as it allows for generating long bit streams based on a short seed. In a nutshell, an FSR is a regularly clocked finite state machine that is composed of a register and an update mapping F.

At each clock, an entry of the state is given out and the state is updated according to the update mapping F.

Definition 2 (Feedback Shift Register (FSR))**.** An *FSR* of length n consists of an internal state of length n and an update functions $f_i(x_0, \dots, x_{n-1})$ for $i = 0, \dots, n-1$. Given some initial state $St_0 = (St_0[0], \dots, St_0[n-1]) \in \mathbb{F}^n$, the following steps take place at each clock t:

1. The value $St_t[0]$ is given out and forms a part of the *output sequence*.
2. The state $St_t \in \mathbb{F}^n$ is updated to St_{t+1} where $St_{t+1}[i] = f_i(St_t)$.

Depending on the form of the update functions $f_i(x_0, \dots, x_{n-1})$ the FSRs are classified into the following categories. When all update functions $f_i(x_0, \dots, x_{n-1})$ for $i = 0, \dots, n-1$ are linear the FSR is called a *Linear Feedback Shift Register (LFSR)*, otherwise it is called *Nonlinear Feedback Shift Register (NLFSR)*. FSRs are usually specified in the *Fibonacci configuration*, meaning that at each clock-cycle all but one state entries are simply shifted or in other words, all update functions except of f_{n-1} are of the form $f_i(x_0, \dots, x_{n-1}) = x_{i+1}$ for $i = 0, \dots, n-2$. Otherwise FSR is in *Galois configuration*.

2.2.2 Security of Stream Ciphers

2.2.2.1 Generic Attacks

There exist different classifications of cryptanalytic attacks according to different characteristics.

Based on the information available to an attacker, they can be classified into the following categories [Jön02]:

1. **Ciphertext-only attack**. In this attack scenario an attacker is given only the ciphertext.
2. **Known-plaintext attack**. These attacks are applied when the ciphertext and all or part of the plaintext is available. For additive stream ciphers this is equivalent of knowing all or part of the keystream.
3. **Chosen-plaintext attack**. In this scenario it is assumed that an attacker has access to an encryption device and can encrypt any number of chosen plaintexts.

4. **Chosen-ciphertext attack**. Here an attacker can decrypt any chosen ciphertext and her aim is to deduce the key.

By the type of access to an encryption/device [BMS06], there exist the following attack classes:

1. **Fixed-key attack**. In this scenario it is assumed that an attacker has access to a black box with one encryption/decryption device. The goal of the attacker is to find the key which remains unchanged during the attack.
2. **Variable key attack** . Here an attacker does not only have access to a black box with the encryption/decryption device, but also to a black box of the key-schedule device. The attacker is allowed to change the keys of the cipher and her goal is to find one of these keys.
3. **Related key attack**. This scenario assumes that the attacker has access to two or more encryption/decryption devices. Moreover, the keys of these devices have certain relations, which are chosen to the attacker, who is also allowed to apply re-keying of the devices.

Another classification is based on the goal of the attack.

1. **Key recovery attack**. This is a method to recover the key.
2. **Prediction attack** . A method for predicting a bit or sequence of bits of the keystream with a probability better than guessing.
3. **Distinguishing attack**. A method to distinguish the keystream from a truly random sequence.

Obviously, the key recovery attack is the most powerful one due to the fact that the knowledge of the key allows to apply both: the prediction and the distinguishing attack.

We now discuss the most important attacks against stream ciphers based on the way how they are applied.

The Exhaustive Key Search (Brute-Force Attack) Cryptanalysis often boils down to the following question. Given a function $F : \mathcal{N} \to \mathcal{N}$ and a value y within the image of F, find a preimage of y, i.e., determine a value $x \in \mathcal{N}$ such that $F(x) = y$. To accomplish this goal, two extreme cases are considered. One approach would be to use brute force search, i.e., randomly pick values $x \in \mathcal{N}$ until $F(x) = y$ does hold. This process would be repeated whenever the attacker aims to invert F.

Precomputation attack (Table lookup) The other extreme approach would be to precompute all possible values beforehand and store them in a large table, i.e., to trade recurring computation effort by memory. This would result in the situation that every subsequent attack is essentially a simple look-up.

Time-Memory-Data-Trade-Off Attack We discuss of this type of attack in more details as they will be used in several parts of this work.

In 1980 Hellman [Hel80] suggested a time-memory-trade-off (TMTO) attack which is probabilistic and falls somewhere in between a brute force attack and a precomputation attack. This initiated a long line of research on different trade-off attacks. A typical trade-off attack consists of two phases: the first is the precomputation phase, often called the offline phase, while the second is referred to as the real-time, or on-line phase. In the offline phase, the attacker precomputes a large table (or sets of tables) using the function F she is trying to invert, while in the online phase the attacker captures a function output and checks if this value is located in her tables. If this attack is successful the attacker can learn the value x for which $y = F(x)$. Usually, this type of attacks is evaluated by looking at the following costs:

- $|\mathcal{N}|$ - the size of the search space $\mathcal{N}$
- T_P - the time effort of the precomputation phase
- T - the time effort of the online phase
- M - memory cost of the attack.
- D - the number of usable data samples, i.e., outputs of F, during the online phase.

Trade-off attacks usually differ in the relation between these values (often expressed by a trade-off curve) and conditions that need to be met. A further distinctive feature is the concrete attack scenario. Here we are interested in two specific scenarios that we term scenario A and B, respectively, and that we explain below.

In scenario A, an attacker is given *one* image $y \in \mathcal{N}$ and tries to find a preimage under F, that is a value $x \in \mathcal{N}$ such that $F(x) = y$. This scenario-A-attacks represent the most general class of attacks. In table 2.1, we list the effort of existing trade-off attacks in scenario A. As one can see, all attacks have in scenario A a precomputation effort which is equivalent to searching the complete search space $\mathcal{N}$. In short, the reason is that a trade-off attack can only be successful if the given image y has been considered during the precomputation phase.

This can be relaxed in scenario B. Here, an attacker is given D images $y_1, \ldots, y_D$ of F and the goal is to find a preimage for any of these points, i.e., a value $x_i \in \mathcal{N}$ such that $F(x_i) = y_i$. The main difference is that for a successful attack, it isn't any longer necessary to cover the whole search space $\mathcal{N}$ during the precomputation phase. Instead, it is sufficient that at least one of the outputs y_i has been considered. An overview of time-memory-data-trade-off attacks for scenario B is given in table 2.2. Note that the parameter R mentioned in the BSW attack stands for the sampling resistance of a stream cipher. In a nutshell, it is connected to the number of special states that can be efficiently enumerated. For example, R can be defined as $2^{-\ell}$ where ℓ is the maximum

Table 2.1: Overview of trade-off attacks for scenario A

Work	Trade-off curve	Restrictions	Precomputation time
Hellman [Hel80]	$\lvert\mathcal{N}\rvert^2 = TM^2$	$1 \leq T \leq \lvert\mathcal{N}\rvert$	$T_P = \lvert\mathcal{N}\rvert$
Oechslin et al. [Oec03]	$\lvert\mathcal{N}\rvert^2 = 2TM^2$	$1 \leq T \leq \lvert\mathcal{N}\rvert$	$T_P = \lvert\mathcal{N}\rvert$
BG [Bab95], [Gol97]	$\lvert\mathcal{N}\rvert = M$	$T = 1$	$T_P = \lvert\mathcal{N}\rvert$
BS [BS00]	$\lvert\mathcal{N}\rvert^2 = TM^2$	$1 \leq T \leq \lvert\mathcal{N}\rvert$	$T_P = \lvert\mathcal{N}\rvert$
BSW [BSW01]	$\lvert\mathcal{N}\rvert^2 = TM^2$	$1 \leq T \leq \lvert\mathcal{N}\rvert$	$T_P = \lvert\mathcal{N}\rvert$
Barkan et al. [BBS06]	$\lvert\mathcal{N}\rvert^2 + \lvert\mathcal{N}\rvert M = 2TM^2$	$1 \leq T \leq \mathcal{N}$	$T_P = \lvert\mathcal{N}\rvert$
Dunkelman [DK08]	$\lvert\mathcal{N}\rvert^2 = TM^2$	$1 \leq T \leq \lvert\mathcal{N}\rvert$	$T_P = \lvert\mathcal{N}\rvert$

Table 2.2: Overview of trade-off attacks for scenario B

Work	Trade-off curve	Restrictions	Precomputation time
BG [Bab95], [Gol97]	$\lvert\mathcal{N}\rvert = TM$	$1 \leq T \leq D$	$T_P = M$
BS [BS00]	$\lvert\mathcal{N}\rvert^2 = TM^2D^2$	$D^2 \leq T \leq \lvert\mathcal{N}\rvert$	$T_P = \lvert\mathcal{N}\rvert / D$
BSW [BSW01]	$\lvert\mathcal{N}\rvert^2 = TM^2D^2$	$(DR)^2 \leq T$	$T_P = \lvert\mathcal{N}\rvert / D$
Barkan et al. [BBS06]	$\lvert\mathcal{N}\rvert^2 + \lvert\mathcal{N}\rvert D^2 M = 2TM^2$	$D^2 \leq T \leq \lvert\mathcal{N}\rvert$	$T_P = \lvert\mathcal{N}\rvert / D$

value for which the direct enumeration of all the special states which generate ℓ zero bits is possible.

Reuse Key Attack Stream ciphers are vulnerable to this attack if the same key is used twice. If two messages M_1 and M_2 are encrypted with the same key κ:

$$\mathsf{Enc}_\kappa(M_1) = M_1 \oplus \kappa, \mathsf{Enc}_\kappa(M_2) = M_2 \oplus \kappa \tag{2.1}$$

the attacker can XOR these two ciphered texts together:

$$\mathsf{Enc}_\kappa(M_1) \oplus \mathsf{Enc}_\kappa(M_2) = M_1 \oplus \kappa \oplus M_2 \oplus \kappa = M_1 \oplus M_2 \tag{2.2}$$

and retrieve the XORed value of the 2 messages.

The Berlekamp-Massey Algorithm is an iterative algorithm that finds one or all of the [Ber68, Mas69] shortest LFSR capable of generating the given sequence. This algorithm forms a universal attack on keystream generators since it carries the potential of substituting any keystream generator by its shortest linear equivalent.

Guess and Determine Attack The basic idea of guess and determine attack is to guess a few parts of the key, and using the knowledge about the keystream generator to solve the rest of the key. This attack is the classical way to cryptanalyse FSR-based stream ciphers, where only the states of a few shortest FSRs are guessed.

Algebraic Attack In an algebraic attack, the attacker is trying to build a system of algebraic equations in the unknown values, usually key bits or state bits, and the known output bits. Solving these systems allows to recover the internal state, and as long as the state update is invertible, also to find the key.

Algebraic attacks [CM03] were described against LFSR-based ciphers. The reason is that once an equation is found in the current state and output bits, analogous equations hold for any successive clocks. This results into a system of equations where each equation has a degree that is at most the degree of the initial equation. Such degree-bound systems of equations then can be solved using the linearization technique where each non-linear term is replaced by a new variable.

It has been demonstrated that the algebraic attack is efficient against a series of stream ciphers based on FSRs.

Correlation Attack The vast body of intensive research literature covers the correlation attacks. Siegenthaler first introduced the correlation attacks [Sie84] in the middle of the 80s. The basic idea is to divide and conquer when the keystream output is correlated to the individual FSR output due to the choice of the bad choice of the combining function. The optimum maximum likelihood decoding strategy results in the answer for the initial state of the FSR. Apparently, the time complexity of the basic correlation attack [Sie84] grows exponential in the length of the FSR, which is impractical for a long FSR. The focus of cryptographers has been on the general problem where the individual FSR may be arbitrarily long. To speed up the attack for the general setting, Meier and Staelbach [MS89] used the probabilistic iterative decoding strategy to refine the basic correlation attack into a so-called **fast correlation attack** to reconstruct each individual FSR.

Linear Sequential Circuit Approximation The Linear Sequential Circuit Approximation approach for analyzing keystream generators was suggested in [Gol96] and later used in [KHK06] for analyzing Trivium cipher. This approach allows for a universal distinguishing attack, i.e., to identify properties in the keystream sequence that makes it possible to find the differences from the random bit sequences. To this end, the approach enables finding a linear equation in the output bits which holds with a probability different than $1/2$.

Fault attack The systematic study of fault attacks against stream ciphers was done in [HS04]. Usually, it is assumed that the attacker can flip one random FSR bit (without knowing its position), produce the required number of keystream bits from this faulted internal state, and then compare it with the keystream, which was produced without any faults in internal state. This process of resetting the device and introducing one fault can be done as many times as it is required for the attacker. This allows to achieve additional information about the processes happening in the state of the cipher and to recover this state.

Side-Channel Attack This type of attacks is based on the information leaked from the implementation, rather than weaknesses of the algorithm itself. For example, timing information, power consumption, electromagnetic leaks, and even sound can help to break the actual implementation of the crypto primitive.

2.2.2.2 Cryptographic Properties of Keystream Sequences

To be cryptographically strong, the keystreams produced by a given KSG should have some indispensable properties. We name and explain those characteristics which are relevant to this work.

Period For a stream cipher to be secure, its keystream must have a period large enough so that the same keystream was never used twice. Otherwise, a cipher will be vulnerable to a "reused attack".

Linear complexity Linear complexity is the length of the shortest LFSR which can generate a given binary sequence. As was already mentioned, the Berlekamp-Massey algorithm [Ber68] [Mas69] is an efficient algorithm to determine the linear complexity. For a binary sequence with linear complexity L the algorithm finds an LFSR of length L which generates it, given a subsequence of length $2 \times L$.

There also exist some other measures of complexity: **Ziv-Lempel complexity**, **2-adic span**, **maximum order complexity**, etc. See [Nie99] for more details.

Nonlinearity Nonlinearity of an m variables boolean function f is the Hamming distance from f to the set of affine functions with m variables: it was first studied in [MS89] as a security measure of cryptographic Boolean functions. A function with low nonlinearity is vulnerable to the linear attacks.

Correlation Immunity This is the measure which is used to detect if the correlation attacks are easy to apply. A Boolean function is said to be correlation-immune of order m if every subset of m or fewer variables in $x_1, x_2 \ldots x_m$, is not correlated with the value of $f(x_1, x_2 \ldots x_m)$. A balanced m-order correlation immune function is called an m-**order resilient function**. [Sie84]

Algebraic Degree To resist algebraic attacks, the **algebraic degree** of a Boolean function should be high. The notion of **algebraic immunity** was introduced in [CM03], and it is defined as the minimum degree of all annihilators of the function and of its inverse.

2.3 Authentication Protocols

In a nutshell, an authentication protocol involves two parties, a prover, and a verifier. The common approach is to use challenge-response protocols based on a secret shared by these two parties. The prover authenticates itself towards the verifier by implicitly proving the knowledge of the secret. This is accomplished by answering the challenges sent by the verifier where the responses depend on the secret.

Due to their hardware restrictions, typical authentication protocols are usually not suited for embedded devices. Consequently, the search for dedicated lightweight authentication protocols became an important topic in cryptography during the last years with high relevance for academia and industry, generating a significant number of different approaches and schemes. In this work we focus on the two common approaches for constructing lightweight authentication schemes today:

- *Cipher-based schemes:* protocols which use lightweight encryption schemes as basic cryptographic operations.
- *LPN-based schemes:* protocols which are based on the well-known learning parity in the presence of noise (LPN) problem, and

We shortly explain both approaches in the following.

2.3.1 Cipher-based Protocols

Cipher-based protocols can be seen as a very straightforward approach for enabling authentication. The basic idea is that the reader chooses a random value and sends as challenge the encryption of this value to the tag. The task of the tag is to decrypt the challenge and to send back the chosen value in plaintext. Trivially, the task of correctly encrypting an unencrypted nonce chosen at random by the verifier is equivalent here. Obviously, the computational effort is mostly dominated by the execution of the deployed cipher.

2.3.2 HB-type Protocols

LPN-based protocols all adapt more or less the following principle: given a challenge $a \in GF(2)^n$, $n \in \mathbb{N}$, the response is computed as $f(a) \oplus e$, where $f : GF(2)^n \longrightarrow GF(2)$ is a secret (linear) function and e some noise bit which takes a value of 1 with a constant probability $p < 1/2$. Straightforwardly, the authentication process comprises of running the above protocol round many times and accepting finally iff the fraction of wrong answers remains below a certain threshold. The security of these kinds of protocols w.r.t passive attackers can be reduced to the widely accepted hardness of the learning parity in the presence of noise (LPN) assumption. The LPN problem can be shortly stated as the problem of distinguishing "from random several noisy inner products of random binary vectors with a random secret vector" [KPC+11].

CHAPTER 3

Evaluation of Existing Cryptographic Primitives in Real-World Scenarios

3.1 Chapter Overview

In this chapter, we evaluate the suitability of different existing cryptographic primitives for the following three real-life scenarios.

In the first scenario, investigated in [AHM14] and discussed in Section 3.2, it is assumed that a strong authentication has to be achieved on ultra low-cost RFID tags in the price range below \$0.10. We summarize the constraints applicable for this scenario and evaluated suitability of different authentication protocols for such constrained devices.

In the second scenario presented in Section 3.3, we investigate the practical consequences of the recent approach where the ciphers constantly access the key stored on the device in non-volatile memory. We summarized the existing constraints for various types of such memory and evaluated the performance of different designs under these limitations. The results were published in [MAM17].

Section 3.4 which is based on the publication [BMA+18], discusses the third scenario, where we assume that the only goal of the designer is to optimize the energy consumption when encrypting the data, whereas the other costs are not important. First, we compare the energy efficiency of block ciphers and stream ciphers. Afterwards, we investigate different stream cipher design choices and their influence on energy consumption and summarize the most critical factors.

3.2 On the Suitability of Different Authentication Protocols for Ultra-constrained RFIDs

3.2.1 Motivation and Overview

As was discussed in Subsection 1.1.1, one of the most common cases where the need for lightweight cryptographic primitives is crucial are RFID tags. Due to the economical reasons, these devices are particularly interesting for industry if their price is as low as possible. This cost pressure directly translates into severe hardware restrictions for the targeted devices. Consequently, the search for appropriate lightweight authentication protocols has become an important topic in cryptography during the last years with high relevance for academia and industry.

In this section, we describe the results of our work [AHM14], written together with Frederik Armknecht and Matthias Hamann, where we focus on authentication protocols between an RFID reader and ultra-constrained RFID tags. More precisely, we targeted devices in the cost range of \$0.05 to \$0.10. The reasons for this specific choice are twofold: Firstly, RFID tags which can be produced at costs of \$0.1 or cheaper, like (variants of) *Electronic Product Codes* (EPCs), have been a common motivation for existing work (see, e.g., [JW05], [FDW04], [MME+11], [CR08]). Secondly, if one allows for only a few additional costs, standard cryptographic primitives like AES become in fact feasible, thus practically removing the need for alternative solutions altogether (see, e.g., also Subsection 3.2.3 or [FDW04]).

Set of Conditions. Our first contribution is that we specified and argued several conditions that need to be satisfied by authentication protocols to be suitable for ultra-constrained RFID devices. These conditions have been derived partly from open literature but most importantly from various discussions with experts from industry. Although these experts were working for different companies and were aiming for RFID-based authentication in different areas, all of them share more or less the same view on what "lightweight" means in the context of ultra-constrained devices and when a scheme can be considered to be relevant for real-world applications. As these conditions mostly result from long lasting experience in hardware production and have not (or only partly) been comprehensively described and summarized in the open literature, we think that this information will be beneficial for assessing the suitability of existing protocols and for providing guidance in the development of new ones.

Evaluation of LPN-based Protocols. Our second contribution is the application of the gained knowledge for evaluating the suitability of *LPN-based protocols*. This branch of research represents the most prominent non-proprietary approach for designing lightweight authentication protocols. It has been initiated by HB [HB01] and HB$^+$[JW05], which became the prototypes for the whole family of protocols that base their security on the hardness of the learning parity in the presence of noise (LPN) assumption (or variant problems). To this end, we extracted particular parameter choices for almost 20 proposals in this section and verified whether these comply with the derived set of conditions. As it turned out, none of the existing LPN-based protocols meet the requirements, i.e., none of them can run on current low-cost RFID hardware.

In Subsection 3.2.2, we summarize the hardware limitations of currently deployed low-cost RFID devices like EPC (Gen2) tags and derive according conditions for lightweight authentication schemes. In Subsection 3.2.3 we show the principle feasibility of cipher-based approaches at the example of PRESENT [BKL$^+$07]. In Subsection 3.2.4, we then revisit existing authentication protocols that are based on the (assumed) hardness of the LPN problem and argue why these, in contrast, do not meet the limitations mentioned above.

We would like to point out that most of the results related to understanding the hardware limitations of low-cost RFID tags were obtained by Matthias Hamann, whereas the author of this thesis mostly focused on implementation and evaluation of the protocols.

3.2.2 Set of Conditions

We are going to discuss the hardware limits imposed on the design of lightweight cryptographic primitives by typical factors like, e.g., chip size, power consumption, and clock speed. As a platform, mainly low-cost RFID tags in the range of \$0.05 to \$0.10 like *Electronic Product Codes* (EPCs) are targeted. Due to their prevalence in the field of lightweight authentication hardware, we focus on *Application-specific Integrated Circuits* (ASICs) in this work. As the name suggests, ASICs are (factory-)customized for a particular task rather than intended for general-purpose usability like the more expensive class of *Field Programmable Gate Arrays* (FPGAs). To the best of our knowledge, low-cost RFID tags are the devices with the most severe limitations, meaning that if a certain cryptographic primitive is feasible on them, it should be supported by most of the different platforms based on ASIC as well. We note, however, that there also exist use-cases, where the cost of the device is less important than the energy consumption (e.g., lifetime of battery in body implants) or speed (real-time encryption of large amounts of data).

3.2.2.1 Area

In 2005 Juels and Weis [JW05] stated the "Security Gate Count Budget" of an EPC tag to be "200-2,000 gates" and, even today, this upper bound of 2,000 GEs is still commonly considered to be the magic number for lightweight cryptographic implementations [AHM14]. From an academic perspective, this conclusion can be drawn based on the fact that many recent works (see, e.g., [SE12, WZ11, PMK$^+$11, MSGAHJ13]) still assume 2,000 GEs to be the upper bound w.r.t. tag area. Some other works assume between 200 and 4,000 GEs [CR08, REC05] but are sometimes not clear about whether they are actually referring to the total area of a low-cost RFID tag or just the amount of GEs available for security purposes. Apart from academic publications, all experts from industry we spoke to in 2014 confirmed that 2,000 GEs still constitute a plausible security gate count budged for low-cost RFIDs, nine years after [JW05] was published in 2005. For comparison, one of the currently smallest known AES implementations due to Moradi et al. [MPL$^+$11] requires about 2,400 GEs, which implies that newly suggested approaches requiring even more area should at least be obliged to justify what additional benefit they bring. This obligation to justify even the need for a single additional gate has straightforward monetary reasons as, according to [CR08], 1,000 additional gates of silicon logic increase a tag's price by \$0.01, which amounts to considerable sums given production volumes of hundreds of millions in the case of low-cost RFID tags. It should also be noted that in addition to the number and placement of logic gates, other (security-related) components contribute to the chip area of an RFID tag as well. Most notably, one way to fix constant bit values (e.g., cryptographic keys) on individual tags is to use fuses/antifuses and "burn" a corresponding selection of them before a tag leaves the factory. As it has to be ensured that no other (i.e., normal logic) components get damaged during this process of burning fuses, considerable area is needed, rendering the technique infeasible when it comes to storing large amounts (i.e., thousands) of constant bits at production time. Finally, providing acceptable side channel security can also significantly increase the number of required GEs, depending on the structure of the cryptographic primitive.

3.2.2.2 Throughput and Timing

An important factor when judging any algorithm running on actual (lightweight) hardware is the expected number of clock cycles needed until completion. Trivially, the higher the underlying clock speed of a device is, the more clock cycles can be safely consumed by the cryptographic authentication process. But as pointed out in Subsubsection 3.2.2.3, factors like the power budget of a passively powered RFID tag impose an upper bound on its clock frequency. In his Ph.D. thesis [Pos09], Poschmann, one of the designers of the established lightweight block cipher PRESENT, assumes 100 kHz to be the prevalent clock speed feasible on lightweight hardware. Again, this value is in line with the information we obtained from the RFID hardware producers who demanded confidentiality. Hence, given that 150 msec is commonly considered to be the maximum amount of time available to execute a full authentication process[1], a clock speed of 100 Hz immediately implies that (even without taking the needs of other, i.e., non-cryptographic, tag components into account) no more than 15,000 clock cycles are available on the tag's side to authenticate successfully. Keep in mind that even for higher clock rates like 1 MHz (given, e.g., by Jules and Weis in [JW05]), not more than 150,000 clock cycles would be available for cryptographic and other purposes upon contact between that tag and the reader.

3.2.2.3 Power

Closely related to the amount of required hardware logic is the question of power consumption. Low-cost RFID tags are commonly powered via an electromagnetic field radiated by the reader (i.e., passively), limiting the total electric energy available over a fixed time interval, e.g., a single authentication run. Analyzing the power consumption of the lightweight authentication protocols is made especially complicated by the fact that, as compared to the introduction of new lightweight block ciphers like PRESENT [BKL$^+$07] or KATAN [DDK09], they often fail to come with an extensive assessment of their real-world hardware cost or, respectively, any reference implementation by their

[1]Perhaps surprisingly, we were told this same upper timing bound of 150 msec by various hardware producers on the basis of rather different reasons. These ranged from human interaction in the presence of additional tag functions to regulations by the automotive industry w.r.t. timing restrictions for component interaction.

Table 3.1: Application fields, transfer rates, and range of RFID technology based on waveband (adapted from [SCU11])

Waveband	Utilization	Bandwidth	Distance
Low Frequency (LF) 30-300 kHz	Animal Identification	< 10 kb/s	0.1-0.5 m
Medium Frequency (MF) 300 kHz - 3 MHz	Contactless Payment	< 50 kb/s	0.5-0.8 m
High Frequency (HF) 3-30 MHz	Access Control	< 100 kb/s	0.05-3 m
Ultra HF (UHF) 300 MHz - 3 GHz	Range Counting	< 200 kb/s	1-5 m
Super HF (SHF) 3 GHz - 30 GHz	Vehicle Identification	< 200 kb/s	ca. 10 m

authors at all. This is problematic as the amount of power an *Integrated Circuit* (IC) consumes depends on multiple design specific factors apart from the afore-mentioned number of required gates[2]. Obviously, ceteris paribus, the higher the clock speed (see Subsubsection 3.2.2.2) of a tag is, the more power must be supplied by the corresponding reader. Hence, if an algorithm depends on high clock rates to perform its operations within a reasonable time span and uses, in addition, power demanding components like EEPROMs, which will be discussed in the following paragraph 2.1.1, the power budget of a lightweight RFID tag my easily be exhausted. Another design choice which may heavily influence a tag's power consumption is the technology library used to implement it. E.g., in [RPLP08], running PRESENT at 100 kHz is compared for the libraries 0.35μ AMIS (3.3 V), 0.25μ IHP (2.7 V), and 0.18μ UMC (1.8 V), leading to different power consumptions of 11.20 μW, 4.24 μW, and 2.52 μW, respectively. These numbers are consistent with the general upper bound of 10 μW given in [JW05] by Jules and Weis for low-cost RFID tags.

3.2.2.4 Transmission Bandwidth

The operating frequency of RFID tags and, closely related, their maximum available transfer rate is determined by several factors. One of the most important is the targeted

[2]Note that the number of gates required to implement a given algorithm itself can also often be heavily influenced by design choices like employing a parallel, round-based, or serialized architecture. (see, e.g., [Pos09], for more information on this topic at the example of PRESENT)

reading distance implied by, inter alia, a tag's purpose. For example, while it may be desirable to read a complete pallet of products with attached EPC tags over a long distance, access control should rather be confined to a close environment, e.g., someone putting his access card right on top of a corresponding reader. Table 3.1 is based on the data provided in [SCU11], one of whose editors, H. Chabanne, is among the developers of the HB-type authentication protocol Trusted hB [BC08] (see Subsection 3.2.4 for further details). It should be noted, however, that there are also many authentication solutions using *Low Frequency* (LF) and that, according to the information provided in Table 3.1, most authentication solutions (i.e., LF- and HF-type implementations) are limited to exchanging data at a rate of at most 100 kb/s between a tag and a reader. Bringing to mind now that, as mentioned in the Subsubsection 3.2.2.2, the whole process of authentication should not take more than 150 msec, this implies that 15,000 bit can be considered as the upper bound for an authentication protocol's communication complexity. Furthermore, this number is even lowered by the fact that, within those 150 msec, the respective data must be processed by the tag and that, as outlined in paragraph 2.1.1, not only non-volatile memory but also volatile memory (see the corresponding data provided by Juels and Weis) is a scarce resource, which heavily limits buffering incoming data.

Random Number Generator (RNG). The hardware means of generating random numbers on a lightweight RFID tag can probably be considered the "magic bullet" with respect to authentication protocols and are most likely the main reason why all of the hardware producers we interviewed demanded to remain unnamed. In [JW05], Juels and Weis state that the random noise bit ν (and probably also the blinding factors required as part of each protocol round; see Sec. 3.2.4) "can be cheaply generated from physical properties like thermal noise, shot noise, diode breakdown noise, metastability, oscillation jitter, or any of a slew of other methods". While the listed physical properties can undoubtedly serve as a source for the generation of random bits, ensuring a sufficient level of entropy in these cases still constitutes a difficult task and is subject to research areas on its own. For example, [TBM07] presents a metastability-based *True Random Number Generator* (TRNG) fabricated in 0.13 μm bulk CMOS technology, which requires 0.145 mm^2 of area and consumes 1 mW of power (at a clock speed of 200 MHz). Even for lower clock speeds (and, hence, lower power consumptions), the required area of 0.145 mm^2 would still render this TRNG infeasible as a component (i.e., one of many parts) of a low-cost RFID tag considering that "10 US cents RFID read only chips have design sizes ranging from 0.16 mm^2 to 0.25 mm^2" [REC05] and that the RNG's "circuit should not occupy more area than $100 \times 100\mu$m" [BB08].[3] TRNGs designed

[3]In [BB08], a 0.13 μm CMOS process is used. For comparison, the AES implementation in [FWR05] is based on a 0.35 μm CMOS process and occupies 0.25 mm^2, which "compares roughly to 3400 gate equivalents" in this context.

particularly for passive RFID tags exist, too, but we are only aware of those like [BB08], which focus on generating 16-bit-long random numbers mainly meant for resolving collisions during communication. Hence, it is unclear to what extend such low-cost RNGs are actually suitable for generating large, continuous amounts of random bits (with sufficient entropy) in time as needed by many HB-type protocols for each authentication instance. For the sake of completeness, we would like to mention that there are also *Pseudorandom Number Generators* (PRNGs) aiming at low-cost scenarios, but, e.g., LAMED [PLHCETR09] still consumes roughly 1,600 GEs, which is about 600 GEs more than the lightweight block cipher PRESENT [RPLP08], which can be used straightforwardly to realize (one-way) authentication in the spirit of [FDW04] without the need for any random numbers at all on tag side. As none of the above TRNG/PRNG solutions seems to fit the scenario implied by HB-type protcols on ultra-constrained devices, at this point, we have resort to information provided to us by different experts from industry, who all agree that generating more than 128 true random bits per authentication on an RFID tag in the price range of \$0.05-\$0.10 seems currently implausible. Note, however, that none of those protocols in appendix B which are currently unbroken were ruled infeasible only because they require more than 128 bits per authentication and, in addition, many protocols exceed this number even by magnitudes. Finally, another problem particular to HB-type protocols is that they depend on a specific probability distribution w.r.t. the noise bit ν and deriving such a fixed distribution from the aforementioned sources is also everything but a trivial task.

Non-Volatile Memory (NVM). While the cost of volatile memory is often implicitly included in the numbers for area in the form of flip-flops/latches (respectively the components needed to build those), non-volatile memory is commonly provided through the use of EEPROMs. One drawback, however, to employing EEPROMs is their high latency. Moreover, from the first EEPROM memory unit on, corresponding charge pumps have to be included in the design in order to supply the high voltages necessary for memory programming. Hence, EEPROMs are not only a major cost driver in terms of money and area but also have a significant impact on a tags power budget when it comes to ultra-constrained RFID devices. Concretely, Ranasinghe and Cole state in [CR08] that, for low-cost RFID tags, the power required for a read operation amounts to 5-10 μW while “a write operation to its EEPROM will require about 50 μW or more”, which would practically allow only read operations (in the field) given the aforementioned power limitations of, e.g., EPC UHF tags, and, hence, inhibit a tag from keeping values across a loss of power (for example, between two separate authentication instances). With respect to area requirements, Nuykin et. al. propose a low-cost 640-bit EEPROM for passive RFID tags fabricated in a 0.18 μm CMOS process, which requires a total area of 0.04 mm^2. They also compare their design to several other recent suggestions, which all require at least twice the area and mostly even offer less memory (i.e., 192 bit). It is therefore not surprising that, as compared to the targeted low-cost EPC-like devices,

Table 3.2: Conditions to be satisfied by authentication protocols to be suitable for ultra-constrained RFID devices.

Cost	Meaning	Max
KC	Key storage complexity, expressed in bits	2048
NR_n, NR_b	# of uniformly distributed random bits (prover's side), required for noise (NR_n) or for blinding factors (NR_b)	128
CC	Total communication complexity, expressed in bits	30000
AS	Total area size of the implemented protocol, expressed in GEs	2,000
$Cycles$	Total # of clock-cycles required to run one full authentication	150000

even significantly more expensive RFID tags like the HITAG 1 by NXP do not provide more than 2,048 bit of EEPROM. In line with this, Juels and Weis assume "128-512 bits of read-only-storage" and "32-128 bits of volatile read-write memory" to be realistic memory resources available on low-cost RFID tags, not considering non-volatile read-write-storage at all [JW05]. Finally, our sources from industry also all agreed that 2048 bit constitute a plausible upper bound for current EEPROM sizes on ultra-constrained RFID tags in the \$0.05 to \$0.10 range.

3.2.3 Cipher-based Authentication

Before we are going to assess dedicated (HB-type) authentication protocols suggested for lightweight hardware, first we point out the existence of an intuitive and, in fact, perfectly feasible approach, which make use of existing encryption schemes: The verifier sends a random challenge to the prover, asking to encrypt it with a secret key and finally checks whether the response is correct, ultimately leading to acceptance or rejection. Typically, due to the harsh resource constraints in lightweight cryptography, only symmetric variants of encryption schemes are used as primitives for this type of protocols. In order to exemplify why, as claimed above, cipher-based authentication schemes are actually feasible in the context of ultra-constrained RFID tags, we will use a (standardized) lightweight block cipher PRESENT [BKL+07]. Subsection 3.2.4 will show that one of the main bottlenecks of HB-type protocols is their massive requirement of random numbers. In contrast, the prover (as compared to the more powerful verifier) does not need to create any random numbers at all in the case of a cipher-based authentication scheme.[4]

Similarly, also the communication complexity is much lower (some HB-type protocols need up to hundreds of thousands of bits per authentication; see Subsection 3.2.4), as in the case of PRESENT, which has a block length of 64 bit and a key length of 80 bit, a challenge consisting of two blocks, i.e., 128 bit, should be sufficient to provide

[4]If the tag should feature an RNG anyhow, PRESENT would even allow for mutual authentication at practically no additional costs as compared to the HB-type protocols assessed in Subsection 3.2.4.

the maximum possible security (otherwise, the pseudorandomness property of the underlying block cipher would be violated). A corresponding bandwidth is available on even the least powerful devices (cf. Subsection 3.2.2). The remaining conditions on low-cost RFID tags as outlined in Subsection 3.2.2 are satisfied by a PRESENT-based authentication scheme as well, according to the following numbers taken from [RPLP08]. Concretely, a serialized implementation of PRESENT requires an area of about 1,080 GEs and 563 clock cycles to process one block, both of which are well below the previously discussed limits of 2,000 GEs and 15,000 clock cycles, respectively. Finally, also the limited power budget of a low-cost RFID tag is respected, for by using 0.18μ UMC (1.8 V) as a library it is possible to reach as low as 2.52 μW given a clock speed of 100 kHz (cf. Subsection 3.2.2).

In summary, the example of PRESENT has shown that it is, in fact, possible to satisfy the conditions of low-cost hardware as outlined in Subsection 3.2.2 and still provide the required level of security. After all, PRESENT remains unbroken so far, even without claiming provable security as several HB-protocols have done in the past (cf. Subsection 3.2.4), many of which were then shown to be insecure shortly after by considering slightly different but nonetheless plausible attack scenarios.

3.2.4 LPN-based Authentication

In this section, we revisit existing protocols that are based on LPN (or related problems) and that have been suggested for lightweight applications. More precisely, we evaluate their suitability for RFID systems based on the conditions presented in Subsection 3.2.2. Our respective results for almost 20 HB-type protocols are summarized in tabular form in Subsection 3.2.5 on page 40 .

We start with a short overview of the most significant proposals for lightweight authentication protocols based on the LPN problem. As this branch of research has been initiated by the introduction of HB [HB01] and HB$^+$ [JW05], which became the prototypes for this family of protocols, these are explained in further detail Subsubsection 3.2.4.2. Moreover, at the example of HB$^+$, we discuss the main parameters which influence the security and the hardware characteristics of HB-type protocols. This allows us to identify the general cost drivers common to the HB family Subsubsection 3.2.4.3. Subsequently, we present our evaluation results for popular follow-up protocols of HB$^+$ Subsection 3.2.5.

3.2.4.1 Overview of the Considered Protocols

In 2,000, the HB [HB01] protocol was proposed, which is proven to be secure against passive attacks [KSS10]. In order to resist active attacks, HB$^+$ [JW05] was introduced that is provably secure in the detection-based model (where the adversary is able to communicate only with the tag before attempting to authenticate itself to the reader).

However, if the attacker is given the ability to modify messages which go from the reader to the tag (GRS model), the HB$^+$ protocol is not secure anymore as it was shown in [GRS05]. As a result, many new HB-type protocols were proposed in order to overcome this and other types of man-in-the-middle (MITM) attacks. In 2006, the HB^{++} protocol was introduced [BCD06], which can be seen as running HB$^+$ twice with correlated challenges and independent secrets. Later, [MP07] proposed the HB-MP protocol, which was designed to be more efficient than HB$^+$ but turned out to be vulnerable w.r.t. passive attacks [GRS08a], which is why HB-MP$^+$ [LMM08] has been suggested. Another attempt to improve the performance of HB$^+$ and to make it resistant against GRS-type MITM attacks was the HB* protocol [DK07]. In 2008 the HB$^\#$ and RANDOM-HB$^\#$ protocols were proposed, where the keys were extended from vectors to matrices [GRS08b]. Another proposal called Trusted hB [BC08] is based on the idea of using a hardware efficient hash function for verifying the integrity of the data in order to resist MITM attacks. PUF-HB [HS08] is a construction which relies on Physically Unclonable Functions (PUFs) as a hardware primitive. In the protocols NLHB [MTSV10] and GHB# [RG12], the linear functions are replaced by non-linear functions while HB^N [BHN11] can be seen as a bilinear variant of HB. In 2011, AUTH [KPC$^+$11] was proposed, where the security is based on a modified LPN problem, called the *subspace LPN problem* [Pie]. One year later, a more efficient proposal building on the ideas from [KPC$^+$11] called Lapin [HKL$^+$12] was introduced, whose security relies on the assumed hardness of the *Ring LPN-problem*.

3.2.4.2 The Procotols HB and HB$^+$ and the Main Parameters

The HB protocol [HB01] was originally developed to be used by humans and with this aim was designed to be very simple. Both the reader (verifier) and the tag (prover) share a $|x|$-bit long secret $x \in \{0,1\}^{|x|}$. The protocol is composed of several rounds that are conceptually all the same. At the beginning of round i, the verifier chooses a random challenge $a^{(i)} \in \{0,1\}^{|x|}$ and sends it to the prover, who replies with $\omega_i = (a^{(i)} \cdot x) \oplus \nu_i$, where $\nu_i \in \{0,1\}$ represents a biased random noise bit satisfying $Prob[\nu_i = 1] = \eta$ for a fixed probability $\eta \in (0, 0.5)$. Then the reader verifies whether the received bit ω_i is equal to $a^{(i)} \cdot x$. If this is the case the response is called correct and otherwise incorrect. The security of HB against passive attacks relies on the LPN problem.

The HB$^+$ protocol [JW05] was developed to resist active attacks in the detection-based model. In extension to the HB protocol, the tag and the reader share an additional secret y of length $|y|$. At the beginning of round i, the tag generates a random blinding factor $b^{(i)} \in \{0,1\}^{|y|}$ and sends it to the reader. Afterwards, similar to the HB protocol, the reader generates a challenge $a^{(i)} \in \{0,1\}^{|x|}$ and sends it to the tag. Then the tag computes $\omega_i = (a^{(i)} \cdot x) \oplus (b^{(i)} \cdot y) \oplus \nu_i$ and responds with it to the reader for verification. In the original proposal [JW05], the challenge $a^{(i)}$, the blinding factor $b^{(i)}$ and the secrets x, y all have the same length $|x| = |y|$, which is implied by the LPN problem. However, later it was shown in [LF06] that, when striving for 80-bit security in the detection-based model, the x-component of the common secret key (x, y) can be restricted to a length of $|x| = 80$ bits while the security of HB$^+$ still relies on the hardness of LPN with parameters η and $|y|$, where $|y| > |x|$ as we will explain shortly.

As already mentioned, both protocols, HB and HB$^+$, run in r rounds. Each additional round increases the confidence of the verifier. To this end, both protocols fix a parameter $u \in (\eta, 0.5)$, such that the authentication is considered to be successful if the number of incorrect answers is less than $t = u \cdot r$. Otherwise, the reader rejects the tag. If the noise probability η is chosen too close to 0.5 then a huge number of rounds is required in order to make the protocol reliable. At the same time, if η is close to 0, then for obtaining the necessary level of security of the protocols, extremely large key lengths $|x|, |y|$ are inevitable. Hence, an appropriate tradeoff needs to be found, which is specified by the choice of η.

However, besides security considerations, there also practical aspects that impact reasonable choices for η. Usually, random number generators are assumed to produce uniformly distributed random bits. In this case, it is much easier to implement instantiations where $\eta = 2^{-j}$, $j \in \mathbb{N}$, as j uniformly distributed bits are sufficient for the generation of one noise bit ν_i. However, for other values of η, many more uniformly distributed random bits may be needed to realize a corresponding random bit generator on top of those. Therefore, we restrict η to the values 0.25 and 0.125, which are in fact typical choices for HB-type protocols [LF06].

The reliability of the protocols depends on the probabilities of the possible errors. On the one hand, an honest tag may be rejected with probability P_{FR} (false rejection probability or completeness error). On the other hand, an adversary answering randomly at each round will be authenticated with probability P_{FA} (false acceptance probability or soundness error). For HB and HB$^+$ these values are computed as follows [GRS08a]:

$$P_{FR} = \sum_{i=t+1}^{r} \binom{r}{i} \eta^i (1-\eta)^{r-i} \tag{3.1}$$

$$P_{FA} = \frac{1}{2^r} \sum_{i=0}^{t} \binom{r}{i} \tag{3.2}$$

According to [LF06], P_{FA} should be less than 2^{-80} for 80-bit security and P_{FR} should be less then 2^{-40}. In order to achieve such bounds for soundness and completeness errors, an appropriate combination of the parameters η, r, u needs to be chosen. In [LF06], for each value of η suitable values for u and r were computed, leading to the following two choices:

- Variant 1: $\eta = 0.25$, $u = 0.348$, $r = 1164$
- Variant 2: $\eta = 0.125$, $u = 0.256$, $r = 441$

Table 3.3: Parameter choices for the protocol HB$^+$

Variant	η	$\|x\|$	$\|y\|$	Sec	r	u	P_{FR}	P_{FA}
1	0.25	80	512	2^{80}	1164	0.348	2^{-45}	2^{-83}
2	0.125	80	512	2^{77}	441	0.256	2^{-45}	2^{-83}
3	0.125	80	512	2^{77}	256	0.1875	0	2^{-81}

In one of the proposed protocols afterwards [GRS08b], the following trick has been suggested based on ideas from [KS06]: if the tag computes in advance r noise bits and keeps them only if the number of **1**s is less than $u \cdot r$, then the completeness error P_{FR} will be equal to 0. The advantage of this approach is that the number of rounds r can be reduced, while the soundness error is kept small. Our evaluation takes this approach into account as well and uses the parameters provided in [GRS08b]: $r = 256, \eta = 0.125, u = 0.1875$ (variant 3). Summing up, in this work we evaluate, where possible, each protocol in all three explained variants.

As mentioned above, the protocols' security relies on the LPN problem. The proofs of security for HB against passive attacks and for HB$^+$ against active attacks were simplified by Katz et al. and extended to the parallel versions of the protocols [KS06, KSS10], which means that several rounds can be performed at the same time. Based on the state-of-the-art heuristic algorithm for solving the LPN problem [LF06], reasonable parameter choices for achieving (almost) 80-bit security are $|x| = 512$ for HB and $|x| = 80$, $|y| = 512$ for HB$^+$. In these cases, solving the LPN-problem would take 2^{89} bytes of memory if $\eta = 0.25$ and 2^{77} bytes of memory if $\eta = 0.125$. Our parameter choices for HB$^+$ implementations are summarized in the Table 3.3.

3.2.4.3 Cost Drivers of LPN-based Protocols

In Subsection 3.2.2, we have established a concrete notion of the term *lightweight* in the RFID context by providing actual hardware limits for low-cost tags. As our goal is to assess for (allegedly) lightweight authentication protocols whether they in fact comply to all of the respective hardware limits, we first need to identify the major cost drivers of such schemes. In particular, we will discuss for each of the following protocol properties how it is linked to the hardware properties of RFID tags in the \$0.05 to \$0.10 cost range discussed in Subsection 3.2.2.

Symmetric Key. All HB-type authentication protocols use symmetric keys[5]. Consequently, the full shared secret, must permanently be available on the (passively powered)

[5]For the sake of simplicity, in this subsection, the term *key* will always be used to refer to the shared secret's unique representation as a binary vector in the corresponding scheme, irrespective of potential blow-up measures like, e.g., the use of Toeplitz matrices. In particular, the key size lower bounds the size of the individual key storage required on each tag.

tag, hence implying the need for some non-volatile *key storage*. Depending on the deployment scenario, multiple (e.g., batches of) RFID tags might share a single key or, in other cases, tag-individual secrets may be required.

Closely related, but even more restrictive w.r.t. key storage options, is a potential need to set or change the secret key of a tag that is already in the field, as compared to irreversibly fixing the key once at production time. In the latter case, key-dependent masks may be used in the factory to apply the secret keys directly to so-called wafers in the process of creating *Integrated Circuits* (ICs) for low-cost RFID tags. However, while this can alleviate the need for additional components like EEPROMs or fuses (cf. Subsection 3.2.2 - *Area*), it inevitably results in the potentially dangerous situation that large quantities of tags will now share the same irreversible key. Concretely, as production costs increase with each new mask (by thousands of U.S. dollars), the size of per-mask-batches must be big enough (i.e., hundreds of thousands or even millions of devices) to allow for per-tag savings (e.g., by removing the need for EEPROMs) which compensate for the additional costs of using multiple masks. At the same time, an attacker's outlook on, e.g., counterfeiting large amounts of items who are all protected by tags using the same key, may now easily justify the costs for retrieving the respective key by means of reverse engineering (for instance, through etching and the use of an electron microscope).

Ultimately, if the deployment scenario requires fully individual keys, the use of masks is clearly not feasible anymore and two other, more flexible options remain: EEPROMs and fuses, whose major hardware properties and limitations w.r.t. low-cost RFID tags were summarized in Subsection 3.2.2. These general preconditions will now be compared to the requirements imposed by how symmetric keys are chosen and used in HB-type protocols. Clearly, EEPROMs offer the highest degree of flexibility as a key storage, allowing, e.g., to redeploy existing tags after changing their keys. On contrast, when resorting to fuses, keys are irreversible and need to be written already at production time. However, unlike masks, fuses allow for individual keys on a per tag basis. Hence, as fuses neither suffer from the high power consumption nor from the latency problems characteristic of EEPROMs, they are a viable option when individual but fixed keys are required.

Unfortunately, in the context of HB-type authentication protocols, key storage options are further restricted by the large key size common to these schemes. In [KPC$^+$11], key sizes for multiple HB-type protocols are specified on the basis of the parameter l denoting the length of an LPN secret. For example, the key size of the original HB$^+$ protocol, i.e., the variant suggested by Juels and Weis in [JW05], is given by $2l$ along with $l = 500$ described as a "typical parameter". Please note that the resulting key length of 1000 bit is even at the lower end of the protocols summarized in [KPC$^+$11] (which range from l bit for the original HB protocol [HB01], over $4.2 \cdot l$ bit for AUTH [KPC$^+$11], up to $80 \cdot l = 40,000$ bit for a MITM-secure protocol also suggested in [KPC$^+$11]; see Subsection 3.2.5 for further details). However, e.g. due to area requirements, already for

1,000 bit it seems highly questionable whether fuses can still be considered a feasible option for storing the secret key on a low-cost RFID tag. Moreover, it is easy to see that, similar to (or even worse than) masks, fuses fail to provide substantial physical security. Ultimately, it depends on the deployment scenario whether this is an actual thread, hence requiring the use of, e.g., EEPROMs instead. Bring to mind, however, that in the context of low-cost RFID devices, EEPROMs typically do not allow for storing more than 2,048 bit. As a result, it must be suspected that many of the HB-type protocols discussed in Subsection 3.2.5 are already precluded by their key size from practical application on RFID tags in the \$0.05 to \$0.10 range.

Challenges, Blinding Factors, and Noise Bits. Another property characteristic of HB-type protocols is their heavy use of challenges and what is often referred to as blinding factors. For most HB-type protocols, the following three phases per round can be identified: (1) The prover creates a vector of random bits, the so-called blinding factor, which is then transmitted wirelessly to the verifier. (2) Just alike, the verifier now also creates a random bit vector and sends it to the tag. (3) Depending on the specific protocol, the prover deterministically computes some 1-bit value based on the blinding factor in (1), the challenge in (2), as well as the secret/shared key. Finally, he needs to produce one more random bit, which, on contrast to the aforementioned challenge and blinding vectors, is not based on the uniform but some other, fixed distribution. Adding this so-called noise bit to the 1-bit value yielded by the previous operation is crucial to the security of HB-type protocols, as described in Subsubsection 3.2.4.2. The resulting bit is then sent to the verifier, who will check whether it is correct or not. In the following paragraph, we will denote the number of protocol rounds per authentication run by r and, for reasons of simplicity, assume that the blinding vector in step (1) as well as the challenge vector in step (2) are both of length l, i.e., the size of the secret key (as done in the original HB$^+$ paper [JW05] and popular follow-up works like [KPC$^+$11]).

Apparently, the protocol scheme we just outlined makes heavy use of at least two hardware resources previously identified as potential bottlenecks for low-cost RFID tags: the transmission bandwidth (Subsection 3.2.2) and the generation of random numbers. Concretely, in each round of the above archetypical example, the communication complexity amounts to $2l + 1$ and the prover needs to obtain l uniformly random bits and 1 differently distributed random bit from his RNG. Hence, a single authentication procedure consisting of r rounds has a communication complexity of at approximately $2 \cdot l \cdot r$ bit and requires at least $r \cdot l$ random bits on the prover's side. As in the previous paragraph about key sizes, let us exemplify the actual consequences of these complexities for HB-type protocols using parameters described as "typical" in [KPC$^+$11]: $l = 500$ and $r = 250$. Moreover, as justified in Subsection 3.2.2, let us consider 150 msec to be the maximum time available for a complete authentication. As a result, at least $2 \cdot 500 \cdot 250 = 250,000$ bit would need to be transmitted within 150 msec, corresponding to a vastly implausible transmission rate of $250,000/0.15$ bit/s ≈ 1.66 Mbit/s (as compared to actual values between 10,000 bit/s and 200,000 bit/s as given in Subsection 3.2.2). Similarly far from reality is the idea that an RFID tag whose production costs are in the \$0.05-\$0.10 range could actually feature an RNG delivering as much as $500 \cdot 250 = 125,000$ uniformly distributed random bits within just 150 msection Apart from the apparent bottlenecks transmission bandwidth and generation of random numbers, the generalizing description of HB-type protocol at the beginning of this paragraph contained a third aspect worth investigating. Concretely, depending on the involved operations, the first computation in step (3) can easily turn out to consume (possibly too) many clock cycles, especially in view of the fact that three operands of bit-length 500 are involved. As this is highly protocol-specific and implementation-dependent (e.g., parallel vs. serial processing in step (3) of HB$^+$) though, the question of computational complexity will be treated, where of importance, in the corresponding of Subsection 3.2.5.

3.2.5 Evaluation and Implementation of the Protocols.

In the following, we explain how the evaluation results for the considered protocols have been derived. To this end, we first provide for each protocol a short description. For the detailed descriptions of the protocols the readers are referred to the original papers, where the protocols were proposed. Afterwards, we justify the chosen parameters and provide the formulas to calculate the costs which do not depend on the implementation choices (communication complexity, required number of random bits generated by the tag, key storage complexity). Most of the protocols can be shown to be infeasible on lightweight RFID hardware already because of these costs. Nevertheless, in order to evaluate the area size and the clock-cycles required for the protocols we implemented them. We explain the implementation issues and our choices and provide the implementation results, which are summarized in the corresponding tables for each of the protocols. All the costs which violate the limits are underlined and are marked in red colour. In this section, we use the notations explained in Table 3.2.

3.2.5.1 HB and HB$^+$

The protocols HB and HB$^+$ have been described already in Subsubsection 3.2.4.2 and the parameters have been justified. Hence, we skip these parts and start with providing the formulas for computing the costs listed in Table 3.2.

Costs evaluation formulas: In case of HB:

$$KC = |x|,$$
$$NR_n = -\log_2(\eta) \cdot r,$$

$$NR_b = 0,$$
$$CC = (|x| + 1) \cdot r$$

In the case of HB$^+$, the *res*pective formulas are:

$$KC = |x| + |y|,$$
$$NR_n = -\log_2(\eta) \cdot r,$$
$$NR_b = |y| \cdot r,$$
$$CC = (|x| + |y| + 1) \cdot r$$

Implementation Issues: In most HB-like protocols the functions which are to be computed by the tag in order to compute the value of the proof ω_i are very simple and can be divided by many similar independent operations. This makes the protocols to be easily scalable.

For example, in case of HB protocol, i.e. it is required that the tag computes the binary product $a^{(i)} \cdot x + \nu$. As it is mentioned in [JW05], it is not necessary to store the whole vector $a^{(i)}$ on the tag side, instead after d bits of $a^{(i)}$ are received from the reader they can be multiplied on the fly with the corresponding d bits of x and the intermediate result of the sum of these d products can be stored in a memory gate. When the last block of d bits of $a^{(i)}$ is received and is multiplied with the corresponding d key bits, the results of multiplication are XORed not only with each other but also with the noise bit ν_i. Obviously, the smaller d is chosen, the more clock-cycles are required to perform all the operations required for the computation of the whole binary product. At the same time, when d is chosen to be bigger more logic gates are necessary to multiply the d bits of $a^{(i)}$ by the corresponding d bits of the secret x.

In the case of HB$^+$ the two vector multiplications have to be performed on the tag side $b^{(i)} \cdot y$ and $a^{(i)} \cdot x$. Again, there is no need to store neither $b^{(i)}$ nor $a^{(i)}$. After the d random bits are generated they can be multiplied by the corresponding bits of the key y and sent out. The second stage (multiplying $a^{(i)} \cdot x$ and adding the noise bit ν_i) is performed the same way as in HB.

We implemented the protocols HB and HB$^+$ for different values of d. The results are shown in the Table 3.6 and Table 3.7.

Even though, the computational part of the protocols is very small for the small values of d, the total area size is rather big. This happens due to the fact that during every clock-cycle it is required that the corresponding block of d key bits is selected before these bits pass to the input of the computational part of the circuit. For the small values of d the selection mechanism requires big number of multiplexers, which increases the area size. Therefore, the total area size required to implement the protocols (the similar holds for most of the protocols discussed in this section) does not differ much for different choices of $d \leq 32$. However, increasing d by the factor of 2 halves the number of clock-cycles required for authentication.

Although, it is already indicated by the implementation-independent values of CC, NR_b and NR_n that the protocols are unrealistic, we implemented them in order to evaluate the area size and the number of clock-cycles required, assuming that all of the other problems are somehow solved.

Evaluation and implementation results

Table 3.4: Evaluation results for HB

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, \lvert x\rvert = 512, r = 1164$	512	2328	0	597132	$CC > 30000$, $NR_n > 128$ for all sets of parameters. .
$\eta = 0.125, \lvert x\rvert = 512, r = 441$	512	1323	0	226233	
$\eta = 0.125, \lvert x\rvert = 512, r = 256$	512	770*	0	131328	

* The average number of random bits is given.

Table 3.5: Evaluation results for HB$^+$

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 1164$	592	2328	59568	690252	$CC > 30000$, $NR_b \gg 128$.
$\eta = 0.125, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 441$	592	1323	225792	261513	
$\eta = 0.125, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 256$	592	770*	131072	151808	

* The average number of random bits is given.

Table 3.6: Implementaion results of HB for the different number of bits d processed per clock-cycle

	$d = 1$		$d = 8$		$d = 16$		$d = 32$		$d = 512$	
	AS	$Cycles$	AS	$Cycles$	AS	$Cycles$	AS	$Cycles$	AS	$Cycles$
$r = 256$	946	131330	971	16442	971	8450	1017	4354	1876	514
$r = 441$	946	226235	971	28667	971	14555	1017	7499	1876	884
$r = 1164$	963	597134	988	75662	989	38414	1034	19790	1893	2330

Table 3.7: Implementation results of HB$^+$ for the different number of bits d processed per clock-cycle

	$d = 1$		$d = 8$		$d = 16$		$d = 512(80)$*	
	AS	*Cycles*	*AS*	*Cycles*	*AS*	*Cycles*	*AS*	*Cycles*
$r = 256$	946	151810	971	19202	971	9728	2195	594
$r = 441$	946	261956	971	33077	971	16760	2203	964
$r = 1164$	963	691418	988	87302	989	44234	2215	2410

* In this case the whole vectors are multiplied $b^{(i)} \cdot y$ and $a^{(i)} \cdot x$ during one clock-cycle

Suitability and security: Both protocols are infeasible on the lightweight RFID hardware because of the communication complexity and required number of random bits generated by the prover (RFID tag). Moreover the straightforward implementation (d=1) also requires infeasible amount clock-cycles to run one authentication of the protocol. HB protocol is insecure w.r.t. active attacks [HB01] and HB$^+$ is insecure w.r.t. GRS-MITM attack [GRS05].

3.2.5.2 HB^{++}

Description: In 2006 Bringer et al. [BCD06] proposed the protocol HB^{++}. In this version the reader and the tag share one secret of 768 bits which is used for generating session keys. The protocol consists of two steps. During the first step the reader and the tag exchange 80-bit challenges, which, together with the shared secret, are used as the inputs for a hash function h. The output of this function are four 80-bit temporary keys x, y, x', y' which are used during the second step. The second step of HB^{++} can be considered as running HB$^+$ twice with correlated challenges and blinding factors and independent temporary keys x, y, x', y'. Similar to HB$^+$, the tag and the reader exchange the blinding factor $b^{(i)}$ and the challenge $a^{(i)}$. Afterwards the tag computes and sends the two values

$$\omega_i = \left(a^{(i)} \cdot x\right) \oplus \left(b^{(i)} \cdot y\right) \oplus \nu_i$$
$$\omega_i' = \left(rot_i\left(f\left(a^{(i)}\right)\right) \cdot x'\right) \oplus \left(rot_i\left(f\left(b^{(i)}\right)\right) \cdot y'\right) \oplus \nu_i',$$

where $rot_i(\beta)$ denotes rotation of β by i bits to the left and f is a function with special properties. A MITM attack against the protocol is presented in [GRS08a].

Parameter choices: [BCD06] gives exact specifications of the hash function which results in secrets of 80 bits length. However, no concrete values have been recommended for the noise rate η and the number of rounds r. According to [GRS08a] the completeness and soundness errors for HB^{++} are given by

$$P_{FR} = 1 - \left(\sum_{i=0}^{t} \binom{r}{i} \eta^i (1-\eta)^{r-i} \right)^2,$$

$$P_{FA} = \left(\frac{1}{2^r} \sum_{i=0}^{t} \binom{r}{i} \right)^2.$$

We calculated the smallest number of rounds r such that $P_{FR} \leq 2^{-40}$ and $P_{FA} \leq 2^{-80}$. The results are the following: $r = 282$ when $\eta = 0.125, u = 0.285$, and $r = 731$ when $\eta = 0.25, u = 0.368$.

Costs evaluation formulas:

$$\begin{aligned} KC &= 768, \\ NR_n &= -\log_2(\eta) \cdot r, \\ NR_b &= 81 \cdot r, \\ CC &= 80 \cdot 2 + (|x| + |y| + 2) \cdot r \end{aligned}$$

Implementation issues: If the session keys are generated before the second step begins, they have to be stored. Hence 320 additional flip-flops are needed for this purpose. Please note that using one flip-flop of the smallest size increases the area by approximately 6 GEs. Another issue which makes HB^{++} not efficient is that the vectors $f\left(a^{(i)}\right)$ and $f\left(b^{(i)}\right)$ are to be rotated by i bits, where i is the number of the current round. In order to rotate these vectors, we had to use additional 80-bits long register. In Table 3.9 we provide implementation results for the HB^{++}.

Evaluation and implementation results: The evaluation and implementation results for HB^{++} are given in the Table 3.8 and Table 3.9 respectively.

Table 3.8: Evaluation results for HB^{++}

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, r = 731$	768	2924	58560	118582	$CC > 30000$,
$\eta = 0.125, r = 282$	768	1692	22640	45844	$NR_b + NR_n > 128$.

Table 3.9: Implementation results for HB^{++}

	d	AS	$Cycles$
$r = 282$	1	6783	57421
$r = 731$	1	6812	148220

Suitability and security: The communication complexity and the required number of random bits in HB^{++} is not feasible on ultra-constrained tags. The area size also significantly exceeds 2,000 GE. MITM attack against the HB^{++} was presented in [GRS08a].

3.2.5.3 HB-MP and HB-MP$^+$

Description: The HB-MP protocol was proposed by Mulilla and Peinado in 2007 [MP07]. In this protocol the reader and the tag share two secrets $x, y \in \{0,1\}^{|k|}$ whose length $|k|$ is bigger than the length of the exchanged messages m. During one round of HP-MP two messages are transmitted instead of three as it is the case for HB$^+$. The tag receives a challenge $a^{(i)} \in \{0,1\}^m$ from the reader and then generates a random binary vector $b^{(i)} \in \{0,1\}^m$. Tag computes the value

$$\omega_i = a^{(i)} \cdot |x|_m^{(i)}$$

where $|x|_m^{(i)} \in \{0,1\}^m$ denotes m least significant bits of $x^{(i)}$. Afterwards the following condition is checked:

$$b^{(i)} \cdot |x|_m^{(i)} = \omega_i \tag{3.3}$$

At each round i, the secret $x^{(i)}$ can be rotated by one bit or not, depending on the value of y_i:

$$x^{(i)} = rot_{y_i}(x^{(i-1)}).$$

Here, $x^{(i)}$ denotes the state of the key x during the i-th round and $x^{(0)} = x$. If condition (3.3) is satisfied then $b^{(i)}$ is sent to the reader. Otherwise, a new vector $b^{(i)}$ is generated. The maximum number of such iterations is n. If after n iterations condition (3.3) has still not been fullfilled, the value $b^{(i)}$ is sent nonetheless.

The HB-MP$^+$ protocol [LMM08] was introduced to resist certain passive attack [GRS08a] and differs from HB-MP in the way how the round secret $|x|_m^{(i)}$ is computed. The authors suggest to include a special one-way function which takes the challenge $a^{(i)}$ and the secret y as the inputs and outputs $|x|_m^{(i)}$.

Parameter choices: It was proven in [MP07] that the protocol security relies on the LPN problem with the parameters m, η, where $\eta = 1/2^{n+1}$. Hence to achieve $\eta = 0.25$ (see the discussion in Subsubsection 3.2.4.2), the value n has to be equal to 1, meaning that the condition (3.3) is tested only one time per round. Consequently for $\eta = 0.125$ condition (3.3) testing is performed at most $n = 2$ times. It is not clear from [MP07] how large k should be chosen. Hence, to get the best performance, we set it in our evaluation as small as possible, i.e., $k + 1 = 513$. Therefore, based on the Table 3.3 we choose the following parameters:

- for $n = 1$, we have $|k| = 513, m = 512, r = 1164$,.
- for $n = 2$, we have $|k| = 513, m = 512, r = 441$.

In [LMM08] it is stated that due to the fact that the secrets are new at each authentication of HB-MP$^+$ it is sufficient to choose $keylengh = 512, m = 224, n = 1, r = 1,164$. We decided to follow these recommendations and provide the costs for HB-MP$^+$ based on these parameters.

Costs evaluation formulas:

$$\begin{aligned} KC &= k \cdot 2, \\ NR_n &= 0, \\ NR_b &= r \cdot \sum_{i=0}^{n} \frac{1}{2^n} m, \\ CC &= 2 \cdot m \cdot r \end{aligned}$$

Implementation issues: In the HB-MP protocol the computation of ω_i can also be done on the fly similar to the cases of HB and HB$^+$. However, in this protocol the value $b^{(i)}$ has to be available before and after checking condition (3.3) is checked, and, therefore, $b^{(i)}$ has to be stored. It requires 512 additional flip-flops which already makes the area size infeasible in lightweight RFID tags. The second implementation issue of this protocol is the need to rotate the secret $x^{(i)}$. To this end, we had to use another register composed of additional 513 flip-flops.

Due to the fact that based on the fulfillment of condition (3.3) during every round the value of $b^{(i)}$ has to be generated either one, two or three times, the actual number of clock-cycles is always different. Therefore, we provide the upper and the lower bounds for this cost.

The exact specifications for the one-way function of HB-MP$^+$ [LMM08] are not provided. Therefore, we cannot implement the protocol. Nevertheless, taking into account the chosen parameters $|k| = 512, m = 224$, it is clear that the area size is several times larger than available 2,000 GEs due to the following reasons. Similar to the case of HB-MP it is required to store the $m = 224$-bits vector $b^{(i)}$ and to rotate the $|k| = 512$-long secret x. Therefore, the minimum number of additional 736 flip-flops are to be used. This amount of memory already requires more than 4,000 GEs.

Evaluation and implementation results: The evaluation and implementation results for the protocols HB-MP and HB-MP$^+$ are given in Table 3.10,Table 3.11 and Table 3.12.

Table 3.10: Evaluation results for HB-MP

Parameters	KC	NR_n	NR_b	CC	Suitability
$n = 1, \|k\| = 513, m = 512, r = 1164$	1026	0	893952*	1191936	$CC > 30000$,
$n = 2, \|k\| = 513, m = 512, r = 441$	1026	0	395136*	451584	$NR_b > 128$..

* The average number of random bits is given.

Table 3.11: Evaluation results for HB-MP$^+$

Parameters	KC	NR_n	NR_b	CC	Suitability
$n = 2, \|k\| = 512, m = 224, r = 1164$	1024	0	391104*	521472	$CC > 30000$, $NR_b \gg 128$.

* The average number of random bits is given.

Table 3.12: Implementation results for HB-MP

	$d = 8$			$d = 16$		
	AS	Min $Cycles$	Max $Cycles$	AS	Min $Cycles$	Max $Cycles$
$r = 441$	9772	57330	113779	9821	29106	45864
$r = 1164$	9917	151320	225816	9982	76826	114072

Suitability and security: For HB-MP, the communication complexity and the number of required random bits to be generated on the prover side are infeasible. Moreover, as one can see from the Table 3.12, in order to have the total number of clock cycles to be less than 150, 000 unconditionally, the number of bits generated and the number of bits transmitted during the time of one clock-cycle has to be $d = 16$, when the number of rounds $r = 1164$ and $d = 8$, when $r = 441$. The area size is bigger than available 2,000 GEs by approximately factor of 5 in all the cases. For HB-MP$^+$, the situation is similar w.r.t. the communication complexity and the possibility to generate required number of random bits. Even though we did not implement it, we expect that the area size will excess the limit of 2, 000 GE, no matter how efficient the one-way function is chosen (see implementation issues part). In [GRS08a] a passive attack against HB-MP is indicated.

3.2.5.4 HB*

Description: In the HB* protocol [DK07], that was proposed by Kim and Duc in 2007, four $|k|$-bit secrets x, y, v, s are used. The reader generates not only a random challenge $a^{(i)} \in \{0,1\}^{|k|}$, but also an additional random bit γ_i. The secret s is used for securely transmitting the value of γ_i to the tag. This is done as following: instead of sending a pair $(a^{(i)}, \gamma_i)$ the reader transmits $(a^{(i)}, \omega_i)$, where

$$\omega_i = (a^{(i)} \cdot v) + \gamma_i$$

The tag computes the value of γ_i as:

$$\gamma_i = (a^{(i)} \cdot v) + \omega_i$$

Afterwards the tag performs the similar actions: generates a random blinding factor $b^{(i)}$ and a random bit γ'_i. If $\gamma_i = \gamma'_i$ then the tag computes ω_i as:

$$\omega_i = (a^{(i)} \cdot x) \oplus (b^{(i)} \cdot y) \tag{3.4}$$

otherwise the roles of x and y are swapped:

$$\omega_i = (a^{(i)} \cdot y) \oplus (b^{(i)} \cdot x) \tag{3.5}$$

The secret v is used for transmitting the value of γ'_i to the reader, i.e. the values $b^{(i)}, \omega'_i, \omega_i$ are transmitted, where

$$\omega'_i = (b^{(i)} \cdot s) + \gamma'_i \tag{3.6}$$

The tag is verified if the value of ω_i is correct.

Interestingly, the papers [PT07, GRS08a, SKII12], where the attacks against HB* are described, use a different description of the protocol. In fact, in our point of view these two protocols are not the same. To be on the safe side we considered both descriptions in our evaluation. To be able to distinguish between the two schemes, we denote the protocol considered in the papers mentioned above as HB^{*1}. Although we were not able to find an official explanation for this fact, we came to the conclusion after a careful examination that HB^{*1} was introduced as an improvement of HB*. We briefly justify our conclusion. The first difference between these two protocols is that the number of secrets in HB^{*1} is reduced by one (v is not used anymore) and the step where the reader generates γ_i and sends it to the tag is omitted. From our point of view this step was redundant anyhow. Instead of checking the condition $\gamma_i = \gamma'_i$, it is sufficient if the tag checks if $\gamma'_i = 1$. This allows for the same level of security, because the probability of swapping x and y while computing ω_i remains 0.5. The second improvement is that the noise bit ν_i is included in the computation of ω_i: if $\gamma'_i = 1$ then $\omega_i = (a^{(i)} \cdot x) \oplus (b^{(i)} \cdot y) \oplus \nu_i$ while in the other case $\omega_i = (a^{(i)} \cdot y) \oplus (b^{(i)} \cdot x) \oplus \nu_i$. This obviously improves the security of the protocol.

Parameter choices: The properties of HB* allow for setting several parameters to smaller values. The first important difference to HB$^+$ protocol is that no noise is added to the value ω_i. Hence the tag is accepted only if all the bits z are correct. This results into the situation when the completeness error is 0 (it is not possible that the reader rejects the honest tag). In order to provide 2^{-80} soundness error, $r = 80$ rounds are enough. Moreover, as it is stated in [DK07], due to the fact that the values γ_i and γ_i' are drawn from the uniform distribution, the protocol's security relies on two instances of LPN problem with noise factor $\eta = 0.5$. According to the results from [LF06], in this case one can choose $k = 256$ for achieving 80-bit security. To sum up, we use the following parameters for HB*: $k = 256, r = 80$;

For the HB^{*1} the values of P_{FA} and P_{FR} are computed in the same way (see (3.1) and (3.2)) as for HB$^+$. Therefore, we evaluated the protocol based on the same parameters as for HB$^+$ (cf. Table 3.3).

Costs evaluation formulas: For HB*

$$\begin{aligned} KC &= |k| \cdot 4, \\ NR_n &= r, \\ NR_b &= r \cdot |k|, \\ CC &= (2 \cdot |k| + 3) \cdot r \end{aligned}$$

and for HB^{*1}

$$\begin{aligned} KC &= |k| \cdot 3, \\ NR_n &= (-\log_2(\eta) + 1) \cdot r, \\ NR_b &= r \cdot |k|, \\ CC &= (2 \cdot |k| + 2) \cdot r \end{aligned}$$

Implementation issues: The implementation results are given in the Table 3.15. Due to the fact that $r = 80$ rounds is enough for HB*, even for $d = 1$ the computational time doesn't exceed 150,000 clock-cycles. It is not the case for the HB^{*1}, where for both sets of parameters only choosing $d = 8$ results in a feasible computation time. The area size of the HB^{*1} protocol is slightly bigger than 2,000 GEs because of the following reasons. The main factor which leads to the increase of the area compared to HB$^+$ is that the total shared secret size is much bigger in HB^{*1}, meaning that more multiplexers have to be used for the key bits selecting mechanism. The second reason is that, as mentioned above, in the protocols HB* and HB^{*1} the value of ω_i can be computed in two different ways (see Equation (3.4), and Equation (3.5)). Therefore we compute the both expressions simultaneously on the fly and save the intermediate results in two memory gates, which doubles the number of logic gates, used for computing the value of ω_i. Also, compared to HB$^+$ protocol, additional logic is required in order to calculate the function (3.6) which serves for secure transmitting of the generated noise bit γ_i' to the reader.

Evaluation and implementation results:

Table 3.13: Evaluation results for HB*

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.5 \vert k \vert, = 256, r = 80$	1024	80	20480	41200	$CC > 30000$, $NR_b \gg 128$.

Table 3.14: Evaluation results for HB^{*1}

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, \vert k \vert = 512, r = 1164$	1536	3492	595968	598296	$CC > 30000$, $NR_b \gg 128$.
$\eta = 0.125, \vert k \vert = 512, r = 441$	1536	1764	225792	226674	

Table 3.15: Implementation results for HB* and HB^{*1}

Variant	r	d	AS	$Cycles$
HB*	80	1	1776	41280
HB^{*1}	441	8	2514	57332
	1164	8	2514	151321

Suitability and security: Even-though, HB* protocol is significantly more efficient than HB$^+$, it still cannot be implemented on lightweight RFID hardware due to the similar reasons, namely the set of parameters violates conditions on the communication complexity and the required number of random bits. The version HB^{*1} appeared to be even less efficient compared to HB$^+$, and except having the same problems, the implementation of the protocol requires more than 2,000 GEs. Based on the fact that even the stronger HB^{*1} version of the protocol is broken [GRS08a], we consider HB* to be insecure as well.

3.2.5.5 Trusted HB

Description: The protocol Trusted HB [BC08] consists of two steps. The first step is equivalent to HB$^+$. If the reader verifies the tag during step 1, then step 2 takes place. The tag sends a hash value of the concatenation of all blinding factors, challenges $b^{(i)}, a^{(i)}$ and ω_i, for every $i \in [1..r]$ to the reader in order to guarantee that these values were not modified and thus protecting from MITM attacks. Nonetheless, the protocol has been also broken in [FS09].

Parameter choices: According to the recommendations of the authors the following parameters were chosen: $|x| = 80, |y| = 512, \eta = 0.25, t = 0.348, r = 1,164$, which is equivalent to variant 1 from the Table 3.3. We didn't consider other parameter sets, as it would have required to modify the hash function used in Trusted HB. It is suggested in [BC08] that the hash function is constructed from the Toeplitz hash family proposed in [Kra94]. This function uses an LFSR whose initial state has to be shared between the reader and the tag, introducing another secret of length 101 bits.

Costs evaluation formulas:

$$\begin{aligned} KC &= |x| + |y| + 101, \\ NR_n &= -\log_2(\eta) \cdot r, \\ NR_b &= |y| \cdot r, \\ CC &= (|x| + |y| + 1) \cdot r + 101 \end{aligned}$$

Implementation issues: According to [BCD06, FS09] to avoid storing all the values $a^{(i)}, b^{(i)}, \omega_i$, the hash function in Trusted HB is computed progressively. The specifications of this family of hash functions [Kra94] implies that during i-th round of Trusted HB the values of $a^{(i)}, b^{(i)}, \omega_i$ are fed into the hash function bit by bit, meaning that choosing d to be larger than 1 does not help to save the time, because the number of clock-cycles required for the hash function is fixed and is equal to $r \cdot (|x| + |y| + 1) = 1,164 \cdot 593 = 690,252$. Moreover, the implementation of this hash function requires two registers: LFSR and the accumulator each of $n = 101$ bits long and also rather large amount of logic gates which significantly increases the area size compared to HB$^+$. The implementation results are provided in the Table 3.17 :

Evaluation and implementation results:

Table 3.16: Evaluation results for Trusted-HB

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, \|x\| = 80, \|y\| = 512, r = 1164$	693	2328	595968	690353	$CC > 30000$, $NR_b \gg 128$

Table 3.17: Implementation results for Trusted HB

	d	AS	Cycles
$r = 1164$	1	4394	691416

Suitability and security: The protocol requires almost the same number of bits to be transmitted between the reader and the tag during one authentication attempt and the same number of randomly generated bits on the tag side as HB$^+$. Hence it is infeasible on ultra lightweight tags. Use of the hash function narrows the implementation choices and also results in the significant increase of the area size. The MITM attack against the protocol is presented in [FS09].

3.2.5.6 Random HB# and HB#

Description: These protocols were developed by Gilbert et al. in 2008 [GRS08b]. Both protocols require only one round for the whole authentication procedure. The main difference between Random HB# and HB$^+$ is that the form of the secrets is extended from $|x|$- and $|y|$-bit vectors x, y into $|x| \cdot r$ and $|x| \cdot r$-binary matrices X, Y. Random HB# can be considered as r rounds of HB$^+$, where each column of X and Y represents a different secret $x^{(i)}$ and $y^{(i)}$, respectively. Because of the fact that the whole protocol runs only for one round, it requires only one pair of challenge $a \in \{0,1\}^{|x|}$ and blinding factor $b \in \{0,1\}^{|y|}$. After receiving the challenge a and generating b the tag computes $z = a \cdot X \oplus b \cdot Y \oplus \nu \in \{0,1\}^r$ and sends the resulting vector to the reader for verification.

Storing secrets of this size $|x| \cdot r$ and $|y| \cdot r$ (which would equal to hundred thousands of bits for the smallest reasonable values for $|x|, |y|$) is infeasible in RFID context. Therefore, in the same paper, a modification of Random HB# was proposed, where the secrets are in the form of Toeplitz matrices, which can be stored in $|x| + r - 1$ and $|y| + r - 1$ memory gates. This variant is called HB#.

Parameter choices: We use the parameters suggested by the authors [GRS08b], which are in fact equivalent to the ones in Table 3.3, with the difference that r represents the number of rows in the matrices X, Y and $|x|, |y|$ represent the number of elements in each row.

Costs evaluation formulas: For Random HB#

$$\begin{aligned} KC &= |x| \cdot r + |y| \cdot r, \\ NR_n &= -\log_2(\eta) \cdot r, \\ NR_b &= |y|, \\ CC &= |x| + |y| + r \end{aligned}$$

and for HB#

$$\begin{aligned} KC &= |x| + |y| + 2 \cdot r - 2, \\ NR_n &= -\log_2(\eta) \cdot r, \\ NR_b &= |y|, \\ CC &= |x| + |y| + r \end{aligned}$$

Implementation issues: Storing $151,552$ (for the smallest value of $r = 256$) bits of the shared secrets in Random HB$^{\#}$ is clearly infeasible. However, if we assume that a solution to this problem was found, it is still necessary to implement the mechanism of selecting the current key bits which are to be multiplied by corresponding values of a and b. Such enormous key size results in a great amount of multiplexers needed for this purpose. We tried to implement the protocol under the smallest values of chosen parameters. However, the compiler was not able to synthesize the design due to the lack of memory. In order to demonstrate that the area size will be definitely much larger than the maximum allowed $2,000$ GEs, we implemented the simplified version of Random HB$^{\#}$, where each key matrix contains only $r = 4$ rows, meaning that the key size is only $2,368$ bits. The area size required for Random HB$^{\#}$ with these parameters is $3,648$ GEs when $d = 1$ and $3,769$ GEs when $d = 8$. Assuming that the multiplexers are evenly distributed between all raws in the key matrix, one can expect the linear increase of the area size with the increment of r, hence, for $r = 256$ the area size will be more than $200,000$ GEs. Based on this result we think that it doesn't make sense to implement the protocols, where the key size is of the order of 10^5 bits, as it is clear that the area size will exceed the limit of $2,000$ GEs by magnitudes.

The protocol HB$^{\#}$ was implemented under various parameter choices. The results are summarized in the Table 3.20. Compared to Random HB$^{\#}$ in HB$^{\#}$ the keys are much smaller, and therefore, less multiplexors are required. However, their number is still large an none of the implementation choices allow for an area size below 2,000 GEs.

Evaluation and implementation results:

Table 3.18: Evaluation results for Random HB$^{\#}$

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 1164$	689088	2328	512	1756	$NR_b + NR_n > 128$. $KC > 2048$
$\eta = 0.125, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 441$	261072	1323	512	1033	
$\eta = 0.125, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 256$	151552	770*	512	848	

* The average number of random bits is given.

Table 3.19: Evaluation results for HB$^{\#}$

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 1164$	2918	2328	512	1756	$NR_b + NR_n > 128$. MITM attacks [OOV08].
$\eta = 0.125, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 441$	1472	1323	512	1033	
$\eta = 0.125, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 256$	1102	770*	512	848	

* The average number of random bits is given.

Table 3.20: Implementation results for HB#

	$d = 1$		$d = 8$	
	AS	$Cycles$	AS	$Cycles$
$r = 256$	3374	151810	3412	19202
$r = 441$	3877	261956	3501	33077
$r = 1164$	9836	691418	5129	87302

Suitability and security: The protocol Random HB# cannot be used in ultra constraint tags due to the extremely high key-storage complexity, which also leads to the enormous area size. The version HB# shows much better efficiency. However, the area size exceeds the maximum possible value of 2,000 GE, implemented under all sets of parameters. Moreover, the number of random bits generated on the tag side, being much more realistic compared to the previously discussed protocols, still significantly exceeds the derived limitations. A MITM attack against these protocols was presented in [OOV08].

3.2.5.7 HB-MAC

Description: An improvement of HB# and Random HB# was suggested by Rizomiliotis in 2009 [Riz09]. The first difference compared to Random HB# is that in this protocol the identification and the authentication procedures are separated. The second improvement is the idea to use a message authentication code (MAC) to guarantee the integrity of the exchanged data, and thus to prevent active attacks. The protocol requires only one shared secret matrix M, and the authors claim that, the number of bits required to store it is smaller than in Random HB# by approximately a factor of 4, providing 80-bits security.

Parameter choices: We use the parameters suggested by the authors [GRS08b]: $\eta = 0.25, |k| = 160, r = 1,164$ and $\eta = 0.125, k = 160, r = 441$.

Costs evaluation formulas:

$$\begin{aligned} KC &= |k| \cdot r, \\ NR_n &= -\log_2(\eta) \cdot r \cdot 2, \\ NR_b &= |k|, \\ CC &= 3 \cdot |k| + 2 \cdot r \end{aligned}$$

Implementation issues: Although no exact specifications for the MAC has been provided, the authors mention a MAC scheme based on using hash functions and cites the paper [Kra94]. The hash function described in this paper is the same as used in Trusted hB. Therefore one can expect a similar increase in time effort and area increase, if the same hash function is used. Moreover, even if a different very lightweight and efficient hash function is chosen, the shared secret size is still extremely large. The required number of multiplexers will result to the area size which exceeds available 2,000 GEs by magnitudes (See discussion on the implementation issues of Random HB$^{\#}$).

Table 3.21: Evaluation results for HB-MAC

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, \lvert k \rvert = 160, r = 1164$	186240	4656	160	2808	$KC > 2048$, $NR_b + NR_n > 128$.
$\eta = 0.125, \lvert k \rvert = 160, r = 441$	70560	2646	160	1362	MITM attacks [Riz09].

Suitability and security: In fact the situation with this protocol is similar to Random HB$^{\#}$. The storage complexity and the required number of randomly generated bits are infeasible on lightweight RFID hardware. Moreover, the large key size results in the situation when the area size is also out of the bounds for ultra constraint tags. According to [Riz09], the protocol is insecure against the MITM attack explained in [OOV08].

3.2.5.8 HB^N

Description: A bilinear version of the HB protocol was proposed by Bosley et al. in [BHN11]. In the HB^N proposal, the shared secret is a square matrix X of the size $n \cdot n$. In this protocol it is not important, when the tag sends the randomly generated vector $b^{(i)} \in \{0,1\}^n$ to the reader. It can be done either before it receives the challenge $a^{(i)}$ from the reader (as it is the case for HB$^+$) or it can send $b^{(i)}$ together with the value ω_i, which is computed using the noisy bilinear function

$$\omega_i = {a^{(i)}}^\top X b^{(i)} + \nu_i$$

The protocol has been proven to be secure against MITM attacks [BHN11].

Parameter choices: The protocol security relies on the Learning Subspaces with Noise (LSN) problem [BHN11]. The authors claim that "it is possible to show that LSN for an n-bit secret is at least as hard as LPN with the secret of length $n-1$". Hence, we chose $n = 513$.

The authors suggest a different probabilistic verification scheme, where the reader adds some noise mirroring the noise from the tag. It leads to different formulas for the completeness error P_{FA} and the soundness error P_{FR}. According to [BHN11] the bounds for these values are the same:

$$P_{FA} \leq 2^{-r\left(\frac{1-4\eta+4\eta^2}{1+4\eta-4\eta^2}\right)},$$

$$P_{FR} \leq 2^{-r\left(\frac{1-4\eta+4\eta^2}{1+4\eta-4\eta^2}\right)}$$

Therefore, in order to achieve $P_{FR} \leq 2^{80}$ for $\eta = 0.125$, at least $r = 522$ rounds are required, and for $\eta = 0.25$, the minimal number of rounds r is $3,921$.

Costs evaluation formulas:

$$\begin{aligned} KC &= n^2, \\ NR_n &= -\log_2(\eta) \cdot r, \\ NR_b &= n \cdot r, \\ CC &= (2 \cdot n + 1)r \end{aligned}$$

Implementation issues: Similar to the previous two examples, namely Random HB# and HB-MAC, because of the extremely large key sizes it is clear that no of the implementation choices will result to the designs where the area is below 2,000 GEs.

Evaluation results:

Table 3.22: Evaluation results for HB^N

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, n = 513, r = 3921$	263169	7842	2011473	4026867	$KC > 2048$, $CC > 30000$.
$\eta = 0.125, n = 513, r = 522$	263169	1566	267786	536094	$NR_b \gg 128$

Suitability and security: In fact this is the most inefficient protocol among the considered ones. Due to the huge key size and large number of rounds all the costs summarized in the Table 3.2 are exceeded, in most cases by magnitudes.

3.2.5.9 GHB#

Description: The protocol called GHB# [RG12] was designed by Rizomiliotis et al. as an improvement of Random HB# [GRS08b]. The authors prove it to be secure against MITM attacks. Note that although the authors always state that the improvement is done with respect to HB#, all characterizations are given for Random HB#. In fact, nothing is said about whether GHB# remains secure against MITM attacks if, analogously to the approach followed for HB#, Toeplitz matrices are used for reducing the size of the secrets. The difference to Random HB# is how the value z is computed. It uses a non-linear function Φ which belongs to the class of Gold functions [Gol68]:

$$z = \Phi(a \cdot X) \oplus \Phi(b \cdot Y) \oplus \nu.$$

Parameter choices: The parameters suggested by the authors [GRS08b] are equivalent to the ones used in Random HB$^{\#}$ proposal and hence are equal to the ones presented in Table 3.3.

Costs evaluation formulas: for Random HB$^{\#}$:

$$\begin{aligned} KC &= |x| \cdot r + |y| \cdot r, \\ NR_n &= -\log_2(\eta) \cdot r, \\ NR_b &= |y|, \\ CC &= |x| + |y| + r \end{aligned}$$

Implementation issues: As it was already mentioned, the sizes of the secrets, suggested by the developers of the protocol are equivalent to the Random HB$^{\#}$. For the same reasons the area size required to implement this protocol is much larger than 2,000 GEs.

Evaluation results:

Table 3.23: Evaluation results for GHB

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, \|x\| = 80, \|y\| = 512, r = 1164$	689088	2328	512	1756	$KC > 2048$, $NR_b + NR_n > 128$.
$\eta = 0.125, \|x\| = 80, \|y\| = 512, r = 441$	261072	1323	512	1033	
$\eta = 0.125, \|x\| = 80, \|y\| = 512, r = 256$	151552	770*	512	848	

* The average number of random bits is given.

Suitability and security: With this protocol we face the same problems as with Random HB$^{\#}$: the key storage complexity, required amount of bits randomly generated by tag, and the area size do not allow for this protocol to be implemented in the low cost tags.

3.2.5.10 HB^b

Description: Another attempt to make Random HB$^{\#}$ resistant against the MITM attack presented in [OOV08] was made by the HB^b proposal [SKII12]. The following improvement was proposed. After the exchange of the blinding factor b and the challenge a between the reader and the tag is over, the tag generates a random bit c. Then it is suggested that if $c = 0$ then the tag chooses ν such that the number of ones in it is equal to $t = u \cdot r$ otherwise ν is chosen to have $t + 1$ ones. The further behaviour of the tag is the same as in Random HB$^{\#}$. However, the authors do not provide any considerations of how the mechanism of choosing the vector with the fixed number of ones can be realized in hardware. Repeating the generation process before the required condition is true may take unpredictable number of iterations, which would make the protocol to be unreliable. However, if such a mechanism exists, the other costs remain equal to the ones for Random HB$^{\#}$.

While it is unclear how many bits, i.e., the value of NR_n, are required for generating the noise, the other costs can be computed exactly as for Random HB$^{\#}$:

$$KC = |x| \cdot r + |y| \cdot r, \;\; NR_b = |y|, \;\; CC = |x| + |y| + r$$

Implementation issues: For the same reasons as in the previous several protocols the area size is extremely huge. Furthermore, the random number generation mechanism is unclear. Thus the protocol was not implemented.

Evaluation results:

Table 3.24: Evaluation results for HB^b

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 1164$	689088	Unclear	512	1756	$KC > 2048$. Noise
$\eta = 0.125, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 441$	261072	Unclear	512	1033	generation
$\eta = 0.125, \lvert x\rvert = 80, \lvert y\rvert = 512, r = 256$	151552	Unclear	512	848	is unclear.

Suitability and security: All the costs except the required number of random bits on tag side are the same as in Random HB$^{\#}$, which indicates that the protocol HB^b is infeasible on low cost RFID tags due to the key storage complexity and the area size.

3.2.5.11 NL-HB

Description: In 2011 a non-linear version of the HB protocol was suggested called NL-HB [MTSV10]. The idea is to use a non-linear function for providing a better resistance against passive attacks instead of if a linear function is used. According to the authors this allows to reduce the key-size by a factor of 4 while preserving the same level of security. However, shortly after it was shown that this attempt fails [Aby10], and that the suggested configuration doesn't provide the claimed level of security. To the best of our knowledge the only reasonable choice of parameters (with respect to the security) is the same as for HB.

Costs evaluation formulas: (equivalent to the ones valid for HB:)

$$KC = |x|, \;\; NR_n = -\log_2(\eta) \cdot r, \;\; NR_b = 0, \;\; CC = (|x| + 1) \cdot r$$

Evaluation results:

Table 3.25: Evaluation results for NL-HB

Parameters	KC	NR_n	NR_b	CC	Suitability
$\eta = 0.25, \lvert k \rvert = 512, r = 1164$	512	2328	-	597132	Similar to HB, the conditions $CC > 30000$, $NR_b + NR_n > 128$.
$\eta = 0.125, \lvert k \rvert = 512, r = 441$	512	1323	-	226233	
$\eta = 0.125, \lvert k \rvert = 512, r = 256$	512	770*	-	131328	

* The average number of random bits is given.

Suitability and security NL-HB has the same communication complexity and requires the same number of random bits as HB. This already points that it is infeasible on lightweight hardware. Furthermore, the protocol implementation would clearly require more area due to a more complicated function involved compared to HB, but would not provide remarkable advantage in security [Aby10].

3.2.5.12 Tree-HB

Another improvement of the HB like protocols was suggested by Halevi in [HSH11]. The variants are called Tree-HB$^+$, Tree-HB$^\#$ etc. The aim of the improvements is to ensure privacy of the tag, meaning that a tag remains anonymous to an attacker during the authentication process. To this end, this protocol includes a tree-based scheme that can be used by the reader to identify the tag without receiving the tag ID from it.

Suitability and security: It was shown in [AMM10] that the developers of Tree-HB didn't reach their goal, and the level of privacy is much weaker than they expected. Therefore, from our point of view, it does not make sense to consider it.

3.2.5.13 PUF-HB

Description: The protocol PUF-HB [HS08] is based on the concept of Physically Unclonable Functions (PUFs), where the same logical circuits produce different outputs depending on the physical properties of the exact device. While this is a promising approach, nothing can be said about its suitability without concretely specifying the deployed PUF type. Given the fact that the development and analysis of PUFs ongoing research, the suitability of this approach is still an open question.

3.2.5.14 AUTH, MAC$_1$, MAC$_2$

Kiltz et al. [KPC$^+$11] introduced a new two round authentication protocol called AUTH which, based on an LPN variant called subset LPN, provably provides active (but not MITM) security. In addition, two actually MITM-secure protocols were suggested.

Table 3.26: Complexities of the 2-round authentication protocols suggested in [KPC+11], with $l = 500$ (key size), $r = 250$ (number of rounds) and $\lambda = 80$ (security parameter).

Construction	Communication	Computation	Key size
AUTH	$l \cdot r \cdot 2.1/c$	$\Theta(l \cdot r)$	$l \cdot 4.2 \cdot c$
MAC_1	$l \cdot r \cdot 2.1/c$	$\Theta(l \cdot r) + \text{PIP}$	$l \cdot 12.6 \cdot c$
MAC_2	$l \cdot r \cdot 1.1/c$	$\Theta(l \cdot r) + \text{PIP}$	$l \cdot \lambda \cdot c$

Suitability and Security: The authors of [KPC+11] provided a precise assessment of complexities, e.g., for communication, computation, and key size, based on theoretical considerations. However, when compared to the hardware constraints of low-cost RFID tags as justified in Subsection 3.2.2, the respective numbers, summarized in Table 3.26, immediately show that these protocols are not suited for such devices. Again, as in subSubsubsection 3.2.4.3, l denotes the length of a respective LPN secret and r denotes the number of protocol rounds. The trade-off parameter c, $1 \leq c \leq r$, between key size and communication complexity is due to Gilbert et. al. [GRS08b] and λ is referred to as a "security parameter". Please note in particular that the term *PIP* in Table 3.26 subsumes the additional computational complexity of evaluating a certain pairwise independent permutation, which, according to the authors, takes $\Theta(m^2)$ time, where $m \approx 1,200$ for MAC_1 and $m \approx 600$ for MAC_2. Clearly, the resulting numbers of additionally required clock cycles are well beyond the limits of what we justified in Subsection 3.2.2 as feasible. In addition, for AUTH as well as for MAC_1 and MAC_2, any choice of c in Table 3.26 will either result in a key size (cf. Subsubsection 3.2.4.3) or in a communication complexity (cf. Subsection 3.2.2) definitely not feasible in the context of low-cost RFID tags.

3.2.5.15 Lapin

For the sake of completeness, let us finally mention Lapin [HKL+12], which is an authentication protocol based on the suggestions by Kiltz et. al. which we just discussed, and whose communication complexity (given as "1,300 bits" in [HKL+12]) is actually feasible for tags in the \$0.05 to \$0.10 range.

Suitability and security: The authors themselves state that they are "targeting lightweight tags that are equipped with (small) CPUs" as compared to "ultra constrained tokens (such as RFIDs in the price range of few cents targeting the EPC market" Moreover, the protocol was strongly criticized in [BL13], where the authors come to the conclusion that Lapin is even less efficient than AES providing less level of security. An FPGA implementation of Lapin was made in [GLS14]. Note, that this implementation requires 36 kb of buffer random access memory (BRAM), which makes it clearly infeasible when transfered to low-cost ASICs. Taking into account all these arguments, we will not discuss this variant in further detail.

3.2.6 Summarizing Results

Table 3.27 summarizes the results of evaluation of all the discussed HB-type protocols. We shortly explain the content of the table:

- For each protocol, we give in the second column (labeled "parameters") the parameter choices our evaluation is based on. These choices have been mostly derived from literature or, whenever necessary, been calculated on our own. In the latter case, the parameters have always been chosen in favour of the protocol. A detailed explanation of how the parameters have been derived is given in Subsubsection 3.2.4.2 for the protocols HB and HB$^+$ and in Subsection 3.2.5 for the remaining considered protocols.
- In columns 3–6, four[6] different cost factors are displayed: key storage complexity, the numbers of random bits required for generating the noise and blinding factors, respectively, and the total communication complexity (see also Table 3.6). Here too, we will later explain in detail how these have been computed.
- The determination of the cost factors now allows to verify whether the conditions explained in Subsection 3.2.2 are fulfilled. If some of these are violated for a certain protocol, this is explicitly stated in the last column. As the bounds given in Subsection 3.2.2 are certainly not tight, we indicate here a violation only if the induced costs would be way above these bounds (often by magnitudes). In these cases, we think that it is very improbable that small changes or the application of implementation tricks would be sufficient to make these protocols suitable. In addition, if man In the middle (MITM) attacks are known against the respective protocol, a reference is given as well.

The conclusion one can draw from these results is that each of the considered protocols would induce costs that are significantly outside of the derived bounds. Furthermore, most of the protocols are insecure against MITM attacks. Although one may debate whether MITM attacks are actually relevant in low-cost use cases, note that there are straightforward authentication schemes on the basis of prevalent lightweight ciphers (cf.Subsection 3.2.3) which are perfectly feasible and do not only provide active but also MITM security.[7] Summing up, it remains an open question to design a protocol based

[6]Please note, that in order to keep the presentation simple we omit from the table the costs (area size and required number of clock-cycles) which were evaluated using implementations of the protocols.

[7]The popular argument that, unlike for cipher-based schemes, active security can actually be "proved" for HB-type authentication protocols is only convincing to a limited extent as this "proof" in fact relies on the *assumed* hardness of the LPN problem.

on LPN (or a related problem) which is secure against MITM attacks and, at the same time, complies to the hardware constraints justified in Subsection 3.2.2.

Table 3.27: Evaluation results for the considered HB-type protocols.

Protocol	Parameters	KC	NR_n	NR_b	CC	Suitability and Security
HB	$\eta = 0.25, k = 512, r = 1164$	512	2328	0	597132	$CC \geq 30000$, $NR_b + NR_n \geq 128$
	$\eta = 0.125, \lvert k \rvert = 512, r = 441$	512	1323	0	226233	for all sets of parameters.
	$\eta = 0.125, \lvert k \rvert = 512, r = 256$	512	770*	0	131328	Active attacks.
HB^+	$\eta = 0.25, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 1164$	592	2328	59568	690252	$CC \geq 30000$, $NR_b + NR_n \geq 128$.
	$\eta = 0.125, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 441$	592	1323	225792	261513	MITM attacks.
	$\eta = 0.125, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 256$	592	770*	131072	151808	
HB^{++}	$\eta = 0.25, r = 731$	768	2924	58560	118582	$CC \geq 30000$, $NR_b + NR_n \geq 128$.
	$\eta = 0.125, r = 282$	768	1692	22640	45844	MITM attacks.
HB-MP	$n = 1, \lvert k \rvert = 513, m = 512, r = 1164$	1026	0	893952*	1191936	$CC \geq 30000$, $NR_b + NR_n \geq 128$.
	$n = 2, \lvert k \rvert = 513, m = 512, r = 441$	1026	0	395136*	451584	Passive attacks.
$HB-MP^+$	$n = 2, \lvert k \rvert = 512, m = 224, r = 1164$	1024	0	391104*	521472	$CC \geq 30000$, $NR_b + NR_n \geq 128$.
HB^*	$\eta = 0.5, \lvert k \rvert = 256, r = 80$	1024	80	20480	41200	$CC \geq 30000$, $NR_b + NR_n \geq 128$.
HB^{*1}	$\eta = 0.25, \lvert k \rvert = 512, r = 1164$	1536	3492	595968	598296	$CC \geq 30000$, $NR_b + NR_n \geq 128$.
	$\eta = 0.125, \lvert k \rvert = 512, r = 441$	1536	1764	225792	226674	MITM attacks [PT07, GRS08a].
Trusted HB	$\eta = 0.25, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 1164$	693	2328	595968	690353	$CC \geq 30000$, $NR_b + NR_n \geq 128$. $\approx 7 \cdot 10^5$ clock-cycles (max. available $1,5 \cdot 10^5$). MITM attacks .
RND-HB#	$\eta = 0.25, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 1164$	689088	2328	512	1756	$NR_b + NR_n \geq 128$.
	$\eta = 0.125, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 441$	261072	1323	512	1033	MITM attacks.
	$\eta = 0.125, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 256$	151552	770*	512	848	
HB#	$\eta = 0.25, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 1164$	2918	2328	512	1756	$NR_b + NR_n \geq 128$.
	$\eta = 0.125, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 441$	1472	1323	512	1033	MITM attacks.
	$\eta = 0.125, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 256$	1102	770*	512	848	
HB-MAC	$\eta = 0.25, \lvert k \rvert = 160, r = 1164$	186240	4656	160	2808	$KC \geq 2048$, $NR_b + NR_n \geq 128$.
	$\eta = 0.125, \lvert k \rvert = 160, r = 441$	70560	2646	160	1362	MITM attacks [Riz09].
GHB#	$\eta = 0.25, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 1164$	689088	2328	512	1756	$KC \geq 2048$, $NR_b + NR_n \geq 128$.
	$\eta = 0.125, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 441$	261072	1323	512	1033	
	$\eta = 0.125, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 256$	151552	770*	512	848	
HB^N	$\eta = 0.25, n = 513, r = 3921$	263169	7842	2011473	4026867	$KC \geq 2048$, $NR_b + NR_n \geq 128$,
	$\eta = 0.125, n = 513, r = 522$	263169	1566	267786	536094	$CC \geq 30000$.
HB^b	$\eta = 0.25, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 1164$	689088	Unclear	512	1756	$KC \geq 2048$. Noise generation
	$\eta = 0.125, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 441$	261072	Unclear	512	1033	mechanism is unclear.
	$\eta = 0.125, \lvert x \rvert = 80, \lvert y \rvert = 512, r = 256$	151552	Unclear	512	848	
NL-HB	$\eta = 0.25, \lvert k \rvert = 512, r = 1164$	512	2328	-	597132	Similar to HB, $CC \geq 30000$,
	$\eta = 0.125, \lvert k \rvert = 512, r = 441$	512	1323	-	226233	$NR_b + NR_n \geq 128$.
	$\eta = 0.125, \lvert k \rvert = 512, r = 256$	512	770*	-	131328	Active attacks.
AUTH						Depending on trade-off parameter c, either CC, KC, or both not feasible (cf. [KPC+11] and [GRS08b]).
MAC_1						Same as AUTH + $CompC$ of $\Theta\left(m^2\right)$, $m = 600$, imposed by pairwise independent permutation (cf. [KPC+11]).
MAC_2						Same as AUTH + $CompC$ of $\Theta\left(m^2\right)$, $m = 1200$, imposed by pairwise independent permutation (cf. [KPC+11]).

KC - Key storage complexity;
NR_n - Number of uniformly distributed random bits required for noise
NR_b - Number of uniformly distributed random bits required for blinding factors
CC - The total communication complexity
$CompC$ - The total computational complexity
$*$ - Is used when the average number of random bits is given

3.3 On the Performance of Ciphers which Continuously Access the Non-volatile Key

3.3.1 Motivation

The community has developed various techniques for designing lightweight ciphers. One approach which is more and more considered is the idea of using the cipher key that is stored on the device not only for initialization of the registers but also to involve it continuously in the encryption/decryption process. Examples include the block ciphers Midori [BBI+15], KTANTAN [CDK09], PRINTcipher [KLPR10], and LED [GPPR11] as well as the stream ciphers A2U2 [DRL11], Sprout and Plantlet. The later two are discussed in Section 5.2. Moreover, this approach is used in practice: the cipher and the authentication protocol of the Megamos Crypto transponder [VGE15] also utilize this principle.

To understand the basic idea, recall that a device incorporating a cipher usually needs to access a key. This key, however, is not dynamically received from an an external entity, but it is already stored on the device in non-volatile memory. We refer to this key as being a *non-volatile key*. Here, the term non-volatile stresses the property that this key cannot be changed by the cipher itself. That is, in most cases the work flow is as follows. After the encryption or decryption process is started, the key is retrieved from the location where it is stored and loaded into registers, i.e, from non-volatile memory NVM into volatile memory VM. For block ciphers, these may be separate registers intended for the key schedule, while for stream ciphers, this is usually the internal state. Adopting the terminology from above, one could say that the value in VM now represents a *volatile value* which changes during the encryption/decryption process whereas the value stored in NVM, the non-volatile key, remains fixed.

It holds for most designs that after the key has been loaded from NVM into VM, the NVM is usually not involved anymore (unless the key schedule or the initialization process needs to be restarted). However, recently several ciphers [BBI+15, CDK09, KLPR10, GPPR11, DRL11, AM15, MAM17], were proposed which require access to the non-volatile key, not only for initialization but also during encryption/decryption process.

One of the main motivations for this approach is that using the values stored in the NVM may allow to reduce the VM, and hence, result in an overall reduction of the area size since NVM is needed for storing the key on the device anyway. Hence, if re-using the key stored in NVM allows to reduce the amount of volatile memory, this would provide a direct benefit. In other words, one could say that non-volatile memory that needs to be present anyhow is “traded” for volatile memory so that in total, the area consumption is reduced.

In this context, we want to point out that a similar approach may be considered for block ciphers to reduce the logic. Block ciphers usually deploy a key schedule to derive several round keys from the secret key. Instead of loading the key into VM and to run the key schedule, one may consider to precompute these round keys once and to store them in NVM, effectively trading it in for volatile memory and logic. While this is certainly an interesting approach with its own benefits, it is not completely in scope of our work as we are mainly looking at ciphers which re-use NVM that needs to be present anyway, while storing the round keys would increase the amount of required non-volatile memory. Furthermore, any potential measures applied to NVM to protect it against physical attacks would also require appropriate extensions to remain effective. Finally, this approach makes only sense for block ciphers with a fixed and relatively small number of round keys and is completely impractical for stream ciphers. This is contrary to the idea of re-accessing the same key in NVM, which has been considered for block and stream ciphers and hence is more general. Nonetheless, the idea of storing precomputed keys is another reason why cipher designs where NVM is continuously accessed can be of interest.

Apart from reducing the demand for resources such as area size, power and energy consumption, the continuous involvement of the key may also result in a higher security level. For example, in Section 4.3 we show that this may increase the resistance of stream ciphers against time-memory-data-tradeoff attacks.

Summing up, the idea of continuously using the key stored in NVM seems to be a promising design approach for several reasons, at least in *theory*. In *practice* however, only little is known how this impacts the practicability of the cipher. In fact, most papers treat NVM as “free” memory from which can be read in “zero” time or at least with the same speed as it is possible to access VM. This is certainly not given in practice and raises the question if and to what extent such designs really help to reduce the consumption of resources. The main problem here is that hardly any information is publicly available. That is, cryptographers without a strong engineering background face the problem that they cannot evaluate whether such designs are indeed reasonable and promising or rather represent “cheating”.

In [MAM17] we revisited the design of ciphers that continuously involve the non-volatile key during the encryption process and discuss these results in the current section. More precisely, our contributions are as follows:

Discussion on Non-volatile Memory: We discuss several use cases and explain what types of NVM are practically relevant for the considered scenario. Here, we have to distinguish between the case that the key is set once and forever or that it is rewritable. For the first case, certain types of NVM can be used, e.g., Mask Read-Only Memory (MROM) and Programmable Read-Only Memory (PROM), where accessing the keybits induces no overhead. That is, in such cases very efficient ciphers are possible but key management is very limited. In the second case, i.e., cases which require the key being rewritable, certain timing limitations for accessing the NVM need to be respected.

Motivated by commercially employed schemes like KeeLoq [Mic01], Hitag (S) [NXP14], or Mifare [NXP11] we then put our focus on the case that EEPROM is used for storing the rewritable key and explain practical implications when aiming for low-area ciphers using EEPROM.

Re-evaluation of Ciphers: Based on our findings, we evaluate existing lightweight ciphers within this scenario with respect to the practical consequences. In case that the NVM needs to be reprogrammable, some ciphers are better suited for this approach than others, which depends on how the key stored in NVM needs to be accessed. We examine for several existing ciphers that continuously access the key stored in non-volatile memory how the use of, on the one hand, various standard serial EEPROM modules using the most common interfaces (see Table 3.29), and on the other hand, customized ASIC EEPROMs (see Table 3.30), would impact performance.

In Subsection 3.3.2, we discuss different technical realizations for NVM and analyze the effort to read from them. Subsection 3.3.3 analyzes the impact on different ciphers which use the non-volatile key "trick" if the cipher would be implemented with various types of NVM.

3.3.2 On Reading from Non-volatile Memory

In this section, required technical background pertaining to NVM is presented. First of all, common implementation approaches for NVM are described with a focus on programmability. Then, prominent serial interfaces suitable for accessing NVM are introduced as well as an effort estimation particularly for reading operations. This section also includes a brief summary of the different interfaces' required number of clock cycles for reading from NVM.

3.3.2.1 Approaches

Generally speaking, there are three different categories regarding NVM production approaches.

The first category comprises technologies where the memory content has to be programmed already by the manufacturer and cannot be changed afterwards. In case of lightweight devices this can be realized either by using MROM—where the information is added to so-called wafers—or by realizing lookup tables (using tie-high and tie-low cells) in standard ASIC library. Utilizing tie-high and tie-low cells has the benefit of not incurring any overhead for accessing the memory. The full key is instantly available and its usage comes at no performance overhead, while the area demand of such a logic is also relatively small.

The second category covers techniques where likewise the key can be set only once. The difference however lies in the time of programming, which in this case happens *after* manufacture. This PROM is typically produced with fuses and antifuses that are blown appropriately during the programming process.

The third category differs significantly from the former two in the sense that the key can be freely rewritten, allowing for a higher level of flexibility.

With respect to the effort of accessing the bits during the encryption/decryption process, the first two approaches would certainly be the preferred option, as each key bit can be directly accessed without any particular overhead.

However, the price one has to pay is the loss of flexibility. More precisely, the first approach means that the key has to be known already before the IC chip is produced. Moreover, this approach becomes cheap only if a large amount of devices share the same key, since the production of individual wafers, from which chips are produced, is rather expensive. In the works [KLPR10] and [DRL11] utilizing a technology called integrated circuit printing is suggested, claiming that for a printer there will be no costs for changing the circuit while producing each of the new devices, thus enabling individual keys. However, according to [KLPR10] the IC printing technology is not yet fully understood. In any case, if this approach is used it holds that once the key is set it can never be changed, which is true for the second approach as well. That is the process of setting the key cannot be reversed and hence has to be settled before a device leaves the factory which enormously complicates key management.

We conjecture that this is one of the main reasons why the third, yet more expensive approach is often used in practice. That is, the key is stored in some NVM which allows to change the key afterwards. Here, different techniques are imaginable. In case of lightweight devices, EEPROM seems to be the favored choice[8] (for more details, see e.g. [AHM14]). In fact, in many commercial products EEPROM is used for storing the secret key. Examples include MIFARE [NXP11], Hitag [NXP14], KeeLoq [Mic01], CryptoRF [Atm14] and Megamos Crypto transponder [VGE15]. We would like to stress that using EEPROM does not necessarily imply very high costs. For instance, the Hitag μ advanced transponder [NXP15] offers 1760 bit of EEPROM memory can be purchased for the price of a few cents per unit from a major retailer. In this section therefore, we focus on a scenario where the key is stored in EEPROM which is continuously accessed by the cipher during the encryption/decryption process and where the goal is to reduce the area size of the cipher without increasing the size of the EEPROM. To this end, we investigate different interfaces for accessing EEPROM in the following and discuss how this impacts the performance of the cipher. As another type of EEPROM-like memory, flash memory is presented in form of a short overview and comparison with a focus on its suitability to lightweight applications in contrast to EEPROM.

3.3.2.1.1 Accessing EEPROM.

In principle, two different options exist for using EEPROM in constrained devices:

[8]We consider the devices with the lowest price available on the market and assume that the more expensive technologies like FRAM are out of the scope of this work.

(i) commercially available external EEPROMs which communicate with the device via certain standard interfaces and (ii) NVM cells, e.g. EEPROM, that are directly integrated into an ASIC by design. Although, the first approach may be much cheaper, it is susceptible to side channel attacks. That is, the data lines between the EEPROM module and the device may be easily eavesdropped on. Nevertheless, this approach may be realistic for a scenario, in which physical access to the device is considered infeasible for an attacker, e.g., medical implants. Since our intention is not to promote commercial EEPROMs, but to provide a holistic view on available memory technologies, excluding commercial EEPROMs from the discussion would render it incomplete. Conversely, integrating EEPROM cells into the ASIC is more expensive, but it also offers a higher level of security and more flexibility pertaining to the interfaces. Although most of the commercially available constrained devices use the second approach, we discuss both alternatives for the sake of completeness.

Standard EEPROMs. In the following, we focus on commercially available EEPROMs and denote these as standard EEPROMs. Standard EEPROMs can be accessed using serial or parallel interfaces. EEPROM modules with serial interfaces have a smaller footprint and lower power consumption compared to EEPROM with parallel interfaces of an equivalent density [PSS+08]. As we examine the use of EEPROM in the context of lightweight ciphers, we consider parallel interfaces, such as those used e.g. in the ARM AMBA protocol, out of the scope of this work.

With respect to commercial EEPROMs with serial interfaces, we examined the data sheets of several low-budget EEPROM devices produced by different manufacturers such as NXP Semiconductors, Microchip Technology Inc, Atmel Corporation, On Semiconductor, Renesas Electronics Corporation. According to these, the following serial interfaces are the most commonly used:

1. I^{2}C [On 14]: If the I^{2}C interface is used, the memory is usually organized in *memory words* with a size of 8 bits. For accessing bits from the EEPROM, the following types of readings are relevant:

   Random (Selective) Reading: This type of reading allows to "jump" to a specific EEPROM address for extracting the values stored there. This allows for the highest level of flexibility, but also incurs additional costs. To perform this type of read operation, first the word address must be reset (which usually requires 19 clock cycles). Then, the control byte has to be sent to the EEPROM which in turn has to acknowledge its receipt. This takes 10 clock cycles. Only if this is accomplished, the EEPROM starts to send the data of the requested memory word (8 clock cycles) and waits for the acknowledgment bit (1 clock cycle) and the `STOP` signal (1 clock cycle). Therefore, reading a word of 8-bits from a freely chosen address requires 39 clock cycles in total.

Current Address Reading: The current address reading uses an internal address counter that is increased after each reading. To initiate the read operation, a control byte preceded by a `START` signal is sent to the EEPROM and gets acknowledged (10 clock cycles). Then, the current memory word is sent (8 clock cycles) and the address counter is increased. The read operation is terminated by a `NoACK` signal followed by a `STOP` signal (additional 2 clock cycles). In total, a current address read, i. e., reading 8 bits from the current memory address, requires 20 clock cycles.

Sequential Reading: Similar to current address reading, sequential read allows to read a memory word from the current address and increases the address afterwards. The main difference is that it keeps on doing so until instructed otherwise. That is sequential read outputs memory words from EEPROM one by one starting at the current address and is preceded by a random or a current address read. After each sent memory word, the EEPROM expects an acknowledgement bit. The read operation continues by increasing the address counter and sending the next memory word until the device receives a `NoACK` followed by a `STOP`. Therefore, reading one memory word sequentially requires an additional clock cycle for the acknowledgement, i. e., 9 clock cycles. Considering the condition that a preceding current address or random read must occur, reading n memory words requires $10 + 9n + 1$ or $29 + 9n + 1$ clock cycles, respectively.

2. Serial Peripheral Interface (SPI) [Mic14]: Usually, the memory is organized in 8-bit words. To read from EEPROM using SPI, the host sends a `READ` instruction (8 bits) followed by a memory address, which may consist of 8, 16, or even 24 bits, to the EEPROM device. After receiving the last address bit, the EEPROM responds by shifting out data on the serial output data pin. Sequentially stored data can be read out for as long as the clock signal is provided. After each provided memory word, the internal address pointer is automatically incremented to point to the next address. If the highest memory address is reached, the address counter "rolls over" to the lowest, and the read cycle can be continued indefinitely. Thus, using an address width of a, reading n memory words requires $8 + a + 8n$ clock cycles, e. g., $16 + 8n$ clock cycles are required in the $a = 8$ case.

3. Microwire [Atm15]: This interface is a predecessor of SPI and provides only a subset thereof. The EEPROM is organized in either 8-bit or 16-bit memory words. A `READ` instruction consists of a start bit, a 2-bit op-code, and 7 or 6 address bits for the requested 8-bit or 16-bit memory word, respectively. After receiving and decoding, the data is transferred from the memory to an output shift register. At first, a dummy 0 bit is sent, followed by the memory word, while each next bit is indicated by the rising edge of the clock signal. The device automatically

Table 3.28: Overview of required clock cycles for reading n consecutive memory words using different interfaces and read operations for 8-bit memory words and addresses. SPI and Microwire only offer selective reading. Sequential reading is therefore not *directly* available, indicated by 'n/a'.

Interface	Selective Reading	Sequential Reading
I^{2}C	$30 + 9n$	$11 + 9n$
SPI	$16 + 8n$	n/a
Microwire	$11 + 8n$	n/a
UNI/O	$50 + 10n$	$30 + 10n$

increments the internal address register and sends subsequent memory words until indicated to stop. However, the dummy 0 bit is only sent once, subsequent memory words are output as a continuous stream of data. If the highest memory address is reached, the address counter rolls over and points to the lowest memory address. Reading n words from memory results in $3 + a + 1 + 8n, a \in \{6, 7\}$ clock cycles, e. g., $11 + 8n$ for memory organized in 8 bits.

4. UNI/O [Mic11]: Memory accessible via this interface type is usually organized in 8-bit memory words. Before a read operation can be performed, a `Start Header` and a memory address of 8 bits each are transmitted, which are followed by an acknowledgement phase in which both the host and the EEPROM device send 1 bit each. Altogether, this setup phase consumes 20 clock cycles. Then, the actual read command comprising 8 bits is sent and acknowledged in the same way. Immediately, the requested memory address is transmitted. A memory address consists of two 8-bit parts, whereas an acknowledgement phase takes place after each part is sent, thus requiring 30 clock cycles for the command and memory address. Besides this selective read operation, a current read command can be used which relies on an internal address pointer that is automatically increased after each read operation. Using this command, the memory address is omitted and only 10 clock cycles for the command and subsequent acknowledge bits are required. Nonetheless, both read operations allow for continuously reading consecutive memory words. The internal address counter rolls over to the lowest memory address after the highest was reached. Each memory word is followed by 2 acknowledge bits, i. e., reading n consecutive bytes from memory requires $50 + 10n$ and $30 + 10n$ for selective and current read operations, respectively.

Although these techniques differ in several details, one can observe that two principal reading methods are possible: accessing the content by providing the address or reading the next word of the EEPROM based on an address counter that is automatically increased. Adopting the terminology of I^{2}C, we refer to these types as *selective reading* and *sequential reading*, respectively. That is, when we speak about sequential reading we will refer to this mode of operation and not necessarily to I^{2}C.

Table 3.28 shows the number of required clock cycles to read n consecutive 8-bit memory words assuming the underlying memory and addressing scheme are as well 8-bit based (or 7 bits as in the Microwire case). One clearly sees that selective reading, i.e., reading from a chosen address, incurs a higher overhead than just reading sequentially the words from the EEPROM. This shows that non-volatile key ciphers that sequentially read the key bits from the EEPROM better match this technology than ciphers that repeatedly have to read from different addresses.

EEPROMs Integrated into ASIC. When a designer develops an ASIC, he can integrate EEPROM cells directly so that he has not to rely on any standard interfaces. By doing so, he can achieve that practically almost no limitations on retrieving key bits from an integrated EEPROM apply. In such cases, any cipher design would be equally good with respect to the effort of accessing the key bits.

However, these benefits do not come for free. First of all, developing an ASIC tailored for certain use cases takes more effort than using standard building blocks. Moreover, it seems that certain ASIC EEPROM designs are favorable compared to others. More precisely, in existing work [BL08, CZDL13, NKTZ12, NXDH06, LCK10] that discusses the construction of ASIC EEPROM designs for restricted devices like passive RFID transponders, the access to the EEPROM is organized word-wise where typical memory word sizes are 8, 16, or 32 bits. According to [NKTZ12], the choice of word size is one of the most important factors with direct impact on the area size of the resulting EEPROM. Obviously, this choice also affects the effectiveness of the procedure that involves the key during encryption/decryption.

Although accessing EEPROM on a word-by-word basis is a common technique, designs utilizing bit-by-bit reading may provide a more conservative power consumption [CZDL13, NXDH06]. Such a customized bit-by-bit interface allows to retrieve any desired bit per clock-cycle. Thus, in addition to 8-, 16-, and 32-bit based parallel interfaces, we consider this type of bit-by-bit interface, as we suppose that it implies realistic limitations on the flexibility of reading from customized EEPROM and can be pertinent in practice.

3.3.2.1.2 Accessing Flash Memory

After investigating cheap candidate devices for lightweight ciphers, e.g., passive RFID tags, flash memory does not seem to be used commonly. However, since this technology is gaining in popularity and decreasing in cost, it may be used for these kinds of devices in the near future. Therefore, for the sake of completeness, we also shortly discuss flash memory.

Technically, flash memory is another type of EEPROM with higher storage density than 'traditional' EEPROM, sacrificing bit alterability for area [Cyp15], such that minimum erasable data units are organized in *blocks* of several kilobytes [KKN+02], e.g., 64, 128, or 256 kilobytes. Therefore, flash is typically used for applications with high memory demand, i.e., in the range of megabits to several gigabits. Also, different architectures (NOR or NAND), data transfer (serial or parallel), and approaches (external or directly integrated) are prevalent, leading to different characteristics and read protocols.

On the one hand, NOR flash memory offers random read access on a byte level making it a suitable EEPROM alternative [Mic]. On the other hand, NAND flash is page-oriented, whereas several pages, typically 512 [KKN+02], 2,048, or 4,096 bytes in size, form one block, and it requires a dedicated memory controller. Usually, NAND flash is used for mass storage, e.g., more than 64 megabits [Cyp15, Mac14].

Having compared several data sheets of available low-price external flash memory ICs, it appears that mostly similar serial interfaces are used for reading as in the case of EEPROM [Fre14, Ade16]. We note that external modules suffer from similar weaknesses as EEPROM, e.g., side channel attacks.

Since the structure of the memory cells is similar to EEPROM, integrating NOR flash into ASIC yields a read process organized on either a bit or word basis and therefore, similar limitations as for integrated EEPROM.

All in all, since these two memory technologies use similar read interfaces in cases where flash memory constitutes a potential replacement of EEPROM, we do *not* further consider flash as a substantially different additional technology with respect to the timing limitations of the ciphers and limit our focus to EEPROM.

3.3.3 Impact of Different EEPROM Types on Throughput of Existing Ciphers

In this section, we examine for several existing ciphers that continuously access the key stored in non-volatile memory how the use of standard serial EEPROM on the one hand and of customized ASIC EEPROM on the other hand would impact performance. This is motivated by the question on how different designs perform with respect to available standard building blocks. As the concrete throughput depends on many parameters unrelated to the deployed EEPROM, we do not state these values but compare them with the case that the same implementation is used but under the assumption that accessing the key incurs no overhead at all.

The considered ciphers are the block ciphers Midori128 and Midori64 [BBI+15], LED [GPPR11], KTANTAN [CDK09] and PRINTcipher [KLPR10] as well as the stream ciphers A2U2 [DRL11], Sprout and Plantlet[9].

[9]The designs of Sprout and Plantlet are discussed in Section 5.2

3.3.3.1 Key Selection Functions of Considered Ciphers

In our analysis we focused on the key selection functions, i.e., the part of the cipher that *continuously* accesses the key stored in non-volatile memory as we expect here the most significant impact. This means for example that we ignore the initialization process which is invoked much less frequently. Therefore, we provide only a description of the key selection function and refer to the respective papers for a full description.

Due to the fact that sequential reading allows for a higher throughput than selective reading, we opt for the first whenever possible when discussing standard interfaces. With respect to sequential reading, we distinguish between two cases that we call *wrap case* and *no-wrap case*, respectively. In the wrap case, it holds that if the last key bit of the EEPROM is read, the internal address counter automatically wraps around such that it points to the first bit of the key again. In contrast if the no-wrap case holds, the internal address counter does *not* automatically wrap around but has to be reset manually to the start address of the key. Furthermore, we make the assumption that at the beginning of the encryption/decryption process, the address already points to the beginning of the key.

Midori. Midori [BBI+15] is a cipher that targets low-energy applications and provides a block size of 64 or 128 bits, i. e., Midori-64 and Midori-128, with a key size of 128 bits for both variants. It does not employ any dedicated key schedule function due to energy consumption concerns. Similar to AES, it consists of a state of 64 bits (or 128 bit for Midori-128) which is transformed by 15 (or 19) rounds of substitution, permutation, and (round) key addition layers. Additionally, key whitening resulting in two extra key additions is performed before the first and after the last round. Note that any round key is obtained by XORing the supplied key with pre-defined round constants, in the 64-bit case however, the supplied 128-bit key is reduced to 64 bits by XORing its most significant and least significant half beforehand.

Thus, for encryption of one 64-bit (or 128-bit) block of data, $1,088$ (or $2,688$) key bits are required. In other words, 136 (or 336) 8-bit memory words need to be retrieved, which results in equally many EEPROM read accesses in the wrap case. In the no-wrap case, at least 9 (or 19) read operations would be *selective*, while all remaining ones could be performed sequentially. Considering round key precomputation, this approach is almost equivalent to the wrap case except for the obvious non-negligible increase in memory demand.[10]

[10]To the advantage of the cipher, we assume that the length of memory addresses covers exactly the NVM and that no additional effort incurs when addressing higher memory regions.

LED. LED [GPPR11] is a block cipher targeting a low hardware footprint with a block size of 64 bits, two *primary*[11] key sizes of 64 or 128 bits, respectively, and no dedicated key schedule. Similar to AES, the cipher consists of several rounds of substitution, permutation, and key addition layers but are applied to a state of 64-bit only. By design, the actual number of rounds depends on the key size. A peculiarity is that 4 rounds constitute one *step* while the key addition is only performed once *before* each step plus one additional key addition after the last step. For 64-bit keys, 32 rounds or 8 steps are performed, which amounts to a total of 9 key additions versus 48 rounds or 12 steps and 13 key additions in the 128-bit case.

For processing one 64-bit block of data, hence, 576 versus 832 key bits or 72 versus 104 8-bit memory words are required, respectively. Due to the absence of any key schedule, storing the key bits redundantly, i. e., in extended form, is essentially equivalent to the wrap case.[12] In the no-wrap case however, after reading all key bits, the memory address needs to point to the first memory word again; this happens 8 times in the 64-bit key case and 6 times in the 128-bit key case. Consequently, 64 versus 98 read operations may be performed sequentially.

KTANTAN. The KTANTAN family [CDK09] represents a set of block ciphers providing 3 different block sizes, i. e., $32, 48, 64$. Independent of the chosen block size, the family uses an 80-bit key and utilizes a key schedule that relies on the current internal state to choose 5 bits from this key. By design, the key is treated as 5 consecutive 16-bit words and the selected bits have the same relative position inside each 16-bit word. Finally, only 2 of these 5 bits are used, whereas the selection is also based on the current internal state. Occasionally, it may happen with a probability of $1/4$ that the same key bit is picked twice. We consider this to be the best case with respect to the throughput and hence assume in favor of the cipher that this always happens. As the selection of the key should not follow any easily predictable pattern, we furthermore assume that each time a key bit is needed, the device has to initiate a selective reading. For sure, one may increase the throughput by for example buffering the recently read key word and checking whether the same key word is needed again. We do not consider such improvements in our analysis here for two reasons. First, this would require additional logic and registers. Second, different selection patterns may result into different timing behavior and hence may allow for side-channel attacks. Thus, we leave it as an open question if and how the throughput of the KTANTAN family could be increased without consuming too much additional area and without compromising the security.

According to [CDK09], KTANTAN uses 254 rounds in which the key schedule is involved, that is, in each round one access to the key is necessary. In other words, for

[11] In fact, any key size from 64 bits up to 128 is imaginable as stated in [GPPR11] by extending keys of more than 64 bits to 128 bits.

[12] We note that the situation is different for any key sizes other than 64 or 128 bits due to the way such keys are handled during the process as of the updated version of the LED specification [GPPR12].

processing 1 block of input data, 254 EEPROM read operations and 254 clock cycles are required.

Of course if one aims to avoid the overhead incurred by selectively accessing the key bits, one could deploy the following approach. Recall that during each of the 254 rounds of KTANTAN, 2 key bits are required that are taken from the 80-bit key. Hence, instead of storing the 80-bit key in EEPROM and being forced to apply selective reading, one could instead store all *necessary* key bit values in the exact order as required by the cipher. That is, given that encryption involves accessing the key 254 times and to extract two bits each, this would result into storing a total of 508 precomputed key bit values. Obviously, this yields a storage overhead since additional 428 bits would be stored in EEPROM compared to storing the 80-bit key only. Any logic responsible for key bit selection can however be omitted. That is logic is traded for EEPROM area in this case. Assuming such a precomputation approach, KTANTAN requires 508 key bits retrieved from EEPROM, i. e., 63.5 sequential read operations for 8-bit memory words.[13] For the sake of comparison, the throughput for this approach is also given in Table 3.29.

PRINTcipher. The PRINTcipher [KLPR10] is a block cipher which is available with different block sizes. PRINTcipher-48 uses a block size of 48 bits with an 80-bit key, whereas PRINTcipher-96 uses a block size of 96 bits with a 160-bit key. The number of encryption rounds per input block is defined by the block size, e. g., PRINTcipher-48 performs 48 rounds, on which we focus in the following.

During encryption, the key is used in two different ways. In each round, the first 48 bits are directly used by XORing them with the current state. This state is then shuffled and combined with a round constant. After this, the state and the remaining 32 key bits are used in the key-dependent permutation layer which takes 3 bits from the state and 2 bits from the remaining 32 key bits to produce the input to an S-box. All these operations can be realized by reading the key bits sequentially from the non-volatile memory in each round.

Obviously, this design would fit best into the wrap case. In the no-wrap case, additional effort needs to be taken to restart reading the key bits from the beginning. Of course, this could be circumvented if the expanded key is stored in EEPROM (similarly as discussed for KTANTAN above). This would result into an increased throughput similar to the wrap case, but at the cost of requiring significantly more EEPROM storage (3, 840 bits instead of 80).

A2U2. The stream cipher A2U2 [DRL11] uses a 56-bit key. For each generated output bit, 5 consecutive key bits are used in the process. This implies that sometimes it is sufficient to retrieve one memory word only (if all five bits are within the same word), but sometimes two words are necessary (if some of the bits fall into the current memory

[13] Technically, 64 read operations would be performed.

word and the remaining ones in the subsequent word). This is independent of the underlying memory word size. As eight bits are a common size for memory words, we assume this value in the following. Thus, producing 8 output bits requires retrieving 40 key bits from memory which translates to 4 1-word and 4 2-word EEPROM read operations. Additionally, only 1 of 8 read cases may be performed as a current address read, in all other cases a previously requested memory address has to be read again which requires a selective read. On average, one has to read 1.5 memory words per output bit where a fraction of $\frac{1}{7}$ can be realized as current address reads.

Table 3.29 provides an overview of the achievable throughput for different interfaces considering both the wrap case and the no-wrap case with an underlying memory word size of 8. Additionally, if provided by the interface, we assume to utilize the current address read which is possible for generating 1 out of 8 output bits. Note that it may happen that the last and first memory word are required to produce an output bit which in the no-wrap case requires 2 selective read operations requesting the 2 memory words separately instead of only 1. However, this occurs 4 times per 56 generated output bits. Hence, there is only a slight difference noticeable between those cases.

Sprout and Plantlet. The stream ciphers Sprout and Plantlet use an 80-bit key and almost identical round key function. In a nutshell, for both ciphers, each of the key bits is accessed in a sequential fashion, i.e., starting with the first bit followed by the subsequent bit and so forth. Although in Sprout, this bit is not necessarily considered for further processing, it is reasonable to assume that 1 key bit is required per produced output bit in both designs. Considering a memory word size of 8 bits, 10 EEPROM read operations are required for producing 80 output bits.

This is particularly useful for the wrap case, as this implies that the read operation needs only be initiated once, while the *current* key bit is available immediately, e.g., to produce output of more than 80 bits.

In the no-wrap case on the other hand, after the 80th key bit is used, a selective read operation is needed to continue output production which then requires reading the first key bit again. This affects the achievable throughput and limits it to different values for the considered interfaces as presented in Table 3.29. Contrary to A2U2, there is an absolute difference noticeable between the wrap case and the no-wrap case.

3.3.3.2 Evaluation of the Ciphers' Throughput

The results of the analysis are summarized in Table 3.29. It displays the throughput of the respective cipher using the respective standard serial EEPROM in comparison to the same implementation where no effort is assumed for accessing the key which represents 100% throughput. For example, in case of Midori64 the throughput decreases

[14]Serialized implementation

Table 3.29: Impact of different standard serial interfaces for EEPROM access on the throughput for several ciphers relative to the original throughput i. e., higher is better.

Cipher	Approach	I^2C	SPI	Microwire	UNI/O
Midori64	wrap	1.4%	1.5%	1.5%	1.2%
	no-wrap	1.2%	1.4%	1.4%	0.9%
LED-128[14]	wrap/no-wrap	81%	82%	82%	81%
KTANTAN32/48/64	std	2.5%	4%	5%	1.6%
	ext	45%	50%	50%	40%
PRINTcipher-48[14]	wrap	17.8%	20%	20%	16%
	no-wrap	13.3%	16.7%	17.6%	10.7%
A2U2	wrap	4.8%	7.1%	8.7%	3%
	no-wrap	4.7%	6.8%	8.4%	3%
Sprout and Plantlet	wrap	88.9%	100%	100%	80%
	no-wrap	66.67%	83.33%	87.91%	53.33%

from 100% to 1.4%, i.e., by a factor of almost 72 when switching from an implementation where accessing the key incurs no overhead compared to using I^2C in wrap case. One exception is KTANTAN where the key bits are accessed in random order. Here, the distinction between wrap case and no-wrap case would not make sense. Instead we distinguish between standard and the extended key approach). Standard approach means that the cipher applies selective reading, i.e., each time the next key bit has to be accessed, the read address needs to be configured before. In the extended approach, the key bits are duplicated and stored in the order they need to be accessed during encryption. This allows to use sequential reading and hence achieves higher throughput, but at the cost of increased area. These two approaches are denoted by std. and ext., respectively.

Furthermore, we investigated for the same ciphers the impact of different EEPROM integrations into ASIC on the throughput. The results are displayed in Table 3.30. Here, we consider different customized EEPROM types where "custom x1" means that it requires one clock to read one bit, "custom x8" that eight-bit words can be read in one clock cycle, and so on. Analogously to above, we do not state concrete values but state the throughput in comparison to an idealized version where the access to the key creates no overhead at all.

Table 3.30: Impact of different ASIC EEPROM realizations on the throughput for several ciphers, i. e., higher is better.

Cipher	Custom x1	Custom x8	Custom x16	Custom x32
Midori64	1.5%	12.5%	25%	50%
LED-128	82%	100%	100%	100%
KTANTAN32/48/64				
std.	57%	57%	57%	62.5%
ext.	57%	100%	100%	100%
PRINTcipher-48	20%	72%	84%	89%
A2U2	20%	67%	77.8%	88.9%
Sprout and Plantlet	100%	100%	100%	100%

3.3.3.3 Discussion of the Results

At this point, we would like to stress that our goal is to show the impact on the throughput of lightweight ciphers that continuously access the non-volatile key in the presence of different realizations of NVM. This is by no means intended as an evaluation, neither a judgement, of the ciphers themselves. Whenever possible we did choose the implementations mentioned in the original publication. However, it may well be that for several ciphers, different implementations would result into a lower or higher decrease of the throughput, e.g., the serialized version of Midori would probably be much less affected by the limitations.

Furthermore, since most of the discussed ciphers are available in different variants with different properties, we decided to evaluate area-conservative implementations as provided in the original publications. As our assessments show, the ciphers LED, Sprout and Plantlet keep almost maximum performance regardless of the NVM type due to sequential processing of key bits which makes them suitable for various target applications. Also the cipher KTANTAN shows high performance if the extended key approach is used. However, applying such approach would expand the key storage effort for KTANTAN from 80 bits to 508 bits, which may lead to additional costs. This does not hold for the ciphers LED, Sprout and Plantlet.

The approach of constantly accessing the key from rewritable non-volatile memory is practical but in certain cases negative impacts may occur. With respect to area size, there is no big difference if the key has to be read only once or continuously during encryption, since the logic for reading the key (at least once) has to be implemented anyway. Small extra logic may be needed for synchronization with NVM—cipher should not be clocked unless key material is ready.This approach often leads to significantly decreased speed and is thus undesirable for use cases with timing or energy constraints. However, designs with round key functions that require sequential access to the key bits are almost unaffected irrespective of the underlying NVM type.

3.4 On the Energy Consumption of Stream Ciphers

3.4.1 Motivation

Some previous works have investigated the energy efficiency of block ciphers. In [BDE+13, KDH+12], an evaluation of several lightweight block ciphers with respect to various hardware performance metrics, with a particular focus on the energy cost was done. In [BBR15], the authors looked at design strategies like serialization and round unrolling and the effect it has on the energy consumption required to encrypt a single block of data. Serialization stretches out the execution of each round function over a number of clock cycles and hence was found to be unsuitable for energy efficiency. The authors then proposed a formal model for energy consumption in any r-round unrolled block cipher architecture. The authors concluded that the energy consumed for encrypting one block of plaintext, for any r-round unrolled implementation had a quasi-quadratic form (a, b, c are constants and R is the number of iterations of the round function prescribed for the design):

$$E_r = (ar^2 + br + c) \cdot \left(1 + \lceil \frac{R}{r} \rceil\right). \tag{3.7}$$

Although an r-round unrolled cipher consumes more energy per cycle for increasing values of r, it takes fewer cycles to complete the encryption operation itself. So to determine the value of r at which the design consumes least energy is an interesting optimization problem. The authors concluded for block ciphers with lightweight round functions like `PRESENT` and `SIMON`, $r = 2$ was the optimal configuration, whereas for "heavier" round functions like in `AES` and `Noekeon`, $r = 1$ was optimal. Building on these ideas, the block cipher family Midori was proposed in [BBI+15] that optimized the energy consumption per encryption. Such a study for stream ciphers was not available in literature as far as we know.

All the previous works in this field [BBR15, BBI+15, BDE+13, KDH+12] have focused on energy per encryption, which is the energy required to encrypt *one* block of data. However it makes more sense to consider the energy consumption required to encrypt a large number of data blocks taken together, since there is seldom any real world protocol that require encrypting one single block of data. In [BMA+18] we showed that when it comes to encrypting significantly large data, a stream cipher may be energy-wise a better solution than a block cipher. Stream ciphers like Grain [HJM07] and Trivium [CP08] use extremely simple state update operation that typically involves computing multiple boolean functions and state rotation. As a result, unrolling multiple rounds of a stream cipher usually only involve accommodating additional copies of the boolean function circuit (if the number of rounds unrolled is small). This means that the power consumption in stream cipher circuits increases very slowly as additional rounds are unrolled. On the other hand, the number of clock cycles required to encrypt a given amount of plaintext drops linearly with such unrolling, and hence, so does the energy required to perform the encryption operation.

As an instructive example it is interesting to compare the energy consumptions of the single and two-round unrolled Grain v1 circuits. A single round implementation of the Grain v1 circuit synthesized with the standard cell library of the STM 90nm logic process, takes around 1164 GE and has an average power consumption of 40.567 μW at a clock frequency of 10 MHz. In order to encrypt 64 bits of data the circuit has to operate for 1 (loading the Key-IV) + 160 (for Key-IV mixing) + 64 = 225 clock cycles. Therefore the energy required for the operation is approximately $40.567 * 225 \approx 912.8$ pJ. On the other hand a two-round unrolled Grain v1 circuit, which performs 2 round operations in one clock cycle, has an area of around 1200 GE and an average power consumption of around 41 μW. However this circuit requires only 1+80+32=113 clock cycles to encrypt 64-bit data, and so the energy requirement is only around 463 pJ. So a 2x unrolling results in approximately a 2x reduction in energy as well. For a cipher like Grain v1 which was specifically designed to allow for efficient unrolling of upto 16 rounds we expect the trend to persist for at least up to 16th degree of unrolling and perhaps beyond that as well. These results were published in [BMA+18].

Table 3.31: Best cipher configurations with respect to energy consumption

Cipher	Security level	Optimal configuration	Energy (nJ) 1000 blocks
PRESENT	80 bits	2x	155.2
Plantlet	80 bits	16x	64.98
Grain v1	80 bits	20x	33.02
Trivium	80 bits	160x	10.15
Lizard	80 bits	16x	80.34
Midori64	128 bits	2x	90.5
Grain 128	128 bits	48x	25.29
Kreyvium	128 bits	128x	11.29
Trivium-2	128 bits	320x	9.77

3.4.2 Evergy Variations in Stream Ciphers

In this section, we investigate factors that may affect the energy consumption of stream ciphers. The aim is to identify parameters that a designer can choose to increase/decrease the energy consumption. To this end, we perform several experiments and draw necessary conclusions about the characteristics that an energy efficient stream cipher should possess. In [BBI+15], it was pointed out that for any given block cipher, the three main factors that determine the quantity of energy dissipated in the circuit:

(a) The clock Frequency,

(b) The architecture of the individual components, and

(c) the number of unrolled rounds.

Since stream ciphers possess the same basic architecture as block ciphers (see Figure 3.1), the same is likely to be true to some extent for a stream cipher as well. Two components characterize the amount of energy dissipated in a CMOS circuit :

- Dynamic dissipation $E_{dynamic}$ due to the charging and discharging of load capacitances and the short-circuit current,
- Static dissipation E_{static} due to leakage current and other current drawn continuously from the power supply.

Thus the total energy dissipation for a CMOS gate can be written as $E_{gate} = E_{dynamic} + E_{static}$. In this section, we investigate various factors that can affect the energy performance of implementations of stream ciphers. In our experiments, we considered the above three factors that would likely affect the energy metric of the encryption algorithm. In all the simulations reported in this work, we maintained the following design flow. The design was implemented at RTL level. A functional verification of the VHDL code was then done using *Mentorgraphics ModelSim*. Thereafter, *Synopsys Design Compiler* was used to synthesize the RTL design. The switching activity of each gate of the circuit was collected by running post-synthesis simulation. The average power was obtained using *Synopsys Power Compiler*, using the back annotated switching activity. The energy was then computed as the product of the average power and the total time taken for the encryption process.

3.4.2.1 Frequency of Operation

As already pointed out in [KDH+12, BBR15], the energy consumption required to compute an encryption operation should be independent of the frequency of operation, as energy is a quantity which is a measure of the total switching activity of a circuit during the process. This is true for sufficiently high frequencies, where the total static energy E_{static} consumed by the system is low across the total number of cycles required for encryption. However it was shown in [BBR15] that for circuits designed with the standard cell library of the STM 90nm low leakage CMOS process, at frequencies lower than 1 MHz, the static energy naturally starts to play a significant role, thereby increasing the energy consumption and that for frequencies higher than 1 MHz, the energy

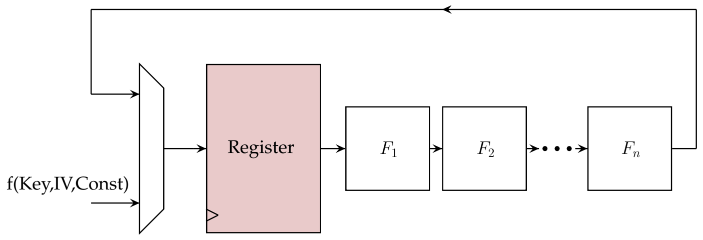

Figure 3.1: n round unrolled implementation of a stream cipher

consumption is more or less invariant with respect to frequency. So in our experiments, we fixed the frequency of operation to 10 MHz (this corresponds to a clock period of 100 ns), so that the leakage power plays minimal role in the energy consumption.

3.4.2.2 Architecture

Often, there are various ways of implementing stream ciphers. We will take a detailed look at a few of them:

A. Scan Flip-Flops vs Regular Flip-Flops Figure 3.1 depicts the diagram of a stream cipher which has been unrolled n times. Unrolling in stream ciphers refers to implementations where we include logic gates for multiple round update functions in the same circuit, so that it is possible to do multiple round computations in a single clock cycle. The storage element of the design is usually preceded by a multiplexer, which in the initial clock cycle filters a combination of the key and IV on to the register and the output of the round function thereafter. The combination of the flip-flop and multiplexer can be replaced with a scan flip-flop which logically achieves the same functionality, while occupying less area and less power. Hence the intuition was that designs using scan flip-flops would be more energy-efficient. This was confirmed by simulations tabulated in Table 3.32. The table shows simulation results for the six hardware-based stream ciphers Grain v1, Grain 128, Trivium, Plantlet, Lizard and Kreyvium synthesized with the standard cell library of the STM 90nm logic process. It displays the energy consumptions for encrypting for both 1 and 1000 blocks of plaintext where one **b**lock is taken to be equal to 64 bits.

For example, Grain v1, takes 1 (loading key-IV) + 160 (initialization) + 64 = 225 cycles to encrypt 1 block of plaintext. So as given in the table, using regular flip-flops, the energy required is given as 225 $\times$ 100 ns $\times$ 40.6 uW $\approx$ 912.8 pJ. Similarly, $161 + 64,000 = 64,161$ cycles are required to encrypt 1,000 blocks, and so the energy required for it can be estimated as $64,161 \times$ 100 ns $\times$ 40.6 uW $\approx$ 260.28 nJ.

Table 3.32: A comparison of energy consumptions for Regular and Scan flip-flops R: Regular flip-flop, S: Scan flip-flop

#	Cipher	FF	Area (GE)	Power* (uW) @ 10 MHz	Energy (pJ) 1 block	Energy (nJ) 1000 Blocks
1	Grain v1	R	1164	40.6	912.8	260.28
		S	1005	38.9	874.8	249.47
2	Grain 128	R	1700	71.5	2287.1	459.23
		S	1455	57.8	1855.4	371.41
3	Trivium	R	1870	78.4	9527.6	510.48
		S	1584	75.6	9194.9	492.26
4	Plantlet	R	886	35.4	1364.7	227.99
		S	785	34.4	1363.1	227.73
5	Lizard[15]	R	1481	51.8	1663.2	332.93
		S	1360	50.4	1617.5	323.78
6	Kreyvium	R	3433	146.2	17792.5	952.53
		S	2892	140.8	17135.4	917.35

*The figures are given for the average power consumption

Main Conclusions: We can draw the following conclusions from the results reported in Table 3.32. First, designs implemented with scan flip-flops are shown to be better both with respect to energy consumption and circuit area. Since all other factors remain equal, reduced area when using scan flip-flops results in reduced power consumption. Since energy is simply the time integral of the power, this also results in reduced energy consumption.

B. Fibonacci vs Galois Configuration Stream ciphers like Grain v1, Grain 128 and Trivium were designed having the Fibonacci configuration of the deployed LFSR. As shown in Figure 3.2A, a Fibonacci shift register updates its states by a one bit shift at each clock cycle. Only the final bit is updated with a round function value computed on the current state. In comparison, in the Galois shift register, each state bit is updated with function f_i computed on the current state, as shown in Figure 3.2B. Note that although it is not shown explicitly in Figure 3.2, each of the functions f_i are computed over the entire state and not just the preceding bit. Galois equivalent circuits for Grain v1 and Grain 128 were proposed in [Dub10, Dub09]. The authors showed that Galois configurations usually have lower circuit latency and thus can be used to make faster throughput designs. In Table 3.33, we tabulate

[15]We use the implementation of Lizard that loads key-IV in one clock cycle

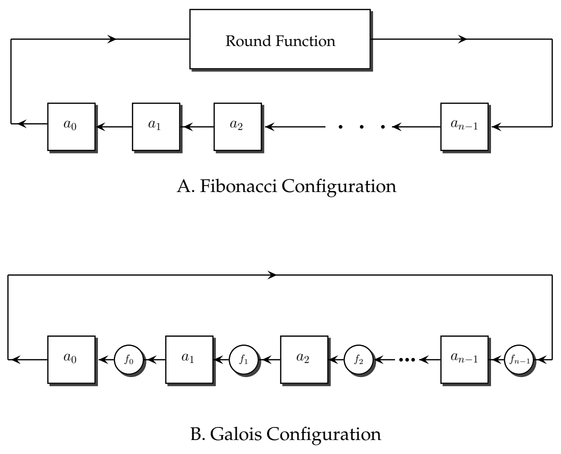

Figure 3.2: Fibonacci and Galois configurations for shift registers

results for a comparison between Galois and Fibonacci configurations. Note that for the Plantlet stream cipher, the non linear register update function takes inputs from the 39-th which is its last bit. As a result it is not possible to realize a Galois configuration for the non-linear register [Dub09]. Also the linear register operates in the Fibonacci mode, and requires its 60-th bit to be held at logic HIGH till the key-initialization is over. As such it is is difficult to realize the exact Galois representation for the linear register. So we omit Plantlet from the list of results in Table 3.33.

Securitywise, both configurations do not offer any significant advantages over the other with respect to classical cryptanalysis. However in [CMM14], it was shown that that Galois registers were more vulnerable to power attacks as compared to its Fibonacci counterpart. The authors performed thresholding results over various sample window lengths of successive clock cycles in case of both the register configurations. They were able to find the initial state of the Galois register using approximately half the number of power traces as compared to Fibonacci registers.

Main Conclusions: Energy and area-wise, a Galois configuration does not seem to offer any significant advantage over its Fibonacci counterpart. With respect to all the ciphers that were primarily designed in the Fibonacci mode, Galois configurations have another significant disadvantage. We will see later that for a stream cipher to be energy efficient, it needs to be unrolled multiple number of times. Galois equivalents of these ciphers that were primarily designed for the Fibonacci mode, cannot be unrolled beyond a certain limit as shown in [Dub09]. This also makes Galois configurations unattractive for low-energy designs.

Table 3.33: Results for Fibonacci vs Galois configurations, 1 block=64 bits, G: Galois, F: Fibonacci configuration

#	Cipher	Conf	Area (GE)	Power* (uW) @ 10 MHz	Energy (pJ) 1 block	Energy (nJ) 1000 Blocks
1	Grain v1	G	1016	39.8	894.4	255.05
		F	1005	38.9	874.8	249.47
2	Grain 128	G	1466	58.9	1890.9	378.52
		F	1455	57.8	1855.4	371.41
3	Trivium	G	1592	76.0	9253.6	495.40
		F	1584	75.6	9194.9	492.26
4	Lizard	G	1366	50.7	1626.0	325.49
		F	1360	50.4	1617.5	323.78
5	Kreyvium	G	2898	141.3	17196.2	724.16
		F	2892	140.8	17135.4	917.35

*The figures are given for the average power consumption

C. Architecture of Round Function The round functions F_i in hardware-based stream ciphers (as shown in Figure 3.1) are generally very simple. These round functions generally involve one bit shift (that can be efficiently implemented by shift registers) and one or multiple small boolean functions to update the terminal bit of the register. We look at three possible ways of approaching these.

1. The first approach is to use a look-up table. For an n-variable boolean function this is a table of $2^n \times 1$ entries. For obvious reasons, although effective for small n, this kind of circuit style is inadvisable, for larger values of n.
2. This approach is to feed the functional description (in terms of the algebraic normal form) of the boolean function to the synthesizer and instructing it to optimize for area and power. In this approach, we depend on the ability of the circuit synthesizer to optimize the circuit according to the requirements of the design.
3. The third approach is to use a Decoder-Switch-Encoder (DSE) style configuration. This approach was previously considered in [BBR15, BBI+15] for designing the 8-bit Rijndael S-box and was shown to be energy efficient in [BBR15, BBI+15]. For implementing boolean functions we have to improvise the corresponding circuit for an S-box. The first step is the same as in the S-box circuit: we implement the decoder first i.e., from an n-bit input, we implement a set of 2^n wires such that only one of them would result in a logical HIGH for any particular input value. It is easy to see that there would

be one wire corresponding to each input value, and we simply logically OR all the wires whose inputs result in a logical HIGH in the truth table of the function we are implementing. In fact it is clear, that we don't even need to expend hardware for constructing wires whose inputs result in a logical LOW in the truth table.

However, the circuit size is still exponential in the length of input and such circuits are also not advisable for large n. Since in both the DSE or the LUT construction, the circuit size is exponential in the number of input variables, we adopted a simple tweak. Whenever the number of input variables of a function exceeded 10, we broke up the function as the GF(2) sum of roughly equal sized component functions of input size less than 10, and constructed the circuits for each of the component functions. Breaking up a function into components is straightforward for some ciphers. For example, in Plantlet, the NFSR update function g is given as

$$\begin{aligned} g = {} & n_0 + n_{13} + n_{19} + n_{35} + n_{39} + n_2 \cdot n_{25} + n_3 \cdot n_5 + n_7 \cdot n_8 + n_{14} \cdot n_{21} + \\ & n_{16} \cdot n_{18} + n_{22} \cdot n_{24} + n_{26} \cdot n_{32} + n_{33} \cdot n_{36} \cdot n_{37} \cdot n_{38} + \\ & n_{10} \cdot n_{11} \cdot n_{12} + n_{27} \cdot n_{30} \cdot n_{31} \end{aligned}$$

Although this is a function of 29 variables, each variable occurs only once and hence there is no intersection of terms between any 2 monomials. Hence it is easy to break up g as a sum of five functions (say g_1 to g_5) each of 5 or 6 variables, such that no two component functions depend on the same input variable. However, this is not always the case. The NFSR update function of Grain v1 for instance, has 13 variables, and breaking it up into functions of disjoint variables is not straightforward. However the DSE construction does not explicitly require that the inputs of the component functions be disjoint. For example, the Grain v1 NFSR function can be written as the sum of four non-disjoint functions of 8, 7, 6, 3 variables each.

Main Conclusions: In Table 3.34 we list the simulation results for all the 3 architectures of the round function. It is clear from the table that LUT or DSE style constructions of the boolean function have no significant advantage over the circuit optimized by the synthesizer.

3.4.2.3 Unrolling Rounds

Unrolling rounds is a design technique, which essentially serves the purpose of speeding up the circuit throughput at the cost of area. For example, a two round unrolled AES circuit consists of two sequentially placed circuits for the round functions, that computes the ciphertext in only 5 clock cycles (i.e. half the time as compared to a single round

Table 3.34: Results for varying architecture of round functions, Lut: Lookup table, Fun: Functional synthesis using Synopsys tool, Dse: DSE configuration

#	Cipher	Conf	Area (GE)	Power (uW)* @ 10 MHz	Energy (pJ) 1 block	Energy (nJ) 1000 Blocks
1	Grain v1	Lut	1071	43.3	973.7	277.68
		Fun	1005	38.9	874.8	249.47
		Dse	1088	41.7	938.4	267.61
2	Grain 128	Lut	1449	57.9	1858.3	371.98
		Fun	1455	57.8	1855.4	371.41
		Dse	4165	76.3	2449.0	490.23
3	Trivium	Lut	1589	75.7	9211.1	493.12
		Fun	1584	75.6	9194.9	492.26
		Dse	1680	78.4	9542.8	510.88
4	Plantlet	Lut	785	34.5	1326.3	221.58
		Fun	785	34.4	1324.6	221.30
		Dse	1143	42.7	1644.1	274.68
5	Lizard	Lut	1327	49.9	1601.8	320.64
		Fun	1360	50.4	1617.5	323.78
		Dse	1946	58.5	1878.5	376.03
6	Kreyvium	Lut	2897	141.2	17184.0	919.96
		Fun	2892	140.8	17135.4	917.35
		Dse	2988	144.0	17524.8	938.20

*The figures are given for the average power consumption

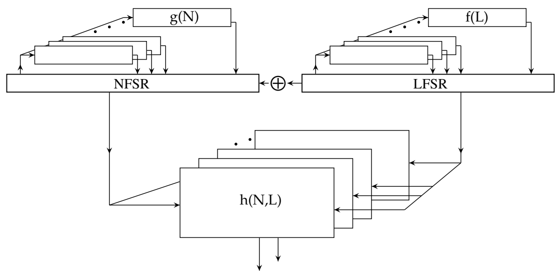

Figure 3.3: Upto 16x implementation of Grain v1

circuit). In [BBR15], a comprehensive analysis of the energy consumption of round unrolled circuits was given.

For stream ciphers, we have already seen that we expect round unrolling to be highly beneficial with respect to energy consumption. Most hardware stream ciphers have very simple round functions (consisting of a logical shift and a boolean function computation). Thus, for the first rounds of unrolling, we do not expect significant increase in algebraic and hence hardware complexity (i.e. lesser number of logic gates) of the circuit . This would naturally limit transient switching activity (signal glitches) from one round to the next [BBR15]. Since less glitches lead to lower power consumption, it is quite often the case that unrolling stream ciphers by one round does not really increase the power consumption by much, whereas it always decreases the number of clock cycles required to encrypt a given amount of data and so the energy consumption decreases with unrolling. A good example of this is the Grain v1 circuit we were alluding to in the introduction. The single round and the 2 round unrolled circuits have an average power consumptions of 40.567 and 41 μW respectively, at a clock frequency of 10 MHz. Since the number of clock cycles required to encrypt data in the 2 round circuit is approximately half as compared to the single round circuit, a 2x unrolling results in approximately a 2x reduction in energy as well.

Unrolling in RTL.

Stream ciphers like Grain v1, Grain 128 and Trivium were specifically designed to allow easy unrolling. In Grain v1 for example, the last 16 bit positions in both the linear and non-linear register are used neither in the round update function nor the output keystream function. This implies that a 16x unrolling of Grain v1 is straightforward [HJM07], and only requires 16 additional copies of the round and update functions to be added to the circuit as shown in Figure 3.3. The same is true for Grain 128 upto 32x and Trivium for upto 64x.

For degrees of unrolling higher than that specified in the design, the algebraic structure of the round update function gets more and more complicated, since simply adding more copies of round functions will no longer lead to correct functionality.

With increased degree of unrolling, the hardware structure of the circuit gets more complicated and hence so do the transient signals produced in the circuit. This leads to a higher power consumption in the circuit. But since each extra unrolling increases the number of keystream bits produced per cycle, it decreases the total number of clock cycles required to encrypt a given amount of data. So for smaller degrees of unrolling, the decrease in the time taken to encrypt outweighs the increase in power, and the energy consumed to encrypt a given amount of data decreases with unrolling. At some point of time, more and more unrolling makes the circuit complex enough to increase the power consumption beyond the leeway provided by the decrease in time to encrypt, and so the energy consumption would increase steadily after this point. The point at which this happens would of course depend on the architecture of the cipher in question.

In Table 3.35, we list the simulation results for energy consumptions for different degrees of unrolling. We use scan based flip-flops to construct the memory element and use functional optimization of the round function circuit. We also compare the results with the corresponding energy performances of lightweight block ciphers `PRESENT` and `Midori64` in the ECB mode. There are several important issues we can observe from the simulation results. We list them one by one.

Block ciphers: We compare our results with block ciphers `PRESENT` and `Midori64`. `PRESENT` has been included as a standard in ISO/IEC 29192-2 and was shown in [BBR15] to be extremely energy efficient. On the other hand the `Midori` block cipher family was designed specifically for low energy consumption. Although a subspace attack [GJN$^+$16, TLS16] that exploits a class of weak keys of `Midori64` has been reported, we use this cipher in our comparisons, as it signifies a lower energy limit achievable with block cipher encryption. Note that, for the block ciphers, in the last column we tabulate the energy required for encrypting data in ECB mode, which is the lowest possible energy the system can consume while encrypting multiple blocks. Usually CBC or CTR mode will include the energy consumption for an additional 128 bit xor gate, but we report the consumption in the ECB mode since we just aim to make a comparison.

Secondly, it is difficult to express the energy consumption of an r-round unrolled stream cipher, in the same way that was done for an r-round block cipher in Equation (3.7). Unlike block ciphers, unrolling a stream cipher by an additional round does not increase the circuit complexity uniformly. As a result the transient signals do not increase uniformly across round functions as in block ciphers, and so it is difficult to algebraically model the energy consumption.

Shorter vs longer data lengths: Note that while for encrypting a single block of data, block cipher based solutions outperform stream ciphers, the opposite is true for larger lengths of data. For shorter lengths of data, the energy consumed by the stream cipher is dominated by the key initialization phase. For example, the 1x implementation of Trivium would take 1217 clock cycles to encrypt 64 bits, of which 1152 is used up by the key initialization function. A one round implementation of `Midori64` would take only 17 cycles to encrypt 64 bits. For longer data, the effect of key initialization on the energy consumption becomes less significant, since it is computed only once. To encrypt 1000 blocks (64,000 bits) of data, Trivium 1x would require only $64,000 + 1,152 + 1 = 65,153$ cycles. Clearly $1,152$ is a much smaller fraction of $65,153$ than of $1,217$. Multiple unrolling decreases the time to encrypt even further. For example, the 160x implementation of Trivium can encrypt 160 bits in a single clock cycle, and so around $1 + \lceil \frac{64,000+1,152}{160} \rceil = 409$ cycles are required for 1000 blocks. On the other hand, the most energy-efficient version of `Midori` (2x) would take $9 * 1,000 = 9,000$ cycles to encrypt 1,000 blocks. As a result we see that for the most energy-efficient configuration of Trivium (160x) is around 9 times more energy efficient than the most energy efficient version of `Midori64`. In Figure 3.5 we plot the energy consumptions for encrypting upto 10 blocks of data with the most energy efficient unrolled configurations of the ciphers. Although for a single block of data, `Midori64` performs best, whereas for 6 blocks of data or more Trivium performs best.

Parabolic behavior with unrolling With respect to unrolling, the energy consumption for stream ciphers follows the same parabolic behavior as block ciphers [BBR15], particularly for longer lengths of data. Which is to say that for smaller degrees of unrolling the energy consumption is very high, the energy consumption comes to a minimum at some fixed degree of unrolling, and the energy consumption increases again if the cipher is unrolled beyond this point. Two contradicting reasons are responsible for the shape of this plot. For lower degrees of unrolling, the energy consumption is obviously high due to **1**) large number of initialization rounds in stream ciphers, as already mentioned, and **2**) lower number of bits encrypted per clock cycle. A single round unrolled version of Grain v1, would encrypt one bit of plaintext per clock cycle. This means that to encrypt one bit, the design has to pay for the energy consumption of the 160-bit register and the associated logic functions per clock cycle. Thus the total electrical work that the battery source would need to do to encrypt 1,000 blocks is significantly large.

For the same reasons, the energy consumption starts to decrease, when the degree of unrolling is increased. Larger degree of unrolling implies less time spent in initialization and more bits encrypted per cycle. For example, a 32x unrolled version of Grain v1, would need only 5 clock cycles for initialization. To encrypt 32 bits of data, the system would have to pay for the energy consumption of the 160-bit register per clock cycle. However, the logic functions in a 32x unrolled version are more than 32 times more complex than in a 1x design, and it is true that more power is consumed in the hardware circuit of the logic functions. Inspite

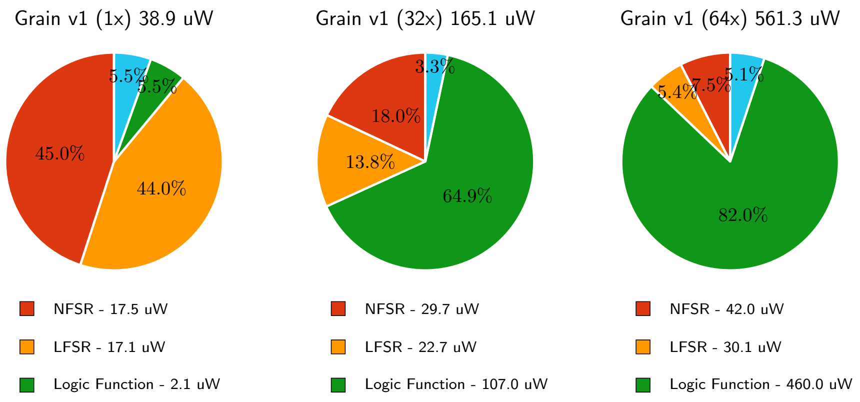

Figure 3.4: Power consumption shares of Grain v1 for 1, 32, 64 degrees of unrolling

of that, we can see that a 32x implementation of Grain v1 is around 6.5 to 7 times more energy efficient than the 1x version for short data lengths, and around 7.5 times better for longer data lengths.

After a certain point of time, increased unrolling results in increased energy consumption. The reason for that is the power consumed in the logic functions increases sharply at that point. This happens due to the reasons which are similar for block ciphers [BBI+15]. In [BBI+15, Figure 2], it was shown that power consumption in sequentially placed logic functions increases uniformly because of increased circuit latency which leads to increased glitch propagation. As a result, unrolling the round functions beyond a fixed number usually proves counterproductive. Figure 3.4, demonstrates the increasing share of power consumed by the logic functions in Grain v1 over 1, 32 and 64 degrees of unrolling. It is easy to see that at 64x, the most power hungry element of the design is the round function.

Light/heavy round function A comparison of the energy consumptions given in Table 3.35, especially for longer data length, is interesting as the results show that the behavior in stream ciphers is similar to block ciphers, when compared with respect to the "lightness" of round functions. It was shown in [BBR15], that block ciphers with light round functions like `PRESENT`, `Twine`, `SIMON` produce less glitches when the circuits for more than one round function are connected serially. Hence, block ciphers with light round functions achieve energy optimality when unrolled twice, in contrast with heavy round functions whose single round versions are most energy efficient. In Table 3.35, it is seen that in ciphers like Grain v1, Lizard and Plantlet whose update functions are more algebraically complex, the energy

Table 3.35: Comparison of energy for different degrees of unrolling.

#	Cipher	r	Area	Avg. Power (uW)	Energy (pJ)	Energy (nJ)	Energy/bit
			(GE)	@ 10 MHz	1 block	1000 Blocks	(pJ)
1	Grain v1	1	1005	38.9	874.8	249.47	3.90
		16	2673	86.6	129.9	34.73	0.54
		20	2888	102.9	133.8	33.02	0.52
		24	3293	129.4	142.3	34.61	0.54
		28	3711	156.5	140.8	35.88	0.56
		32	3934	165.1	132.1	33.12	0.52
		48	5751	343.1	205.9	45.91	0.72
		64	7474	561.3	280.7	56.30	0.88
2	Grain-128	1	1455	57.8	1855.4	371.41	5.80
		32	3579	126.8	139.4	25.47	0.40
		40	4178	158.1	142.3	25.42	0.40
		48	4749	188.8	151.0	25.29	0.40
		56	5321	235.2	164.6	27.02	0.42
		64	6336	282.7	169.6	28.41	0.44
		80	7078	407.7	203.7	32.81	0.51
3	Trivium	1	1870	78.4	9527.6	510.48	7.97
		64	3051	128.7	257.4	13.11	0.20
		80	3457	148.1	251.7	12.08	0.19
		96	3839	169.4	237.1	11.51	0.18
		128	4593	207.1	227.8	10.56	0.17
		160	5409	248.2	223.4	10.15	0.16
		192	6179	306.2	244.9	10.44	0.16
		256	7755	419.5	251.7	10.73	0.17
		288	8584	490.0	294.0	11.17	0.17
4	Plantlet	1	785	34.4	1324.6	221.30	3.46
		8	1630	88.5	433.7	71.15	1.11
		16	2254	161.6	404.0	64.98	1.02
		32	3451	651.5	847.0	131.02	2.05
5	Lizard	1	1360	50.4	1617.5	323.78	5.06
		8	2565	101.7	417.0	81.70	1.28
		16	3954	200.0	420.0	80.34	1.26
		32	6778	672.4	739.6	135.09	2.11
6	Kreyvium	1	2892	140.8	17135.4	917.35	14.33
		64	4579	202.8	405.6	20.66	0.32
		80	5045	224.0	380.8	18.28	0.29
		96	5480	248.8	348.3	16.92	0.26
		128	5050	221.4	243.5	11.29	0.18
		160	7268	364.7	328.2	14.92	0.23
		192	8149	430.7	344.6	14.69	0.23
		256	8612	452.6	271.6	11.59	0.18
		288	10836	696.1	417.7	15.87	0.25
7	`PRESENT`	1	1440	52.2	172.3	172.3	2.69
		2	1968	91.3	155.2	155.2	2.43
		3	2500	149.0	178.8	178.8	2.79
8	`Midori64`	1	1542	60.6	103.0	103.0	1.61
		2	2017	100.6	90.5	90.5	1.41
		3	2826	273.8	191.7	191.7	3.00

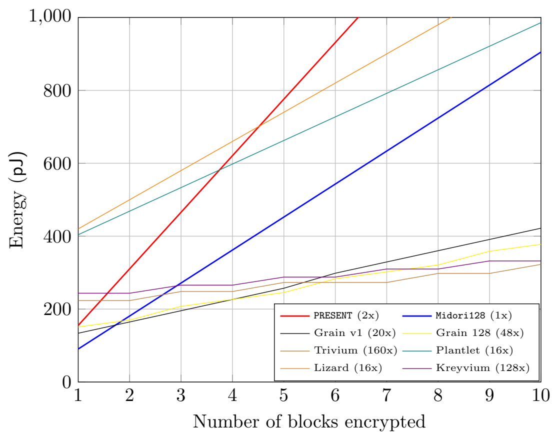

Figure 3.5: Energy consumptions for upto 10 blocks for most energy-efficient implementations

optimality is achieved at smaller degrees of unrolling. This is in contrast with Trivium which has an extremely simple round update function consisting of 3 and gates and 6 xor gates. Trivium achieves energy optimality at 160x unrolling.

Also there seems to be a distinct advantage for unrolled stream ciphers with simple update functions like Trivium, especially for encryption of longer data streams. For example, Trivium unrolled at 160x, would for the encryption of 160 bits in every clock cycle, pay for only the power consumed in the 288 bit register and the circuit for the logic function. Unlike Grain v1, the logic update of Trivium is very simple, and this ensures that even at 160x, the algebraic and hardware complexity of the round update is not significant. At 160x, the round function in Trivium consumes only 134 μW which is only around 54% of the total power. This is in contrast with the 64x Grain v1 implementation, which consumes around 460 μW, which is 82 % of the total power.

Comparison with Kreyvium Since Kreyvium builds upon the Trivium structure by adding two additional registers for key and IV rotation, and two additional xor gates, we have seen that 1x unrolled versions of Kreyvium consume 1.5 to 2 times more energy as Trivium even for longer data lengths. This trend is also seen in higher round unrolled versions except for implementations where the number of unrolled rounds is a multiple of 128. These versions do not need additional registers to implement key and IV rotation since they can be assumed to be available on the wires, and hence these implementations have lower energy consumption. Inspite of these the additional complexity of 2 xor gates in the round function implies that even for multiples of 128, the most energy-efficient configuration of Kreyvium consumes around 10% more energy than Trivium.

From the discussion in this section, it is clear that in the longer run (encryption of longer streams) stream ciphers with **s**impler update functions have a distinct advantage. These are easier and more energy-efficient to unroll for **h**igher degrees of unrolling. Higher degrees of unrolling allows to encrypt more bits in one clock cycle, which is crucial in bringing down the number of clock cycles required to encrypt a given length of data, and hence the energy consumption. On the other hand, higher degree of unrolling, does imply that the logic for the update function becomes more complex and hence needs more power to operate, but if the update function is kept simple enough, it ensures that the additional power consumption is small enough not to outweigh the natural advantages obtained from unrolling. Lastly, the number of initialization rounds does affect the energy numbers for shorter data packets, but its effect becomes minimal with the increase in the length of plaintext to be encrypted.

CHAPTER 4

Proposed Approaches and Techniques

4.1 Chapter Overview

This chapter is devoted to new implementation techniques and approaches which lead to the more lightweight security solutions.

In Section 4.2, we discuss an implementation technique for increasing the throughput of stream ciphers without usage of additional memory. The technique adapts the well-known principle of pipelining of the output function. The core difference however is that we identify "unused" regions within the FSR and use them for storing intermediate values, hence mitigating the need for additional memory. Here, "unused" means registers of the FSRs that are only used for storing values and that are neither involved in the update function nor the output function. We consisely describe sufficient conditions for the applicability of this technique and demonstrate it on the stream ciphers Grain-128 and Grain-128a. The results presented in the section are based on the papers [AM14] and [AM].

In Section 4.3 we explore an extension in the common design approach for stream ciphers which allows to reduce their area size. There exists a well-known rule of thumb which says that the internal state size of a stream cipher should be at least twice the security parameter, otherwise, the cipher can be broken. This makes it difficult to realize stream ciphers with low area size, as long as memory is usually the most expensive part of the design. In this section, we demonstrate that a simple shift in the common design paradigm, namely, involving the key into the update process of the internal state of a cipher, allows to improve its resistance against certain attacks, and hence, to develop secure designs with lower internal state size. Section 4.3 is based on the paper [AM15].

4.2 On Increasing the Throughput of Stream Ciphers

4.2.1 Motivation

A common approach for increasing the maximum throughput of a circuit which implements a given function is the *pipelining technique*. In order to reduce the delay of the critical path of the circuit, it is divided into several parts which are computed in parallel. Pipelines can be implemented at a number of layers. During each clock cycle, the output of each block is stored in a memory stage. At the next layer the values from the previous layer are combined to new blocks and so on until the last layer outputs the result of the function. However, each layer of the pipelining except the last one requires additional memory for storing the intermediate results, which is the most expensive part in terms of the area size and power consumption and, according to [MD10, MD13], induces additional latency.

4.2.2 High Level Description

Before we explain the proposed transformation in detail, we provide a high level description.

Our technique can be seen as a special case of pipelining but where existing structures are cleverly re-used for avoiding additional memory as much as possible. It is motivated by the observation that FSRs are usually implemented in the Fibonacci configuration. This means that at each clock, all but one state entries are simply shifted while only the remaining entry requires more involved computations.

The idea is now that some of the computations that should take place in the output function are "outsourced" to the FSR update functions and to store the intermediate results in the FSR. Assume that the output function Out (see Equation 4.2) can be written as

$$z_t = \mathsf{Out}(St_t, B_t) = St_t\beta + \mathsf{Out}_1(St_t, B_t) + \mathsf{Out}_2(St_t, B_t) \tag{4.1}$$

with $B_t = (b_t, \ldots, b_{t+\ell-1})$. In principle, the transformation removes Out_1 from Out and inserts it into the update function f_β:

$$(f_\beta, \mathsf{Out} = St_t[\beta] + \mathsf{Out}_1 + \mathsf{Out}_2) \overset{\text{Transf.}}{\Rightarrow} (f'_\beta := f_\beta + \mathsf{Out}_1, \mathsf{Out}' = St_t[\beta] + \mathsf{Out}_2)$$

Observe that the overall computational effort has not been increased. Moreover, it is not necessary to insert additional memory for storing the intermediate value $\mathsf{Out}_1(St_t, B_t)$ as it is stored in the FSR register with index β. Finally, one sees that Out' produces the same output as Out but its complexity has been reduced. If possible, one may repeat this step several times for different parts of Out so that eventually the output function becomes a linear function where simply a selection of FSR state entries is added.

Of course, care needs to be taken that the transformation is preserving. To ensure that the FSR output, i.e., the value at index 0, is not affected by the transformation, we apply the following trick: we integrate Out_1 into *two* different update functions f_α and f_β for $\alpha < \beta$. The modification of f_β insert the value $\mathsf{Out}_1(St_t, B_t)$ as explained above such that it can be used directly in the output function. In the subsequent clocks, this value is simply shifted until it reaches position α. Here the modification of f_α has been such that the value is canceled out again. The consequence is that the transformation possibly changes the FSR entries at indexes $\alpha \ldots \beta$ but leaves the state entries outside of this interval unchanged, including the value at index 0 which defines the FSR output. This property is proven for different variants of the transformation (see the proofs of Theorem 1, Lemma 1, and Theorem 2).

While the approach is conceptually simple, several aspects need to be considered:

1. It may not be sufficient to ensure that the FSR output is preserved. If the cipher uses functions where part of the inputs are taken from the interval $\alpha \ldots \beta$, then

the output of this function would be altered as well. To avoid such effects, we focus on intervals of state entries such that all other values are independent of this interval in a certain sense. This is made precise by the notion of *isolated intervals* (see Definition 4).

2. For canceling out the change at position α, one cannot simply replace f_α to $f_\alpha + \mathsf{Out}_1$. The reason is that Out_1 would use state entries that are $\beta - \alpha$ clocks later than when the change at index β has been introduced. Thus it is necessary that the value $\mathsf{Out}_1(St_t, B_t)$ can still be reproduced $\beta - \alpha$ clocks later. We capture this formally by saying that a function has *sustainable outputs* (Definition 5).

In fact, the concrete transformation (including the prerequisites and the proof of correctness) is technically involved. Therefore we split the technical treatment into two parts. In section 4.2.3 we introduce a preserving transformation for *FSRs* where an external bit stream is integrated into some of the update functions (Theorem 1). As we make no assumptions on this bitstream, the transformation might be of independent interest. Afterwards we explain an extension where this bitstream may depend on the FSR itself (Lemma 1). This requires a careful argumentation why the FSR-transformation is still preserving. For practical reasons, we restrict to bitstreams that depend on values only which are accessible over several clocks without requiring additional memory. These transformations represent the basic building block for the cipher transformation presented and discussed in section 4.2.4.

4.2.3 New Preserving FSR-Transformation

4.2.3.1 Feedback Shift Registers with External Input.

While stream ciphers commonly use a discussed specific type of FSRs (See definition 1), we consider the following, significantly broader class of FSRs:

Definition 3 (FSR with External Input). A *FSR with external input* FSR_E of length n consists of an internal state of length n, an external source which produces a bit sequence $(b_t)_{t\geq 0}$, and update functions $f_i(x_0, \ldots, x_{n-1}, y_0, \ldots, y_\ell)$ for $i = 0, \ldots, n-1$. Given some initial state $St_0 = (St_0[0], \ldots, St_0[n-1]) \in \mathbb{F}^n$, the following steps take place at each clock t:

1. The value $St_t[0]$ is given out and forms a part of the *output sequence*.

2. The state $St_t \in \mathbb{F}^n$ is updated to St_{t+1} where $St_{t+1}[i] = f_i(St_t, b_t, \ldots, b_{t+\ell-1})$.

We denote by $\mathrm{seq}(\mathsf{FSR}_\mathsf{E}, St_0, (b_t)_{t\geq 0})$ the output sequence of FSR_E given an initial state St_0 and an external bit sequence $(b_t)_{t\geq 0}$.

Observe that we make no restrictions on the update functions whereas stream ciphers commonly deploy FSRs which are in Fibonacci configuration. A further relaxation is that the update of the state depends on the current assignment *and* possibly on bits from an external source. Observe however that the bits $(b_t)_t$ of the external source are not affected by the state bits.

Two FSRs $\mathsf{FSR_E}$ and $\mathsf{FSR_E}'$ of the same length with access to the same external source are called *equivalent*, denoted by $\mathsf{FSR_E} \equiv \mathsf{FSR_E}'$, if for any initial state St_0 for $\mathsf{FSR_E}$ there exists an initial state St'_0 for $\mathsf{FSR_E}'$ (and vice versa) such that both produce the same output sequence for any external bit sequence $(b_t)_{t\geq 0}$. A transformation which takes as input some FSR $\mathsf{FSR_E}$ (and possible other inputs) and outputs a FSR $\mathsf{FSR_E}'$ such that $\mathsf{FSR_E} \equiv \mathsf{FSR_E}'$ is called *preserving*.

The majority of stream ciphers can be characterized as follows: They deploy one or several regularly clocked finite state machines, typically including at least one FSR. At each clock several values of these components are fed into an output function Out which eventually produces the current keystream bit. We assume that the keystream bit is computed before the FSRs are updated. In principle we investigate if and how certain computations that would take place in Out can be shifted to one of the deployed FSRs such that (i) the resulting cipher remains equivalent but (ii) the throughput is increased. Consequently to keep the description as simple and readable as possible and to cover a maximally broad class of stream ciphers, we consider stream ciphers which contain three components only: an FSR $\mathsf{FSR_E}$ of length n, an output function Out, and an external block EB. Here we make no restrictions on the processes running inside of EB but consider it as a black box which may contain further FSRs, additional memory, etc. The only assumption we make is that a bitstream $(b_t)_{t\geq 0}$ can be specified which contains all bits produced within EB which are relevant for the state updates of $\mathsf{FSR_E}$ and/or for computing the next keystream bit. Observe that this does not exclude the case that EB may contain several components, each producing its own bitstream. For example if we consider ℓ components where the i-th component produces a bitstream $(b_t^{(i)})_{t\geq 0}$, these bitstreams can be joined to one bitstream as follows: $(b_t)_{t\geq 0} = b_0^{(1)}, \ldots, b_0^{(\ell)}, b_1^{(1)}, \ldots$. Adopting the notation from Definition 3, the output function Out is defined over some variables $\mathsf{Out}(x_0, \ldots, x_{n-1}, y_0, \ldots, y_\ell)$ and the t-th keystream bit is

$$z_t = \mathsf{Out}(St_t, b_t, \ldots, b_{t+\ell-1}). \tag{4.2}$$

We adopt the notions of *equivalence* and *preserving transformation* from the topic of FSRs for ciphers in a straightforward manner.

4.2.3.2 Preserving FSR transformation

We now provide a detailed technical description of a new preserving transformation for FSRs which is an integral part of the cipher transformation in section 4.2.4. It requires that the FSR state contains an *isolated interval*, meaning a part of the state which has almost no impact on the update of the remaining part and, if existent, is independent of some output function Out. The formal definition is as follows:

Definition 4 (Isolated Interval)**.** Consider an FSR-based stream cipher, being composed of an FSR FSR_E of length n with update mapping $F = (f_0, \ldots, f_{n-1})$, an external block with bit stream $(b_t)_{t\geq 0}$, and a function $\mathsf{Out}(x_0, \ldots, x_{n-1}, y_0, \ldots, y_\ell)$. Interval $[\alpha \ldots \beta]$ with $0 \leq \alpha \leq \beta \leq n-1$ of the FSR-state is *isolated* with respect to F and Out if the following conditions are met:

1. The feedback functions $f_{\alpha-1}, \ldots, f_{\beta-1}$ have all the form

   $$f_i(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{\ell-1}) = x_{(i+1) \bmod n} + g_i(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{\ell-1})$$

   with $\mathrm{supp}(g_i) \cap \{x_\alpha, \ldots, x_\beta\} = \emptyset$. That is, these feedback functions depend on the values at indices in $[\alpha, \beta]$ but only in the sense that these values are shifted.

2. The remaining feedback functions $f_0, \ldots, f_{\alpha-2}, f_\beta, \ldots, f_{n-1}$ and the output function Out are completely independent of the values at indices $[\alpha, \beta]$, that is $\mathrm{supp}(f_i) \cap \{x_\alpha, \ldots, x_\beta\} = \emptyset$ for all $i \in [n-1] \setminus [\alpha, \beta]$.

We now describe and prove a basic preserving transformation where an external bit stream $(b_t)_{t\geq 0}$ is integrated into some update functions:

Theorem 1 (Preserving FSR Transformation)**.** *Consider an FSR* FSR_E *with update mapping* $F = (f_0, \ldots, f_{n-1})$ *and an external source producing a bitstream* $(b_t)_{t\geq 0}$*. Let* $[\alpha, \ldots, \beta]$ *be an interval which is isolated with respect to F. We define an FSR* FSR_E' *with update mapping* $F' = (f'_0, \ldots, f'_{n-1})$ *which is derived from F as follows:*

$$f'_{\alpha-1} := f_{\alpha-1} + y_{\beta-\alpha}, \quad f'_\beta := f_\beta + y_1, \quad f'_i := f_i \text{ for all } i \neq \alpha, \beta$$

Then both FSRs are equivalent, i.e, $\mathsf{FSR}_\mathsf{E} \equiv \mathsf{FSR}_\mathsf{E}'$.

Proof of Theorem 1. We have to show that for any initial state $St_0 \in \mathbb{F}^n$ there exists a corresponding initial state $St'_0 \in \mathbb{F}^n$ such that the output sequences $(St_t[0])_{t\geq 0}$ and $(St'_t[0])_{t\geq 0}$ are equal for any assignment of $(b_t)_{t\geq 0}$. This is an immediate consequence of the following claim:

Claim: We define for each $t \geq 0$ the vector

$$\Delta_t := (0^{\alpha-1}, b_{t+\beta-\alpha}, b_{t+\beta-\alpha-1}, \ldots, b_{t+1}, b_t, 0^{n-\beta})$$

where 0^r denotes r-times the value zero. Let $St_0 \in \mathbb{F}^n$ be an arbitrary initial state and define $St'_0 := St_0 + \Delta_0$. Then it holds that for each clock t that $St_t + St'_t = \Delta_t$.

It follows from the claim that $St_t[0] + St'_t[0] = \Delta_t[0] = 0$ for each clock if $St_0 + St'_0 = \Delta_0$. That is both produce the same output bitstream which proves the theorem.

It remains to prove the claim which we do by induction. For $t = 0$, the claim holds by definition of St'_0. Next assume that $St_t + St'_t = \Delta_t$ for some clock $t \geq 0$. Recall that by definition of isolated intervals (see Definition 4) that the functions g_i for $i \in [\alpha-1, \beta-1]$ and f_i for $i \notin [\alpha-1, \beta-1]$ are all independent of the variables $x_\alpha, \ldots, x_\beta$. Moreover, we have $St_t[i] = St'_t[i]$ for all $i \notin [\alpha-1, \beta-1]$. Hence, we have $g_i(St_t) = g_i(St'_t)$ for $i \in [\alpha-1, \beta-1]$ and $f_i(St_t, b_t, \ldots, b_{t+\ell-1}) = f_i(St'_t, b_t, \ldots, b_{t+\ell-1})$ for $i \notin [\alpha-1, \beta-1]$. We investigate now the difference $St_{t+1} + St'_{t+1}$ index by index.

For $i \in [0, \ldots, \alpha-2, \beta+1, \ldots, n-1]$, we have

$$\begin{aligned} St_{t+1}[i] &= f_i(St_t, b_t, \ldots, b_{t+\ell-1}) = f_i(St'_t, b_t, \ldots, b_{t+\ell-1}) \\ &= f'_i(St'_t, b_t, \ldots, b_{t+\ell-1}) = St'_{t+1}[i], \end{aligned}$$

showing the zeros at the beginning and the end of $St_{t+1} + St'_{t+1}$.

For $i = \alpha, \ldots, \beta-1$, it holds

$$\begin{aligned} St_{t+1}[i] &= f_i(St_t, b_t, \ldots, b_{t+\ell-1}) = St_t[i+1] + g_i(St_t, b_t, \ldots, b_{t+\ell-1}) \\ &= St'_t[i+1] + \Delta_t[i+1] + g_i(St'_t, b_t, \ldots, b_{t+\ell-1}) \\ &= St'_t[i+1] + g_i(St'_t, b_t, \ldots, b_{t+\ell-1}) + \Delta_t[i+1] = St'_{t+1}[i] + \Delta_{t+1}[i]. \end{aligned}$$

In the last equation we made use of the fact that by definition it holds that $\Delta_t[i] = b_{t+\beta-i}$ for all $i = \alpha, \ldots, \beta$ and hence $\Delta_t[i+1] = \Delta_{t+1}[i]$ for $i = \alpha, \ldots, \beta-1$.

For $i = \alpha - 1$, we have

$$\begin{aligned} St_{t+1}[\alpha-1] &= f_{\alpha-1}(St_t, b_t, \ldots, b_{t+\ell-1}) = St_t[\alpha] + g_{\alpha-1}(St_t, b_t, \ldots, b_{t+\ell-1}) \\ &= St'_t[\alpha] + \Delta_t[\alpha] + g_{\alpha-1}(St'_t, b_t, \ldots, b_{t+\ell-1}) \\ &= St'_t[\alpha] + g_{\alpha-1}(St'_t, b_t, \ldots, b_{t+\ell-1}) + y_{t+\beta-\alpha} \\ &= f'_{\alpha-1}(St'_t, b_t, \ldots, b_{t+\ell-1}) = St'_{t+1}[\alpha-1]. \end{aligned}$$

Finally, it holds that

$$\begin{aligned} St_{t+1}[\beta] &= f_\beta(St_t) = f_\beta(St'_t, b_t, \ldots, b_{t+\ell-1}) = f_\beta(St'_t, b_t, \ldots, b_{t+\ell-1}) + b_{t+1} + b_{t+1} \\ &= f'_\beta(St'_t, b_t, \ldots, b_{t+\ell-1}) + b_{t+1} = St'_{t+1}[\beta] + b_{t+1}. \end{aligned}$$

Hence $St_{t+1}[\beta] + St'_{t+1}[\beta] = b_{t+1}$ which concludes the claim. □

Summing up Theorem 1 ensures that XORing an external value to the state entry which marks the beginning of an isolated interval preserves the FSR as long as this value is cancelled out later before it "leaves" this interval. In practice this requires to have access to the same value at two different clocks. Certainly, a simple solution would be to store this value in some external memory until it is not longer needed but this would increase the hardware area. Hence, in practice, it is preferable to use values which can be reconstructed several clocks later without the need for additional memory. To this end, we introduce the notion of *sustainability*:

Definition 5 (Function with Sustainable Output)**.** Consider an FSR-based stream cipher, being composed of an FSR $\mathsf{FSR_E}$ of length n with update mapping $F = (f_0, \ldots, f_{n-1})$, an external block with bit stream $(b_t)_{t \geq 0}$, and a function $\mathsf{Out}(x_0, \ldots, x_{n-1}, y_0, \ldots, y_\ell)$. We say that Out produces values which are *r-sustainable* if there exists a supplemental function $\mathsf{Out}^*(x_0, \ldots, x_{n-1}, y_0, \ldots, y_\ell)$ such that

$$\mathsf{Out}(St_t, b_t, \ldots, b_{t+\ell-1}) = \mathsf{Out}^*(St_{t+r}, b_{t+r}, \ldots, b_{t+r+\ell-1}) \quad \forall t \geq 0.$$

Remark 1. Informally the definition means that the output of Out at some clock t can likewise be computed r clocks later by Out^* without requiring additional storage. Although it may seem like an artificial and strong assumption at the first sight, it is in fact quite often naturally given for FSRs in Fibonacci configuration: as soon as a new state entry is computed, it is only shifted until it forms the output. In particular it remains part of the state for $n-1$ clocks.

We can now extend the transformation considered in Theorem 1 by replacing the external bits b_t by the outputs of a function that produces $(\beta - \alpha)$-sustainable outputs:

Lemma 1 (Preserving FSR Transformation based on Sustainable Functions)**.** *newline Consider a FSR-based stream cipher, being composed of an FSR $\mathsf{FSR_E}$ of length n with update mapping $F = (f_0, \ldots, f_{n-1})$, an external block with bitstream $(b_t)_{t \geq 0}$, and an output function $\mathsf{Out}(x_0, \ldots, x_{n-1}, y_0, \ldots, y_\ell)$. Let $[\alpha, \ldots, \beta]$ be an interval which is isolated with respect to F and Out. Moreover assume that Out produces $(\beta - \alpha)$-sustainable outputs with Out^* being the corresponding supplementary function. We define an FSR $\mathsf{FSR_E}'$ with update mapping $F' = (f'_0, \ldots, f'_{n-1})$ which is derived from F as follows:*

$$f'_{\alpha-1} := f_{\alpha-1} + \mathsf{Out}^*, \quad f'_\beta := f_\beta + \mathsf{Out}, \quad f'_i := f_i \text{ for all } i \neq \alpha - 1, \beta$$

Then both FSRs are equivalent, i.e, $\mathsf{FSR_E} \equiv \mathsf{FSR_E}'$.

Proof of Lemma 1. We define $\tilde{b}_t := \mathsf{Out}(St_t, b_t, \ldots, b_{t+\ell-1})$ for each $t \geq 0$. Assume for a moment that $(\tilde{b}_t)_{t \geq 0}$ is an independent from the FSR and $(b_t)_{t \geq 0}$. We define an FSR $\mathsf{FSR_E}''$ with update mapping $F'' = (f''_0, \ldots, f''_{n-1})$ which is derived from F as follows:

$$f''_{\alpha-1} := f_{\alpha-1} + \tilde{y}_{\beta-\alpha}, \quad f''_\beta := f_\beta + \tilde{y}_1, \quad f''_i := f_i \text{ for all } i \neq \alpha - 1, \beta$$

Here, $\tilde{y}_i$ represents a variable which takes the value $\tilde{b}_{t+i}$ at each clock t. It follows directly from Theorem 1 that $\mathsf{FSR_E}$ and $\mathsf{FSR_E}''$ are equivalent. Moreover the proof shows that at each clock $t \geq 1$, the state St_t and S''_t of $\mathsf{FSR_E}$ and $\mathsf{FSR_E}''$, respectively, differ by

$$\Delta_t := (0^{\alpha-1}, \tilde{b}_{t+\beta-\alpha}, \tilde{b}_{t+\beta-\alpha-1}, \ldots, \tilde{b}_{t+1}, \tilde{b}_t, 0^{n-\beta}).$$

if the initial states differ by Δ_0 (what we assume in the following). As the interval $[\alpha, \dots, \beta]$ is isolated with respect to Out by assumption, hence, $\mathsf{Out}(St_t, b_t, \dots, b_{t+\ell-1}) = \mathsf{Out}(S''_t, b_t, \dots, b_{t+\ell-1})$ and $\mathsf{Out}^*(St_{t+r}, b_{t+r}, \dots, b_{t+r+\ell-1}) = \mathsf{Out}^*(S''_{t+r}, b_{t+r}, \dots, b_{t+r+\ell-1})$ for $r = \beta - \alpha$. Therefore we can rephrase the definitions of the update functions $f''_{\alpha-1}$ and $f'_\beta := f_\beta + \mathsf{Out}$ by $f''_{\alpha-1} := f_{\alpha-1}$ and $f''_\beta := f_\beta + \mathsf{Out}$, respectively. As this results into the exact definition of $\mathsf{FSR_E}'$, we have $\mathsf{FSR_E}' = \mathsf{FSR_E}''$ and in particular $\mathsf{FSR_E} \equiv \mathsf{FSR_E}'$, showing the claim. □

4.2.4 A Preserving Cipher-Transformation

4.2.4.1 Technical Description

We now present the proposed preserving cipher transformation. The idea is to identify appropriate terms of the output function and to integrate them into the update functions of the FSR. This way the delay of the output function can be decreased without any (or very small) increase of the delay of the FSR. More precisely the transformation is as follows:

Theorem 2 (Preserving Cipher Transformation). *Consider a FSR-based stream cipher $\mathcal{SC}$, being composed of an FSR $\mathsf{FSR_E}$ of length n with update mapping $F = (f_0, \dots, f_{n-1})$, an external block with bit stream $(b_t)_{t \geq 0}$, and an output function $\mathsf{Out}(x_0, \dots, x_{n-1}, y_0, \dots, y_\ell)$. Assume that* Out *can be written as*

$$\begin{aligned} \mathsf{Out}(x_0, \dots, x_{n-1}, y_0, \dots, y_\ell) \quad = \quad & x_\beta + \mathsf{Out}_1(x_0, \dots, x_{n-1}, y_0, \dots, y_\ell) + \\ & \mathsf{Out}_2(x_0, \dots, x_{n-1}, y_0, \dots, y_\ell) \end{aligned} \tag{4.3}$$

such that the outputs of Out_1 *could be computed one clock earlier as well. Formally, this means that there exists a function $g((x_0, \dots, x_{n-1}, y_0, \dots, y_\ell))$ such that it holds for all clocks $t \geq 1$:*

$$\mathsf{Out}(St_t, b_t, \dots, b_{t+\ell-1}) = St_t[\beta] + g(St_{t-1}, b_{t-1}, \dots, b_{t+\ell-2}) + \mathsf{Out}_2(St_t, b_t, \dots, b_{t+\ell-1})$$

Moreover, the following conditions need to be met:

1. *There exist integers $1 \leq \alpha < \beta < n-1$ such that the interval $[\alpha, \dots, \beta]$ is isolated with respect to F and* Out_2 *and the interval $[\alpha+1, \dots, \beta+1]$ is isolated with respect to g.*

2. *g produces $(\beta - \alpha)$-sustainable outputs with g^* being the corresponding supplementary function.*

A second cipher is defined as $\mathcal{SC}'$ with an FSR $\mathsf{FSR_E}'$ *and an output function* Out' *which are derived from* $\mathsf{FSR_E}$ *and* Out, *respectively. The update mapping $F' = (f'_0, \dots, f'_{n-1})$ of* $\mathsf{FSR_E}'$ *is defined as*

$$f'_{\alpha-1} := f_{\alpha-1} + g^*, \quad f'_\beta := f_\beta + g, \quad f'_i := f_i \text{ for all } i \neq \alpha, \beta$$

and the output function Out' *of* $\mathcal{SC}'$ *as*

$$\mathsf{Out}'(x_0,\ldots,x_{n-1},y_0,\ldots,y_\ell) = x_\beta + \mathsf{Out}_2(x_0,\ldots,x_{n-1},y_0,\ldots,y_\ell).$$

Then both ciphers are equivalent.

Remark 2 (On the Relation between Out_1 and g). As elaborated before, the goal of the transformation is to remove parts of the output function (here Out_1) and to integrate it into the update functions of the FSRs. Here one needs to carefully pay attention to the *order of the computations*. By definition, at each clock t the output function first computes the output and the FSRs are updated afterwards. Hence, when the modified output function is invoked it is necessary that the output of Out_1 is already present in the FSR state. The only possibility is that this computation of Out_1 has been executed at least one clock before, which is accomplished by the function g.

Proof of Theorem 2. We show that for any initial state of $\mathcal{SC}$, there exists a corresponding initial state of $\mathcal{SC}'$ such that both ciphers produce the same keystream.[1] We assume that in both ciphers, the same external block EB is used, producing the same bitstream $(b_t)_{t\geq 0}$. Let St_0 be an arbitrary initial state of $\mathsf{FSR_E}$. We define for each $t \geq 0$ the vector

$$\Delta_t := (0^{\alpha-1}, g(St_{t+r}, b_{t+r},\ldots,b_{t+r+\ell-1}),\ldots,g(St_t,b_t,\ldots,b_{t+\ell-1}),0^{n-\beta})$$

with $r = \beta-\alpha$. Assume now that $\mathsf{FSR_E}'$ is initialized with $St'_0 := St_0 + \Delta_0$. It follows from Lemma 1 (and the arguments used in its proof) that at each clock $t \geq 1$, the states St_t and St'_t of $\mathsf{FSR_E}$ and $\mathsf{FSR_E}'$, respectively, differ by Δ_t. As the interval $[\alpha,\beta]$ is assumed to be isolated with respect to F, and Out_2, and the interval $[\alpha+1,\ldots,\beta+1]$ is isolated with respect to g, it follows that for all t it holds that $f_\beta(St'_t,b_t,\ldots,b_{t+\ell-1}) = f_\beta(St_t,b_t,\ldots,b_{t+\ell-1})$ and likewise for g and Out_2. We compare now the keystream bits $(z_t)_{t\geq 0}$ and $(z'_t)_{t\geq 0}$ of $\mathcal{SC}$ and $\mathcal{SC}'$, respectively. For each $t\geq 0$:

$$\begin{aligned}
z'_t &= \mathsf{Out}'(St'_t,b_t,\ldots,b_{t+\ell-1}) = St'_t[\beta] + \mathsf{Out}_2(St'_t,b_t,\ldots,b_{t+\ell-1})\\
&= f'_\beta(St'_{t-1},b_{t-1},\ldots,b_{t+\ell-2}) + \mathsf{Out}_2(St'_t,b_t,\ldots,b_{t+\ell-1})\\
&= f_\beta(St'_{t-1},b_{t-1},\ldots,b_{t+\ell-2}) + g(St'_{t-1},b_{t-1},\ldots,b_{t+\ell-2}) + \mathsf{Out}_2(St'_t,b_t,\ldots,b_{t+\ell-1})\\
&= f_\beta(St_{t-1},b_{t-1},\ldots,b_{t+\ell-2}) + g(St_{t-1},b_{t-1},\ldots,b_{t+\ell-2}) + \mathsf{Out}_2(St_t,b_t,\ldots,b_{t+\ell-1})\\
&= St_t[\beta] + \mathsf{Out}_1(St_t,b_t,\ldots,b_{t+\ell-1}) + \mathsf{Out}_2(St_t,b_t,\ldots,b_{t+\ell-1}) = z_t.
\end{aligned}$$

This shows that both ciphers produce the same keystream. □

[1] The other direction can be shown analogously.

4.2.4.2 Discussion

On the Preconditions. We shortly argue why one might expect in general that the preconditions of Theorem 2 are fulfilled. In a nutshell these conditions are that the output function contains a linear term x_β such that β represents the endpoint of an interval $[\alpha, \beta]$ which is isolated with respect to update functions f_i with $i \notin [\alpha, \beta]$ and the remainder of the output function. First observe that already isolated intervals of length 1, i.e., where $\beta = \alpha$, are sufficient for "outsourcing" Out_1 to the FSR. Second ciphers often deploy FSRs in Fibonacci configuration, meaning that all but one update functions are simple (i.e., only shift a value) and in particular depend on one value only. Moreover these FSRs have high length (for security reasons) and the only non-simple update function is sparse (for efficiency reasons). Thus we found several examples where our approach could be applied, Grain-128 being one of them (cf. section 4.2.5). For example in the case of Grain-128 the relation between Out_1 and g (see Remark 2) is only a shift in the indices of the variables.

Analysis. In the following, we analyze the change of the delay induced by our transformation. As the exact delay strongly depends on the technology used, we discuss from a qualitative point of view how the delay of the cipher depends on the cipher components and under which conditions the transformation increases of the throughput. As our approach considers transformations in the output function and the FSR, no gain can be expected if the critical path goes through the external block EB. Hence we exclude this case in the following and assume that the delay of EB is always smaller than the delay of the output function and of the FSR, i.e, we ignore the delay of EB for simplicity. In fact all expressions can be easily adapted to take $\mathsf{Del}(\mathsf{EB})$ into account as well. Observe in this context that if the cipher contains more than one FSR, we can apply the transformation with respect to that FSR which is the most appropriate and consider the others as part of the external block.

For the analysis, we distinguish between two different cases. We start with the simpler mode A where the update of $\mathsf{FSR_E}$ is independent of the output of Out. This implies both two components can operate in parallel and hence $\mathsf{Del}(\mathcal{SC}) = \max\{\mathsf{Del}(\mathsf{FSR_E}), \mathsf{Del}(\mathsf{Out})\}$. As the transformation aims for decreasing $\mathsf{Del}(\mathsf{Out})$, we restrict to the case of $\mathsf{Del}(\mathsf{Out}) > \mathsf{Del}(\mathsf{FSR_E})$ as otherwise the transformation would not bring any gain. Let Out' and $\mathsf{FSR_E}'$ denote the output function and the FSR, respectively, after the transformation.

To apply the transformation explained in Theorem 2, it is necessary that Out can be split accordingly, i.e., $\mathsf{Out} = x_\beta + \mathsf{Out}_1 + \mathsf{Out}_2$. Observe that after the transformation the output function becomes smaller, i.e., $\mathsf{Out}' = x_\beta + \mathsf{Out}_2$ and therefore likely has smaller delay. In general we have

$$\mathsf{Del}(\mathsf{Out}') = \mathsf{Del}(x_\beta + \mathsf{Out}_2) \leq \mathsf{Del}(x_\beta + \mathsf{Out}_1 + \mathsf{Out}_2) = \mathsf{Del}(\mathsf{Out}).$$

While the concrete delay of a function depends on the deployed technology, a good approximation is the delay of a depth-optimal tree of 2-input AND and 2-input XOR gates implementing this function. In particular terms with the highest algebraic degree tend to induce the biggest delay. Hence preferable choices of Out_1 should contain terms with high algebraic degree.

An FSR of length n is composed of n stages and update functions $f_0, \ldots, f_{n-1}$ which are implemented by flip-flops and logical gates, respectively. The update functions are computed simultaneously but at each clock-cycle, the two steps have to be performed sequentially:

- the values of the update functions have to be computed
- the stages have to be updated with the new values

Therefore the delay of a FSR is the sum of the delay of a flip-flop Del_{fl} and of the delay of the slowest of the update functions:

$$\begin{aligned}\mathsf{Del}(\mathsf{FSR_E}) &= \max\{\mathsf{Del}(f_0), \mathsf{Del}(f_1) \cdots \mathsf{Del}(f_{n-1})\} + \mathsf{Del}_{fl} \\ &= \mathsf{Del}(f_\mu) + \mathsf{Del}_{fl} \end{aligned} \tag{4.4}$$

if μ denote the index of the slowest update function. Observe that the update functions within the isolated interval $[\alpha, \beta]$ are simply the shift operators. Hence we can assume that $\mu \notin [\alpha, \beta]$.[2] Recall that the transformation only changes the update functions f_α and f_β. Hence it follows

$$\mathsf{Del}(\mathsf{FSR_E}') = \max\{\mathsf{Del}(f_\mu), \mathsf{Del}(f_\alpha + g^*), \mathsf{Del}(f_\beta + g)\} + \mathsf{Del}_{fl}.$$

Because of $\mathsf{Del}(\mathcal{SC}') = \max\{\mathsf{Del}(\mathsf{FSR_E}'), \mathsf{Del}(\mathsf{Out}')\}$ and $\mathsf{Del}(f_\mu) + \mathsf{Del}_{fl} < \mathsf{Del}(\mathsf{Out})$, the transformation decreases the overall delay if

$$\max\left\{\begin{array}{l}\mathsf{Del}(x_\beta + \mathsf{Out}_2), \\ \mathsf{Del}(f_\alpha + g^*) + \mathsf{Del}_{fl}, \\ \mathsf{Del}(f_\beta + g) + \mathsf{Del}_{fl}\end{array}\right\} < \mathsf{Del}(x_\beta + \mathsf{Out}_1 + \mathsf{Out}_2). \tag{4.5}$$

Next we consider mode B where some of the update functions of $\mathsf{FSR_E}$ depend on the output of Out. We denote by F^{Out} the part of the circuit that implements $\mathsf{FSR_E}$ which requires the output of Out as input and by $F \setminus F^{\mathsf{Out}}$ the remaining part of the $\mathsf{FSR_E}$ circuit. In this case it implies that

$$\mathsf{Del}(\mathcal{SC}) = \max\{\mathsf{Del}(F^{\mathsf{Out}}) + \mathsf{Del}(\mathsf{Out}), \mathsf{Del}(F \setminus F^{\mathsf{Out}})\}.$$

Again if $\mathsf{Del}(F \setminus F^{\mathsf{Out}}) \geq \mathsf{Del}(F^{\mathsf{Out}}) + \mathsf{Del}(\mathsf{Out})$, no gain can be expected by shifting computations from Out to the FSR. Hence we restrict to the case that $\mathsf{Del}(F \setminus F^{\mathsf{Out}}) <$

[2]Otherwise the whole FSR would consists only of a cyclic shift of its internal state, rendering it useless for cryptographic purposes.

$\mathsf{Del}(F^{\mathsf{Out}}) + \mathsf{Del}(\mathsf{Out})$. Recall that by definition, the update functions within the isolated interval simply shift the preceding state entry. Hence, we can assume that $f_\alpha, f_\beta \notin F^{\mathsf{Out}}$. In particular, $\mathsf{Del}(F^{\mathsf{Out}})$ will not change after the transformation. Analogouesly to mode A, we define by μ the index of the slowest update function in $F \setminus F^{\mathsf{Out}}$. Thus, after the transformation, the delay in the set of update functions which are independent of Out is given by $\max\{\mathsf{Del}(f_\alpha + g^*), \mathsf{Del}(f_\beta + g)\}, \mathsf{Del}(f_\mu)\}$. An increase of the delay in this set is tolerable as long as it stays below $\mathsf{Del}(\mathcal{SC}) = \mathsf{Del}(F^{\mathsf{Out}}) + \mathsf{Del}(\mathsf{FSR_E})$ which is given for $\mathsf{Del}(f_\mu)$. A further condition for an improvement is (as in mode A) that the delay of the resulting output function Out' is indeed less than the delay of the original output function Out. Taking both conditions together yields the following condition for an improvement of the throughput:

$$\max \left\{ \begin{array}{l} \mathsf{Del}(f_\alpha + g^*) + \mathsf{Del}_{fl} - \mathsf{Del}(F^{\mathsf{Out}}), \\ \mathsf{Del}(f_\beta + g) + \mathsf{Del}_{fl} - \mathsf{Del}(F^{\mathsf{Out}}), \\ \mathsf{Del}(x_\beta + \mathsf{Out}_2)\} \end{array} \right\} < \mathsf{Del}(\mathsf{Out}). \tag{4.6}$$

Observe that if none of the update functions depend on Out, it holds that $F^{\mathsf{Out}} = \emptyset$. Then, Eq. 4.6 simplifies to Eq. 4.5 and we are back in mode A.

We want to stress that our approach is applicable in both modes while it is stated in [MD13] that the pipelining method cannot be used in mode B (at least for Grain-128a). However, the resulting $\mathcal{SC}'$ requires the additional computation of g^* (compared to the original cipher $\mathcal{SC}$) which may induce a (preferably small) increase in the area size.

4.2.5 Application to Grain-128 and Grain-128a

In this section, we demonstrated our technique by applying it to the stream ciphers Grain-128 [HJMM06] presented in [AM14] and Grain 128a [ÅHJM11] provided in [AM]. Both ciphers have a very similar structure with the main difference in the update and output functions. Since all the steps of our technique are similar in both cases, for the sake of simplicity and in order to avoid overloading this work with technical details, we focus on one cipher, namely Grain-128, and use it as an example to demonstrate the application of the transformation. However, we present the implementation results for both ciphers in Subsubsection 4.2.5.7.

Grain-128 consists of an 128-bit LFSR, an 128-bit NLFSR, and an output function Out. In the original description of Grain-128 both FSRs are used in Fibonacci configuration, meaning that all bits except the 127th are updated just by shifting the adjacent value. The concrete updates and the output function are given in Subsubsection 4.2.5.4. Similar to most of the existing stream ciphers Grain-128 uses two different modes: initialization and keystream generation. During the initializing mode the cipher does not produce any output for 256 clock-cycles. Instead the outputs of Out are fed back to the LFSR and NLFSR. In the keystream generation mode, the result of Out forms the output.

4.2.5.1 Setup.

To the best of our knowledge, the previously most efficient implementation of Grain-128 with the currently highest throughput was given in [MD10]. Following [MD10] we used the Cadence RTL Compiler[3] for synthesis and simulation. Two implementations with different compiler settings were examined: optimizing the output for timing and optimizing for area size, respectively. It is well known that changing the compiler setting can lead to unpredictable effects. For example, although our transformation includes additional computations, the area size of the time-optimized solution *reduced* from 1794 GE to 1748 GE. We assume the following reason: when the compiler is set to optimize timing, bigger functions are implemented by gates which consume more area. It seems that such tricks are not necessary anymore (or at least to a lesser extent) after our transformation. In other words, routing of the gates becomes easier for the transformed version of the cipher which outweights the slightly increased logic count. To minimize such effects as far as possible and to get a preferably unbiased view on the results of our transformation, our implementation contains only these blocks that are affected by the transformation. For example, we excluded on purpose the counter that is used in the initialization mode for counting the 256 cycles.

4.2.5.2 Preparation.

Recall that our transformation aims for reducing the delay of the output function. Unfortunately, in the original specification of Grain-128 the critical path goes through the FSRs. Hence, before we applied our transformation, we modified the FSRs to decrease their delays. More precisely, we changed the configurations of the FSRs from Fibonacci to Galois. The idea is to spread the monomials of one update function amongst the others, in order to make them being computed in parallel. The new update functions together with initial state mappings are given in Appendix 4.2.5.5. This transformation increased the maximal frequency from 1.03 GHz to 1.11 GHz in the initialization mode (approx. +8%), from 1.29 GHz to 1.45 GHz in the keystream generation mode (approx. +12%) for the timed-optimized solution. For the area-optimized solution the improvement is from 0,42 GHz to 0,60 GHz by 42 % in the initialization mode and from 0,89 GHz to 0,90 GHz in the keystream generation mode. In particular the transformation led to the critical path going through the output function, making our approach applicable. Of course we used this configuration as the benchmark for estimating the gain of our transformation.

4.2.5.3 Transformation.

For getting the preferably best results from the transformations, we used the following strategy. Originally, the output function Out contains several linear and quadratic terms

[3]See http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx

and one cubic term (degree 3). As the terms with the highest degree tend to induce the highest delay, we aimed for shifting all monomials with degree bigger than 1 to the FSRs. The exact transformation is explained in App. 4.2.5.6. Observe that we took both FSRs into account, i.e., some monomials have been integrated into the LFSR and others into the NLFSR. As both FSRs deploy update functions of degree 2 or less, there was no increase in the delay of the FSRs when the quadratic monomials from Out have been integrated. For the single cubic term of Out, the situation is different as this may increase the delay of the FSRs. Therefore we implemented both variants, i.e., the cubic term remaining in Out or being moved to the FSRs, and it turned out that moving the term yielded the better result.

4.2.5.4 Specification of Grain-128

Grain-128 consists of an 128-bit LFSR L with update mappings $F = (f_0, \dots, f_{127})$, an 128-bit NLFSR N with update mappings $Q = (q_0, \dots, q_{127})$, and an output function Out. In the original description of Grain-128 both FSRs are used in Fibonacci configuration, meaning that all bits except the 127th are updated just by shifting the adjacent value. We denote at clock t the state of the LFSR to be $A_t = (A_t[0], \cdots, A_t[127])$ and the state of the NLFSR as $B_t = (B_t[0], \cdots, B_t[127])$. The updates of L and N are as follows:

$$\begin{aligned}
A_{t+1}[i] &= A_t[i+1] \text{ and } B_{t+1}[i] = B_t[i+1] \quad \text{for } i = 0, \dots, 126 \\
A_{t+1}[127] &= A_t[0] + A_t[7] + A_t[38] + A_t[70] + A_t[81] + A_t[96] \\
B_{t+1}[127] &= A_t[0] + B_t[0] + B_t[26] + B_t[56] + B_t[91] + B_t[96] + B_t[3]B_t[67] + \\
&\quad B_t[11]B_t[13] + B_t[17]B_t[18] + B_t[27]B_t[59] + B_t[40]B_t[48] + \\
&\quad B_t[61]B_t[65] + B_t[68]B_t[84]
\end{aligned}$$

The output function Out of Grain-128 is:

$$\begin{aligned}
\mathsf{Out} &= B_t[2] + B_t[15] + B_t[36] + B_t[45] + B_t[64] + B_t[73] + A_t[93] + B_t[89] + \\
&\quad B_t[12]A_t[8] + A_t[13]A_t[20] + B_t[95]A_t[42] + A_t[60]A_t[79] + \\
&\quad B_t[12]B_t[95]A_t[95]
\end{aligned}$$

4.2.5.5 Specification of the Change of the FSR Configurations

We specify the update functions of the FSRs after changing the configuration from Fibonacci to Galois. To distinguish the original update functions from the update functions after the change, we use the upper index (^{g}). That is $(f_0, \dots, f_{127})$ and $(q_0, \dots, q_{127})$ denote the original update functions of the LFSR L and the NLFSR N, respectively, while $(f_0^g, \dots, f_{127}^g)$ and $(q_0^g, \dots, q_{127}^g)$ refer to the update functions after changing the configuration. Likewise we denote at clock t the state of the LFSR to be $A_t^g = (A_t^g[0], \cdots, A_t^g[127])$ and the state of the NLFSR as $B_t^g = (B_t^g[0], \cdots, B_t^g[127])$. The update functions of the

Table 4.1: The update functions of the FSRs after yransforming into Galois configuration

LFSR L^g:

$$\begin{array}{ll}
f^g_{127} = A^g_t[0] & f^g_{111} = A^g_t[112] + A^g_t[80] \\
f^g_{123} = A^g_t[124] + A^g_t[3] & f^g_{103} = A^g_t[104] + A^g_t[46] \\
f^g_{119} = A^g_t[120] + A^g_t[30] & f^g_{97} = A^g_t[98] + A^g_t[51] \\
q^g_j = B^g_t[j+1], 0 \leq j \leq 127, & j \notin \{127, 123, 119, 111, 103, 97\}
\end{array}$$

NLFSR N^g:

$$\begin{array}{ll}
q^g_{127} = A^g_t[0] + B^g_t[0] & q^g_{113} = B^g_t[114] + B^g_t[77] \\
q^g_{125} = B^g_t[126] + B^g_t[1]B^g_t[65] & q^g_{111} = B^g_t[112] + B^g_t[80] \\
q^g_{123} = B^g_t[124] + B^g_t[7]B^g_t[9] & q^g_{100} = B^g_t[101] + B^g_t[34]B^g_t[38] \\
q^g_{121} = B^g_t[122] + B^g_t[20] & q^g_{99} = B^g_t[100] + B^g_t[40]B^g_t[56] \\
q^g_{119} = B^g_t[120] + B^g_t[9]B^g_t[10] & q^g_{98} = B^g_t[99] + B^g_t[11]B^g_t[19] \\
q^g_{117} = B^g_t[118] + B^g_t[17]B^g_t[49] & q^g_{97} = B^g_t[98] + B^g_t[26] \\
q^g_i = B^g_t[i+1], 0 \leq i \leq 127, & i \notin \{127, 125, 123, 121, 119, 117, 113, 111, 100, 99, 98, 97\}
\end{array}$$

FSRs of Grain-128 in Galois configuration are given in the Table 4.1. In order to get the same output after this transformation as in original Grain-128, the initial state has to be changed. A general treatment of this topic can be found in [Dub10]. For our configuration the initial state needs to be changed as follows.

$$\begin{array}{rcl}
A^g_0[i] & = & A_0[i], 0 \leq i \leq 97 \\
A^g_0[i] & = & A_0[i] + f^g_{i-1}(A_0) + f^g_{i-2|_{+1}}(A_0) + \cdots + f^g_{97|_{+i-98}}(A_0), 98 \leq i \leq 127 \\
B^g_0[j] & = & B_0[j], 0 \leq j \leq 97 \\
B^g_0[j] & = & B_0[j] + q^g_{j-1}(B_0) + q^g_{j-2|_{+1}}(B_0) + \cdots + q^g_{97|_{+j-98}}(B_0), 98 \leq j \leq 127
\end{array}$$

were $q^f_{i|_{+k}}$ and $f^f_{i|_{+k}}$ denote that every index in the arguments of the functions $q^f_{i|_{+k}}$ and $f^f_{i|_{+k}}$ is increased by k.

For example consider the following initial state of Grain-128:

$$\begin{array}{rcl}
A_0 & = & (1010010111100011110011000101000110010111011101111100110100011001 \\
 & & 1010011001110100110101000111000100101001011100011100011001001100) \\
B_0 & = & (1100100100010001001011111100110000010111000100101101011011101101 \\
 & & 1010100010101100101001111011110101101001101100100001101101100001)
\end{array}$$

Table 4.2: The update and output functions after our transformation

$q^T_{89} = B^T_t[90] + B^T_t[3]$	$q^T_{87} = B^T_t[88] + B^T_t[1]$
$q^T_{73} = B^T_t[74] + B^T_t[13]A^T_t[9]$	$q^T_{71} = B^T_t[72] + B^T_t[11]A^T_t[7]$
$q^T_{64} = B^T_t[65] + A^T_t[14]A^T_t[21]$	$q^T_{62} = B^T_t[63] + A^T_t[12]A^T_t[19]$
$q^T_{36} = B^T_t[37] + B^T_t[96]A^T_t[43]$	$q^T_{34} = B^T_t[35] + B^T_t[94]A^T_t[41]$
$q^T_{15} = B^T_t[16] + B^T_t[13]B^T_t[96]A^T_t[96]$	$q^T_{13} = B^T_t[14] + B^T_t[11]B^T_t[94]A^T_t[94]$
$f^T_{93} = A^T_t[94] + A^T_t[61]A^T_t[80]$	$f^T_{91} = A^T_t[92] + A^T_t[59]A^T_t[78]$
$\mathrm{Out}^T = B^T_t[15] + B^T_t[36] + B^T_t[45] + B^T_t[64] + B^T_t[73] + B^T_t[89] + A^T_t[93]$	

Table 4.3: Mapping of the initial states after our transformation

$B^T_0[89] = B^g_0[89] + B^g_0[2]$	$B^T_0[88] = B^g_0[88] + B^g_0[1]$
$B^T_0[73] = B^g_0[73] + B^g_0[12]A^g_0[8]$	$B^T_0[72] = B^g_0[72] + B^g_0[11]A^g_0[7]$
$B^T_0[64] = B^g_0[64] + A^g_0[13]A^g_0[20]$	$B^T_0[63] = B^g_0[63] + A^g_0[12]A^g_0[19]$
$B^T_0[36] = B^g_0[36] + B^g_0[95]A^g_0[42]$	$B^T_0[35] = B^g_0[35] + B^g_0[94]A^g_0[41]$
$B^T_0[15] = B^g_0[15] + B^g_0[12]B^g_0[95]A^g_0[95]$	$B^T_0[14] = B^g_0[14] + B^g_0[11]B^g_0[94]A^g_0[94]$
$A^T_0[93] = B^g_0[93] + A^g_0[60]A^g_0[79]$	$A^T_0[92] = B^g_0[92] + A^g_0[59]A^g_0[78]$

Then the corresponding initial states after transformation to Galois configuration would be:

$$
\begin{aligned}
A^g_0 &= (101001011110001111001100010100011001011101110111110011010001100 1\\
&\quad 1010011001110100110101000111000100110011101100010001100011011010)\\
B^g_0 &= (110010010001000100101111110011000001011100010010110101101110110 1\\
&\quad 1010100010101100101001111011110101100110100000011000110100111111)
\end{aligned}
$$

4.2.5.6 Specification of Our Transformation

We use the upper index (^{T}) to indicate that the FSR states and the update function correspond to the configuration after our transformation is done. The exact transformations are provided in the Table 4.2. All the other update functions are the same as in the configuration explained in App. 4.2.5.5. Observe that the modified output function Out^T is linear as opposed to the cubic output function Out of original Grain-128.

In Tab. 4.3, we provide the concrete mapping between initial states of FSRs before and after the transformation and before it, which is necessary to get the same output. All the other initial state entries are also the same as in the previous configuration.

We would like to remark that the same transformation was applied to Grain-128a

[ÅHJM11] which is a recent improvement of Grain-128. The concrete transformation details are not included as they are similar to the case of Grain 128. We refer to [AM] for more details.

4.2.5.7 Results and Discussion.

Table 4.4: The performance results for Grain-128 [AM14] and Grain-128a[AM]

Grain-128						
Config.	Area-opimizing			Time-optimizing		
	Initialization		Keystream gen.	Initialization		Keystream gen.
	Area size*	Througput**	Througput	Area size	Througput	Througput
Original	1626	0,42	0,89	1853	1,03	1,29
Galois	1627	0,60 (+42%)	0,90	1794	1,11 (+8 %)	1,45 (+12 %)
Our	1656	0,73 (+20 %)	1,06 (+18 %)	1748	1,31 (+18 %)	1,8 (+24 %)

Grain-128a						
Config.	Area-opimizing			Time-optimizing		
	Initialization		Keystream gen.	Initialization		Keystream gen.
	Area size	Througput	Througput	Area size	Througput	Througput
Original	1640	0,57	0,84	1888	1,02	1,23
Galois	1632	0,61 (+7%)	0,90 (+7%)	1816	1,17 (+15 %)	1,51 (+22 %)
Our	1652	0,73 (+20 %)	1,06 (+18 %)	1736	1,3 (+11 %)	1,78 (+18 %)

* Area size is given in gate equivalents (GE)
** Throuhput is given in gigahertz (GHz)

The results are provided in the table 4.4. We presented a new approach for parallelizing computations in stream ciphers based on feedback shift registers (FSRs). As opposed to the common pipelining technique, existing structures are re-used for avoiding (or at least reducing) an increase of memory. The transformation has been applied to Grain-128 and Grain-128a ciphers, where the throughput for a time-optimized implementation is increased in the initialization mode by 18% and 11% and in the keystream generation mode by 24% and 18% correspondingly. When the compiler was set to optimize the area size the throughput of both ciphers is increased by 20 % in initialization mode and by 18% in the keystream generation mode. As opposed to other solutions, no additional memory is required.

An interesting problem is to automate this approach, i.e., finding an algorithm which automatically finds a (nearly) optimal solution. Our technique is tailored for improving the throughput of FSR-based stream ciphers with a *non-linear* output function by transforming it into a cipher with *linear* output function. Interestingly many recently proposed stream ciphers use a linear output function already in their original configuration. Our transformation may be an indication that when designing FSR-based stream ciphers, it is sufficient to restrict to designs that use a linear output function. In general we expect that the presented technique and theory may be helpful in the design phase already. As the idea is to re-use existing memory, these ideas might be used right from the start for developing new stream ciphers with further decreased hardware size.

4.3 On Lightweight Stream Ciphers with Shorter Internal State

This section is based on the paper [AM15].

4.3.1 Motivation

Stream ciphers usually allow for a higher throughput but require a larger area size compared to block ciphers. The latter is mainly caused by time-memory-data trade-off (TMDTO) attacks which aim to recover the internal state of the stream cipher [Gol97, Bab95, BS00]. The attack effort is in $O(2^{\sigma/2})$, where σ denotes the size of the internal state of a stream cipher. This results into a rule of thump that for achieving κ-bit security level, the size of internal state should be at least $\sigma = 2 \cdot \kappa$. It means that in order to implement such a cipher at least $2 \cdot \kappa$ memory gates are required which is usually the most area and power-consuming resource.

We investigated an extension in the common design for stream ciphers which allows to realize secure lightweight stream cipher with an area size beyond the trade-off attack bound mentioned above.

The core idea is to split the set of internal states into 2^κ equivalence classes such that a TMDTO attack has to consider each of these classes at least once. To achieve this goal, we suggest to involve the key into the update process of the internal state.

Theoretically, the overall approach is still to have a sufficiently large internal state which determines the keystream bits. The main difference though is that part of this state is the secret key itself and not only a state that has been derived from this key. If one considers the case that the key is fixed for the device, one can make use of the fact that storing a fixed key is significantly less area consuming than deploying a register of the same length. In fact, a similar idea has been used in the design of KATAN/KTANTAN [CDK09]. Moreover, the approach may allow for designs where the overall state size is smaller than 2κ.

4.3.2 Trade-off Attacks against KSGs

To explain the idea of the proposed approach let us first take a deeper look into Time-Data-Memory trade-off attacks (see paragraph 2.2.2.1) with respect to their application against keystream generators. In principle, two different approaches can be considered here, depending on what function the attacker aims to invert.

Recovering the Key. The most obvious approach is to invert the whole cipher. That is one considers the process which takes as input a secret key $k \in \mathcal{K} = \mathrm{GF}(2)^\kappa$ and outputs the first κ keystream bits as a function $F_{\mathsf{KSG}} : \mathrm{GF}(2)^\kappa \to \mathrm{GF}(2)^\kappa$. The search space would be $\mathcal{N} = \mathcal{K} = \mathrm{GF}(2)^\kappa$ in this case. As already explained, trade-off attacks for scenario A would require a precomputation time which is equivalent to exhaustive search in the key space. If we say that a security level of κ expresses the requirement that a successful attack requires at least once a time effort in $\mathcal{O}(2^\kappa)$, then such attacks do not represent a specific threat.

Observe that although an attacker may have knowledge of significantly more than κ bits, scenario B trade-off attacks are not applicable here (at least not in general). To see why, let $F^t_{\mathsf{KSG}} : \mathrm{GF}(2)^\kappa \to \mathrm{GF}(2)^\kappa$ be the function that takes as input the secret key and outputs the keystream bits for clocks $t, \ldots, t+\kappa-1$. That is it holds that $F^0_{\mathsf{KSG}} = F_{\mathsf{KSG}}$ from above. Then, the knowledge of $D+\kappa-1$ keystream bits translates to knowing images of $F^0_{\mathsf{KSG}}, \ldots, F^{D-1}_{\mathsf{KSG}}$ and in fact, inverting one of these would be sufficient. However, these functions are all different. In particular, any precomputation done for one of these, e.g., F^i_{KSG}, cannot be used for inverting another one, e.g, F^j_{KSG} with $i \neq j$.

Recovering the Internal State. An alternative approach is to invert the output function Out only, that is used in the keystream generation phase. More precisely, let $F_{\mathsf{Out}} : \mathrm{GF}(2)^\sigma \to \mathrm{GF}(2)^\sigma$ be the function that takes the internal state $St_t \in \mathrm{GF}(2)^\sigma$ at some clock t as input and outputs the σ keystream bits $z_t, \ldots, z_{t+\sigma-1}$. The search space would be $\mathcal{N} = S$. A scenario-A trade-off attack would again require a precomputation time equal to $|\mathcal{N}| = |S|$ which implies that $\sigma \geq \kappa$ if one aims for a security level of κ.

As each keystream segment $z_t, \ldots, z_{t+\sigma-1}$ is an output of the same function F_{Out} and as the knowledge of one internal state St_t allows to compute all succeeding keystreams bits z_r for $r \geq t$ (and as Upd is assumed to be reversible, the preceeding keystream bits as well), scenario B attacks are suitable. As can be seen from table 2.2, each attack would require at least once a time effort of about $\sqrt{|S|} = 2^{\sigma/2}$. This implies the already rule of selecting $\sigma \geq 2\kappa$.

4.3.3 The Proposed Design Approach

In this section, we discuss a conceptually simple adaptation of how keystream generators are commonly designed (see definition 1). The goal is to make stream ciphers more resistant against TMDTO attacks such that shorter internal states can be used. To this end, let us take another look at trade-off attacks. An attacker who is given a part of the keystream aims to find an internal state which allows to compute the remaining keystream. Let $F^{\mathrm{compl.}}_{\mathsf{Out}}$ denote the function that takes as input the initial state and outputs the complete keystream. Here, ”complete” refers to the maximum number of keystream bits that are intended by the designer. If no bound is given, then we simply assume that 2^σ keystream bits are produced as this refers to the maximum possible period. From an attacker's point of view, any internal state that allows for reconstructing the keystream is equally good. This brings us to the notion of keystream-equivalent states:

Definition 6 (Keystream-equivalent States)**.** Consider a KSG with a function $F^{\text{compl.}}_{\mathsf{Out}}$ that outputs the complete keystream. Two states $St, St' \in S$ are said to be *keystream-equivalent* (in short $St \equiv_{\text{kse}} St'$) if there exists an integer $r \geq 0$ such that $F^{\text{compl.}}_{\mathsf{Out}}(\mathsf{Upd}^r(St)) = F^{\text{compl.}}_{\mathsf{Out}}(St')$. Here, Upd^r means the r-times application of Upd.

Observe that keystream-equivalence is an equivalence relation.[4] For any state $St \in S$, we denote by $[St]$ its equivalence class, that is

$$[St] = \{St' \in S | St \equiv_{\text{kse}} St'\}$$

To see why this notion is important for analyzing the effectiveness of a TMDTO attack, let us consider an arbitrary KSG with state space S. As any state is member of exactly one equivalence class, the state space can be divided into ℓ distinct equivalence classes:

$$S = \left[St^{(1)}\right] \dot{\cup} \ldots \dot{\cup} \left[St^{(\ell)}\right]$$

Now assume a TMDTO attacker who is given some keystream (z_t), based on an unknown initial state St_0. Recall that the strategy of a trade-off attack is not to exploit any weaknesses in the concrete design but to efficiently cover a sufficiently large fraction of the search space. In this case if none of the precomputations were done for values in $[St_0]$, the attack cannot be successful unless the online phase searches all equivalence classes that have been ignored during the precomputation phase. This leads to the following observation: a TMDTO attack on the KSG will be a union of TMDTO attacks, one for each equivalence class. That is we have ℓ TMDTO attacks with search spaces $\mathcal{N}_i = \left[St^{(i)}\right]$, $i = 1, \ldots, \ell$, respectively. As each of these attacks has a time effort of at least 1, we get a lower bound of ℓ for the attack effort. Now, if one designs a cipher such that $\ell \geq 2^\kappa$, then one has achieved the required security level against trade-off attacks. This is exactly the idea behind the design approach discussed next.

We are now ready to discuss our proposed design. The basic idea is to achieve a splitting of the internal state space in sufficiently many equivalence classes. To achieve this, we divide the internal state into two parts: a variable part that may change over time and a fixed part. For practical reasons the fixed part will be realized by simply re-using the secret key (more on this later). The main difference to a KSG as given in definition 1, the update function Upd will compute the next variable state from the current variable state *and* the fixed secret key. We call such a construction a KSG with keyed update function, to be defined below. Observe that this definition is in fact covered by definition given in [MVO96].

Definition 7 (Keystream Generator With Keyed Update Function)**.** A keystream generator (KSG) with *keyed update function* comprises three sets, namely

[4]This is due to the fact that for any state $St \in S$, the sequence $(\mathsf{Upd}^r(St))_{r \geq 0}$ is cyclic and that Upd is reversible by assumption.

- the key space $\mathcal{K} = \mathrm{GF}(2)^{\kappa}$,
- the IV space $\mathcal{IV} = \mathrm{GF}(2)^{\nu}$,
- the variable state space $S = \mathrm{GF}(2)^{\sigma}$,

and the following three functions

- an initialization function $\mathsf{Init} : \mathcal{IV} \times \mathcal{K} \to S$
- an update function $\mathsf{Upd} : \mathcal{K} \times S \to S$ such that $\mathsf{Upd}_k : S \to S$, $\mathsf{Upd}_k(St) := \mathsf{Upd}(k, St)$, is bijective for any $k \in \mathcal{K}$, and
- an output function $\mathsf{Out} : S \to \mathrm{GF}(2)$.

The internal state ST is composed of a variable part $St \in S$ and a fixed part $k \in \mathcal{K}$. Initialization and keystream generation work analogously to definition 1 with the only difference that the state update also depends on the fixed secret key.

Let us take a look at the minimum time effort for a TMDTO attack against a KSG with keyed update function. We make in the following the assumption that any two different states $ST = (St, k)$ and $ST' = (St', k')$ with $k \neq k'$ never produce the same keystream, that is $F^{\text{compl.}}_{\mathsf{Out}}(ST) \not\equiv_{\text{kse}} F^{\text{compl.}}_{\mathsf{Out}}(ST')$. Hence, we have at least 2^{κ} different equivalence classes. As the effort grows linearly with the number of equivalence classes, we assume in favor of the attacker that we have exactly 2^{κ} equivalence classes. This gives a minimum time effort of 2^{κ}.

Observe that similar techniques are present in stream cipher modes for block ciphers like OFM or CTR. However, as far as we know it has never been discussed for directly designing stream ciphers with increased resistance against TMDTO-attacks. In this context, we think that this approach has two interesting consequences with respect to saving area size in stream cipher implementations:

1. Apparently one can achieve a security level of κ independent of length σ of the variable state. This allows to use a shorter internal state which directly translates to saving area size.[5]
2. For technical reasons, storing a fixed value (here: the key) can be realized with significantly less area size than is necessary for storing a variable value. This effect has been used for example in the construction of the block cipher KTANTAN ([CDK09]). It allows for further savings compared to KSGs with an register of length $\geq 2\kappa$.

[5]Of course, σ shouldn't be too small. Otherwise, the period of the KSG may become too short and the cipher may also become vulnerable for other attacks like guess-and-determine.

We demonstrate the feasibility of this approach by proposing concrete ciphers named Sprout and Plantlet see section 5.2. Our implementations showed that these ciphers need significantly less area size than existing ones with comparable efficiency and claimed security level.

CHAPTER 5 New Designs

5.1 Chapter Overview

This chapter discusses the new proposed cryptographic solutions.

In Section 5.2 we describe low-area stream ciphers Sprout and Plantlet. These ciphers were developed following the design approach discussed in Section 4.3 which allows to realize secure KSGs with shorter internal state. Both ciphers are based on the Grain[HJMM08] family of ciphers and have similar structure. Plantlet has several improvements which were included in order to resist the attacks found against Sprout. These ciphers use the approach of continuously accessing the non-volatile key discussed in section Section 3.3 in such a way that their throughput is almost unaffected, no matter what type of non-volatile is used. After describing the specifications of the design, we explain the design rationale, provide security analysis and the implementation results. This section is based on the papers [AM15] and [MAM17].

Section 5.3 is devoted to the new low-energy stream cipher Trivium-2 which was designed based on findings provided in Section 3.4. Trivium-2 has almost identical structure with Trivium [CP] but has two times larger internal state size and uses 128-bit keys, whereas the key size of Trivium is 80 bit. The goal of this design was to develop the most energy-efficient cipher among the existing ones with the same level of security. After providing the full specifications of Trivium-2, we describe design rationale and discuss security of the cipher. The implementation results are presented at the end of the section. Trivium-2was developped together with Subhadeep Banik, Frederik Armknecht, Takanori Isobe, Willi Meier, Andrey Bogdanov, YuheiWatanabe and Francesco Regazzoni while working on the paper [BMA$^+$18]. Although the design of Trivium-2 was not included, we plan to publish these results in the full version of the paper.

Section 5.4 presents a data protection scheme which was designed to enhance the security of Low-Power Networks (LPNs). The scheme follows the NIST recommendations and can be embedded into the application level of LPNs in order to guarantee data confidentiality and authenticity. This solution has been deployed on the water distribution network of the City of Antibes in France. The section is based on the paper [MGAM17].

5.2 Low-Area Stream Ciphers Sprout and Plantlet

5.2.1 Overview

In this section we describe two concrete keystream generators with keyed update function, namely Sprout [AM15] and Plantlet[MAM16].

Sprout was first purposed in [AM15] and was designed to demonstrate the feasibility of the approach discussed in Section 4.3 of realizing stream ciphers with shorter internal state by continuously involving the key into the state update during the keystream generation phase. Sprout builds upon the Grain 128a [ÅHJM11] cipher but uses shorter registers. Unfortunately, there were weaknesses mistakenly included in the design which leaded to the serious attacks [Ban15, LN15, EK15, MSBD15, LYR15, ZG15].

Plantlet is an improvement of Sprout which has several security enhancements. In particular, it inherits the overall structure from Sprout depicted in figure 5.1, adopts the continuous key involvement, but at the same time implements fixes for discovered vulnerabilities, e. g., stronger round key function and avoiding the all-zero state.

The way how both ciphers access the key is well-aligned with the different types of NVM as discussed in Section 3.3.

After describing the specifications of the design, we explain the design rationale, provide security analysis and the implementation results. This section is based on the papers [AM15] and [MAM17].

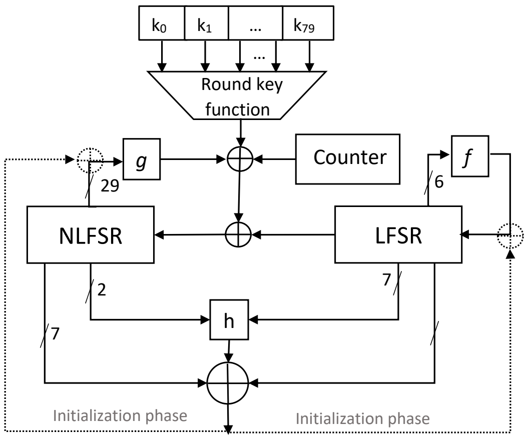

Figure 5.1: The overall structure of Sprout and Plantlet ciphers.

We use the following notation:

- t - the clock-cycle number
- $L^t = (l_0^t, l_1^t, \cdots, l_{|L|-1}^t)$ - state of the $|L|$-bit LFSR during the clock-cycle t
- $N^t = (n_0^t, n_1^t, \cdots, n_{39}^t)$ - state of the NLFSR during the clock-cycle t
- $C^t = (c_0^t, c_1^t, \cdots, c_8^t)$ - state of the counter during the clock-cycle t
- $k = (k_0, k_1, \cdots, k_{79})$ - key
- $iv = (iv_0, iv_1, \cdots, iv_{|iv|-1})$ - initialization vector of length $|iv|$
- k^* - the round key bit generated during the clock-cycle t

Table 5.1: Area size of the eStream finalists, lightweight block ciphers and Sprout

Cipher	Area size (GE)	Throughput (Kb/s)*	Logic process	Source
Block ciphers				
PRESENT 80	1570	200	0.18μm	[BKL+07]
PRESENT 80	1000	11.4	0.35μm	[RPLP08]
KATAN32	802	12.5	0.13μm	[CDK09]
KATAN48	927	18.8	0.13μm	[CDK09]
KATAN64	1054	25.1	0.13μm	[CDK09]
KTANTAN32	462	12.5	0.13μm	[CDK09]
KTANTAN48	588	18.8	0.13μm	[CDK09]
KTANTAN64	688	25.1	0.13μm	[CDK09]
Stream ciphers				
Mickey	3188	100	0.13μm	[GB08]
Trivium	2580	100	0.13μm	[GB08]
Grain 80	1294	100	0.13μm	[GB08]
Grain 80	1162	100	0.18μm	This work
Sprout*	813	100	0.18μm	This work
Plantlet*	928	100	0.18μm	This work

*- The throughput is given for the clock frequency of 100KHz

- z_t - the keystream bit generated during the clock-cycle t

Note that we use the index $^{(s)}$ when we refer to Sprout and the index $^{(p)}$ when referring to Plantlet. We use none of the indexes when a corresponding notation holds in both cases.

5.2.2 Design Specifications

Both ciphers are composed of an LFSR (40-bit long in Sprout and 61-bit long in Plantlet), a 40-bit NLFSR, a 9-bit counter, a round key function (RKF), and an output function. 80-bit key is used in both cases. Sprout uses the iv of length $|iv|^{(s)} = 70$, whereas Plantlet of length $|iv|^{(p)} = 90$.

As can be seen from Table 5.1 both ciphers use significantly less area than comparable existing lightweight stream ciphers and have higher throughput as compared to the existing block ciphers.

Initialization Phase. In the initialization phase the 40 NLFSR stages of both ciphers are loaded with the first 40 IV bits, i.e., $n_i = iv_i, 0 \leq i \leq 39$, and the remaining IV bits are loaded into the first LFSR stages. That is, in case of Sprout: $l^{(s)}_{i-40} = iv_i, 40 \leq i \leq 69.$; and in case of Plantlet: $l^{(p)}_{i-40} = iv_i$ for $40 \leq i \leq 89$. The last bits of the LFSR are filled with '0' and '1' as follows. In Sprout: $l^{(s)}_i = 1, 30 \leq i \leq 38, l^{(s)}_{39} = 0$; and in Plantlet: $l^{(p)}_i = 1, 50 \leq i \leq 58, l^{(p)}_{59} = 0, l^{(p)}_{60} = 1$.

Then, both ciphers are clocked 320 times without producing any keystream bits. Instead, the output is fed back and XORed with the LFSR and NLFSR inputs. In the initialization phase, the first 60 LFSR bits of Plantlet are updated, while the bit l_{60} remains fixed to 1, i.e., it is not involved in the LFSR update. After 320 clock cycles, the initialization phase is complete and the ciphers start to generate keystream bits. Here, the update function of the Plantlet LFSR is changed such that l_{60} is also updated.

LFSR of Sprout uses the primitive feedback polynomial which guarantees a period of $2^{40} - 1$:

$$P(x) = x^{40} + x^{35} + x^{25} + x^{20} + x^{15} + x^{6} + 1 \tag{5.1}$$

Double-Layer LFSR of Plantlet. Plantlet utilizes 2 different phase-dependent LFSRs, i. e., a 60-bit LFSR during the initialization phase and a 61-bit LFSR during the keystream generation phase. The novel aspect is that in both phases, the LFSR is instantiated with almost the same hardware. That is the LFSR used in the keystream generation phase reuses significant parts of the LFSR from the initialization phase. More precisely, in the initialization phase the LFSR uses a feedback polynomial P_I, whereas during the keystream generation phase, another polynomial P_K is used. However, both polynomials are almost the same which allows to reuse the same register and almost all of the taps. To ensure maximum period, both polynomials P_I and P_K are primitive and defined as follows:

$$\begin{aligned} P_I(x) &= x^{60} + x^{54} + x^{43} + x^{34} + x^{20} + x^{14} + 1 \\ P_K(x) &= x^{61} + x^{54} + x^{43} + x^{34} + x^{20} + x^{14} + 1 \end{aligned}$$

This means that during the initialization phase, the update function of the LFSR works as follows:

$$\begin{aligned} 0 \leq t \leq 319 : l^{(p)t+1}_{60} &= 1 \\ l^{(p)t+1}_{59} &= l^{(p)t}_{54} + l^{(p)t}_{43} + l^{(p)t}_{34} + l^{(p)t}_{20} + l^{(p)t}_{14} + l^{(p)t}_{0} + z_t \\ l^{(p)t+1}_{i} &= l^{(p)t}_{i+1}, 0 \leq i \leq 58 \end{aligned}$$

Afterwards when the keystream generation phase starts, the LFSR is updated as follows:

$$\begin{aligned} t \geq 320 : l^{(p)t+1}_{60} &= l^{(p)t}_{54} + l^{(p)t}_{43} + l^{(p)t}_{34} + l^{(p)t}_{20} + l^{(p)t}_{14} + l^{(p)t}_{0} \\ l^{(p)t+1}_{i} &= l^{(p)t}_{i+1}, 0 \leq i \leq 59 \end{aligned}$$

NLFSR and Counter. The NLFSR and the counter have the same specifications in both ciphers. Their respective lengths are 40 and 9 bits. The NLFSR is updated as defined in the following:

$$\begin{aligned} n_{39}^{t+1} &= g(N_t) + k^* + l_0^t + c_4^t \\ &= k^* + l_0^t + c_4^t + n_0^t + n_{13}^t + n_{19}^t + n_{35}^t + n_{39}^t + n_2^t n_{25}^t + n_3^t n_5^t + n_7^t n_8^t + n_{14}^t n_{21}^t + n_{16}^t n_{18}^t \\ &\quad + n_{22}^t n_{24}^t + n_{26}^t n_{32}^t + n_{33}^t n_{36}^t n_{37}^t n_{38}^t + n_{10}^t n_{11}^t n_{12}^t + n_{27}^t n_{30}^t n_{31}^t \\ n_i^{t+1} &= n_{i+1}^t, 0 \le i \le 39 \end{aligned}$$

The counter is a simple 9 bit register where the value is continuously increased as follows. The first seven bits ($c_0^t \cdots c_6^t$) of the counter are used to count cyclically from 0 to 79, i. e., it resets to 0 after 79 is reached. The two most significant bits realize a 2-bit counter to determine the number of elapsed clock cycles in the initialization phase, i. e., it is triggered by the resets of the lower 7 bits. Hence, it is reset to 0 after $4 \times 80 = 320$ clock cycles and indicates the end of the initialization phase.

Round Key Function. The RKF is responsible for making the update function key dependent. In Sprout at each clock t, RKF computes one round key bit k_t^* as follows:

$$\begin{aligned} k_t^{*(s)} &= k_t, \ 0 \le t \le 79; \\ k_t^{*(s)} &= (k_{t \ mod \ 80}) \cdot (l_4^s + l_{21}^s + l_{37}^s + n_9^s + n_{20}^s + n_{29}^s), \ t \ge 80; \end{aligned}$$

As it turned out (see Section 5.2.4 for details), the RKF of Sprout appeared to be the most important weakness of the cipher. Therefore, it was improved and in Plantlet where it simply cyclically selects the next key bit which is provided as input to the NLFSR. That is, it holds

$$k_t^{*(p)} = k_{(t \ mod \ 80)}, t \ge 0$$

Output Function. Both ciphers use the same output function. It has a nonlinear part $h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8$ where $x_0, \cdots, x_8$ correspond to the state variables n_4^t, l_6^t, l_8^t, l_{10}^t, l_{32}^t, l_{17}^t, l_{19}^t, l_{23}^t, n_{38}^t, respectively. The entire output function is determined by

$$z_t = h(x) + l_{30}^t + \sum_{j \in B} n_j^t$$

where $B = \{1, 6, 15, 17, 23, 28, 34\}$.

5.2.3 Design Rationale

We explain some aspects of the design rationale behind the suggested ciphers. We focus on the stream cipher Plantlet as long as there are several weaknesses in Sprout and the design rationale had flaws. We explain the main security problems of Sprout and argue why we introduced changes when designing Plantlet.

Choice of General Design As already mentioned, both ciphers adopt the generic idea behind the Grain family. This has been done for several reasons. First of all, our primary goal was to show the feasibility of the approach discussed in section 4.3. Therefore, we decided against designing a new cipher from scratch (which may have eventually turned out to be vulnerable against other attacks) but rather to build upon an existing established design. To this end, our focus was to pick a stream cipher that is already lightweight, is scalable (at least to some extent), and has undergone already some cryptanalysis.

Internal State Size Our goal was to show that is possible to develop a secure stream cipher that uses a register of size σ significantly below 2κ. For Sprout we decided to have the internal state size to be equal to the key size. However, there have been several indications that the internal state size of Sprout is too small. When designing Sprout we believed that the key cannot be easily found even when the entire internal state is known because the round key bits influence the internal state before they propagate to the output function and can be recovered. However, it was shown in [EK15] that if the cipher is clocked backwards, then the key bits can be recovered from the output sequence if the internal state is known. Therefore, having an internal state size equal to the security parameter is a design flaw in this context. In general, all the attacks against Sprout exploit the property of the internal state being too short. Yet, the main reason for increasing the internal state size in Plantlet is to rule out attacks based on the guess and determine technique. Possible ramifications were to increase the size of the LFSR, the NLFSR, or both of them. We chose to increase the size of the LFSR by 21 bits what at the same time allowed for a higher period of the output sequences.

The choice of the size was driven by the attack discussed in [Ban15]. Our analysis revealed that increasing the LFSR by 15 bits already made the attack less efficient than a brute-force attack. We added further 6 bits to the total length to increase the security margin.

Double-Layer LFSR The main reason for involving an LFSR in the Grain family is to ensure a minimum period. Consequently, the feedback polynomials of the LFSRs used in Sprout and Plantlet are primitive to guarantee a maximum period of $2^{40} - 1$ and $2^{61} - 1$ respectively. A further design criteria was to choose a polynomial with not too few terms in order to increase the resistance of the cipher against correlation attacks. However, in the initialization phase both ciphers use do not output any keystream, but feed it back to the NLFSR and to the LFSR inputs. This is done in order to assure both registers depend on all the key and IV bits. Unfortunately, for Sprout, where the normal LFSR of length 40 is used, it may lead to the case that the LFSR falls into an all-zero state after the initialization phase. As long as during the keystream generation phase the feedback from the output function is not used, the LFSR would remain in the all-zero state during the entire encryption process which would result in existence of keystream sequences with very short period. In [Ban15], a key recovery attack against Sprout was suggested based on this weakness.

Therefore, this weakness was improved in Plantlet. A typical countermeasure is to set one LFSR bit to 1 once the initialization phase is complete. However, this means that there are always two inputs to the cipher that lead to the same initial state, representing another weakness.

To avoid this problem, in Plantlet, we use two different LFSRs in different phases, such that the LFSR used in the keystream generation phase is obtained from the LFSR used in the initialization phase by extending it with one additional bit set to 1. With respect to the lightweight hardware implementation, it is preferable that the two update polynomials of these LFSRs share as many terms as possible. To achieve this, we found two primitive polynomials of degrees 60 and 61 which only differ in the maximum-degree term. In hardware, a mechanism to decide how the stages with indexes 60 and 61 are updated depending on the current phase can be easily implemented by using two multiplexers. We call this approach a *double-layer LFSR* and consider it of independent value.

NLFSR Update Function Plantlet and Sprout use ths same NLFSR. The update function of the NLFSR $g(N)$ is XORed with the LFSR output l_t and round key bit k_t^*. Each of these parts has different purpose and we will discuss them separately.

$g(N)$ is the nonlinear function which has the same form as in Grain 128a, where it was carefully selected in order to resist against different types of attacks [ÅHJM11]. As the used NLFSR is shorter than the one of Grain 128a, different indexes had to be chosen. Nonetheless, the relevant cryptographic properties remained: It is balanced, has a nonlinearity of $2,674,03264$, a resiliency of 4, and the set of the best linear approximations is of size 2^{14}. Each of the functions from this set has a bias of $63 \cdot 2^{-15}$ [ÅHJM11]. This function hasn't revealed any unexpected weaknesses over the time why we decided to stick to it.

The LFSR output is XORed with the NLFSR update function the same way as it is done in Grain family so that each of the NLFSR state bits is balanced.

Output Function The output function has the same form as the one used in Grain 128a. This function has nonlinearity of $61,440$. The best linear approximation of the nonlinear part h has a bias of 2^{-5}, and there are 2^8 such linear approximations [ÅHJM11].

Round Key Function The task of the round key function is to generate a round key bit that has to be involved in the update function. From a security perspective, one may prefer involved, non-linear functions that compute the round key bit from several key bits. However, more involved functions usually require more logic which increases the area size and often the power consumption as well. Hence, for practical reasons simpler designs should be preferred. Moreover, as elaborated in Subsection 3.3.2, the use of EEPROM for storing the key advocates solutions where the key bits are cyclically taken from the full key. With this respect, the round key functions of Sprout and Plantlet are perfectly suited for reading the key from EEPROM. In both ciphers, the round key function also cyclically steps through the key, but in Sprout it is only included into the NLFSR update with a probability of 0.5, i. e., only if the linear combination of several state bits is equal to 1. This has been exploited by several attacks [LN15, Ban15, EK15], for example by looking for longer periods where no key bit is involved. This imbalanced key involvement turned out to be a major weakness in Sprout. To avoid such attacks, Plantlet simply incorporates a key bit during each clock cycle. The modified *linear* key update function utilized by Plantlet balances key involvement such that the key always influences the state, which is also in accordance with the mitigation strategies suggested in [LN15, Ban15, EK15].

5.2.4 Security

Since several weaknesses were indicated in Sprout [MSBD15, LYR15, ZG15, LN15, EK15, Hao15, Ban15], in this section, we focus on discussing the security of Plantlet and explain which countermeasures were involved into the design in order to overcome the weaknesses of Sprout. Plantlet is designed for a small area size, the maximum possible period has to be shorter in comparison to other ciphers that deploy a larger internal state. Hence, we assume that distinguishing attacks might be possible but consider them only applicable to scenarios that are less relevant in the context of lightweight devices, e.g., encrypting long data streams. Nonetheless, we will shortly discuss distinguishing attacks at the end of this section. Moreover, as we consider that the key is fixed, variable-key attacks (including related-key attacks) attacks are also out of scope.

Similar to Sprout, our goal for Plantlet is to achieve 80-bit security against key-recovery attacks in the single-key but chosen IV scenario.

In this section, we discuss the attacks that were found against Sprout [AM15], and explain what countermeasures we introduced in Plantlet to render these attacks inapplicable and argue why.

5.2.4.1 Merging and Sieving Technique

The first key recovery attack against Sprout was published in [LN15]. The basic idea of this attack is to cleverly enumerate possible states of the two registers used in Sprout. Then, given the observed keystream a sieving technique is applied which allows to discard states that cannot produce the observed bits. It was shown that during the process of sieving, it is feasible to not only reduce the number of possible states, but also to recover some of the key bits. The attack allows for recovering the whole key with a time effort equivalent to roughly 2^{69} of Sprout encryptions. We note that the sieving is viable mainly due to the nonlinear influence of the key on the update function, in other words, due to the fact that the key bits do not affect the internal state in every clock cycle. Therefore, with the new round key function of Plantlet, which takes one key bit every clock cycle and uses it in the state update process, this approach is not applicable anymore. Moreover, the complexity of this attack against Sprout is around 2^{69} encryptions and it directly depends on the sizes of the shift registers. Consequently, even if a sieving attack with similar efficiency would be possible, due to the increase of the internal state size by 21 bits, the total effort would exceed the effort of a brute force attack.

5.2.4.2 Guess-and-Determine Attacks

In a guess-and-determine attack, the attacker guesses part of the internal state and aims to recover the remaining parts with an overall effort lower than brute force. In [MSBD15] it is shown that if the attacker partially guesses the internal state of Sprout, it is possible to create a system of nonlinear equations which can be solved by a SAT solver in reasonable time to recover the key and remaining state bits. The paper does not provide the complexity of solving such systems, however, some experimental results are provided. For example, when 54 out of 80 bits of the internal state are known, the SAT solver finds 6.6 key candidates on average within approximately 77 seconds. In addition, this paper presents a fault attack on Sprout.

An improved guess-and-determine attack has been described in [Ban15]. It likewise demonstrates the possibility to recover the key from partial knowledge of the cipher state, yet in a more efficient way compared to the attack explained in [MSBD15]. According to the experiments conducted by the author, it holds that if 50 bits of the internal state are guessed, the remaining bits and the key can be found in around 31 seconds on an average PC. In [EK15] it was estimated that the time complexity of this attack equals roughly 2^{70} Sprout encryptions and hence would constitute an attack. Observe however that no theoretical analysis of the effort has been given.[1] Consequently, to estimate the susceptibility of Plantlet to this attack, we conducted several lines of experiments on our own, which were performed using the Cryptominisat 2.9.10 [Cry] solver in combination with SAGE 6.9 [Sag] and the same source code as the author of [Ban15]. In the first line of experiment, we repeated the experiments reported in [Ban15] to establish a base line for comparison. In our case, when 50 bits of the internal state were guessed for different key/internal state values, the solver needed 30 to 430 seconds for finding the key and the remaining state bits. In order to understand how the new round key function of Plantlet influences the complexity of this attack, we performed a second line of experiments on Sprout but replaced the round key selection mechanism with the one used in Plantlet. In this case, the unknown bits could be determined in 1300 seconds on average, which

[1]In fact, the author of [Ban15] referred to his result as an observation rather than an attack.

is significantly longer compared to the unmodified round key function as defined in Sprout.

Finally, we conducted a third line of experiments and performed the attack against Plantlet directly. For different numbers of guessed bits, we measured the time to set up the equations and to compute the solutions. The results are given in Table 5.2.

Table 5.2: Time required (in seconds) for applying the attack from [Ban15] against Plantlet.

Guessed bits	Set up time			Solve time			Total time		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
95	0.057	0.096	0.075	0.549	0.949	0.764	0.61	1.04	0.83
90	0.058	0.097	0.078	0.877	1.559	1.193	0.935	1.652	1.271
85	0.089	0.150	0.112	6.400	23.36	12.95	6.49	23.45	13.06
80	0.087	0.095	0.091	33.73	1416.99	669.45	33.81	1417.08	664.05
75			0.095			8992.36			8992.45

For all cases except for the case where 75 bits are guessed, we conducted five experiments for different randomly selected key and state values. For the case of 75 guessed bits, we only carried out one experiment as already this case turned out to be quite time consuming[2]. Observe that one can perform within one second around 2^{10} Plantlet encryptions on the same PC with a non-optimized Sage implementation of the cipher. Hence, the results indicate that each of the considered variants of the attack is much more expensive than exhaustive key search.

Another guess-and-determine attack is reported in [LYR15]. The first step of the attack is to transform the NLFSR used in Sprout to an equivalent NLFSR with a simplified output function, but a more involved update function. More precisely, the goal of this transformation is to decrease the number of variables used in the output function and to introduce several update functions into the NLFSR (Galois configuration). Then, for several rounds at each clock cycle, the attacker guesses all but one bit of the internal state which go into the output function. The last one, which was not guessed can be computed directly from the output function because the keystream bit is known. Then, this process is continued. At some clock cycles, it is possible to compute the bits using not only the output function, but also the feedback functions. As soon as the state is known, the key can be found very efficiently. This attack projects an average number of guesses of around $2^{70.87}$. However, we expect that the increase of the internal state size incurs an effort beyond 2^{80} which is thus worse than exhaustive key search.

[2]In our experiment, the attack required 8992 seconds which is about 2.5 hours.

5.2.4.3 Trade-off Attacks

The authors of [EK15] demonstrated the first practical key recovery attack against Sprout. It allows to find the key in 2^{33} encryption time if the attacker is given data of 2^{40} bits of the keystream output with a memory requirement of 770 terabytes in total. The offline phase requires solving of approximately 2^{42} systems of linear equations with 20 unknowns each. The attack idea is to look for sequences where the key bits are not involved in the update function and to apply the common birthday paradox trade-off attack to these sequences.

A further trade-off attack on Sprout is given in [ZG15]. The time of the attack is 2^{79-x-y} with memory complexity of $[c \cdot (2x + 2y - 58) \cdot 2^{71-x-y}]$ and data complexity of 2^{9+x+y}, where x and y are the numbers of forward and backwards clockings of the cipher under the assumption that there is no involvement of the key bits in the state update.

Both trade-off attacks [EK15, ZG15] against Sprout require the existence of keystream sequences of certain lengths which are generated without any influence of the key. However, due to the fact that in Plantlet one key bit is involved in the state update at every clock cycle and hence propagates to the keystream, we do not see a possibility for these attacks to still be effective.

5.2.4.4 All-zero LFSR State

In [Ban15] it is shown that for every key approximately 2^{30} IVs exist for which the LFSR gets into the all-zero state during the keystream generation phase. This allows to find Key-IV pairs which result in the sequences which have a period equal to 80 and also to mount a key recovery attack with the complexity equivalent to $2^{66.7}$ Sprout encryptions with negligible memory requirements.

Such a situation cannot occur in Plantlet due to the use of the double-layer LFSR. That is, one of the LFSR bits of the longer LFSR is definitely set to "1" before the keystream generation phase begins. Moreover, the LFSR update polynomial is primitive, hence excluding the all-zero state.

5.2.4.5 Distinguishing Attacks

As explained at the beginning, we are aware that similar to Sprout distinguishing attacks are probable due to the relatively short period (although Plantlet should provide higher periods because of the use of a longer LFSR). From our point of view, it is a general problem that decreasing the area size makes it more and more difficult to ensure high periods. Moreover, one may argue that common scenarios deploying resource-constrained devices do not require the encryption of very long data streams. Nonetheless, we take a short look at existing distinguishing attacks against Sprout and discuss if and how they would apply to Plantlet as well.

In fact the first attack on Sprout was a related-key distinguishing attack presented in [Hao15]. However, we consider such attacks less relevant in the case of hardware ciphers as changing the key takes significant effort or is even impossible. Observe that attacks that use related IVs in the sense that initialization involves the same key but different, yet related public values do not apply here either as the internal update procedure depends on the (fixed) key.

However, Plantlet may also be subject to the distinguishing attack explained in [Ban15] against Sprout. The attack looks for two IVs that result in $(80 \cdot P)$-bit shifted keystreams, where P is a positive integer. This attack is possible due to the simplicity of the round key function. It would be possible to make the design resistant against this attack by either choosing a more complicated key-selection function or by further increasing internal state size. The first countermeasure would by all means result in a lower throughput because of the timing issues inherent to reading from serial EEPROM, the second would result in higher area size. On the other hand, the memory complexity of this distinguishing attack against Plantlet is at least 2^{58} which is about $32,768$ terabytes. As long as we are aiming for ultra lightweight devices, we think that it is not reasonable to introduce countermeasures against this attack at the cost of area and performance characteristics.

5.2.5 Implementation Results

Next, we explain and discuss our implementation of Plantlet. The implementation results for Plantlet are provided in Table 5.3. We do not consider any costs for *storing* the key in the EEPROM memory because we assume that it has to be provided by the device anyway, independent of whether it needs to load the key only once for initialization or requires constant access to the EEPROM. However, we do consider the costs incurred by the control logic which is required to *synchronize* the cipher with commercial EEPROM modules. That is, for the clock cycles when the key bits are not output by the EEPROM, we switch off the clocking of the cipher.

For the implementations discussed in this section, we used Cadence RTL Compiler[4] for synthesis and simulation, and the technology library UMCL18G212T3 (UMC 0.18μm process). We considered the same clock frequency of 100 kHz which is the most common rate for lightweight devices and supported by most of the existing commercial EEPROMs.

In such a scenario and due to the fact that Plantlet simply reads out the key bits sequentially, there is no need for using multiplexers for selecting the round key bits as this is directly provided by the discussed serial interfaces. Thus, we consider it as reasonable and fair approach to re-use the existing mechanisms when designing a cipher.

[3]By hard-wired we mean all the techniques which do not allow for changing the key, i.e. MROM or PROM

[4]See http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx

Table 5.3: Implementation results for Plantlet considering that the key is continuously read from different types of NVM.

Hard-wired[3]		Integrated EEPROM				
Area (in GE)	Throughput (in kbit/s)	Area (in GE)				Throughput (in kbit/s)
		x1	x8	x16	x32	
928	100	807	852	897	929	100

Commercial serial EEPROM modules

Interface	wrap case		no-wrap case	
	Area (in GE)	Throughput (in kbit/s)	Area (in GE)	Throughput (in kbit/s)
I^2C	822	88.9	885	66.7
SPI	807	100	860	83.3
Microwire	807	100	860	87.9
UNI/O	831	80	898	53.3

Depending on the chosen interface, our implementations of Plantlet require a minimum of 807 GE (e.g. for SPI in the wrap case) and a maximum of 898 GE (e.g. for UNI/O in the no-wrap case).

We also investigated cases in which the key is stored in customized EEPROM. The most common architectures convey the possibility to read either one bit from a random memory location or an n-bit word with a given address per memory access. Here, typical word sizes are 8, 16, and 32 bits. Our smallest implementation with respect to these different word sizes requires mere 807 GE. As another alternative, we examined using a lower frequency for accessing the EEPROM compared to the clock-frequency of the cipher itself for which then buffer registers are used. The resulting area sizes are also shown in Table 5.3. From our point of view, this approach might be practically relevant as it could potentially allow for reducing the power consumption of reading from EEPROM considerably.

For the case of Sprout, assuming that the key is read directly from EEPROM, and no additional logic is required, it needs 692 GE.

We contrast this scenario to the case when the key is stored in PROM (i.e. using fuses and antifuses) or MROM, which incurs additional logic for retrieving the current key bit as required by the respective round. The Sprout implementation assuming the key being burnt into the device needs 813 GE. The corresponding implementation for Plantlet results in 928 GE. For comparison, Table 5.4 provides information on several lightweight stream and block ciphers and their respective area requirements, however, since in the previously published papers the authors were not considering the approach of constantly reading from rewritable non-volatile memory, the figures are given for *fixed* keys where applicable.

Table 5.4: Overview of different lightweight ciphers regarding key and block size, throughput at 100 kHz, the logic process (denoted Tech.), and the required area size. The figures are given for the case, when the key is stored in non-rewritable memory (e.g. MROM or PROM). Note that for A2U2, no logic process information is given and the area size is an estimate as reported in [DRL11]. As there were no throughput figures given for Midori in [BBI+15], we computed them based on the number of rounds and the block size as described for the original implementation.

Cipher	Key size	Block size	Throughput [kbit/s]	Tech. [μm]	Area [GE]
		Stream ciphers			
A2U2 [DRL11]	56	1	100	–	300
Sprout [AM15]	80	1	100	0.18	810
Grain 80 [AM15]	80	1	100	0.18	1162
Trivium [GB08]	80	1	100	0.13	2580
Plantlet	80	1	100	0.18	928
		Block ciphers			
KTANTAN [CDK09]	80	32	12.5	0.13	462
		48	18.8		588
		64	25.1		688
LED [GPPR12]	64	64	5.1	0.18	688
	128		3.4		700
Midori [BBI+15]	128	64	400	0.09	1,542
		128	640		2,522
PRESENT-80 [BKL+07]	80	64	200	0.18	1,570
PRINTcipher [KLPR10]	80	48	100	0.18	503
	160	96			967

5.3 Low-Energy Cipher Trivium-2

5.3.1 Motivation

Our principal finding from the experiments discussed in Section 3.4 was that the 160x unrolled implementation of Trivium is about 9 times more energy efficient than any block cipher based solution for encrypting long data streams, and that unrolled stream ciphers in general outperform block ciphers in this domain. Since Trivium only offers 80-bit security, our motivation was to find a design that is both energy efficient for longer data streams and provides 128-bit security. We present the stream cipher Trivium-2 (based on the Trivium design) that in addition to providing 128-bit security is optimized with respect to energy consumption. Among stream ciphers that provide 128-bit security, for encryption of longer data streams the energy consumption of the cipher is around 2.5 times better than Grain-128 and around 15% better than Kreyvium (see Table 5.5 for comparison).

Table 5.5: Best cipher configurations with respect to energy consumption

Cipher	Security level	Optimal configuration	Energy (nJ) 1000 blocks
PRESENT	80 bits	2x	155.2
Plantlet	80 bits	16x	64.98
Grain v1	80 bits	20x	33.02
Trivium	80 bits	160x	10.15
Lizard	80 bits	16x	80.34
Midori64	128 bits	2x	90.5
Grain 128	128 bits	48x	25.29
Kreyvium	128 bits	128x	11.29
Trivium-2	128 bits	320x	9.77

We also argue the security of the cipher by performing extensive cryptanalysis on reduced round variants of the design.

5.3.2 Design Specifications

The design of Trivium-2 is similar to Trivium [CP08] but includes several changes to allow for a higher energy efficiency. The probably most notable difference is that Trivium-2 uses a 576-bit internal state, being two times larger than the state size of Trivium. Trivium-2 is composed of three NFSRs of sizes 186, 168 and 222 bits respectively. Trivium-2 uses 4,032 initialization rounds compared to Trivium that has 1,152 initialization rounds. Both ciphers effect three quadratic updates per round. For Trivium this gives 12 quadratic updates per state bit in average, while for Trivium-2 we get around

21 quadratic updates per state bit during initialization. The structure of Trivium-2 is depicted in the figure 5.2.

Naturally, the use of Trivium-2 starts with an initialization phase where an internal state is derived from a 128-bit secret key and a 96-bit public IV, followed by the keystream generation phase. As both phases deploy the same state update procedure, we explain it first before describing the two different phases.

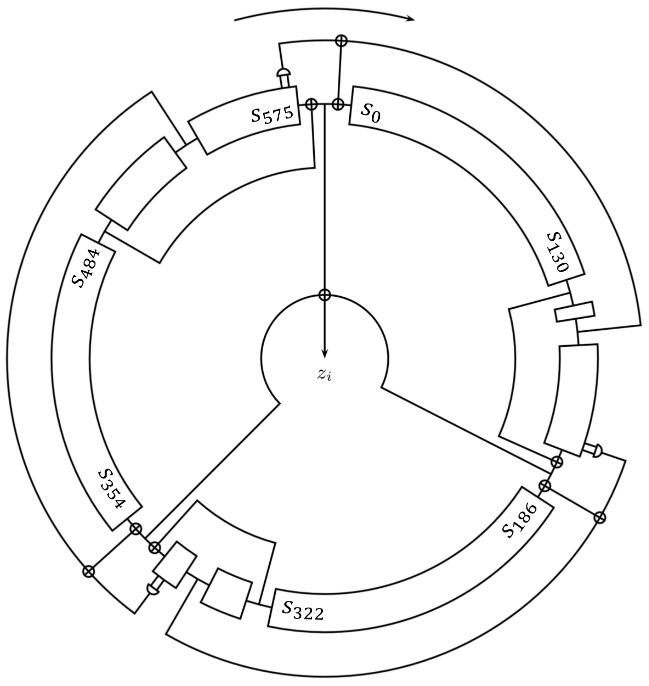

Figure 5.2: The structure of Trivium-2 .

State Update We denote the internal state by $St = (s_0, \ldots, s_{575})$. The state update function updates St to $St' = (s'_0, \ldots, s'_{575})$ as follows:

$$\begin{aligned}
s'_0 &\leftarrow s_{484} + s_{575} + s_{570} \cdot s_{571} + s_{136} \\
s'_{186} &\leftarrow s_{130} + s_{185} + s_{180} \cdot s_{181} + s_{340} \\
s'_{354} &\leftarrow s_{322} + s_{353} + s_{348} \cdot s_{349} + s_{526} \\
s'_i &\leftarrow s_{i-1}, \quad 0 \le i \le 575, i \notin \{0, 186, 353\}
\end{aligned} \tag{5.2}$$

As can be seen, the tap locations for Trivium-2 have been chosen at locations that are roughly two times the locations used in Trivium. Some tap locations were adjusted for the following reasons:

a) We used $(185, 353, 575)$ as tap locations instead of $2 \times (92, 176, 287)$ so that the state update function is one-to-one and invertible.

b) As in Trivium, we ensured that the tap locations feeding the AND gates were taken from successive register indices.

Since, Trivium-2 has twice the register length but the same number of tap locations as Trivium, the main concern was whether there existed linear approximations of the update function that resulted in any statistically significant bias in the distribution of any linear combination of the output keystream bits. However, after analyzing the design using the matrix method in [KHK06], we found that the maximum bias in the distribution in any linear combination of the keystream is 2^{-72}. This is low enough to prevent a linear attack for a 128 bit key. We refer to Section 5.3.4.2 for more details.

Initialization Phase At the beginning of the initialization phase, the 128-bit secret key $(k_0, \ldots, k_{127})$ and the publicly known 96-bit IV $(iv_0, \ldots, iv_{96})$ are loaded into the internal state as follows:

$$\begin{aligned}
(s_0, \ldots s_{127}) &\leftarrow (k_0, \ldots, k_{127}) \\
(s_{128}, \ldots, s_{223}) &\leftarrow (iv_0, \ldots, iv_{95}) \\
(s_{224}, \ldots, s_{255}) &\leftarrow (iv_0, \ldots, iv_{31}) \\
(s_{256}, \ldots, s_{383}) &\leftarrow (k_0, \ldots, k_{127}) \\
(s_{384}, \ldots, s_{447}) &\leftarrow (1, \ldots, 1) \\
(s_{448} \ldots s_{511}) &\leftarrow (iv_{32} \ldots iv_{95}) \\
(s_{512}, \ldots s_{572}, s_{573}, s_{574}, s_{575}) &\leftarrow (1, \ldots, 1, 0, 0, 0)
\end{aligned} \tag{5.3}$$

Then the state is updated 4,032 times, using the state update function (cf. (5.2)). While designing the Initialization Phase, we were mindful of three principal considerations:

a) Key-IV mixing should occur as early as possible: Fast diffusion of key and IV bits to all bits of the internal state is a desirable characteristic of any shift register based cryptosystem. The initialization phase ensures that all the 576 bits of the internal state is a function of all the 128 bits of the key and 96 bits of the IV after 1062 initialization rounds.

b) Diffusion of differences: We ensure that for every difference introduced by the IV, there are 2 bit differences introduced into the internal state. After experimenting with single bit differences we found that: when a difference is introduced in the

54th IV bit, then it affects all bits of the internal state after 1067 initialization rounds. This is the longest any single bit difference propagates without affecting atleast one state bit.

c) Cube/Conditional differential attacks: We discuss this issue in detail in Section 5.3.4.5, 5.3.4.6. The initialization ensures that the maximum number of rounds that can be attacked is 1,684.

We choose the number of initialization rounds equal to 4,032 to give us enough security margin, in keeping with the above observations. We choose an asymmetric 128-bit initialization constant ($1^{64}||1^{61}||0^{3}$) to prevent slide attacks reported in [CKP08]. Also a large initialization constant prevents easy enumeration of key-IV pairs that generate correlated keystreams [BMS12b, Ban15].

Keystream Generation Phase After the initialization phase is over, the cipher state is continuously updated with the state update function explained above. In addition, after each update the cipher outputs a keystream bit z_t as follows:

$$z_t = s_{130} + s_{185} + s_{322} + s_{353} + s_{484} + s_{575}$$

The maximum amount of keystream bits generated under the same (key, IV) pair is 2^{64}. Then the IV has to be changed and the cipher needs to be reinitialized.

5.3.3 Design Rationale

The main design principle was to enable a high degree of unrolling such that each copy of the update and output functions are as simple as possible. Theoretically, the most simple approach to achieve this goal is probably to use just one NFSR, possibly of large size, that uses conceptually simple update and output functions. Unfortunately, our internal analysis has shown that such a construction is very vulnerable to the attacks based on the linear approximation technique [KHK06]. A natural countermeasure is to deploy more complicated update and output functions but this would directly violate our main design goal mentioned above.

Thus, the next logical step is to use two NFSRs instead of one. Unfortunately, also here we faced problems similar to the case of one NFSR. For example, we considered variants of the cipher Bivium [Can06], which is a reduced variant of Trivium using two NFSRs only and has a similar structure as Trivium but with a shorter internal state. We were investigating extensions of Bivium with a considerably larger internal state and with more complicated update functions. But also here, either the designs were subject to linear approximation attacks or resulted into too complicate functions. While our research does not exclude the existence of secure and energy efficient designs based on one or two NFSRs, it seemed to be most promising to adopt the general structure of Trivium with three NFSRs.

Given this, the next question was how the design of Trivium should be modified to achieve a higher level of energy efficiency. At first, we investigated to slightly shift the taps of the update functions of the registers in order to make unrolling easier. However, all variants of Trivium that we found that would significantly improve the energy consumption were likewise vulnerable to linear approximation attacks.

Therefore, our next attempt was to increase the state size of Trivium. This not only allowed us designs that achieve better energy performance, but also to use a 128-bit key instead of an 80-bit key as used by Trivium. Note that Trivium itself would not provide 128-bit security even if the key length would be increased from 80-bits to 128-bits. The reason is the state recovery attack proposed in [MB07] where the complexity of this attack does not depend on the key length, but only on the state size (see Section 5.3.4.3 for more details). We note that using the structure of Kreyvium[CCF+16] instead of Trivium, which includes protection mechanisms against this attack, wouldn't be the optimal decision because Kreyvium always consumes more energy than Trivium due to the slightly more complex update function (see table 3.34).

We considered different designs which use the same Trivium structure with increased state sizes, with identical update functions in which we scale the location of the tap positions[5], and the number of initialization rounds. By doing so, we got scaled versions of Trivium which we call Trivium-i. Trivium-i has a state size of $288 \cdot i$ and the characteristics described above. As will be shown in the Section 5.3.5, Trivium-2 has the best energy performance.

5.3.4 Security

In this section we discuss the security of Trivium-2 with respect to known attacks against stream ciphers. The security of Trivium-2 was analyzed against various attack paradigms reported in literature. In Table 5.6, we summarize the main security results.

5.3.4.1 Period

Although the use of NFSRs is considered since several years for cryptographic designs, there is still only little known about the period of NFSRs. Since Trivium-2 is mainly composed of NFSRs, we cannot assure a minimum period[6]. Due to the similarity between the designs and the fact that the internal state of Trivium-2 is twice as large as the one of Trivium, we expect the cycle lengths of Trivium-2 to be at least as large as the ones of Trivium. To the best of our knowledge, there have been no indications that Trivium may have short periods.

[5]After indexes of the tap positions for both output and feedback functions were multiplied by i, they were continuously decremented by the value of 1, step by step unless designs secure against linear approximation attacks [KHK06] were found.

[6]However, note that this likewise applies to Trivium.

Table 5.6: Summary of security analysis for Trivium-2.

#	Attack type	Security Margin	Comments
1	Linear Approximation	Maximum linear bias $=2^{-72}$	Key recovery attack using a Fast Walsh transform will take atleast 2^{144} time
2	Guess and Determine	Complexity atleast $2^{148.3}$	Using the technique in [MB07], the initial stage requires guessing 148 bits
3	TMD tradeoff attack		Not feasible in time less than brute force due to large state size
4	Conditional Differential attack	Maximum attacked rounds =1431	We use a 20 bit differential and impose additional 20 conditions on the IV bits
5	Cube attack	Maximum attacked rounds =1684	We use a 46 dimensional cube and use the techniques of [Liu17]

Moreover, since 128 bits of the initial state are fixed (see (5.3)) an attacker has only partial control over the internal state. Therefore, even if short cycles exist, we believe, it will be difficult to find such Key/IV pairs which fall into these.

5.3.4.2 Linear Approximation

We applied this approach from [KHK06] to Trivium-2. The state update functions of Trivium-2 have 3 quadratic terms, where each of them can be approximated using 4 different linear functions: for example, the term $s_{180} \cdot s_{181}$ can be approximated as 0 or s_{180} or s_{181} or $s_{180} + s_{181}$. Therefore, in total there exist $4^3 = 64$ possible linear circuit approximations. The best results were given when all three quadratic monomials in the state update function (see (5.2)) were approximated with the constant zero function, yielding:

$$Pr(z_{576} + z_{439} + z_{421} + z_{403} + z_{284} + z_{266} + z_{248} + z_{177} + z_{146} + \tag{5.4}$$
$$z_{122} + z_{111} + z_{91} + z_{86} + z_{55} + z_{31} + z_0 = 0) = 1/2 + 2^{-72} \tag{5.5}$$

To use this property for distinguishing a keystream sequence from a purely random bit sequence, the common [KHK06] approach would be to use a chi-square test. However, this would result into a time complexity of about $(2^{72})^2 = 2^{144}$ which would take more time than exhaustive key search.

5.3.4.3 Guess and Determine

The best guess-and-determine attack against Trivium was proposed in [MB07]. It has a time effort of $c \cdot 2^{83.5}$, where c is the complexity of solving a system of 192 sparse linear equations. The attack is actually applicable against a whole family of Trivium-like ciphers which can be described as a combination of three NFSRs with certain properties: each NFSR is composed of three building blocks, where each has a size divisible by 3. These building blocks are connected to each other as shown in the Figure 5.3.

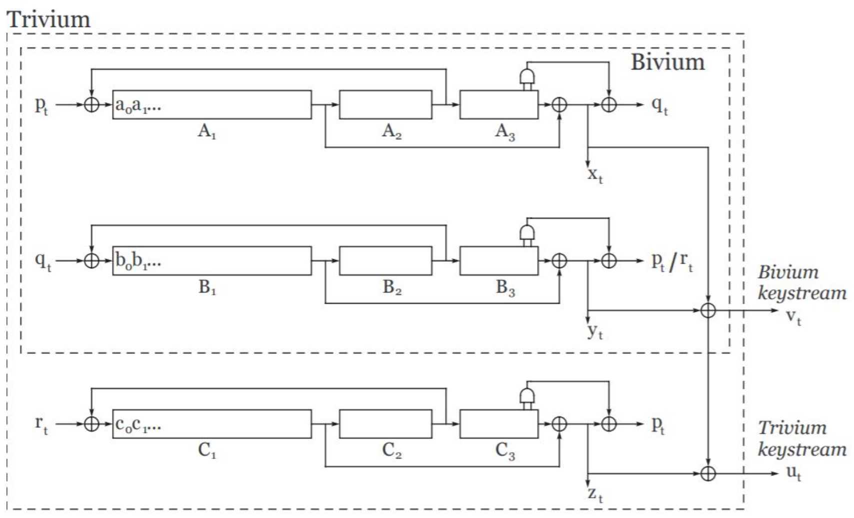

Figure 5.3: Alternative representation of trivium-like ciphers [MB07] .

Although this condition does not hold for Trivium-2, e.g. A_1 is equal to 131, which is not divisible by 3, we expect that this attack could still be applied with some minor modifications. This makes it necessary to discuss the lower bound on the complexity. The attack consists of two phases.

1. During the first phase, every third bit of each block of the register is determined. Most of these bits are found by guessing. However due to the very simple output function, it is possible to obtain $d = min\{A_1, B_1, D_1\}$ linear equations on these bits. This allows to reduce the number[7] of guesses to $(|S| - min\{A_1, B_1, C_1\})/3$ where $|S|$ denotes the state size. In case of Trivium-2, it would be required to guess $(576 - 131)/3 = 148.3$ state bits.

[7]In case of Trivium-2 , this number is not an integer since the size of the block A_1 is not divisible by 3. However, this fact would only lead to additional effort for the attacker, so that we can keep this for estimating a lower bound.

2. At the second phase, these bits are used for recovering the remaining state bits.

Since during the first phase it is already required to guess at least 148 bits, the overall complexity will be higher than of a brute force attack.

5.3.4.4 TMD Tradeoff Attacks

The classical time-memory-data tradeoff attack (BG attack) has a lower bound on the complexity of $\mathcal{O}(\sqrt{S})$ known data, memory and time, where the S is the size of the set of all possible internal states of the cipher. Hence in order to avoid this attack, the state size has to be at least two times larger than the key size, which is the case for Trivium-2 .

The same holds for the improvement (BS attack) suggested in [BS00] which gives the tradeoff curve of

$$T \cdot M^2 \cdot D^2 = S^2,$$

with the constraint of $D^2 \leq T$, where T, M, D are the time, memory, and data complexity, respectively. Obviously, it needs to holds that $\min\{T, M, D\} \geq \sqrt[5]{(2^{512})^2} \geq 2^{204}$. Thus, the effort of this attack is far beyond the effort of a brute force attack.

5.3.4.5 Conditional Differential Cryptanalysis

Conditional differential cryptanalysis is a variation of differential attacks for NFSR-based ciphers [KMNP10]. Typically the attacker introduces some differences, and imposes some conditions on the values of public variable IV to control the propagation of differences. Depending whether these conditions involve secret variables or not, she can mount key-recovery or distinguishing attacks. The technique extends to higher-order differential cryptanalysis.

We evaluate conditional differential cryptanalysis using higher-order differential characteristics against Trivium-2 by investigating the probability to find a zero-sum distinguisher. To be more precisely, we check whether $\sum_{c \in L(e_1,\ldots,e_d)} f(k, iv \oplus c) = 0$, where f is the update function of the stream cipher, and $L(e_1, \ldots, e_d)$ is the set of all 2^d linear combinations of $e_1, \ldots, e_d$.

We carefully choose arrangements of differences and conditions in the initial state to control propagation of differences in the AND operations. The best result of our experiments is a $1,431$-round (out of $4,032$) zero-sum distinguisher, obtained when the following 20-bit differences and 20-bit conditions are imposed on IV:

Differences: iv_{2i} for $0 \leq i \leq 19$.

Conditions : $iv_{2i+1} = 0$ for $0 \leq i \leq 19$.

We also search several such distinguishers (i.e. not only zero-sum) by considering differences and conditions in IV. As a result, the zero-sum distinguisher was best with respect to the number of round.

Table 5.7: Comparison with Trivium and Trivium-2 with respect to the security of cube attacks. LB of #round (key/IV) and LB of #round (IV) are lower bounds on the maximum number of rounds of NOT achieving maximum degree for Trivium-2 with all the key and IV bits as variables and all the IV bits as variables, respectively. #round of CT is the lower bound of attacked round by the cube tester.

Cipher	LB of #round (key/IV)	LB of #round (IV)	#round of CT (cube size)
Trivium	907 [Liu17]	793 [Liu17]	837 (37) [Liu17]
Trivium-2	1869	1635	1684 (46)

Thus, full round Trivium-2 should provide a sufficient security margin against the conditional differential cryptanalysis.

5.3.4.6 Cube Attacks

The cube attack [DS09], which is an extension of the higher-order differential cryptanalysis, exploits low-degree polynomial equations in the output of stream ciphers. To evaluate the security of cube attacks, we use a method for estimating the algebraic degree of NFSR-based cryptsystem which is recently proposed by Liu [Liu17][8]. Specifically, we implemented Algorithm 2 in their paper [Liu17] to obtain lower bounds on the maximum number of rounds of not achieving maximum degree for Trivium-2 with all the key and IV bits as variables for cube attacks. We found that Trivium-2 does not achieve the maximum degree 224 after an initialization of 1869 rounds. We also obtain lower bounds on the maximum number of rounds of not achieving maximum degree for Trivium-2 with only all the IV bits as variables, and found that Trivium-2 does not achieve the maximum degree 96 after an initialization of 1635 rounds. As shown in Table 5.7, Trivium-2 requires twice number of rounds than Trivium to achieve the required maiximum degree. Since the number of initialization of Trivium-2 is 3.5 times larger than Trivium, we consider that Trivium-2 has sufficient security margin.

In addition, to obtain the lower bound on the number of attacked rounds by cube tester [ADMS09], we did an exhaustive search on the sets of input variables which have size of around half length of the IV by ithm 2 as with Liu's evaluation of Trivium [Liu17]. As a result, the output of 1,684-round Trivium-2 has degree strictly less than 46 over a subset of IV bits with size 46, and thus the outputs of 1,684-round Trivium-2 over this cube always sum to 0.

[8]Todo et al. [TIHM17] also proposed a new method for evaluating cube attacks based on a division property. Since Liu's method is more efficient than their method and does not requires any solver, we use Liu's methods. We think that there is no much diffrence in the number of requierd rounds between these methods.

5.3.4.7 Algebraic Attacks

The classical algebraic attacks are normally not applicable in those cases where non-linear update functions are used. In fact, all algebraic attacks against Trivium discussed so far [MCP08, SFP08, TWB$^+$14] did apply to reduced-size versions only and could not be extended to the full version. Since Trivium-2 is likewise using a non-linear update function and has an even much larger internal state size, we do not expect that this type of attacks is feasible against Trivium-2.

5.3.4.8 Side Channel Attacks

Fault attacks on stream ciphers is a well researched topic as is apparent from numerous papers in literature [BMS12a, HR08, BM13]. Trivium itself is vulnerable from such attacks [HR08], and if the attacker is able to apply time synchronized bit flipping faults to the state register, he may be able to determine the internal state of the cipher by comparing the faulty and fault-free keystream bits and formulating enough equations to solve for the internal state, and so we do not claim security from fault attacks. As far as power attacks are considered, we could mount a differential power analysis (DPA) attack on Trivium-2, using the same techniques as in [FGKV07]. However, due to the low multiplicative complexity of the round function, threshold implementations as reported in [BGN$^+$14] could be designed in a similar manner.

5.3.5 Implementation Results

We simulated the energy performance of different versions of Trivium-i. In Table 5.8, we tabulate the energy figures for $i = 1, 2, 4$, where we have used an initialization of $1,152$, $4,032$ and $8,064$ rounds respectively. As in Trivium-2, the tap locations for Trivium-i were placed at register bits $i \times \mathcal{A}$, where $\mathcal{A}$ denotes the set of tap locations for the original Trivium stream cipher, with the above listed modifications required to make the state update one-to-one and to ensure that successive bit locations feed the AND gates. Due to the fact that the best performance for Trivium is at 160x, intuitively we guess that the optimal energy efficient configuration of Trivium-i will be at $i \cdot 160$x.

Table 5.8: Comparison of energy consumptions for Trivium-i, r denotes # unrolled rounds, Energy/bit figure calculated over 1000 blocks.

Cipher	r	Area (GE)	Power (uW) 10 MHz	Energy (pJ) 1 block	Energy (nJ) 1000 Blocks	Energy/bit (pJ)
Trivium	128	4593	207.1	227.8	10.56	0.17
	160	5409	248.2	223.4	10.15	0.16
	256	7755	419.5	251.7	10.73	0.17
Trivium-2	128	5811	246.4	813.1	13.13	0.21
	256	9172	412.0	700.4	11.00	0.17
	320	10136	456.5	639.1	9.77	0.15
	512	14488	768.2	691.4	10.29	0.16
Trivium-4	256	12424	511.7	1688.6	13.76	0.22
	512	19177	864.5	1469.7	11.67	0.18
	640	22513	1089.8	1525.7	11.87	0.19

The implementation results show that at the optimum configuration (320x) Trivium-2 performs slightly better than Trivium itself. Thus Trivium-2 seems to be a good candidate w.r.t. to energy efficiency to provide 128 bit security. Compared with the optimal configuration (48x) of Grain 128, which also provides 128 bit security, Trivium-2 is around 2.5 times more energy efficient and is around 15% more efficient than Kreyvium (see Table 5.5).

5.4 End-to-End Encryption Scheme for Low-Power Networks

5.4.1 Overview of LPNs

More and more enterprises are nowadays aiming to connect both new and legacy physical assets to their system landscapes in order to capture the data from these assets, generate insights and derive value out of the latter. This requires to retrofit existing physical assets in order to leverage them as part of the (connected) physical (IoT) infrastructure.

A major part of this growing amount of ”things” (devices) is expected to be low-powered, i.e., devices which are restricted to consume only very little energy to operate (and therefore to communicate). This has numerous consequences in practice; for instance, one cannot expect these devices to hold an active link but rather to communicate on-demand only. To this effect, these devices will have to communicate not only with a reduced packet size, but to embrace both a higher latency and a lower throughput at run-time. This takes us to the concept of Low-Power Connectivity, materialized as Low-Powered Wide-Area Networks (LPWANs) or Low-Power Networks (LPNs). LPNs offer an economically viable option to physically deploy new sensors along with the necessary communications infrastructure in order to generate, transport and ingest data coming from any type of asset. This means that part of the great potential of the LPNs rely on the cost effectiveness of retrofitting "old" assets with new (low-power) sensors and (low-power) connectivity, making this type of approach the first choice when targeting legacy assets and landscapes.

However, even if the connectivity is achieved, security is often an equally important requirement - in particular end-to-end data protection from the devices (i.e. the first end) all the way to the backend applications (i.e. the second end). The involvement of multiple actors in an IoT scenario (e.g. device, network, platform, application, professional services providers, etc) together with LPN constraints makes the fulfillment of an end-to-end data protection (i.e. confidentiality and integrity) a challenging endeavor.

5.4.2 Security Goals

We now discuss the requirements for the security solution which need to be fulfilled so that the solution was applicable in broad class of scenarios and use-cases. One of the most important needs is that the content of the data has to be concealed all the way from the time the data is generated (device) to the time the data consumed (application/dashboard). This requirement is driven by the need to be compliant to the EU General Data Protection Regulation [Com17]. A further natural requirement is that the integrity of the data has to be ensured as it strongly determines the reliability of distribution network. For practical reasons, it is also often required that part of the encrypted data is of the same format as the plaintext data. This allows to store the encrypted data in the same data-bases, where previously plain-text data was located, hence there is no need to make changes to the data-base, and therefore such a scheme can be used with higher flexibility.

Based on these discussions and own experience, we came up with the following list of security requirements for a LP-WAN security solution:

- It has to guarantee end-to-end confidentiality, authenticity, and integrity of the data.
- All components have to follow NIST recommendations i.e. [BR12, Dwo01, Dwo05, Dwo16].
- It has to be applicable to different existing low-power networks, even if the maximal payload size is as small as 12 bytes (e.g. SIGFOX).
- It needs to be deployable on the low-power devices.
- It must be applicable in scenarios which are not (yet) supporting bidirectional communication.

- It must be compliant with different encryption algorithms while meeting the compute power restrictions of constrained devices.

5.4.3 Use Case

We now discuss the concrete use case, for which this solution was developed.

The City of Antibes, France, instruments 300 kms of water pipelines with 2000+ sensors that capture a variety of data, e.g. debit, temperature, pressure, water storage levels. The collected information is sent over an ultra-narrow band network, and pushed to a central application (a predictive maintenance dashboard), depicted in Figure 5.4. The latter implements a hydraulic model of the network with the purpose of predicting disruptions in the water supply of the city. This application enables the City of Antibes to optimize their operational budget and to better allocate its work force (as well as external contractors) on the field.

Given that the sensor data carries critical information for a number of operational levels of the city, it is of the utmost importance to meet specific security requirements, being confidentiality and integrity of the sensor data the top priority.

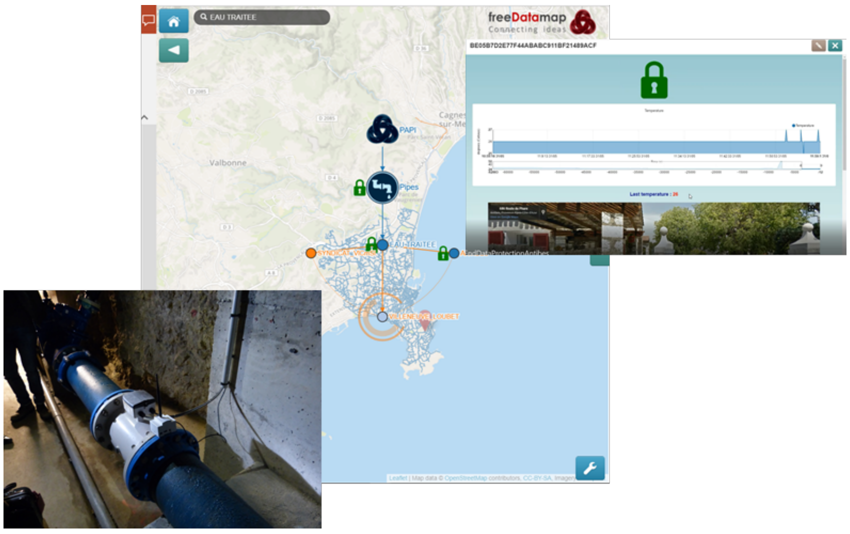

Figure 5.4: Predictive maintenance dashboard, city of Antibes

5.4.4 Design Specifications

Within this section, we propose a concrete solution that fulfills the security requirements formulated in Section 5.4.2. We first explain the underlying reference architecture and describe afterwards the data protection scheme.

5.4.4.1 Reference Architecture

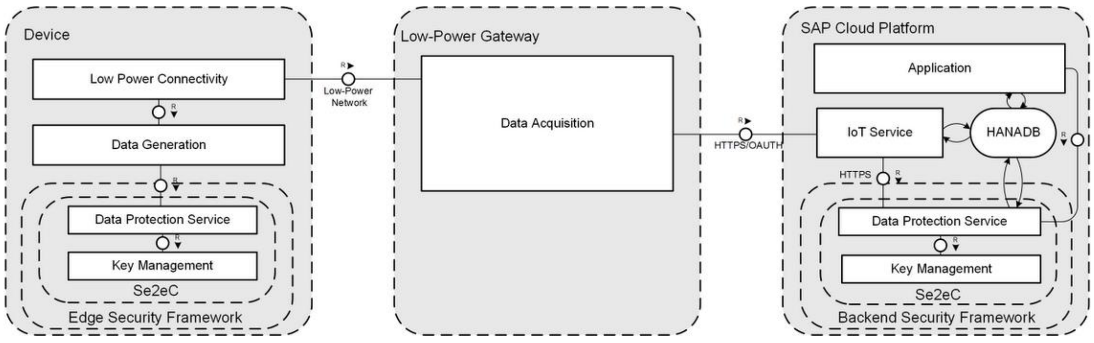

Figure 5.5: Reference architecture, secure end-to-end communications

As depicted in Figure 5.5, our reference architecture is organized end-to-end i.e., from the edge to the backend. Edge refers to those devices that are not under direct control of the backend system. Within the edge, a device connected to the actual sensors provides an entry point of data into the landscape. At the edge, devices have three basic data services: acquisition, protection, and communication of the data toward either the edge gateway or directly to the backend system. Gateways are in this context the bridges between rather constrained devices and the backend systems, enabling the required connectivity. The protection of sensor data is realized by a *Data Protection Service*. The data is sent from sensors over a LPN to a low-power gateway which then forwards it to the backend.

On the backend, the central application uses this *Data Protection Service* to check the validity of the protected data. That is, it detects and reports any attempts for data injection or replay attacks. Once data validity is confirmed, the data is decrypted.

On both, the device and backend, the *Data Protection Service* delegates the task of key management to the *Key Management* component. In our architecture, the device stores only one master key that is used for enabling both encryption and authentication. On the backend, the *Key Management* maintains a list of the master keys of all the devices involved.

5.4.4.2 Data Protection Scheme

Next we provide the full specification of the proposed data protection scheme. The used notation is summarized in Table 5.9.

Table 5.9: Notation

	Key Management
K_{master}	Master key. A 128-bit key preshared between the user application and the device. This key is never changed and is used to generate intermediate keys.
i	Intermediate key index. An integer value used for computing the next intermediate key. This value is updated by being incremented by 1 every SN_{max} times.
K_i	Intermediate key. A 128-bit key generated from the current intermediate key index and the master key, used for generating encryption and authentication keys.
K_{Enc}	Encryption key. A 128-bit key used for encrypting the data
K_{MAC}	Message authentication code key. A 128-bit key used for computing message authentication code tags.
	Data
M	Message. The sensor value that has to be transmitted from the node to the backend application.
C	Ciphertext
T	Authentication tag
	Meta data
ID	Device ID. The unique identifier of a device.
SN	The sequence number of the packet generated using the current intermediate key.
	Algorithms
$\mathsf{Enc}(\mathsf{K_{Enc}}, \mathsf{M})$	An algorithm that encrypts a message M using a secret key K_{Enc} and produces a ciphertext C.
$\mathsf{Dec}K_{Enc}, C$	An algorithm that decrypts a ciphertext C using the secret key K_{Enc} and outputs a plaintext message M.
$\mathsf{MAC}()$	An algorithm that computes a message authentication code for some data using authentication key K_{MAC}.
$\mathsf{CMAC}()$	Cipher-based message authentication code. [Dwo05]
	Parameters
SN_{max}	The maximum value for the sequence number. Represents the maximum number of packets to be processed using the same intermediate key K_i.
L	Length of the authentication tag T.
n^{max}	The maximal number of invalid packets with the same meta-data (ID,SN) for current intermediate key index i.
	Backend stored values
$\mathcal{S}^i$	The list of sequence numbers already used with the same key index i for a given device.
n^i_{SN}	The number of packets with the same meta-data (ID,SN) for the intermediate key index i currently received by the backend side.
$Decrypted[ID, SN, i]$	Boolean value which is true if a packet with the given meta-data (ID, SN) for intermediate key index i was already decrypted.

In a nutshell, the scheme can be divided into two parts: the key management part and the data protection part. As explained above, the scheme builds on master keys that are pre-shared between each of the devices and the backend application. With respect to key management, the scheme generates single-use keys which are applied to encrypt and authenticate every new packet to be sent. To achieve synchronization, it implements intermediate keys and sequence numbers.

The encryption and authentication processes are independent from each other. In order to keep the communication complexity low, we are using format preserving encryption algorithms where the ciphertext size equals the plaintext size, e.g. AES in counter mode [Dwo01] and FF1 [Dwo16]. Authentication is realized by computing a message authentication code (MAC) of the cipher text.

Data Protection Scheme on the Device Here we describe the operations performed on each of the devices. The pseudo code is given in algorithm 1.

Prerequisites:

- encryption scheme $\mathsf{Enc}()$;
- master key K_{master};
- intermediate key index i;
- sequence number SN;
- Device Id ID.

```
Input   : Plaintext message $M$
Output  : Payload $PL$
if ( $SN \neq SN_{max}$ ) then
    $SN \leftarrow SN + 1$;
else
    $SN \leftarrow 0$;
    $i \leftarrow i + 1$ ;
    $K_i \leftarrow \mathsf{CMAC}(\mathsf{K_{master}}, \mathsf{i})$
end
$K_{Enc} \leftarrow \mathsf{CMAC}(\mathsf{K_i}, \mathsf{SN} \parallel \mathsf{ID} \parallel 0)$ ;
$K_{MAC} \leftarrow \mathsf{CMAC}(\mathsf{K_i}, \mathsf{SN} \parallel \mathsf{ID} \parallel 1)$ ;
$C \leftarrow \mathsf{Enc}(\mathsf{K_{Enc}}, \mathsf{M})$;
$T \leftarrow \mathsf{CMAC}(\mathsf{K_{MAC}}, \mathsf{C}))$;
$PL \leftarrow (C \parallel T \parallel SN \parallel ID)$;
Return($PL$);
```

Algorithm 1: Data protection scheme: encryption and authentication on the constrained device side

Each device stores its device ID ID and a master key K_{master} in protected (access controlled) memory. These values are set during the deployment of the devices into a landscape and are known to the central application. The master key is used implicitly for both, encryption and authentication, by deriving appropriate intermediate keys from the master key; more precisely, the data sent to the backend is divided into packets and those are protected by applying different keys. Packets are grouped in *sequences* of length SN_{max} each. Our scheme uses a different intermediate key K_i for each sequence. When ever the sequence number reaches SN_{max}, a new sequence is started using a new intermediate key.

Moreover, the device keeps track of the sequence number, referring to the number of the packet in the current sequence. For each packet within a given sequence, an encryption key K_{Enc} and an authentication key K_{MAC} are derived from the current intermediate key K_i. Then the message is encrypted and the ciphertext is authenticated. The payload to be sent is composed of the ciphertext C, authentication tag T, sequence number SN, and device ID ID.

Data Protection Scheme on the Backend Next we explain the operations which take place on the backend. To distinguish between the values received from the nodes and those ones which are computed, we use the upper indexes ^{rec} and ^{com} respectively. For example, T^{rec} denotes the value of the authentication token in a received packet, while T^{com} denotes the token computed at the backend side.

Let $K_{master}, i, \mathcal{S}^i, n^i_{SN}$ be the values used by the device with identifier ID. The scheme is shown in Algorithm 2. At first it is checked if packets with the same meta data were already processed by the algorithm, if one of them was already verified and decrypted $Decrypted[ID, SN, i]$, or if the number of attempts exceeded the maximum n^{max}. In all these cases the algorithm returns corresponding errors. Otherwise, the authentication tag is verified. In case that the tag is valid the decryption process begins.

Encryption Algorithms We consider two different variants of encryption algorithms, which allow to preserve the size[9] of the plaintext while computing the ciphertext. These are:

- AES encryption in counter mode [Dwo01]:
  $C = \mathsf{Enc}(\mathsf{K_{Enc}}, \mathsf{M}) = M \oplus AES_{128}(K_{Enc}, IV^0)$

  where IV^0 is a 16-byte zero vector;
- the format preserving encryption scheme:
  $FF1_{enc}$[Dwo16].$C = \mathsf{Enc}(\mathsf{K_{Enc}}, \mathsf{M})$
  $= FF1_{enc}(K_{Enc}, TW, M)$.

  The tweak TW is computed as the concatenation of the device ID and sequence number: $TW = (ID \parallel SN)$.

[9]This is required to adapt the packet size to possible length restrictions.

Prerequisites : For each of the devices which communicate with the application:

- encryption scheme Enc();
- master key K_{master};
- current intermediate key index, i;
- all sequence numbers $\mathcal{S}^i$ already used for current intermediate key index i
- information if corresponding packets were already verified and decrypted $Decrypted[ID, SN, i]$;
- the number of messages received with the same sequence number n^i_{SN}

Input : Payload PL; length of MAC L
Output : Device ID ID, Sequence number SN, Message M, or error

```
$(C^{rec}, TAG^{rec}, SN, ID) \leftarrow PL$ // splitting payload
Use $K_{master}$, $i$, $\mathcal{S}^i$, $n^i_{SN}$ for the device ID = $ID$;
if $(SN \in \mathcal{S}^i)$ then
    if $(Decrypted[ID, SN, i])$ then
        Return(Error: replay attack detected);
    end
    $n^i_{SN} \leftarrow n^i_{SN} + 1$;
    if $(n^i_{SN} > n^{max})$ then
        Return(Error: too many attempts);
    end
else
    $\mathcal{S}^i \leftarrow \mathcal{S}^i \cup \{SN\}$.
end
$K_i \leftarrow \mathsf{CMAC}(\mathsf{K_{master}}, \mathsf{i})$ ;
$K_{MAC} \leftarrow \mathsf{CMAC}(\mathsf{K_i}, \mathsf{SN} \parallel \mathsf{ID} \parallel 1)$;
$T^{com} \leftarrow \mathsf{CMAC}(\mathsf{K_{MAC}}, \mathsf{C^r}))$;
if $(T^{rec} \neq T^{com})$ then
    Return(Error: Injection detected);
else
    $K_{Enc} \leftarrow \mathsf{CMAC}(\mathsf{K_i}, \mathsf{SN} \parallel \mathsf{ID} \parallel 0)$;
    $M^{com} \leftarrow \mathsf{Dec}K_{Enc}, C^{rec}$
end
$Decrypted[ID, SN, i] \Leftarrow true$;
if $SN = SN_{max}$ then
    $i \leftarrow i + 1$
end
Return($M$);
```

Algorithm 2: Data protection scheme: authentication check and decryption at the backend side

5.4.5 Design Rationale

The goal was to design a scheme that meets the requirements in section 5.4.2.

5.4.5.1 Key Management

Master Key The master key is a preshared 128-bit secret value which has to be stored in the secure memory of the device. It has to be properly generated for each of the unique device identifiers. From the master key all the other keys are derived. We note that most of the existing LPN technologies/providers offer a possibility to equip the devices with master keys.

Key Generation Approach For the generation we are using the cipher-based message authentication code CMAC [Dwo05], as it is recommended by the NIST for secure key generation from preshared secrets [BR12].

Intermediate Keys Intermediate keys are used together with sequence numbers to make sure that no two messages are encrypted and/or authenticated using the same secret keys K_{Enc}, K_{MAC}.

Sequence Number The reasons to include a sequence number are the following. First of all, it allows each time to generate different authentication and encryption keys, even under the same intermediate key. Due to this property, we do not need to update the intermediate key index and to write to the non-volatile memory every time, which is a rather energy consuming operation. Moreover, the sequence number is included into the payload to increase the reliability of the communication. For example, if a packet is lost or delayed, the backend will immediately get this information from the value of the sequence number of the next packet. Since the sequence number has to be included into the payload, we want to keep its size as small as possible. For example, in our current setting we use 16-bit long sequence numbers. This allows us to achieve both: update the intermediate key only rarely, while reducing to the outmost the increase of the payload.

Frequency of Intermediate Keys Update Every SN_{max} times of being used, the intermediate key is updated. The value SN_{max} depends on the length of the sequence number. For example, if the sequence number is 16 bit long, we set SN_{max} to be equal to 65535. This is done in order to avoid the situation when the same (i,SN) pair are used twice on the same device.

In addition, when the device is reset[10], the intermediate key index is incremented and the sequence number is zeroed before any other computations begin. This allows that

[10]Resetting the device may be easily done by an attacker, e.g., by interrupting the power supply or even by using a hard-reset button, which is available/accessible on most devices.

pairs (intermediate key / sequence number) are only used once.

Frequency of Encryption and Authentication Keys Update We use new encryption and authentication keys for every packet to be sent. This ensures that an attacker cannot collect multiple data connected to the same keys. Moreover, even if one of the packets and its corresponding keys are compromised, this does not hint to get information about the other packets. Although, it may seem that such approach would require a lot of computational effort, our experiments show that the scheme is efficient as demonstrated in section 5.4.7.

5.4.5.2 Authentication and Encryption

Different Keys for Authentication and Encryption A standard solution for achieving authentication and encryption with symmetric cryptography is to use CCM mode as recommended by the NIST[Dwo04]. It would probably be more efficient compared to the solution presented in this paper because it allows to achieve authentication and encryption using the same key for both. However, this mode is not flexible and does not support different encryption algorithms, contradicting our agnostic principles. As we need that the data protection scheme is compatible with other encryption schemes such as FF1, we realize authentication and encryption separately based on different single-time keys.

Encrypt then Authenticate We first encrypt and then apply a chosen MAC on the encrypted data rather than computing MAC of the plaintext. This option provides the integrity of both the plaintext and the ciphertext [BN00]. If the ciphertext is wrong, it is filtered out immediately and there is no need for decryption.

AES in Counter Mode In most of the real-life cases we use AES in CTR mode to have short payload and low computational complexity (one call of AES per packet for sensor data size below 128-bit). We note that this mode is among the ones recommended by the NIST [Dwo01]. We keep the initialization vector constant IV^0 since we change the encryption key K_{Enc} with every new packet.

Format Preserving Encryption Scheme $FF1_{enc}$ For the cases when it is required that the ciphertext not only has the same size as the plaintext but also has to be stored in the same format on the backend side, we are using the FF1 algorithm recommended by the NIST [Dwo16]. In comparison with the second recommended alternative called FF3, FF1 is more flexible as it accepts more formats of the input data and uses size tweaks. Moreover, there are recent indications of weaknesses in the FF3 algorithm [DV17].

Tweak Generation for $FF1_{enc}$ The main recommendation for the FF1 tweak generation [Dwo16] is that it should vary with each instance of the encryption as much as possible. Note that, the tweak has to be associated with the given plaintext and doesn't necessarily need to be secret. To fulfill these requirements, we use the meta-data of a given packet, which is associated with each plaintext per se, and generate the tweak by concatenating the sequence number and the device ID.

Choice of Message Authentication Codes We follow the NIST recommendations [Dwo05] and use CMAC for the authentication.

5.4.6 Security

Attacker Model In our security model we assume that an attacker has full access to the communication channel between the device and the backend. That is, an attacker can eavesdrop all the exchanged messages and modify them freely. We also give an attacker the possibility to reset a device as many times as needed. However, we do not consider side-channel attacks, as these depend on concrete devices and implementations.

Authentication of the Sender Each message to be sent by the device contains a cryptographic token (authentication tag) that is computed based on the single-time used authentication key. Relying on the NIST [Dwo05] we assume that without knowing the authentication key the chances of an attacker to apply the forgery attack against a given packet with the same meta-data(ID,SN for current i) is not higher than $2^{-L} \cdot n^{max}$.

To keep the payload size low, in our most lightweight implementation we consider L to be as low as 32 bits and fix n^{max} to 1, meaning that if a packet with the given meta-data is rejected, all the other similar packets are rejected without further verification. This leads to a risk of 2^{-32} which is sufficient for our desirable securtiy level. However, we note that increasing the length of the tag would allow to achieve higher security levels for the cost of higher communication complexity.

Data Integrity Similar discussions apply to the integrity of the message. If it is corrupted or changed it will be accepted as a valid one only with probability 2^{-L}.

Data Confidentiality Both encryption schemes that we discuss in the paper, namely AES 128 in counter mode and the format preserving encryption FF1 were selected strictly following the recommendations by the NIST[Dwo01, Dwo16]. Relying on these, we assume that there are no attacks with a complexity lower than 2^{128} which would break any of these encryption schemes.

Replay Attacks Each message contains a sequence number which is verified by the backend in order to discard replay attempts. If the message contains a sequence number which is repeated for the same intermediate key index and the same device, the replay attack is detected. We recall that the described data protection scheme excludes the possibility of using the same pair (key index, sequence number) twice.

Side-Channel Attacks The security of the scheme with respect to side-channel attacks depends on the concrete implementation. Therefore, no general arguments can be given about vulnerability. We use primitives from the TinyCrypt library where certain generic side-channel attack countermeasures are implemented [11].

5.4.7 Implementation

For reading the data from the sensors and applying protection mechanisms, our nodes are equipped with Intel Quark C1000 Microcontroller Units (MCUs) connected through LoRaWan modules. At the backend side we are using a decryption service implemented in Java and deployed in the SAP Cloud Platform.

We evaluate the performance of the proposed data protection scheme on Intel Quark C1000[12], an Intel Microcontroller Unit (MCU). This MCU is equipped with 8 KB of cache, 32 MHz clock speed, 80 KB SRAM, and 384 KB integrated Flash. In our experiments the MCU was powered with 5.07V.

As depicted in Figure 5.5, data is collected by the C1000, protected using the *Data Security Service*, and are sent over LoRa to a gateway. On C1000, the *Data Security Service* is implemented in C, hosted on ZephyrOS[13]. We use cryptographic primitives from its TinyCrypt library.

Protected data is then forwarded to the SAP Cloud Platform[14]. On the backend, the *Data Security Service*, implemented in Java, checks the validity of the protected data against replay attacks and injections. Once the validity of the protected data is confirmed, data is decrypted by the *Data Security Service*.

5.4.8 Evaluation

In this section, we discuss the overhead on battery (power consumption), memory, and time introduced by the data protection scheme. We consider the overhead on a cloud backend as negligible, as long as the resources are theoretically unlimited there. Our

[11]See http://zephyr-docs.s3-website-us-east-1.amazonaws.com/online/dev/crypto/tinycrypt.html

[12]See http://www.intel.com/content/www/us/en/embedded/products/quark/mcu/se-soc/overview.html

[13]See https://www.zephyrproject.org/

[14]See https://cloudplatform.sap.com/

Table 5.10: Battery and CPU performance on 10K data

	Data Acquisition	**Data Protection**		**LoRa Communication**
		AES-CTR	FFX	
Battery	<1 mAh	3 mAh	4 mAh	16 mAh
CPU	13.44 s	46.16 s	69.44 s	68 s

evaluations have been conducted for the two mentioned encryption algorithms: AES in counter mode, and FF1.

5.4.8.1 Battery and Time Consumption

In Table 5.10, we summarize the results on the battery and time consumption over 10k temperature data. We distinguish between three steps in this evaluation: (i) data acquisition, (ii) data protection, and (iii) data transmission over LoRaWAN. At data acquisition, we read temperature data from the sensor attached to the C1000. At data protection, we protect the data using the proposed scheme. We have here two implementations of encryption: AES in counter mode and FF1. At data transmission over LoRaWAN, we send data through the C1000 LoRa built-in module to an Intel LoRa gateway.

On 10k data the overall process takes 150.88s, and consumes 21mAh with FF1, and 127.6s and 20mAh with AES-CTR. The impact of data protection scheme with AES in counter mode is overall 36.17% on time and 15% on battery consumption. In case of FF1, the overall impact is 45.06% on time and 19.04% on battery consumption.

5.4.8.2 Memory Consumption

Regarding the total flash memory consumption, the data protection scheme occupies 0.56% while 15.49% is reserved for the cryptographic libraries, TinyCrypt, and 1.56% for LoRaWAN communication.

5.4.9 Discussion

Overall, the estimated overhead incurred on time and battery is at most 33%. The flash memory footprint overhead is negligible.

For a regular 16,750mAh battery, 6.700k sensor data can be acquired and then encrypted with AES in counter mode and sent over LoRaWAN. It allows to process 12 years of data when data is sent every minute, or 190 years of data if sent every 15 minutes.

CHAPTER 6 Conclusion

In this thesis, we investigated several problems which refer to the field of lightweight cryptography.

In Chapter 3, we explored different real-life scenarios and evaluated how suitable the existing cryptographic solutions for these scenarios are. In Section 3.2 we revisited lightweight authentication schemes and discussed their suitability for RFID systems which use tags in the cost range of 0.05 to 0.10. The evaluation was based on a set of conditions that need to be satisfied by real-world applications. These conditions have been derived from open literature but mostly from numerous discussions with hardware experts from industry. As our analysis reveals the probably most prominent approach for designing lightweight authentication schemes, i.e., basing the security on the hardness of the famous LPN problem, has, so far, not lead to any practical solutions feasible for ultra-constrained devices. While this neither questions the significance nor the security of such designs, it indicates that for real-world applications in the context of low-cost RFID tags other approaches should be considered. As we argue, one possibility is the rather straightforward approach using lightweight encryption schemes. However, this immediately raises the question whether other approaches exist that are explicitly tailored to lightweight authentication and may enable even more efficient solutions. We think that this is an interesting and open question for both academia and industry.

In Section 3.3, we investigated the limitations and consequences of the design approach when ciphers continuously access the key from the non-volatile memory. After a discussion on reasonable approaches for storing a key in non-volatile memory, motivated and validated by several commercial products, we focus on the case that the key is stored in EEPROM. Here, we highlight existing constraints and conclude that some designs are better suited for reducing the area size than others.

In Section 3.4 we looked at some of the issues related to low-energy encryption. We experimented with various design parameters that affect the energy consumption of the encryption process and were able to draw several conclusions out of it. Our investigations showed that although block ciphers are more energy-efficient solutions for encryption of short data streams, for longer data streams, multiple round unrolled stream ciphers perform better. Stream ciphers with simple update functions are found to be more energy-efficient since these are easy to unroll. More degrees of unrolling lets us encrypt a higher number of bits in one clock cycle, which is crucial for bringing down the number of clock cycles required to encrypt a a data of a given length, and hence the energy consumption. We found that the Trivium structure was best suited for this purpose. The 160x unrolled implementation of Trivium was not only around 9 times better than the best block cipher based solution in terms of energy consumption of 1000 data blocks, but also the cipher easily avoids linear approximations which can become an issue with ciphers with lightweight update functions.

In Chapter 4, we proposed new approaches. In Section 4.2, we presented a technique for reducing the critical path of a circuit while implementing stream ciphers based on feedback shift registers (FSRs) in ASICs. As opposed to the common pipelining technique, existing structures are re-used for reducing the memory increase. The transformation has been applied to Grain-128 and Grain-128a where the maximal throughput is increased by around 20% . As opposed to other solutions, no additional memory is required.

In Section 4.3, we discussed a different approach for realizing keystream generators. The core idea is to design a cipher where the set of internal states is split into a large number of equivalence classes such that any trade-off attack has to consider every class at least once. As a concrete approach for realizing this property, we suggest to involve the secret key not only in the initialization process but also in the update procedure. Although the change is conceptually simple, it may allow to avoid the rule of thumb that the internal state size needs to be at least twice the desired security level.

In Chapter 5, we presented new cryptographic solutions. Exploiting the fact that storing fixed values consumes less area than using registers, in Section 5.2, we were able to present two new stream ciphers named Sprout and Plantlet which have significantly smaller area size than existing ciphers of the same security level and providing the same throughput. While Sprout was broken by multiple attacks, Plantlet which was designed to improve the weaknesses of Sprout remains secure to the best of our knowledge at the time of writing.

Following our findings on low energy ciphers discussed in Section 3.4, in Section 5.3 we presented the Trivium-2 design. It has the similar structure as Trivium but has a double state size. Increasing the internal state allowed us to achieve 128-bit security. We showed that the new cipher performed slightly better than Trivium with respect to energy performance when encrypting long data streams.

We also proposed a secure end-to-end data protection scheme for low-power networks in Section 5.4. Our solution provides data confidentiality and integrity from (IoT) devices to central backend applications by relying on established cryptographic schemes and frequent key updates. Moreover, it respects the technical constraints imposed by low-power devices (e.g. CPU, memory) and low-power connectivity (e.g. high latency, low throughput). Our solution has been implemented on industrial IoT devices and deployed on the water distribution network of the City of Antibes. The results show that the data protection scheme incurs an increase in battery consumption which is four times less than what is required for transmitting the data over LoRaWAN.

Bibliography

[Aby10] Mohammad Reza Sohizadeh Abyaneh. On the security of non-linear HB (NLHB) protocol against passive attack. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, pages 523–528. IEEE, 2010.

[Ade16] Adesto Techonologies. *AT25SF041 4-Mbit, 2.5V Minimum SPI Serial Flash Memory with Dual-I/O and Quad-I/O Support*, 2016. http://www.adestotech.com/wp-content/uploads/DS-AT25SF041_044.pdf.

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2009.

[ÅHJM11] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of Grain-128 with optional authentication. *IJWMC*, 5(1):48–59, 2011.

[AHM14] Frederik Armknecht, Matthias Hamann, and Vasily Mikhalev. Lightweight authentication protocols on ultra-constrained RFIDs - myths and facts. 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 1–18. Springer, 2014.

[AM] Frederik Armknecht and Vasily Mikhalev. Revisiting a Recent Resource-Efficient Technique for Increasing the Throughput of Stream Ciphers. International Conference on Security and Cryptography, Vienna, Austria, 2014.

[AM14] Frederik Armknecht and Vasily Mikhalev. On increasing the throughput of stream ciphers. In *Topics in Cryptology–CT-RSA 2014*, pages 132–151. Springer, 2014.

[AM15] Frederik Armknecht and Vasily Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. In Gregor Leander, editor, *FSE*, volume 9054 of *Lecture Notes in Computer Science*, pages 451–470. Springer, 2015.

[AMM10] Gildas Avoine, Benjamin Martin, and Tania Martin. Tree-based rfid authentication protocols are definitively not privacy-friendly. In SiddikaBerna Ors Yalcin, editor, *Radio Frequency Identification: Security and Privacy Issues*, volume 6370 of *Lecture Notes in Computer Science*, pages 103–122. Springer Berlin Heidelberg, 2010.

[Atm14] Atmel Corporation. *CryptoRF EEPROM Memory 13.56MHz, 4 Kilobits, Summary Datasheet*, 2014. `http://www.atmel.com/Images/Atmel-8672S-CryptoRF-AT88RF04C-Datasheet-Summary.pdf`.

[Atm15] Atmel Corporation. *3-wire Serial EEPROM 1K (128 x 8 or 64 x 16), Data sheet*, 2015. `http://www.atmel.com/Images/Atmel-5193-SEEPROM-AT93C46D-Datasheet.pdf`.

[Ava18] Roberto Avanzi. From software security to cryptographic research in the ARM World. Lightweight Crypto Day, 2018.

[Bab95] Steve Babbage. Improved "exhaustive search" attacks on stream ciphers. In *Security and Detection, 1995., European Convention on*, pages 161–166. IET, 1995.

[Ban15] Subhadeep Banik. Some results on Sprout. In Alex Biryukov and Vipul Goyal, editors, *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*, volume 9462 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2015.

[BB08] G.K. Balachandran and R.E. Barnett. A 440-nA true random number generator for passive RFID tags. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 55(11):3723–3732, Dec 2008.

[BBI$^+$15] 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, *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

[BBR15] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Exploring Energy Efficiency of Lightweight Block Ciphers. In Orr Dunkelman and Liam Keliher, editors, *SAC*, volume 9566 of *Lecture Notes in Computer Science*, pages 178–194. Springer, 2015.

[BBS06] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *Advances in Cryptology-CRYPTO 2006*, pages 1–21. Springer, 2006.

[BC08] Julien Bringer and Hervé Chabanne. Trusted-HB: a low-cost version of HB+ secure against man-in-the-middle attacks. *arXiv preprint arXiv:0802.0603*, 2008.

[BCD06] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. HB^{++}: a lightweight authentication protocol secure against some attacks. In *Security, Privacy and Trust in Pervasive and Ubiquitous Computing, 2006. SecPerU 2006. Second International Workshop on*, pages 28–33. IEEE, 2006.

[BCG$^+$12] 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, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BDE$^+$13] 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, *RFIDSec*, volume 8262 of *Lecture Notes in Computer Science*, pages 103–112. Springer, 2013.

[Ber68] Elwyn R Berlekamp. Algebraic coding theory. 1968.

[BGN$^+$14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-Order Threshold Implementations. *IACR Cryptology ePrint Archive*, 2014:751, 2014.

[BHN11] Carl Bosley, Kristiyan Haralambiev, and Antonio Nicolosi. HB^N: An HB-like protocol secure against man-in-the-middle attacks. *IACR Cryptology ePrint Archive*, 2011:350, 2011.

[BKL$^+$07] 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.

[BL08] Raymond E Barnett and Jin Liu. An EEPROM programming controller for passive UHF RFID transponders with gated clock regulation loop and current surge control. *Solid-State Circuits, IEEE Journal of*, 43(8):1808–1815, 2008.

[BL13] Daniel J. Bernstein and Tanja Lange. Never trust a bunny. In *Proceedings of the 8th International Conference on Radio Frequency Identification: Security and Privacy Issues*, RFIDSec'12, pages 137–148, Berlin, Heidelberg, 2013. Springer-Verlag.

[BM13] Subhadeep Banik and Subhamoy Maitra. A Differential Fault Attack on MICKEY 2.0. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2013.

[BMA$^+$18] Subhadeep Banik, Vasily Mikhalev, Frederik Armknecht, Takanori Isobe, Willi Meier, Andrey Bogdanov, Yuhei Watanabe, and Francesco Regazzoni. Towards low energy stream ciphers. *IACR Transactions on Symmetric Cryptology*, 2018(2):1–19, Jun. 2018.

[BMS06] Alex Biryukov, Sourav Mukhopadhyay, and Palash Sarkar. Improved time-memory trade-offs with multiple data. In *Selected Areas in Cryptography*, pages 110–127. Springer, 2006.

[BMS12a] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A differential fault attack on grain-128a using macs. In *Security, Privacy, and Applied Cryptography Engineering*, pages 111–125. Springer, 2012.

[BMS12b] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. Some Results on Related Key-IV Pairs of Grain. In Andrey Bogdanov and Somitra Kumar Sanadhya, editors, *SPACE*, volume 7644 of *Lecture Notes in Computer Science*, pages 94–110. Springer, 2012.

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

[BR12] Elaine Barker and Allen Roginsky. SP 800-133. Recommendation for cryptographic key generation. *NIST Special Publication*, 800:133, 2012.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data tradeoffs for stream ciphers. In *Advances in Cryptology–ASIACRYPT 2000*, pages 1–13. Springer, 2000.

[BSW01] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of a5/1 on a pc. In *Fast Software Encryption*, pages 1–18. Springer, 2001.

[Can06] Christophe De Cannière. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[CCF$^+$16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. In Thomas Peyrin, editor, *FSE*, volume 9783 of *Lecture Notes in Computer Science*, pages 313–333. Springer, 2016.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[CKP08] Christophe De Cannière, Özgül Küçük, and Bart Preneel. Analysis of Grain's Initialization Algorithm. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 276–289. Springer, 2008.

[CM03] Nicolas Courtois and Willi Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 345–359. 2003.

[CMM14] Abhishek Chakraborty, Bodhisatwa Mazumdar, and Debdeep Mukhopadhyay. Fibonacci LFSR vs. Galois LFSR: Which is More Vulnerable to Power Attacks? In Rajat Subhra Chakraborty, Vashek Matyas, and Patrick Schaumont, editors, *SPACE*, volume 8804 of *Lecture Notes in Computer Science*, pages 14–27. Springer, 2014.

[Com17] European Commission. EU General Data Protection Regulation, 2017.

[CP] Christophe De Canniere and Bart Preneel. Trivium specifications. *eSTREAM, ECRYPT Stream Cipher Project*, 2006.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[CR08] Peter H. Cole and Damith C. Ranasinghe. *Networked RFID Systems and Lightweight Cryptography: Raising Barriers to Product Counterfeiting*. Springer Berlin Heidelberg, 1 edition, 2008.

[Cry] Cryptominisat-2.9.10. http://www.msoos.org/cryptominisat2/.

[Cyp15] Cypress Semiconductor Corporation. Parallel NOR Flash Memory: An Overview, 2015. http://www.cypress.com/file/202491/download.

[CZDL13] Zhaoxian Cheng, Xiaoxing Zhang, Yujie Dai, and Yingjie Lu. Design techniques of low-power embedded EEPROM for passive RFID tag. *Analog Integrated Circuits and Signal Processing*, 74(3):585–589, 2013.

[DDK09] C. De Cannière, O. Dunkelman, and M. Knežević. KATAN and KTANTAN – A family of small and efficient hardware-oriented block ciphers. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems (CHES) 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009.

[DK07] Dang Nguyen Duc and Kwangjo Kim. Securing HB$^+$ against GRS man-in-the-middle attack. In *Institute of Electronics, Information and Communication Engineers, Symposium on Cryptography and Information Security*, 2007.

[DK08] Orr Dunkelman and Nathan Keller. Treatment of the initial value in time-memory-data tradeoff attacks on stream ciphers. *Information Processing Letters*, 107(5):133–137, 2008.

[DRL11] Mathieu David, Damith C Ranasinghe, and Torben Larsen. A2u2: a stream cipher for printed electronics rfid tags. In *RFID (RFID), 2011 IEEE International Conference on*, pages 176–183. IEEE, 2011.

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009.

[Dub09] Elena Dubrova. A transformation from the Fibonacci to the Galois NLFSRs. *IEEE Trans. Information Theory*, 55(11):5263–5271, 2009.

[Dub10] E. Dubrova. Finding matching initial states for equivalent NLFSRs in the Fibonacci and the Galois configurations. *Information Theory, IEEE Transactions on*, 56(6):2961–2966, 2010.

[DV17] F. Betül Durak and Serge Vaudenay. Breaking the ff3 format-preserving encryption standard over small domains. Cryptology ePrint Archive, Report 2017/521, 2017. http://eprint.iacr.org/2017/521.

[Dwo01] Morris J Dworkin. *SP 800-38A. Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. National Institute of Standards & Technology, 2001.

[Dwo04] Morris J Dworkin. *SP 800-38C. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality*. National Institute of Standards & Technology, 2004.

[Dwo05] Morris J Dworkin. *SP 800-38B. Recommendation for block cipher modes of operation: The CMAC mode for authentication*. National Institute of Standards & Technology, 2005.

[Dwo16] Morris J Dworkin. *SP 800-38G. Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption*. National Institute of Standards & Technology, 2016.

[EK15] Muhammed F Esgin and Orhun Kara. Practical cryptanalysis of full Sprout with TMD tradeoff attacks. In *International Conference on Selected Areas in Cryptography*, pages 67–85. Springer, 2015.

[FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer Berlin Heidelberg, 2004.

[FGKV07] Wieland Fischer, Berndt M. Gammel, O. Kniffler, and Joachim Velten. Differential Power Analysis of Stream Ciphers. In Masayuki Abe, editor, *CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 2007.

[Fre14] Fremont Micro Devices. *FT25H04/02 DATASHEET*, 2014. http://www.fremontmicrousa.com/pdf/FT25H04_Rev1p0.pdf.

[FS09] D. Frumkin and A Shamir. Untrusted-HB: Security vulnerabilities of Trusted-HB. Cryptology ePrint Archive, Report 2009/044, 2009.

[FWR05] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. Aes implementation on a grain of sand. *Information Security, IEE Proceedings*, 152(1):13–20, Oct 2005.

[GB08] Tim Good and Mohammed Benaissa. Hardware performance of estream phase-iii stream cipher candidates. In *Proc. of Workshop on the State of the Art of Stream Ciphers (SACS'08)*, 2008.

[GJN$^+$16] Jian Guo, Jérémy Jean, Ivica Nikolic, Kexin Qiao, Yu Sasaki, and Siang Meng Sim. Invariant Subspace Attack Against Midori64 and The Resistance Criteria for S-box Designs. *IACR Trans. Symmetric Cryptol.*, 2016(1):33–56, 2016.

[GLS14] L. Gaspar, G. Leurent, and F.-X. Standaert. Hardware implementation and side-channel analysis of lapin. In *Topics in Cryptology – CT-RSA 2014*, volume 8366 of *LNCS*, pages 206–226. Springer, 2014.

[Gol68] Robert Gold. Maximal recursive sequences with 3-valued recursive cross-correlation functions (corresp.). *Information Theory, IEEE Transactions on*, 14(1):154–156, 1968.

[Gol96] J. D. Golic. Linear models for keystream generators. *IEEE Transactions on Computers*, 45(1):41–49, Jan 1996.

[Gol97] JovanDj. Golić. Cryptanalysis of alleged a5 stream cipher. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer Berlin Heidelberg, 1997.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[GPPR12] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED Block Cipher. Cryptology ePrint Archive, Report 2012/600, 2012. `http://eprint.iacr.org/2012/600`.

[GRS05] Henri Gilbert, Matthew Robshaw, and Herve Sibert. Active attack against HB$^+$: a provably secure lightweight authentication protocol. *Electronics Letters*, 41(21):1169–1170, 2005.

[GRS08a] Henri Gilbert, Matthew JB Robshaw, and Yannick Seurin. Good variants of HB$^+$ are hard to find. In *Financial Cryptography and Data Security*, pages 156–170. Springer, 2008.

[GRS08b] Henri Gilbert, Matthew JB Robshaw, and Yannick Seurin. HB$^\#$: Increasing the Security and Efficiency of Efficiency of HB+. In *Advances in Cryptology–EUROCRYPT 2008*, pages 361–378. Springer, 2008.

[Hao15] Yonglin Hao. A related-key chosen-IV distinguishing attack on full Sprout stream cipher. *IACR Cryptology ePrint Archive*, 2015:231, 2015.

[HB01] Nicholas J Hopper and Manuel Blum. Secure human identification protocols. In *Advances in cryptology—ASIACRYPT 2001*, pages 52–66. Springer, 2001.

[Hel80] Martin E Hellman. A cryptanalytic time-memory trade-off. *Information Theory, IEEE Transactions on*, 26(4):401–406, 1980.

[HJM07] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing*, 2(1):86–93, 2007.

[HJMM06] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A stream cipher proposal: Grain-128. In *Information Theory, 2006 IEEE International Symposium on*, pages 1614–1618. IEEE, 2006.

[HJMM08] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The grain family of stream ciphers. In *New Stream Cipher Designs*, pages 179–190. Springer, 2008.

[HKL$^+$12] Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. Lapin: an efficient authentication protocol based on Ring-LPN. In *Fast Software Encryption*, pages 346–365. Springer, 2012.

[HKM17] Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD - A Lightweight Stream Cipher for Power-constrained Devices. *IACR Trans. Symmetric Cryptol.*, 2017(1):45–79, 2017.

[HR08] Michal Hojsík and Bohuslav Rudolf. Differential Fault Analysis of Trivium. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2008.

[HS04] Jonathan J Hoch and Adi Shamir. Fault analysis of stream ciphers. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 240–253. Springer, 2004.

[HS08] Ghaith Hammouri and Berk Sunar. PUF-HB: A tamper-resilient HB based authentication protocol. In *Applied Cryptography and Network Security*, pages 346–365. Springer, 2008.

[HSH11] Tzipora Halevi, Nitesh Saxena, and Shai Halevi. Tree-based hb protocols for privacy-preserving authentication of rfid tags. *J. Comput. Secur.*, 19(2):343–363, April 2011.

[ISO12] ISO/IEC 29192-2:2012. Information technology - Security techniques - Lightweight cryptography - Part 2: Block ciphers, 2012.

[Jön02] Fredrik Jönsson. *Some results on fast correlation attacks*. Department of Information Technology, Lund Univeristy, 2002.

[JW05] A. Juels and S. A. Weis. Authenticating pervasive devices with human protocols. In *Proceedings of Crypto 2005*, volume 3621 of *LNCS*, pages 293–308. Springer, 2005.

[KDH$^+$12] Stéphanie Kerckhof, François Durvaux, Cédric Hocquet, David Bol, and François-Xavier Standaert. Towards Green Cryptography: A Comparison of Lightweight Ciphers from the Energy Viewpoint. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 390–407. Springer, 2012.

[KHK06] Shahram Khazaei, Mahdi M Hasanzadeh, and Mohammad S Kiaei. Linear Sequential Circuit Approximation of Grain and Trivium Stream Ciphers. *IACR Cryptology ePrint Archive*, 2006:141, 2006.

[KKN$^+$02] Jesung Kim, Jong Min Kim, Sam H Noh, Sang Lyul Min, and Yookun Cho. A space-efficient flash translation layer for CompactFlash systems. *IEEE Transactions on Consumer Electronics*, 48(2):366–375, 2002.

[KLPR10] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.

[KMNP10] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2010.

[KPC$^+$11] Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi. Efficient authentication from hard learning problems. In *Advances in Cryptology–EUROCRYPT 2011*, pages 7–26. Springer, 2011.

[Kra94] Hugo Krawczyk. LFSR-based hashing and authentication. In *Advances in Cryptology—CRYPTO'94*, pages 129–139. Springer, 1994.

[KS06] Jonathan Katz and Ji Sun Shin. Parallel and concurrent security of the HB and HB$^+$ protocols. In *Advances in Cryptology-EUROCRYPT 2006*, pages 73–87. Springer, 2006.

[KS17] Jonathan Katz and Hovav Shacham, editors. *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*. Springer, 2017.

[KSS10] Jonathan Katz, Ji Sun Shin, and Adam Smith. Parallel and concurrent security of the HB and HB$^+$ protocols. *Journal of cryptology*, 23(3):402–421, 2010.

[LCK10] Kyoung-Su Lee, Jung-Hoon Chun, and Kee-Won Kwon. A low power CMOS compatible embedded EEPROM for passive RFID tag. *Microelectronics Journal*, 41(10):662–668, 2010.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In *Security and Cryptography for Networks*, pages 348–359. Springer, 2006.

[Liu17] Meicheng Liu. Degree Evaluation of NFSR-Based Cryptosystems. In Katz and Shacham [KS17], pages 227–249.

[LMM08] Xuefei Leng, Keith Mayes, and Konstantinos Markantonakis. HB-MP+ protocol: An improvement on the HB-MP protocol. In *RFID, 2008 IEEE International Conference on*, pages 118–124. IEEE, 2008.

[LN15] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of full Sprout. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 663–682. Springer, 2015.

[LYR15] Gefei Li, Yuval Yarom, and Damith Chinthana Ranasinghe. Exploiting transformations of the Galois configuration to improve guess-and-determine attacks on NFSRs. *IACR Cryptology ePrint Archive*, 2015:1045, 2015.

[Mac14] Macronix International Co.. Ltd. Introduction to NAND in Embedded Systems, 2014.

[MAM16] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On Ciphers that Continuously Access the Non-Volatile Key. *IACR Trans. Symmetric Cryptol.*, 2016(2):52–79, 2016.

[MAM17] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On ciphers that continuously access the non-volatile key. *IACR Transactions on Symmetric Cryptology*, 2016(2):52–79, 2017.

[Mas69] James Massey. Shift-register synthesis and bch decoding. *IEEE transactions on Information Theory*, 15(1):122–127, 1969.

[MB07] Alexander Maximov and Alex Biryukov. Two Trivial Attacks on Trivium. In *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, pages 36–55, 2007.

[MCP08] Cameron McDonald, Chris Charnes, and Josef Pieprzyk. An algebraic analysis of Trivium ciphers based on the boolean satisfiability problem. In *Proceedings of the 4th International Workshop on Boolean Functions: Cryptography and Applications*, pages 173–184, 2008.

[MD10] Shohreh Sharif Mansouri and Elena Dubrova. An improved hardware implementation of the Grain stream cipher. In *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pages 433 –440, sept. 2010.

[MD13] Shohreh Sharif Mansouri and Elena Dubrova. An improved hardware implementation of the Grain-128a stream cipher. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology – ICISC 2012*, volume 7839 of *Lecture Notes in Computer Science*, pages 278–292. Springer Berlin Heidelberg, 2013.

[MGAM17] Vasily Mikhalev, Laurent Gomez, Frederik Armknecht, and José Márquez. Towards end-to-end data protection in low-power networks. In *Computer Security*, pages 3–18. Springer, 2017.

[Mic] Micron Technology, Inc. NOR | NAND Flash Guide. https://www.micron.com/~/media/documents/products/product-flyer/flyer_nor_nand_flash_guide.pdf.

[Mic01] Microchip. *KeeLoq Code Hopping Encoder, Product data sheet*, 2001. http://ww1.microchip.com/downloads/en/DeviceDoc/21137f.pdf.

[Mic11] Microchip Technology Inc. *1K-16K UNI/O Serial EEPROM Family Data Sheet*, 2011. http://ww1.microchip.com/downloads/en/DeviceDoc/22067J.pdf.

[Mic14] Microchip Technology Inc. *SPI Serial EEPROM Family Data Sheet*, 2014. http://ww1.microchip.com/downloads/en/DeviceDoc/22040A.pdf.

[MME$^+$11] Honorio Martin, Enrique San Millán, Luis Entrena, Julio César Hernández Castro, and Pedro Peris-Lopez. Akari-x: A pseudorandom number generator for secure lightweight systems. In *IOLTS*, pages 228–233, 2011.

[MP07] Jorge Munilla and Alberto Peinado. HB-MP: A further step in the HB-family of lightweight authentication protocols. *Computer Networks*, 51(9):2262–2267, 2007.

[MPL$^+$11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: a very compact and a threshold implementation of aes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 69–88. Springer, 2011.

[MS89] Willi Meier and Othmar Staffelbach. Nonlinearity criteria for cryptographic functions. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 549–562. Springer, 1989.

[MSBD15] Subhamoy Maitra, Santanu Sarkar, Anubhab Baksi, and Pramit Dey. Key recovery from state information of Sprout: Application to cryptanalysis and fault attack. *IACR Cryptology ePrint Archive*, 2015:236, 2015.

[MSGAHJ13] Joan Melià-Seguí, Joaquin Garcia-Alfaro, and Herrera-Joancomartí. J3Gen: A PRNG for low-cost passive RFID. *Sensors*, 13(3):3816–3830, 2013.

[MTSV10] Mukundan Madhavan, Andrew Thangaraj, Yogesh Sankarasubramanian, and Kapali Viswanathan. NLHB: A non-linear Hopper-Blum protocol. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 2498–2502. IEEE, 2010.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[Nie99] Harald Niederreiter. Some computable complexity measures for binary sequences. In *Sequences and their Applications*, pages 67–78. Springer, 1999.

[NKTZ12] Andrey Nuykin, Alexander Kravtsov, Sergey Timoshin, and Igor Zubov. A low cost EEPROM design for passive RFID tags. In *Communications and Electronics (ICCE), 2012 Fourth International Conference on*, pages 443–446. IEEE, 2012.

[NXDH06] Yan Na, Tan Xi, Zhao Dixian, and Min Hao. An Ultra-Low-Power Embedded EEPROM for Passive RFID Tags. *Chinese Journal of Semiconductors*, 27(6), 2006.

[NXP11] NXP Semiconductors. *MIFARE Classic 1K - Mainstream contactless smart card IC for fast and easy solution development, Product short data sheet*, 2011. `http://www.nxp.com/documents/short_data_sheet/MF1S50YYX.pdf`.

[NXP14] NXP Semiconductors. *HITAG S transponder IC., Product short data sheet*, 2014. `http://www.nxp.com/documents/short_data_sheet/HTSICH56_48_SDS.pdf`.

[NXP15] NXP Semiconductors. *HITAG μ transponder IC.HTMS1x01; HTMS8x01. Product data sheet*, 2015. `http://www.nxp.com/documents/data_sheet/HTMS1X01_8X01.pdf`.

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology-CRYPTO 2003*, pages 617–630. Springer, 2003.

[On 14] On Semiconductor. *1-Kb, 2-Kb, 4-Kb, 8-Kb and 16-Kb I2C CMOS Serial EEPROM, Data sheet*, 2014. `http://www.onsemi.com/pub_link/Collateral/CAT24C01-D.PDF`.

[OOV08] Khaled Ouafi, Raphael Overbeck, and Serge Vaudenay. On the Security of HB$^{\#}$ against a Man-in-the-Middle Attack. In *Advances in Cryptology-ASIACRYPT 2008*, pages 108–124. Springer, 2008.

[Pie] K. Pietrzak. Subspace LWE. Manuscript available at `http://homepages.cwi.nl/~pietrzak/publications/SLWE.pdf`.

[PLHCETR09] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador, and A. Ribagorda. LAMED - a PRNG for EPC Class-1 Generation-2 RFID specification. *Comput. Stand. Interfaces*, 31(1):88–97, January 2009.

[PMK$^+$11] A. Poschmann, A. Moradi, K. Khoo, C. Lim, H. Wang, and S. Ling. Side-channel resistant crypto for less than 2,300 GE. *Journal of Cryptology*, 24(2):322–345, 2011.

[Pos09] Axel York Poschmann. Lightweight cryptography: Cryptographic engineering for a pervasive world, 2009.

[PSS$^+$08] Jivan Parab, Santosh A Shinde, Vinod G Shelake, Rajanish K Kamat, and Gourish M Naik. *Practical aspects of embedded system design using microcontrollers*. Springer Science & Business Media, 2008.

[PT07] Selwyn Piramuthu and Yu-Ju TU. Modified HB authentication protocol. *WEWoRC*, pages 41–44, 2007.

[REC05] Damith C. Ranasinghe, Daniel W. Engels, and Peter H. Cole. Low-cost rfid systems: Confronting security and privacy. In *In: Auto-ID Labs Research Workshop*. Portal, 2005.

[RG12] Panagiotis Rizomiliotis and Stefanos Gritzalis. GHB$^\#$: A provably secure HB-like lightweight authentication protocol. In *Applied Cryptography and Network Security*, pages 489–506. Springer, 2012.

[Riz09] Panagiotis Rizomiliotis. HB-MAC: Improving the Random- HB$^\#$ authentication protocol. In *Trust, Privacy and Security in Digital Business*, pages 159–168. Springer, 2009.

[RPLP08] Carsten Rolfes, Axel Poschmann, Gregor Le, and Christof Paar. Ultralightweight implementations for smart devices - security for 1000 gate equivalents. In *in Proceedings of the 8th Smart Card Research and Advanced Application IFIP Conference – CARDIS 2008, ser. LNCS*, pages 89–103. Springer-Verlag, 2008.

[Sag] Sage mathematics software. http://www.sagemath.org/.

[SCU11] J. Susini, H. Chabanne, and P. Urien. *RFID and the Internet of Things*, chapter RFID and the Internet of Things, page 304. ISTE - John Wiley & Sons, 2011.

[SE12] Markku-Juhani O. Saarinen and Daniel W. Engels. A do-it-all-cipher for rfid: Design requirements (extended abstract). *IACR Cryptology ePrint Archive*, 2012:317, 2012. informal publication.

[SFP08] Ilaria Simonetti, Jean-Charles Faugere, and Ludovic Perret. Algebraic attack against trivium. In *First International Conference on Symbolic Computation and Cryptography, SCC*, volume 8, pages 95–102, 2008.

[Sha49] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[Sie84] Thomas Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications (corresp.). *IEEE Transactions on Information theory*, 30(5):776–780, 1984.

[SKII12] Xuedi Song, Kazukuni Kobara, Kentaro Imafuku, and Hideki Imai. HB^b protocol for lightweight authentication; Its information theoretic indistinguishability against MITM attack watching reader's response. In *Information Theory and its Applications (ISITA), 2012 International Symposium on*, pages 536–540. IEEE, 2012.

[SSA$^+$07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher clefia. In *International workshop on fast software encryption*, pages 181–195. Springer, 2007.

[TBM07] C. Tokunaga, D. Blaauw, and T. Mudge. True random number generator with a metastability-based quality control. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 404–611, Feb 2007.

[TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube Attacks on Non-Blackbox Polynomials Based on Division Property. In Katz and Shacham [KS17], pages 250–279.

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear Invariant Attack - Practical Attack on Full SCREAM, iSCREAM, and Midori64. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, pages 3–33, 2016.

[TWB$^+$14] Sui-Guan Teo, Kenneth Koon-Ho Wong, Harry Bartlett, Leonie Simpson, and Ed Dawson. Algebraic analysis of Trivium-like ciphers. In *Proceedings of the Twelfth Australasian Information Security Conference-Volume 149*, pages 77–81. Australian Computer Society, Inc., 2014.

[VGE15] Roel Verdult, Flavio D Garcia, and Baris Ege. Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer. In *Supplement to the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 703–718, 2015.

[WZ11] Wenling Wu and Lei Zhang. LBlock: A lightweight block cipher. In *Applied Cryptography and Network Security*, volume 6715 of *LNCS*, pages 327–344. Springer, 2011.

[ZG15] Bin Zhang and Xinxin Gong. Another tradeoff attack on sprout-like stream ciphers. *Accepted at AsiaCrypt 2015*, 2015.

Erklärung der Urheberschaft

Eidesstattliche Versicherung gemäß § 7 Absatz 2 Buchstabe c) der Promotionsordnung der Universität Mannheim (Stand: 11. Juni 2012) zur Erlangung des Doktorgrades der Naturwissenschaften:

1. Bei der eingereichten Dissertation zum Thema
   Lightweight Symmetric Cryptography
   handelt es sich um mein eigenständig erstelltes eigenes Werk.

2. Ich habe nur die angegebenen Quellen und Hilfsmittel benutzt und mich keiner unzulässigen Hilfe Dritter bedient. Insbesondere habe ich wörtliche Zitate aus anderen Werken als solche kenntlich gemacht.

3. Die Arbeit oder Teile davon habe ich bislang nicht an einer Hochschule des In- oder Auslands als Bestandteil einer Prüfungs- oder Qualifikationsleistung vorgelegt.

4. Die Richtigkeit der vorstehenden Erklärung bestätige ich.

5. Die Bedeutung der eidesstattlichen Versicherung und die strafrechtlichen Folgen einer unrichtigen oder unvollständigen eidesstattlichen Versicherung sind mir bekannt.

Ich versichere an Eides statt, dass ich nach bestem Wissen die reine Wahrheit erklärt und nichts verschwiegen habe.

Ort, Datum	Unterschrift