Doctoral Thesis

Cryptanalysis of hardware-oriented ciphers the Knapsack generator, and SHA-1

Author(s):
Knellwolf, Simon

Publication Date:
2012

Permanent Link:
https://doi.org/10.3929/ethz-a-007554419

Rights / License:
In Copyright - Non-Commercial Use Permitted
Cryptanalysis of Hardware-Oriented Ciphers
the Knapsack Generator, and SHA-1

A dissertation submitted to

ETH Zurich

for the degree of
Doctor of Science

presented by

Simon Knellwolf
MSc ETH Mathematics

born May 25, 1982
citizen of Herisau (AR)

accepted on the recommendation of

Prof. Dr. Ueli Maurer, examiner
Prof. Dr. Willi Meier, co-examiner
Prof. Dr. Lars Knudsen, co-examiner

2012
Acknowledgments

I am very grateful to Ueli Maurer and Willi Meier for accepting me as a PhD student. Thank you a lot for your great support during the last three years. It was a privilege to work with Willi in the same office. His passion for cryptanalysis passed on very quickly and I profited a lot from his enormous experience. On the other hand, Ueli opened my mind to the many other fields of cryptography.

It is an honor to me to have Lars Knudsen as a co-examiner. Many thanks for investing your time. Many thanks also to my co-authors. It was a pleasure to work with Jean-Philippe and Shahram in the beginning of my PhD, and it was a great chance that María came to Switzerland for her post-doc. We worked hard on conditional differential cryptanalysis and I gladly remember those few hours when we believed having broken Trivium with “infinitely” many rounds. One of the highlights of my PhD was the internship at Microsoft Research Redmond. I am very grateful to Dmitry, for having me invited and for having been such a great mentor.

I acknowledge the Hasler Foundation and the University of Applied Sciences Northern Switzerland (FHNW) for paying my salary and for providing a workplace that was only a few minutes away from our apartment. At FHNW I was a member of the Institute for Aerosol and Sensor Technology (IAST). Many thanks to all members of the institute, for the valuable discussions and for being such an interested audience during my talks at the IAST-meetings. Many thanks also to the members of Ueli’s group at ETH Zurich. I appreciated your ICS-talks, the Christmas events, and running for the “Alice and Bob”-Team at the SOLA-Stafette.

Finally, I thank my parents, brothers, and close friends for their constant support in various phases of my life. No words can express my gratitude to Yvonne, for being my girlfriend since many years and for being such a wonderful mother to Anna.
Abstract

Symmetric key cryptographic algorithms provide confidentiality, integrity, and authentication in modern communication systems. Our confidence in these algorithms is largely based on the fact that intense cryptanalysis has been carried out over several years without revealing any weakness. This thesis makes three independent contributions to the cryptanalysis of symmetric key primitives and hash functions. First, conditional differential cryptanalysis is proposed as a general framework for the analysis of a large class of hardware-oriented ciphers that build on non-linear feedback shift registers. As main applications, various improved distinguishing and key recovery attacks on reduced-round variants of the stream ciphers Grain v1, Grain-128, Grain-128a, and Trivium are obtained. Second, the security of the knapsack generator, a stream cipher construction proposed by Rueppel and Massey in 1985, is studied. A surprisingly effective guess-and-determine attack is shown that recovers large parts of the $n^2 + n$ secret key bits if only $n$ bits are known. Quite different from standard techniques of symmetric cryptanalysis, our approach uses Babai’s closest vertex algorithm and lattice reduction. Finally, meet-in-the-middle preimage attacks on hash functions are revisited. A new differential cryptanalytic perspective is proposed which is very suitable for hash functions with linear message expansion. As an application, previous preimage attacks against reduced variants of SHA-1 are significantly improved.
Zusammenfassung

## 5 Analysis of the Knapsack Generator

5.1 Introduction .................................. 67
5.2 The Knapsack Generator ....................... 68
5.3 Attack Description .............................. 71
5.4 Finding Good Approximation Matrices ........... 74
5.5 Experimental Results ........................... 76
5.6 Summary and Conclusion ........................ 79

## 6 New Preimage Attacks Against Reduced SHA-1

6.1 Introduction .................................. 81
6.2 Differential Meet-in-the-Middle Framework ........ 84
6.3 Application to SHA-1 ............................ 91
6.4 Accelerated Brute-Force Search .................. 97
6.5 Summary and Conclusion ........................ 97

### A On the Applicability of the Cube Attack

101

### B Collisions for CubeHash Variants

103

### C Attack Parameters for SHA-1

107

Bibliography ....................................... 121
Chapter 1

Introduction

The transmission of secret information is a very old problem. Until the end of World War II, the problem arose almost exclusively in military and diplomatic applications. Governments have operated special departments, responsible for inventing secret codes and in parallel, breaking the codes of others. At this time, cryptography was an art and only a handful of artists worked in the field.

Modern Cryptography

Modern cryptography is different in many aspects. It is not only concerned with encryption, but with all aspects of a secure and reliable treatment of information. Besides confidentiality, this includes data integrity, authentication, non-repudiation, and many other security goals. Notions of security have been formalized which allows to prove the security of some systems under well-defined assumptions. This was essential for the transition of cryptography as an art to a legitimate science.¹ The introduction of scientific reasoning into cryptography is often attributed to Shannon’s 1949 paper Communication theory of secrecy systems [109]. Further milestones in the early times of modern cryptography were the introduction of public-key cryptography in 1976 by Diffie and Hellman [38], the publication of the first cryptographic standard, the Data Encryption Standard (DES), in 1977 by the U.S.

¹The comparison of art and science is found in many introductory texts. The earliest reference we found is the famous paper by Luby and Rackoff [80].
National Institute of Standards and Technology (NIST), and the discovery of the first practical public-key encryption and signature scheme (RSA) in 1978 by Rivest, Shamir, and Adleman [96]. In 1981, the first CRYPTO conference was held, and one year later, the International Association for Cryptologic Research (IACR) was formed. Today, the IACR annually organizes three flagship conferences and four workshops in different fields and the development of cryptography is driven by academic research rather than by governmental military departments. Nevertheless, the applicability of results in practice is more important in cryptography than in other sciences. This is particularly true for the field of symmetric key cryptography, where the design of secure and efficient algorithms for everyday use is a major goal. The design of such algorithms goes hand in hand, with their analysis which is the subject of this thesis.

The Role of Symmetric Cryptanalysis

The design and analysis of symmetric key primitives is a field of cryptography that has preserved some of the original flavor of code development and code breaking. However, designing a modern cipher is much more a delicate engineering task rather than an art. The algorithms are not only required to be secure, but fast and easily implementable on different platforms.

Although we know how an ideal block cipher (or stream cipher, or hash function) should behave, we are quite far away from efficient constructions which provably achieve this behavior under practical assumptions. As in pre-modern times of cryptography, an algorithm is considered secure as long as it is not broken. What has changed a lot is the fact that, in accordance with Kerckhoffs’ principle [60], most modern cryptographic algorithms are publicly known. Different design approaches are analyzed, discussed, and compared by a lively community of people from academia and private industry. Quite a number of general construction principles have emerged and have been studied from different perspectives. Prominent examples are the Feistel network [45,80,82], or the Merkle-Damgård construction [37,91]. This is complemented by technical background on individual building blocks, such as non-linearity of modular addition over the binary field [100,112], properties of certain S-boxes [103], and many other results in this spirit.

In spite of well understood construction principles and solid background results, putting things together in order to obtain a secure and efficient algorithm is not an easy task. It requires care, experience, and luck. In this
context, cryptanalysis plays an important role assigned with the following tasks:

**Detect flaws in newly proposed designs.** For good reasons, a new proposal is not trusted until it underwent intense cryptanalysis, even when it was designed by experienced cryptographers. The interplay between designers and cryptanalysts turned out to be very valuable in the past. For example in the case of the stream cipher Grain v1 [54], a flaw in the initial design, has been pointed out by third-party cryptanalysts [14]. Fixing the flaw required only a relatively small modification and finally, Grain v1 was selected for the eSTREAM portfolio of hardware-oriented stream ciphers [43]. It happened that serious flaws were not detected by cryptanalysts for many years. An example is the hash function MD5 which turned out to have very weak collision resistance only after thirteen years of its publication [119]. Fortunately, this seems not to be the general rule.

**Give hints to optimize the security/efficiency tradeoff.** Many cryptographic algorithms admit one or several parameters to trade between security and efficiency (e.g. the number of rounds). Detailed cryptanalysis of reduced-round variants is indispensable to find a good balance. As an example, Daemen and Rijmen write in their final specification of Rijndael (after its selection as the AES): “We have determined the number of rounds by looking at the maximum number of rounds for which shortcut attacks have been found and added a considerable security margin.” [36, Section 7.6].

**Develop general attack strategies and countermeasures.** Resistance against known attack strategies is an important design criteria for new algorithms. This is impressively demonstrated by the resistance of DES against differential cryptanalysis. It is known that the designers of DES were aware of differential cryptanalysis long before it has been publicly introduced and resistance against this type of attack was a design goal [34]. Due to the carefully chosen S-boxes, DES was indeed very resistant while other ciphers at that time were completely broken. As a matter of fact, modern ciphers are designed to resist differential cryptanalysis and other general techniques such as linear cryptanalysis or meet-in-the-middle attacks.

**Provide an appropriate estimate of security in different scenarios.** Compromising a $\kappa$-bit encryption algorithm is generally expected to be roughly
as expensive as testing all $2^κ$ keys. Implicitly, a scenario is assumed where the adversary knows only a few bytes of plaintext with the corresponding ciphertext. In general, this has no implications on the security in other scenarios, for example when megabytes of data or side channel information (information gained from the physical implementation) are available, or when the key is not chosen properly. A well known example is the use of the stream cipher RC4 in the Wired Equivalent Privacy (WEP) protocol. Eavesdropping to a WEP-protected network, the adversary can recover 104 bits of the secret key within a few seconds, exploiting the key derivativation algorithm [49] that was not intended by Rivest when he designed RC4.

As a summary, symmetric cryptanalysis is crucial to have trustworthy and efficient symmetric key algorithms and hash functions for practical use. This was also recognized by the NIST which successfully focuses the design and cryptanalysis efforts of the community by organizing competitions such as the AES competition or the ongoing SHA-3 competition.

Publications and Outline of the Thesis

This thesis has three parts with independent contributions to symmetric cryptanalysis. The first part is on conditional differential cryptanalysis of hardware-oriented ciphers which build on non-linear feedback shift registers. A general analysis approach is proposed and applied to the KATAN/ KTANTAN family of block ciphers and to the stream ciphers Trivium, Grain v1, Grain-128, and Grain-128a. Most ideas and results have appeared in:


The second part is a security analysis of the knapsack generator, a stream cipher construction proposed by Rueppel and Massey in 1985. Quite different from standard techniques of symmetric cryptanalysis, our approach uses Babai’s closest vertex algorithm and lattice reduction. This chapter is a modified version of:


The third part revisits meet-in-the-middle preimage attacks on hash functions. We propose a new framework from a differential cryptanalytic perspective. As a main application, we obtain significantly better preimage attacks against reduced variants of SHA-1. This work will appear in:


The appendix contains an observation on the cube attack, a summary of prize-winning collision attacks on the SHA-3 candidate CubeHash, and the detailed parameters for the SHA-1 preimage attacks. The results on Cube-Hash have appeared in:


Contributions and workshop publications which are not part of this thesis:


Chapter 2

Preliminaries of Symmetric Cryptanalysis

We use “symmetric cryptanalysis” as a shorthand notation for cryptanalysis of symmetric key primitives and hash functions. Hash functions are included, because their design is heavily inspired by block ciphers and similar analysis techniques apply. This chapter aims at introducing some common terminology. It describes the target primitives, the notion of a cryptanalytic attack, and differential cryptanalysis as a principal analysis techniques. We don’t attempt completeness, but focus on those aspects which are relevant for the later parts of this thesis.

2.1 Symmetric Key Primitives

Algorithms for symmetric key encryption can be divided into two classes: block ciphers and stream ciphers. Both are mainly used for symmetric encryption, but have important other applications as well, for example the CBC-MAC for block cipher based message authentication. Traditionally, stream ciphers were mainly used for hardware applications and for high throughput requirements, block ciphers for software applications and very high security requirements. However, this separation is not that strict anymore.
2.1.1 Block Ciphers

A block cipher transforms an $n$-bit plaintext block into an $n$-bit ciphertext block under a $\kappa$-bit secret key, $n$ is called the block length and $\kappa$ the key length. To allow unique decryption, the transformation must be invertible.

**Definition.** A block cipher is a function $E : \{0, 1\}^\kappa \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, such that for each $K \in \{0, 1\}^\kappa$, $E(K, \cdot)$ is a permutation on $\{0, 1\}^n$.

**Expected Behavior.** The ideal behavior of a block cipher is captured by the notion of a random permutation. An adversary not knowing $K$ should not be able to distinguish the input/output behavior of $E(K, \cdot)$ from a permutation that was chosen uniformly at random from the set of all permutations on $\{0, 1\}^n$.

The random permutation characterization gives an idea of the difficulty of constructing a good block cipher. The challenge is to describe a family of $2^\kappa$ permutations such that randomly choosing a member of this family looks like randomly choosing an element from the set of all $2^n!$ permutations.

**Design Principles.** Most block ciphers are iterated constructions. They perform several rounds, where each round is an invertible transformation that depends on a round key. Figure 2.1 illustrates the basic structure of an iterated block cipher with $N$ rounds. The transformation of the key $K$ into the $N$ round keys $K_1, \ldots, K_N$ is called the key schedule or the key expansion. The plaintext $P$ and the first round key $K_1$ are input to the first round function $R_1$. The output of $R_1$ is called the state after one round, it is the input to the second round, and so on. The output of the last round function is output as the ciphertext $C = E(K, P)$. The round transformations have to be carefully designed in order to provide good diffusion and confusion. Most ciphers follow one of two classical construction principles: Feistel ciphers and substitution-permutation networks. Another classification can be made with respect to how the ciphers achieve diffusion and confusion: AXR-constructions use a combination of addition, exclusive-or, and word-wise rotations, while S-Box-based constructions use diffusion layers (often linear) in combination with well-chosen non-linear substitution-boxes.\(^2\)

\(^2\)A more formal characterization is required for security proofs. We refer to random systems framework introduced by Maurer [83].

\(^3\)We refer to The Block Cipher Companion by Knudsen and Robshaw [73] for the state of the art in block cipher design and analysis.
Prominent examples of modern block ciphers are Rijndael (chosen as the AES in 2001), Serpent (the strongest competitor of Rijndael in the AES competition), IDEA (not Feistel and not SP-network), Kasumi (used in mobile phone communication standards), and PRESENT (dedicated to light-weight hardware implementations). In Chapter 3, the KATAN/KTANTAN family of hardware-oriented block ciphers is analyzed, whose design is heavily inspired by stream ciphers.

2.1.2 Stream Ciphers

Stream ciphers are divided into synchronous and self-synchronizing stream ciphers. Only synchronous stream ciphers are considered in this thesis and occasionally the word “synchronous” will be omitted.\(^4\) A synchronous stream cipher maintains an internal state which is initialized by a key and an initial value (IV). After an initialization phase where the key and the IV are mixed, keystream is generated and the state is continuously updated. Most commonly, encryption and decryption is done by adding the keystream to the plaintext or the ciphertext, respectively.

**Definition.** A synchronous stream cipher with key length \(\kappa\) and IV length \(n\) consists of an internal state of \(s\) bits, a state initialization function \(G : \{0,1\}^\kappa \times \{0,1\}^n \rightarrow \{0,1\}^s\), a state update function \(H : \{0,1\}^s \rightarrow \{0,1\}^s\), and an output function \(F : \{0,1\}^s \rightarrow \{0,1\}^m\) (typically, \(m = 1\)). On input \((K, IV)\), an

\(^4\)We refer to [89] for self-synchronous stream ciphers.
initial state $S_0 = G(K, IV)$ is computed. Then, for $i = 1, 2, \ldots$, $z_i = F(S_{i-1})$ is output and the state is updated as $S_i = H(S_{i-1})$.

**Expected Behavior.** An adversary not knowing the key (the IV is typically assumed to be known) should not be able to distinguish the keystream from a sequence of independent uniformly distributed bits. Moreover, for different IVs, the keystreams should look independent from each other.

**Design Principles.** As a general rule, the initialization phase should produce a state whose bits depend in a complicated way from each bit of the key and the IV (this process is called key/IV mixing). The challenge is similar to the design of a block cipher, $G$ should look like a random selection of $2^\kappa$ functions $\{0, 1\}^n \rightarrow \{0, 1\}^s$. The role of the output function is to prevent that the state can be recovered from the keystream.

Many stream ciphers, namely the hardware-oriented ones, are based on binary feedback shift registers. Well known examples are E0 (used in Bluetooth technology) and A5/1 (used in GSM networks). A prominent exception is RC4 which operates on bytes and is very fast in software. More recent software oriented stream cipher designs have emerged from the ECRYPT stream cipher project eSTREAM [43].

**Feedback Shift Registers.** Binary feedback shift registers are the main building block of hardware-oriented stream ciphers (“binary” is omitted in the following). A feedback shift register of length $\ell$ consists of an internal state of $\ell$ bits, denoted by $(s_0, \ldots, s_{\ell-1})$, and an update function $g : \{0, 1\}^\ell \rightarrow \{0, 1\}$. A clock or update of the register means the following operation:

$$t \leftarrow g(s_0, \ldots, s_{\ell-1})$$

$$(s_0, \ldots, s_{\ell-1}) \leftarrow (t, s_0, \ldots, s_{\ell-2}).$$

One bit is discarded and a new bit is generated by $g$. The discarded bit is sometimes considered as the output of the register. The register is called a linear feedback shift register (LFSR) if $g$ is a linear function, otherwise it is called a non-linear feedback shift register (NLFSR). A register that, starting from any non-zero state, returns to this state only after $2^\ell - 1$ clocks is said to

---

5For stream ciphers, a textbook in the spirit of *The Block Cipher Companion* is missing at this time. A collection of modern design approaches can be found in [97]. Classical references are Rueppel’s thesis [100] and the chapter on stream ciphers in [89].
2.2 Hash Functions

A cryptographic hash function is an unkeyed function that takes an input of arbitrary length, called the message, and returns a fixed length output of \( n \) bits, called the hash value. The hash value can be considered as a short identification of the message and is sometimes interpreted as its digital fingerprint. Typical output length are 128, 160, 256, or 512 bits.

**Definition.** Let \( \{0, 1\}^* \) be the set of arbitrary length binary strings. A hash function with output length \( n \) is a function \( H : \{0, 1\}^* \to \{0, 1\}^n \).

\(^6\)We refer to [89] and Fischer’s thesis [47] for these constructions and attacks.
Expected Behavior. A hash function should behave like a random oracle [12], that is, a publicly accessible black box that on each input returns a uniformly chosen string of $n$ bits. In particular this implies the following classical security properties:

- Collision resistance: It should not be feasible to find two different messages $M$ and $M'$ such that $H(M) = H(M')$.
- Preimage resistance: Given a hash value $h$, it should not be feasible to find a message $M$ such that $H(M) = h$.
- Second preimage resistance: Given a message $M$, it should not be feasible to find a second message $M'$ such that $H(M) = H(M')$.

Design Principles. It is an old idea to build hash functions from block ciphers. The idea goes back to Rabin [95] who proposed a hash function based on DES. Other proposals soon followed and, generalizing from previous work, Preneel, Govaerts, and Vandewalle [94] systematically considered the construction of a hash function from a block cipher. But also dedicated hash functions such as MD5, SHA-1, and the SHA-2 family are implicitly built on block ciphers. They use the Merkle-Damgård construction [37, 91] to transform a function with fixed length input $F : \{0, 1\}^\kappa \times \{0, 1\}^n \to \{0, 1\}^n$, called the compression function, into a function with arbitrary length input: a message is padded and split into $\kappa$-bit message blocks $M = M_1 \parallel M_2 \parallel \ldots \parallel M_\ell$, then, each block is processed separately by the compression function as illustrated in Fig. 2.2.

The popularity of the Merkle-Damgård construction comes from the fact that it preserves the collision resistance of the compression function. Unfortunately, the construction is vulnerable to length extension attacks and it has other undesirable properties [35, 59]. Promising new design approaches emerged from the ongoing SHA-3 competition. A notable example is the sponge construction by Bertoni, Daemen, Peeters, and Van Assche [17].

---

7Formalizing the ideal behavior of a hash function is less straightforward than for keyed primitives. The problem is the absence of a secret key. The notion of indifferentiability, introduced by Maurer, Renner, and Holenstein [84], provides a framework to circumvent this problem. Random oracle indifferentiability became an important security argument for candidates of the SHA-3 competition [3].

8There are several ways to formally define the below security properties. Rogaway and Shrimpton [98] give seven different definitions and work out implications and separations among them.
SHA-3 Competition. In 2007, the NIST announced a call for the design of a new cryptographic hash function standard in response to serious vulnerabilities identified in the existing hash standards MD5 and SHA-1. For the first round, 51 submission got accepted and published by the NIST. After a short public comment period, sixteen second round candidates got selected which received a lot of cryptanalytic attention. In December 2010, the number of candidates was reduced to five: BLAKE, Grøstl, JH, Keccak, and Skein. The winner is expected to be announced in the mid of 2012.

2.3 Cryptanalytic Attacks

It turns out that the word “attack” is used very frequently in papers on symmetric cryptanalysis.\footnote{This is true also for this thesis. We used the word already 19 times until here, and we are going to use it 269 more times.} Below we describe the meaning of the word in the context of this thesis.

2.3.1 Notions of Security

Different notions of security are used in different fields of cryptography. In the setting of provable security, information-theoretic and asymptotic security notions are commonly used. A system is information-theoretically secure if it cannot be broken by any (non-telepathic) adversary even when she has unlimited computational resources. The classical example of an

![Figure 2.2: The Merkle-Damgård construction.](image-url)
information-theoretic scheme is the one-time pad. When considering asymptotic security, the adversary’s resources and her success probability are considered as functions of a security parameter, for example the key length, and a system is secure, if the success probability is negligible for any adversary with polynomially bounded resources.

Both of these notions are not suitable to evaluate symmetric key primitives with fixed parameters (e.g. AES with block length $n = 128$ bits and key length $\kappa = 128$). The security of such a primitive is evaluated by estimating the concrete amount of resources that is required to achieve a certain goal (e.g. key recovery). Three types of resources are typically considered separately: time, memory, and data. A security proof would have to provide concrete lower bounds on these resources.

2.3.2 What is an Attack?

Considering practical security, one could say that an attack is a technique that achieves a certain goal with a feasible amount of resources. However, such a notion would heavily depend on available technology. The following notion does not depend on technology and has become standard in the cryptanalytic community: an attack is a technique that achieves a certain goal in a certain scenario with less resources than the best generic technique would require to achieve the same goal in the same scenario. A generic technique is a technique that does not exploit any internal properties of the target algorithm. This is a very generous characterization and leaves room for attacks without or with very small implications on the security of an algorithm. Moreover, the word “attack” is also used when considering reduced-round variants of target algorithms. Whether an attack is relevant or not heavily depends on how much the target primitive is reduced, on the attack goal that is achieved, and on the attack scenario which is required.

2.3.3 Attack Goals and Scenarios

Common attack goals, attack scenarios, and generic techniques, are discussed in the following.

**Key Recovery Attacks.** Recovering the secret key is the most devastating attack against keyed primitives and is usually the most challenging attack
2.3 Cryptanalytic Attacks

The generic key recovery technique is brute-force search. The adversary needs to observe $\kappa$ bits of input/output information and then simply tries all possible keys, which requires $2^\kappa$ computations of the cipher and essentially no memory. If the input value can be chosen, the adversary can precompute the list of $2^\kappa$ outputs for all different keys. This variant requires memory to store the $2^\kappa$ values, but the actual attack is a single table lookup. Intermediate variants are known as time-memory tradeoffs. The classical time-memory tradeoff is described by Hellman [55].

For stream ciphers, state recovery attacks are sometimes considered. For most ciphers this boils down to a key recovery attack, because their initialization process is invertible.

**Distinguishing Attacks.** Distinguishing an algorithm from an ideal primitive is a much more general attack goal than key recovery, but as the latter, it only applies to keyed primitives. Let us consider the case of a block cipher $E$. A distinguisher for $E$ is an algorithm that has oracle access to a permutation on $\{0, 1\}^n$ (and eventually to its inverse) and, after a certain number of queries, it outputs a single bit. We write $D^P = 1$ for the event that $D$ outputs 1 when interacting with oracle $P$. The advantage of $D$ is then defined as

$$\text{Adv}(D) = \left| \Pr[D^{E(K, \cdot)} = 1] - \Pr[D^P = 1] \right|$$

for $P$ chosen uniformly at random from the set of all permutations on $\{0, 1\}^n$ and $K$ chosen uniformly at random from $\{0, 1\}^\kappa$.

The notion of indistinguishability is crucial in the context of provable security. It allows to make precise what it means to “behave like an ideal primitive” with respect to a certain class of adversaries. In the cryptanalytic context, a distinguishing attack often points out some weakness that is later exploited for more serious attacks like key recovery.

**Attack Scenarios for Keyed Primitives.** It is always assumed that the cipher specification is known and that the secret key is chosen uniformly at random. For block ciphers, the following attack scenarios are the most common (listed with increasing capabilities of the adversary):

- **Ciphertext-only:** The adversary can observe outputs of $E(K, \cdot)$ and has some knowledge of the inputs (e.g. the language).

---

10 A ranking of four goals of block cipher attacks is given by Knudsen and Robshaw [73]. We only consider the extreme cases of key recovery and distinguishing attacks in this thesis.
- **Known-plaintext:** The adversary can observe outputs of $E(K, \cdot)$ and knows the corresponding inputs.

- **Chosen-plaintext:** The adversary has oracle access to $E(K, \cdot)$, that is, he can choose the inputs for which he wants to obtain the outputs.

- **Chosen-plaintext, chosen-ciphertext:** The adversary has oracle access to $E(K, \cdot)$ and $E^{-1}(K, \cdot)$.

The scenarios can be further refined, for example for the chosen plaintext scenario it can be specified whether the adversary is able to adaptively choose its input or not. Moreover, each scenario has a related-key variant. In a related-key chosen-plaintext scenario, for example, the adversary has access to $E(K, \cdot)$ and $E(K', \cdot)$, where $K$ and $K'$ have a known or chosen relation, typically $K \oplus K' = \Delta K$ for a fixed difference $\Delta K$.

The scenarios for stream ciphers are very similar, with the plaintext replaced with the IV and the ciphertext replaced with the keystream.

**Attacks on Hash Functions.** Attack goals are to find a collision, a preimage, or a second-preimage. Additional specifications can make attacks more challenging (e.g. one-block collisions, short preimages, meaningful message content) or easier to achieve (e.g. ignoring the padding, very long preimages, pseudo-preimages). For an $n$-bit hash function, finding a preimage or a second-preimage with generic techniques requires $2^n$ computations of the hash function in average and essentially no memory. For collisions only $2^{n/2}$ computations are required due to the birthday paradox.\(^{11}\) A basic variant of the birthday attack requires memory to store $2^{n/2}$ hash values, but essentially memoryless variants are known.

### 2.4 Differential Cryptanalysis

Differential cryptanalysis is certainly the most general and most widely used technique in symmetric cryptanalysis. It was introduced to the public in 1990 by Biham and Shamir [19, 20] and lead to the first attack on the full 16-round DES [21]. Although it was originally developed for block cipher

\(^{11}\)The birthday paradox concerns the problem that, in a set of $N$ randomly chosen people, two have the same birthday. For $N = 23$ the probability for this is more than 0.5, and for $N = 57$ it is already more than 0.99.
cryptanalysis, differential cryptanalysis turned out to be very effective on stream ciphers and hash functions as well. Numerous variants of the original technique have been proposed over time. Notable examples are higher order differential cryptanalysis by Lai [74], truncated differential cryptanalysis by Knudsen [71], impossible differential cryptanalysis by Knudsen [72] and by Biham, Biryukov, and Shamir [18], and the Boomerang attack by Wagner [117].

Many years after the introduction of differential cryptanalysis it sometimes happens that a seemingly new technique can be explained in a simpler way when it is reformulated in terms of differential cryptanalysis, and often this leads to more effective attacks. Examples are given in Chapter 4 for high order differential attacks on certain stream ciphers and in Chapter 6 for meet-in-the-middle preimage attacks on hash functions.

### 2.4.1 Terminology

Consider a block cipher as illustrated in Fig. 2.1. Differential cryptanalysis looks at pairs of plaintexts \((P, P \oplus \Delta P)\) for a fixed difference \(\Delta P\). Assuming that the key \(K\) and the plaintext \(P\) are chosen uniformly at random, the propagation of the initial difference is analyzed round by round. Let \(P_0 = P\) and \(\Delta P_0 = \Delta P\). For \(1 \leq i \leq N\) we say that \(\Delta P_{i-1}\) propagates to \(\Delta P_i\), denoted by \(\Delta P_{i-1} \rightarrow \Delta P_i\), if

\[
R_i(K_i, P_{i-1}) \oplus R_i(K_i, P_{i-1} \oplus \Delta P_{i-1}) = \Delta P_i.
\]

A pair \((\Delta P_{i-1}, \Delta P_i)\) is called a (1-round) differential and the probability that \(\Delta P_{i-1} \rightarrow \Delta P_i\) is called its differential probability. Differentials can extend over several rounds, for example \((\Delta P_i, \Delta P_{i+s})\) is an \(s\)-round differential. A multi-round differential may be realized by several differential characteristics. An \(s\)-round characteristic is a sequence of \(s + 1\) differences \(\Delta P_i, \Delta P_{i+1}, \ldots, \Delta P_{i+s}\) and its probability is the probability that \(\Delta P_i \rightarrow \Delta P_{i+1} \rightarrow \ldots \rightarrow \Delta P_{i+s}\). The probability of a differential is the sum of the probabilities of all characteristics that realize it. Computing the probability of a differential characteristic is not simple in general. Typically, it requires relatively strong independency assumptions that might not be met in reality. Obtaining an appropriate estimate is a delicate task.
2.4.2 Applications

Differential cryptanalysis has numerous applications. Three of them are listed below. Key recovery on DES was the original application, output prediction is used in Chapters 3 and 4, and collision attacks on hash functions is used in Appendix B.

**Key Recovery Attacks on DES-like block ciphers.** Assume that a high probability \((N - 1)\)-round differential \((\Delta P_0, \Delta P_{N-1})\) exists. Then, the adversary chooses a random sample \(P\) of plaintexts and for each \(P \in P\) she queries \(P\) and \(P' = P \oplus \Delta P_0\) in order to obtain the ciphertexts \(C\) and \(C'\). Making a guess of \(K_N\) she computes \(P_{N-1} = R_{N-1}^{-1}(K_{N-1}, C)\) and \(P'_{N-1} = R_{N-1}^{-1}(K_{N-1}, C')\). If the guess of \(K_N\) was correct, many pairs \(P_{N-1}\) and \(P'_{N-1}\) have the difference \(\Delta P_{N-1}\). A refined variant of this approach has lead to the first attack on full 16-round DES [21].

**Output Prediction.** This is probably the most straightforward application of differential cryptanalysis. Assume that a high probability \(N\)-round differential \((\Delta P_0, \Delta P_N)\) exists. Then, the adversary can predict the ciphertext \(C' = E(K, P \oplus \Delta P_0)\) as \(C \oplus \Delta P_N\) when she knows \(C = E(K, P)\). Primarily, this gives rise to a distinguishing attack, but often key dependencies of \(\Delta P_N\) can be exploited to turn the distinguisher into a key recovery attack.

**Collision Attacks on Hash Functions.** Consider a hash function that uses an iterated compression function with \(N\) rounds. For a differential collision attack one requires a message difference \(\Delta M\) that cancels out after \(N\) rounds. Suitable differentials are often found by linear algebra using a linearized version of the hash function. This idea is first used by Chabaud and Joux [31] for SHA-0. If \(p\) is the differential probability, one out of \(1/p\) random message pairs \(M\) and \(M \oplus \Delta M\) are expected to produce a collision. In order to be faster than the birthday attack it is required that \(1/p \geq 2^{n/2}\).

The approach lead to breakthrough results when it was realized that messages should not be chosen at random, but rather in a specific way to control the propagation of the difference through the first few rounds. The first collision attacks on full MD5 and SHA-1 by Wang, Yin, and Yu [118, 119] have been obtained in this way.
Chapter 3

Conditional Differential Cryptanalysis of NLFSR

Non-linear feedback shift registers are the main building block of a variety of modern hardware-oriented cryptographic algorithms. For such constructions, we propose a differential cryptanalysis framework for distinguishing and key recovery attacks, that we call conditional differential cryptanalysis.

In this chapter, the general principle and analysis tools are described and applied to the KATAN/KTANTAN family of hardware-oriented block ciphers and the stream cipher Grain v1. None of the primitives is broken, but the best known results for Grain v1 are obtained. In Chapter 4, the framework will be extended to higher order differential attacks.

3.1 Introduction

For constrained environments like RFID tags or sensor networks a number of cryptographic primitives, such as stream ciphers and light-weight block ciphers have been developed, to provide security and privacy. Well known examples are the stream ciphers Grain v1 [54] and Trivium [27] that have been selected in the eSTREAM portfolio of promising stream ciphers for small hardware [43], and the block cipher family KATAN/KTANTAN [28]. All these constructions build essentially on non-linear feedback shift registers (NLFSRs). These facilitate an efficient hardware implementation and at
the same time enable to counter algebraic attacks. In contrast to LFSR-based stream ciphers however, there is a lack of general tools to assist the security of NLFSR-based constructions. Important design decisions include the choice of the update functions (degree, density) and the number of initialization rounds. Basic experimental tools such as bit-flip tests and cube testers are typically used to evaluate the key/IV mixing. These tools treat the cipher as a black box, which makes them easy to apply, but on the other hand, they don’t provide much confidence against adversaries that exploit specific properties of the construction.

The analysis framework presented in this chapter allows for a more in-depth security analysis, heavily exploiting the cipher’s specification. The application of the framework is more involved than black box cryptanalytic tests, but tools are proposed that greatly facilitate the analysis. Differential cryptanalysis, originally introduced for the analysis of block ciphers, studies the propagation of a difference in the plaintext through an iterated construction. In the case of NLFSR-based constructions, the propagation of the differences depends on polynomial expressions that can be written out explicitly. The degree and the density of these polynomials, however, grow very quickly. After the initialization phase they are expected to look like random polynomials of full degree containing each monomial with uniform probability. It is completely out of reach to write out such a polynomial for the purpose of a symbolic analysis. However, by carefully studying the first few rounds, conditions can be imposed on certain variables in order to virtually cancel out many terms which can be detected by a bias in the output difference.

The framework is applied to the KATAN/KTANTAN family of block ciphers and to the stream cipher Grain v1. Table 3.1 summarizes the results. All attacks recover part of the key and are experimentally verified on a PC.

Related Work. Differential cryptanalysis was first proposed by Biham and Shamir [21]. Ben-Aroya and Biham [13] introduced the notion of a conditional differential characteristics with an application to DES-like cryptosystems. The basic idea is essentially the same as for NLFSR-based constructions in this chapter. The analysis and tools, however, are quite different. Our analysis is originally inspired by message modification techniques that play a crucial role in collision attacks on MD5 and SHA-1 presented by Wang, Yin, and Yu [118, 119].
Chapter Outline. Section 3.2 describes the general principle of conditional differential cryptanalysis. In Sections 3.3 and 3.4 the framework is applied to the KATAN/KTANTAN family and to the stream cipher Grain v1. The figures included at the end of this chapter will be referenced in Chapter 4 as well. They show basic statistical information on the initialization process of different NLFSR-based constructions.

Preview on Chapter 4. The framework is extended to higher order differential attacks. While not giving improved results for KATAN/KTANTANT and for Grain v1, higher order conditional differential cryptanalysis turns out to be effective against reduced-round variants of Grain-128, Grain-128a, and Trivium. Based on our findings in both chapters, different design approaches of Grain and Trivium are discussed at the end of the Chapter.

3.2 Framework and Tools of Analysis

NLFSR-based constructions use one or several NLFSRs in combination with other building blocks, for example with LFSRs. Our framework applies to any such construction that uses at least one NLFSR. To simplify the presentation we suppose that a single NLFSR is used. Let \( \ell \) be the length of this NLFSR, denote its state by \((s_0, \ldots, s_{\ell-1})\), and let the state update be defined

<table>
<thead>
<tr>
<th>Cipher</th>
<th>( \kappa )</th>
<th>( n )</th>
<th>Scenario</th>
<th>Rounds</th>
<th>Queries</th>
<th>Key eq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>KATAN and KTANTAN</td>
<td>80</td>
<td>48</td>
<td>single-key</td>
<td>81 (32%)</td>
<td>( 2^{20} )</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td></td>
<td>71 (28%)</td>
<td>( 2^{26} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>70 (28%)</td>
<td>( 2^{35} )</td>
<td></td>
</tr>
<tr>
<td>KATAN</td>
<td>80</td>
<td>32</td>
<td>related-key</td>
<td>120 (47%)</td>
<td>( 2^{31} )</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>48</td>
<td></td>
<td>103 (41%)</td>
<td>( 2^{25} )</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td></td>
<td>90 (35%)</td>
<td>( 2^{27} )</td>
<td>1</td>
</tr>
<tr>
<td>Grain v1</td>
<td>80</td>
<td>64</td>
<td>single-key</td>
<td>97 (61%)</td>
<td>( 2^{27} )</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>104 (65%)</td>
<td>( 2^{35} )</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>related-key</td>
<td>133 (83%)</td>
<td>( 2^{35} )</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of results: \( \kappa \) is the key length and \( n \) is either the block length or the length of the IV. The last column indicates the number of recovered equations in key bits.
Conditional Differential Cryptanalysis of NLFSR

as

\[ t \leftarrow g(s_0, \ldots, s_{\ell-1}) \]
\[ (s_0, \ldots, s_{\ell-1}) \leftarrow (t, s_0, \ldots, s_{\ell-2}), \]

where \( g \) is a non-linear function. Denote by \( f : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\} \) the function mapping a key \( k = (k_0, \ldots, k_{n-1}) \) and an IV \( x = (x_0, \ldots, x_{n-1}) \) to a keystream bit \( z \).\(^{12}\) The bit \( z = f(k, x) \) is supposed to be computed as follows: the NLFSR is initialized with \( k \) and \( x \) in some way and updated several times, before \( z \) is derived from the resulting state.

Toy Example (1/4). Throughout the section we consider a toy example for illustration. The example uses an NLFSR of length 12 with the following update transformation:

\[ t \leftarrow s_{11} + s_7 + s_8s_5 + s_3, \]
\[ (s_0, \ldots, s_{11}) \leftarrow (t, s_0, \ldots, s_{10}). \]

On input \( x = (x_0, \ldots, x_7) \) and \( k = (k_0, \ldots, k_3) \), the register is initialized as follows:

\[ (s_0, \ldots, s_7) \leftarrow (x_0, \ldots, x_7) \]
\[ (s_8, \ldots, s_{11}) \leftarrow (k_0, \ldots, k_3). \]

Then, the register is clocked 18 times and the state bit generated at round 18 is considered as the output \( z = f(k, x) \). The example is chosen small enough such that \( z \) can be explicitly written out as a polynomial in the variables of \( k \) and \( x \):

\[
z = x_0x_1x_2x_3x_5 + x_0x_1x_2x_5x_7 + x_0x_1x_2x_5k_0 + x_0x_1x_2x_5k_3 + x_0x_1x_3k_0 + x_0x_1x_3 + x_0x_1x_4x_5k_0 + x_0x_1x_4k_3 + x_0x_1x_6 + x_0x_1x_7k_0 + x_0x_1x_7 + x_0x_1k_0k_3 + x_0x_1k_2 + x_0x_1k_3 + x_0x_2x_3x_4x_5k_0 + x_0x_2x_3x_4x_7 + x_0x_2x_3x_4k_3 + x_0x_2x_3x_4 + x_0x_2x_3x_5x_6x_7 + x_0x_2x_3x_5x_6k_0 + x_0x_2x_3x_5x_6k_3 + x_0x_2x_3x_5x_7 + x_0x_2x_3x_5k_0 + x_0x_2x_3x_5k_3 + x_0x_2x_3x_6 + x_0x_2x_3x_7k_0 + x_0x_2x_3x_7 + x_0x_2x_3k_0k_3 + x_0x_2x_3k_0 + x_0x_2x_3k_3 + x_0x_2x_5x_6k_0 + x_0x_2x_5x_7k_1 + x_0x_2x_5x_7 + x_0x_2x_5k_0k_1 + x_0x_2x_5k_1k_3 + x_0x_2x_5k_3 + x_0x_2x_6x_7 + x_0x_2x_6k_3 + x_0x_2x_7 + x_0x_2k_3 + x_0x_3x_4x_5x_6k_0 + x_0x_3x_4x_5 + x_0x_3x_4x_6k_3 + x_0x_3x_5x_6k_0 + x_0x_3x_5x_6k_3 + x_0x_3x_6k_2 + x_0x_3x_6 + x_0x_3k_1 + x_0x_3k_3 + x_0x_4x_5k_0k_1 + x_0x_4x_5k_0 + x_0x_4x_5k_3 + x_0x_4k_1k_3 + \]

\(^{12}\)In the case of a block cipher, the IV is replaced by the plaintext and the keystream by the ciphertext.
3.2.1 Difference Propagation Through NLFSRs

The state bits of the NLFSR are considered as Boolean functions in the variables of the key and the IV. At each clock, a single state bit is newly generated, and the other state bits are merely shifted. Thus, it is enough to consider the propagation of differences to newly generated state bits. We write \( t_i \) for the state bit generated at round \( i \).

Let \( \Delta x = (\Delta x_1, \ldots, \Delta x_n) \in \{0, 1\}^n \) be an IV difference. We write \( x + \Delta x = (x_1 + \Delta x_1, \ldots, x_n + \Delta x_n) \), and we say that \( \Delta x \) propagates to \( t_i \) if

\[
\Delta t_i = t_i(k, x) + t_i(k, x + \Delta x) = 1.
\]

Consider \( \Delta t_i \) as a polynomial in the key and the IV variables. The value of \( \Delta t_i \) determines whether the difference \( \Delta x \) propagates to \( t_i \) or not. A differential characteristic for \( r \) rounds of the NLFSR is a sequence of bits \( \delta = (\delta_0, \ldots, \delta_{r-1}) \) that describes the propagation of the difference through \( r \) rounds.

**Toy Example (2/4).** We consider the propagation of \( \Delta x = 0x80 \) (difference in bit \( x_7 \)) through the NLFSR of our toy example, the polynomial \( \Delta t_i \) is indicated for each round:

**Round 0:** 1

**Round 1:** \( x_4 \)

**Rounds 2-4:** 0

**Round 5:** \( x_4 \)
Round 6: $x_2$
Round 7: $x_1 x_4$
Round 8: 1
Round 9: $x_0 + x_2 x_5 + x_4 + k_0$
Round 10: $x_1 x_4 + x_2 + x_4 x_5 k_0 + x_4 k_3$

3.2.2 Imposing Conditions on Difference Propagations

The goal is to find a sample of IVs, denoted by $X$, for which the difference $\Delta z$ is biased. Such a sample is derived from conditions that control the difference propagation through the early clocks of the NLFSR. The effect of such conditions can be illustrated in symbolic form for our example.

Toy Example (3/4). The variables $x_4$ and $x_2$ control the propagation of the difference at the 1 and 6, respectively. We illustrate the effect of preventing the difference propagation at these rounds:

without conditions:

$$\Delta z = x_0 x_1 x_2 x_5 + x_0 x_1 k_0 + x_0 x_1 + x_0 x_2 x_3 x_4 + x_0 x_2 x_3 x_5 x_6 + x_0 x_2 x_3 x_5 + x_0 x_2 x_3 k_0 + x_0 x_2 x_3 + x_0 x_2 x_5 k_1 + x_0 x_2 x_5 + x_0 x_2 x_6 + x_0 x_2 + x_0 x_3 x_6 k_0 + x_0 x_5 k_0 + x_0 x_5 + x_0 k_0 k_1 + k_1 x_4 + x_2 x_3 x_5 x_6 + x_2 x_3 x_5 + x_2 x_4 x_6 + x_2 x_4 + x_2 x_5 x_6 + x_2 x_5 k_1 + x_2 x_5 + x_2 x_6 k_0 + x_2 k_0 + x_3 x_4 x_6 + x_3 x_4 + x_3 x_6 k_0 + x_3 k_0 + x_4 k_1 + x_4 + k_0 k_1 + k_2$$

with conditions $x_2 = 0, x_4 = 0$:

$$\Delta z = x_0 x_1 k_0 + x_0 x_1 + x_0 x_3 x_6 k_0 + x_0 x_5 k_0 + x_0 x_5 + x_0 k_0 k_1 + x_3 x_6 k_0 + x_3 k_0 + k_0 k_1 + k_2$$

The specific conditions cancel out many terms in $\Delta z$ (note that arbitrary conditions have a much weaker effect). Here, it would have been possible to derive the conditions from the symbolic representation of $z$, but such a representation is not available in real world applications.
3.2 Framework and Tools of Analysis

3.2.3 Key Dependent Conditions

Two types of key dependent conditions may occur: conditions that involve both, key and IV variables, and conditions in key variables only. The latter will typically define classes of weak keys, but both types can also be exploited for key recovery. We refer to the applications for concrete examples.

3.2.4 Tools for Condition Analysis

Imposing the right conditions is the crucial part of conditional differential cryptanalysis. One has to tradeoff between the following aims:

- Control a maximum number of propagations.
- Find a large sample of IVs that satisfy the conditions.

Analyzing the conditions by hand, is prone to error and quickly gets impossible. Using a mathematics software package that provides arithmetic in Boolean polynomial rings facilitates things a lot. The idea is to represent conditions as polynomials. Imposed conditions are “collected” in a polynomial ideal $J$, and new conditions are analyzed modulo $J$.

As an example, Algorithm 3.1 implements a fully automatic strategy to find conditions for the following use case: one starts with a low-weight difference $\Delta x$ and one wants to prevent the difference propagation whenever possible in the first $q$ rounds. The conditions are returned as a basis of $J$. If $g_1, \ldots, g_s$ is such a basis, the sample $X$ has to be derived from the conditions $g_1 = 0, \ldots, g_s = 0$.

One can think of many variations of Algorithm 3.1. The procedure might stop after a fixed number of conditions, or certain propagations might be forced instead of prevented. As a main limitation, such algorithms only work as long as the state bits can be written out symbolically. It heavily depends on the target primitive for how many rounds this is possible. Figures 3.1 to 3.6 at the end of this chapter show the evolution of degree and number of monomials in state bits of the algorithms considered in this thesis. They show that the number of rounds that can be written out symbolically varies from about 7% (for KATAN64) to about 39% (for Grain-128).\(^\text{13}\) Some more rounds can be sometimes controlled by a combination of by hand analysis and automatic tools (as in our application to Grain v1).

\(^{13}\)The data is obtained with SAGE [113] using the polybori library [25] on a Intel Core 2 Duo E8400 3.0 GHz Processor with 4 GB of RAM.
Algorithm 3.1 Prevent difference propagation whenever possible.

**Input:** $\Delta x, q$

**Output:** a set of conditions

```plaintext
J \leftarrow \langle 0 \rangle 

for $i = 0$ to $q - 1$ do
    $p \leftarrow \Delta t_i$
    if $p \not\equiv 1 \mod J$ then
        $J = J + \langle p \rangle$
    end if
end for

return a basis of $J$
```

**Toy Example (4/4).** Let us apply Algorithm 3.1 to our toy example with $\Delta x = 0x80$ and $q = 18$. The following conditions are returned: $x_0 + k_0 = 0$, $x_1 x_6 + x_1 k_2 + x_3 x_6 k_2 + x_6 k_2 = 0$, $x_2 = 0$, $x_3 k_0 + k_2 = 0$, $x_4 = 0$. These conditions control the propagation through all the 18 rounds, and result in $\Delta z = 0$. Note that these conditions are not trivial to find, even not when the symbolic representation of $z$ is known as in this toy example.

### 3.2.5 Statistical Terminology

The aim of our attacks is to experimentally detect a bias in the output $\Delta f(k, \cdot)$ on a sample $X$. The statistical terminology is briefly explained in the following.

Let $z_1, \ldots, z_N$ be the outputs of $\Delta f(k, \cdot)$ on $X$, where $N = |X|$ is the sample size. Let $S = \sum_{i=1}^{N} z_i$ be the sum of the outputs. We say that $\Delta f(k, \cdot)$ is biased on $X$ if

$$
\Phi \left( \left| \frac{2S - N}{\sqrt{N}} \right| \right) > 0.9995,
$$

where $\Phi$ is the standard normal cumulative distribution function,

$$
\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{1}{2}u^2} du.
$$

The test detects a bias in a perfect random source with a probability of $\alpha = 0.001$ (this is also called the significance level).
If we say that a bias is detected (without further indications), we always mean that it was detected in almost all of sufficiently many independent experiments. The size of the required sample is always indicated as it determines the query complexity of the attack.

### 3.3 Application to KATAN/KTANTAN

KATAN/KTANTAN is a family of hardware-oriented block ciphers proposed by De Cannière, Dunkelman, and Knežević in [28]. The family consists of six ciphers denoted by KATAN\(n\) and KTANTAN\(n\) for \(n = 32, 48, 64\) indicating the block length of the cipher. All instances accept an 80-bit key and use two NLFSRs as their main building blocks. The only difference between KATAN\(n\) and KTANTAN\(n\) is the key scheduling.

**Previous Cryptanalytic Results.** Bogdanov and Rechberger [24] present a meet-in-the-middle key recovery attack against the full KTANTAN\(n\) ciphers. The attack exploits a serious flaw in the KTANTAN key expansion and recovers the full key about \(2^{25}\) times faster than by brute-force search. The attacks were later improved by Wei at al. [120]. The same weakness is exploited by Ågren [1] for practical key recovery attacks in the related-key scenario. We are not aware of previous cryptanalytic results on KATAN.

### 3.3.1 Description of KATAN32 and KTANTAN32

A unified notation is used in this thesis for the description of NLFSR-based ciphers. The notation is essentially the one used in [27] for Trivium.

The internal state of KATAN32 consists of two NLFSRs with lengths 19 and 13, respectively. The registers, denoted by \((s_0, \ldots, s_{18})\) and \((s_{19}, \ldots, s_{31})\), are initialized with the plaintext \(x = (x_0, \ldots, x_{31})\) as follows:

\[
(s_0, \ldots, s_{18}) \leftarrow (x_0, \ldots, x_{18}) \quad (s_{19}, \ldots, s_{31}) \leftarrow (x_{19}, \ldots, x_{31}).
\]

The key \(k = (k_0, \ldots, k_{79})\) is expanded to 508 bits according to the following linear recursion:

\[
k_{i+80} = k_i + k_{i+19} + k_{i+30} + k_{i+67}, \quad 0 \leq i < 428.
\]
The encryption process has 254 rounds. At round \( i \), the key bits \( k_{2i} \) and \( k_{2i+1} \) are used as well as a constant \( c_i \). The round constants are generated by an 8-bit LFSR which is used as a counter as well. It is initialized by \((c_0, \ldots, c_7) = (1, \ldots, 1, 0)\) and expanded according to

\[
c_{i+8} = c_i + c_{i+1} + c_{i+3} + c_{i+5}, \quad 0 \leq i < 246.
\]

Round \( i \) corresponds to the following transformation of the state:

\[
\begin{align*}
t_1 & \leftarrow s_{31} + s_{26} + s_{27}s_{24} + s_{22}c_i + k_{2i} \\
t_2 & \leftarrow s_{18} + s_7 + s_{12}s_{10} + s_8s_3 + k_{2i+1} \\
(s_0, \ldots, s_{18}) & \leftarrow (t_1, s_0, \ldots, s_{17}) \\
(s_{19}, \ldots, s_{31}) & \leftarrow (t_2, s_{19}, \ldots, s_{30})
\end{align*}
\]

After 254 rounds, the state is output as the ciphertext.

The only difference between KATAN\( n \) and KTANTAN\( n \) is the key expansion. The key expansion of KTANTAN is given by Table 3.2. The number of rounds, the key expansion, and the round constants do not depend on the block length. Differences lie in the length of the two NLFSRs and in the round transformation.

### 3.3.2 Description of KATAN48 and KTANTAN48

The NLFSRs have length 29 and 19, respectively. They are initialized as follows:

\[
\begin{align*}
(s_0, \ldots, s_{28}) & \leftarrow (x_0, \ldots, x_{28}) \\
(s_{29}, \ldots, s_{47}) & \leftarrow (x_{29}, \ldots, x_{47}).
\end{align*}
\]

Round \( i \) corresponds to the following transformation of the state:

\[
\begin{align*}
t_1 & \leftarrow s_{47} + s_{41} + s_{44}s_{36} + s_{35}c_i + k_{2i} \\
t_2 & \leftarrow s_{46} + s_{40} + s_{43}s_{35} + s_{34}c_i + k_{2i} \\
t_3 & \leftarrow s_{28} + s_{19} + s_{21}s_{13} + s_{15}s_6 + k_{2i+1} \\
t_4 & \leftarrow s_{27} + s_{18} + s_{20}s_{12} + s_{14}s_5 + k_{2i+1} \\
(s_0, \ldots, s_{28}) & \leftarrow (t_1, t_2, s_0, \ldots, s_{26}) \\
(s_{29}, \ldots, s_{47}) & \leftarrow (t_3, t_4, s_{29}, \ldots, s_{45})
\end{align*}
\]
### Table 3.2: KTANTAN key expansion. The expanded key is a sequence of the original key bits. Reading example: entry 21 in the second column of the third row indicates that bit $33 = 32 + 1$ of the expanded key equal to $k_{21}$ of the original key.

|    | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0  | 63 | 31 | 31 | 63 | 31 | 63 | 15 | 47 | 14 | 14 | 60 | 76 | 40 | 40 | 49 | 17 |
| 16 | 35 | 67 | 54 | 22 | 45 | 77 | 58 | 26 | 37 | 69 | 74 | 10 | 69 | 69 | 74 | 10 |
| 32 | 53 | 21 | 43 | 43 | 71 | 7 | 63 | 79 | 30 | 62 | 45 | 45 | 11 | 11 | 54 | 70 |
| 48 | 28 | 60 | 41 | 41 | 3 | 19 | 38 | 70 | 60 | 28 | 25 | 73 | 34 | 34 | 5 | 21 |
| 64 | 26 | 74 | 20 | 52 | 9 | 41 | 2 | 18 | 20 | 68 | 24 | 56 | 1 | 33 | 2 | 2 |
| 80 | 52 | 68 | 24 | 56 | 17 | 49 | 3 | 35 | 6 | 6 | 76 | 76 | 72 | 8 | 49 | 17 |
| 96 | 19 | 51 | 23 | 55 | 15 | 63 | 14 | 46 | 12 | 28 | 24 | 72 | 16 | 48 | 1 | 49 |
| 112| 2 | 34 | 4 | 20 | 40 | 72 | 48 | 16 | 17 | 65 | 18 | 50 | 5 | 53 | 10 | 58 |
| 128| 4 | 36 | 8 | 8 | 64 | 64 | 64 | 0 | 65 | 1 | 51 | 19 | 23 | 55 | 47 | 47 |
| 144| 15 | 15 | 78 | 78 | 76 | 12 | 73 | 9 | 67 | 3 | 55 | 23 | 47 | 47 | 63 | 31 |
| 160| 47 | 79 | 62 | 30 | 29 | 77 | 26 | 58 | 5 | 37 | 10 | 26 | 36 | 68 | 56 | 24 |
| 176| 33 | 65 | 50 | 18 | 21 | 69 | 42 | 42 | 5 | 5 | 58 | 74 | 20 | 52 | 25 | 57 |
| 192| 3 | 51 | 6 | 38 | 12 | 12 | 56 | 72 | 16 | 48 | 33 | 33 | 3 | 3 | 70 | 70 |
| 208| 60 | 28 | 41 | 41 | 67 | 3 | 71 | 71 | 78 | 14 | 77 | 13 | 59 | 27 | 39 | 39 |
| 224| 79 | 15 | 79 | 79 | 62 | 30 | 45 | 45 | 59 | 27 | 23 | 71 | 46 | 46 | 13 | 29 |
| 240| 42 | 74 | 52 | 20 | 41 | 73 | 66 | 2 | 53 | 69 | 42 | 42 | 53 | 21 | 27 | 75 |
| 256| 38 | 38 | 13 | 13 | 74 | 74 | 52 | 20 | 25 | 57 | 35 | 35 | 7 | 7 | 62 | 78 |
| 272| 44 | 44 | 73 | 9 | 51 | 67 | 22 | 54 | 29 | 61 | 11 | 43 | 6 | 22 | 44 | 76 |
| 288| 72 | 8 | 65 | 65 | 50 | 18 | 37 | 37 | 75 | 11 | 55 | 71 | 46 | 46 | 77 | 13 |
| 304| 75 | 75 | 70 | 6 | 61 | 29 | 27 | 59 | 39 | 39 | 15 | 31 | 46 | 78 | 76 | 12 |
| 320| 57 | 73 | 34 | 34 | 69 | 5 | 59 | 75 | 38 | 38 | 61 | 29 | 43 | 75 | 70 | 6 |
| 336| 77 | 77 | 58 | 26 | 21 | 53 | 43 | 43 | 7 | 23 | 30 | 78 | 44 | 44 | 9 | 25 |
| 352| 18 | 66 | 36 | 36 | 9 | 9 | 50 | 66 | 36 | 36 | 57 | 25 | 19 | 67 | 22 | 54 |
| 368| 13 | 45 | 10 | 10 | 68 | 68 | 56 | 24 | 17 | 49 | 19 | 51 | 7 | 39 | 14 | 30 |
| 384| 28 | 76 | 40 | 40 | 1 | 1 | 66 | 66 | 68 | 4 | 57 | 25 | 35 | 35 | 55 | 23 |
| 400| 31 | 79 | 30 | 62 | 13 | 61 | 10 | 42 | 4 | 4 | 72 | 72 | 48 | 16 | 33 | 33 |
| 416| 51 | 19 | 39 | 71 | 78 | 14 | 61 | 77 | 26 | 58 | 21 | 53 | 11 | 59 | 6 | 54 |
| 432| 12 | 44 | 8 | 24 | 32 | 64 | 64 | 0 | 49 | 65 | 18 | 50 | 37 | 37 | 11 | 27 |
| 448| 22 | 70 | 28 | 60 | 9 | 57 | 2 | 50 | 4 | 52 | 8 | 40 | 0 | 0 | 48 | 64 |
| 464| 32 | 32 | 65 | 1 | 67 | 67 | 54 | 22 | 29 | 61 | 27 | 59 | 7 | 55 | 14 | 62 |
| 480| 12 | 60 | 8 | 56 | 0 | 32 | 0 | 16 | 16 | 64 | 32 | 32 | 1 | 17 | 34 | 66 |
| 496| 68 | 4 | 73 | 73 | 66 | 2 | 69 | 5 | 75 | 11 | 71 | 7 | 7 | 7 | 7 | 7 |
3.3.3 Description of KATAN64 and KTANTAN64

The NLFSRs have length 39 and 25, respectively. They are initialized as follows:

\[
\begin{align*}
(s_0, \ldots, s_{38}) & \leftarrow (x_0, \ldots, x_{38}) \\
(s_{39}, \ldots, s_{63}) & \leftarrow (x_{39}, \ldots, x_{63}).
\end{align*}
\]

Round \(i\) corresponds to the following transformation of the state:

\[
\begin{align*}
t_1 & \leftarrow s_{63} + s_{54} + s_{59}s_{50} + s_{48}c_i + k_{2i} \\
t_2 & \leftarrow s_{62} + s_{53} + s_{58}s_{49} + s_{47}c_i + k_{2i} \\
t_3 & \leftarrow s_{61} + s_{52} + s_{57}s_{48} + s_{46}c_i + k_{2i} \\
t_4 & \leftarrow s_{38} + s_{25} + s_{33}s_{21} + s_{14}s_9 + k_{2i+1} \\
t_5 & \leftarrow s_{37} + s_{24} + s_{32}s_{20} + s_{13}s_8 + k_{2i+1} \\
t_6 & \leftarrow s_{36} + s_{23} + s_{31}s_{19} + s_{12}s_7 + k_{2i+1}
\end{align*}
\]

\[
\begin{align*}
(s_0, \ldots, s_{38}) & \leftarrow (t_1, t_2, t_3, s_0, \ldots, s_{35}) \\
(s_{39}, \ldots, s_{63}) & \leftarrow (t_4, t_5, t_6, s_{39}, \ldots, s_{60}).
\end{align*}
\]

3.3.4 Reduced-Round Key Recovery Attacks

The target bit of our attack is either the last bit of the first register or the last bit of the second register. An input difference \(\Delta x\) together with a characteristic for \(q\) rounds is determined such that a single bit difference remains in the state after \(q\) rounds. Such a difference and a corresponding characteristic are computed by “decrypting” the desired state difference through the first \(q\) rounds with a linearized round transformation. The fixed characteristic determines the conditions on the plaintext and the key. Several key dependant samples are tested and the right key guess is detected by the bias.

Application to KATAN32 and KTANTAN32

A characteristic is used for \(q = 18\) rounds with input difference

\[
\Delta x = 0x10040080
\]

(differences at bits \(x_7, x_{18},\) and \(x_{28}\)). This difference results in a single bit difference in state bit \(s_{14}\) after 18 rounds if the following conditions are satisfied:
Round 0: No conditions.

Round 1: \( x_2 = 0 \)

Round 2: No conditions.

In the following, rounds without conditions are omitted.

Round 3: \( x_9 = 0 \)

Round 5: \( x_5 = 0 \)

Round 7: \( x_1 = 0 \)

Round 12:

\[
x_3x_8 + x_7 + x_8x_{10}x_{19} + x_{10}x_{12} + x_{16}x_{19} + x_{18} + x_{19}k_5 + x_{23} + k_1 + k_{16} = 0
\]

Round 14: \( x_{21} + x_{23}x_{26} + x_{25} + x_{30} + k_2 = 0 \)

Round 16: \( x_6 + x_{17} + x_{23}x_{26} + x_{25} + x_{26} + x_{30} + k_2 + k_3 + k_{10} = 0 \).

In order to simplify the condition at round 12, we impose an additional condition \( x_{19} = 0 \). As a summary, the following eight conditions are imposed:

\[
x_2 = 0, x_9 = 0, x_5 = 0, x_1 = 0, x_{19} = 0,
\]
\[
x_3x_8 + x_7 + x_{10}x_{12} + x_{18} + x_{23} + k_1 + k_{16} = 0,
\]
\[
x_{21} + x_{23}x_{26} + x_{25} + x_{30} + k_2 = 0,
\]
\[
x_6 + x_{17} + x_{23}x_{26} + x_{25} + x_{26} + x_{30} + k_2 + k_3 + k_{10} = 0.
\]

Three of the conditions depend on the key (key bits are printed in boldface). The corresponding expressions will be guessed and \( 2^3 \) different samples will be derived.

Let \( K_1 = k_1 + k_{16} \), \( K_2 = k_2 \), and \( K_3 = k_2 + k_3 + k_{10} \) be the unknown key expressions and write \( K = (K_1, K_2, K_3) \). Then, for each \( K \in \{0, 1\}^3 \) a sample \( X_K \) can be generated as follows:

1. Set \( X_K \leftarrow \{ x \in \{0, 1\}^n \mid \right. \begin{align*}
    x_2 &= 0, x_9 = 0, x_5 = 0, x_1 = 0, \\
    x_{19} &= 0, x_6 = 0, x_7 = 0, x_{21} = 0 \}
\).
2. For all \( x \in X_K \), adjust \( x_7, x_{21}, \) and \( x_6 \) according to \( K \):
\[
\begin{align*}
x_7 & \leftarrow x_3 x_8 + x_{10} x_{12} + x_{18} + x_{23} + K_1, \\
x_{21} & \leftarrow x_{23} x_{26} + x_{25} + x_{30} + K_2, \\
x_6 & \leftarrow x_{17} + x_{23} x_{26} + x_{25} + x_{26} + x_{30} + K_3.
\end{align*}
\]

Samples of size \( N = 2^{23} \) can be generated in this way, but smaller subsamples can be easily derived by fixing some bits to random values.

For the correct guess, a bias is detected in the difference of \( s_{18} \) after 81 rounds. With \( N = 2^{15} \), the correct guess was identified for 108 of 128 random keys. With \( N = 2^{16} \), the correct guess was always identified correctly. The attack requires \( 2^4 \cdot N \) chosen plaintext queries, essentially no memory and no time, and it recovers three bits of information on the key.

For KTANTAN32, only the key bits are different, the expressions \( k_{31} + k_{35}, k_{31}, \) and \( k_{63} + k_{60} \) are recovered.

**Application to KATAN48 and KTANTAN48**

IV difference: \( \Delta x = 0x000001208000 \)

Conditions: \( x_6 + x_{13} = 0, x_7 = 0, x_{29} = 0, x_1 x_{10} + x_8 x_{16} + x_{14} + x_{23} + k_5 = 0, \)
\( x_0 = 0, x_{13} = 0, x_4 + k_{15} = 0, x_5 = 0, x_{12} = 0, x_{30} = 0, x_{39} = 0, x_{36} + x_{42} = 0, \)
\( x_{18} + x_{27} + x_{33} = 0, x_{34} + x_{35} x_{43} + x_{40} + x_{46} + k_0 + k_1 k_4 + k_4 k_{14} + k_{21} = 0. \)

Bias is detected in \( \Delta s_{27} \) after 71 rounds for \( N = 2^{23}. \)

Key recovery KATAN48: \( k_5, k_{15}, k_0 + k_1 k_4 + k_4 k_{14} + k_{21}. \)

Key recovery KTANTAN48: \( k_{63}, k_{17}, k_{63} + k_{31} + k_{31} k_{49} + k_{77}. \)

**Application to KATAN64 and KTANTAN64**

IV difference: \( \Delta x = 0x02000001000080040 \)

Conditions: \( x_{48} = 0, x_{20} = 0, x_{31} = 0, x_{11} = 0, x_1 = 0, x_7 = 0, x_{18} = 0, \)
\( x_{47} + x_{49} x_{58} + x_{53} + x_{62} + k_0 = 0, x_8 x_{13} + x_{24} + x_{37} + x_{43} + x_{52} + k_1 + k_6 = 0, \)
\( x_5 = 0, x_{43} + x_{45} x_{54} + x_{49} + x_{58} + k_2 = 0, x_{13} = 0, x_{41} = 0, x_{17} + x_{23} + x_{30} + \)
\( x_{36} + x_{45} + k_1 + k_5 + k_{12} = 0. \)

Bias is detected in \( \Delta s_{36} \) after 70 rounds for \( N = 2^{30}. \)

Key recovery KATAN64: \( k_0, k_1 + k_6, k_2, k_1 + k_5 + k_{12}. \)

Key recovery KTANTAN64: \( k_{63}, k_{31} + k_{15}, k_{31}, k_{31} + k_{63} + k_{40}. \)
3.3 Application to KATAN/KTANTAN

3.3.5 Reduced-Round Related-Key Attacks

We now assume a related-key attack scenario, where the adversary can choose a particular difference between two keys. The principle of the attacks is the same as in the single-key scenario, but the initial differences and the corresponding characteristic are chosen in a slightly different way. Due to the linear key expansion of KATAN it is easy to compute a key difference that does not introduce differences during 39 consecutive rounds, say from round \( q \) to \( q + 39 \). For such a difference, a plaintext difference and a corresponding characteristic is computed (again by linearized decryption), such that no difference is present in the state after \( q \) rounds. Differences are then re-introduced only at round \( q + 39 \).

Application to KATAN32

The following difference for the IV and the key is found for \( q = 21 \):

\[
(\Delta x, \Delta k) = (0x00080240, 0x00000000100002004040).
\]

All differences cancel out after 21 rounds and are re-introduced only at round 60, if the following conditions are satisfied:

**Round 1, 2, 3, 4, 5, and 6:**

\[
x_{11} + 1 = 0, \ x_1 = 0, \ x_7 = 0, \ x_8 = 1, \ x_{22} = 0, \ x_4 = 0
\]

**Round 8:**

\[
x_5 + x_{10} + x_{16} + k_5 = 0
\]

**Round 10:**

\[
x_6 + x_9 + x_{17} + k_3 = 0
\]

**Round 13:**

\[
x_0 + x_3x_{10} + x_3x_{16} + x_3k_5 + k_{15} = 0
\]

**Round 15:**

\[
x_2x_{20} + x_2x_{24} + x_2x_{29} + x_2k_4 + x_{12} + k_{13} = 0
\]
Round 18:

\[ x_3x_{10}x_{21} + x_3x_{10}x_{23}x_{26} + x_3x_{10}x_{25} + x_3x_{10}x_{30} + x_3x_{10}k_2 + x_3x_{16}x_{21} + x_3x_{16}x_{23}x_{26} + x_3x_{16}x_{25} + x_3x_{16}x_{30} + x_3x_{16}k_2 + x_3x_{19} + x_3x_{21}x_{24} + x_3x_{21}k_5 + x_3x_{23}x_{26}k_5 + x_3x_{23} + x_3x_{25}k_5 + x_3x_{28} + x_3x_{30}k_5 + x_3k_2k_5 + x_3k_6 + x_3 + x_9 + x_{10}x_{12}x_{19} + x_{10}x_{12}x_{21}x_{24} + x_{10}x_{12}x_{23} + x_{10}x_{12}x_{28} + x_{10}x_{12}k_6 + x_{10}x_{12} + x_{17} + x_{18}x_{19} + x_{18}x_{21}x_{24} + x_{18}x_{23} + x_{18}x_{28} + x_{18}k_6 + x_{18} + x_{19}x_{23} + x_{19}k_1 + x_{19}k_{16} + x_{20}x_{23} + x_{21}x_{23}x_{24} + x_{21}x_{24}k_1 + x_{21}x_{24}k_{16} + x_{21}k_6 + x_{23}x_{26}k_5 + x_{23}x_{28} + x_{23}k_1 + x_{23}k_6 + x_{23}k_{16} + x_{23} + x_{25}k_{15} + x_{27} + x_{28}k_1 + x_{28}k_{16} + x_{30}k_{15} + k_1k_6 + k_1 + k_2k_{15} + k_3 + k_6k_{16} + k_8 + k_{25} = 0 \]

The last condition, in particular, is more complicated than what we had so far, but the principle of the attack remains the same. Ten expressions in key bits have to be guessed: \( k_1, k_2, k_3, k_4, k_5, k_6, k_{13}, k_{15}, k_{16}, \) and \( k_8 + k_{25} \). For the correct guess, a bias is detected in \( s_{18} \) after 120 rounds. The correct guess is always identified correctly with samples of size \( N = 2^{20} \). The attack requires \( 2^{31} \) chosen plaintext queries and recovers ten bits of information on the key.

Application to KATAN48

IV difference: \( \Delta x = 0x30c010280c06 \)

Key difference: \( \Delta k = 0x00000000000000000000100 \)

Conditions: \( x_{36} = 0, x_{12} = 0, x_{11} + x_{19} + 1 = 0, x_{18} = 0, x_{31} = 0, x_2 + x_{19} = 0, x_1 + x_{10} + 1 = 0, x_{30} = 0, x_3 = 0, x_{10} + x_{19} + 1 = 0, x_9 = 0, x_{29} + x_{35} + x_{41} + k_6 = 0, x_6 + x_{15} + x_{13}x_{21} + x_{19} + x_{28} + x_{34} + x_{35} + x_{37} + x_{37}x_{41} + x_{37}k_6 + x_{40} + k_1 + k_6 = 0, x_{42} + k_4 = 0, x_{7}x_{13}x_{15}x_{21} + x_{7}x_{13}x_{21} + x_{7}x_{15}x_{19} + x_{7}x_{15}x_{28} + x_{7}x_{15}x_{34} + x_{7}x_{15}x_{35}x_{37} + x_{7}x_{15}x_{37}x_{41} + x_{7}x_{15}x_{37}k_6 + x_{7}x_{15}x_{40} + x_{7}x_{15}k_1 + x_{7}x_{15}k_6 + x_{7}x_{19} + x_{7}x_{28} + x_{7}x_{34} + x_{7}x_{35}x_{37} + x_{7}x_{37}x_{41} + x_{7}x_{37}k_6 + x_{7}x_{40} + x_{7}k_1 + x_{7}k_6 = 0. \)

Bias is detected in \( \Delta s_{28} \) after 103 rounds for \( N = 2^{21} \).

Key recovery: \( k_1, k_4, k_6. \)

Application to KATAN64

IV difference: \( \Delta x = 0x1c000007003801c0 \)

Key difference: \( \Delta k = 0x0000000000000000200004 \)
Conditions: \(x_{50} = 0, x_{49} = 0, x_{21} + x_{33} + 1 = 0, x_{13} + x_{20} + x_{32} + 1 = 0, x_{12} + x_{31} = 0, x_{11} = 0, x_{3} = 0, x_{2} = 0, x_{1} = 0, x_{9} = 0, x_{8} + x_{20} + 1 = 0, x_{7} + x_{19} + 1 = 0, x_{18} = 0, x_{45} + x_{47}x_{56} + x_{51} + x_{60} + k_2 = 0, x_{44} + x_{46}x_{55} + x_{59} + k_2 = 0, x_{43} + x_{47}x_{54}x_{56} + x_{51}x_{54} + x_{54}x_{60} + x_{54}k_2 + x_{58} + k_2 = 0,

Bias is detected in \(\Delta s_{39}\) after 90 rounds for \(N = 2^{25}\).

Key recovery: \(k_2\).

### 3.4 Application to Grain v1

Grain v1 is a stream cipher designed by Hell, Johansson, and Meier in [54] and has been selected for the final eSTREAM portfolio [43]. It accepts an 80-bit key and a 64-bit IV.

**Previous Cryptanalytic Results.** De Cannière, Küçük, and Preneel [29] point out a sliding property in the initialization of Grain v1 that can be used to speed up exhaustive search by a factor two. The property was later exploited by Lee, Jeong, Sung, and Hong [76] for a related-key attack on the full Grain v1. The attack requires 3 related keys and a practical amount of data and time to recover the full key of Grain v1. Further in [29], a differential cryptanalysis is carried out. An attack on 112 rounds is claimed requiring an impossible amount of \(2^{72}\) chosen IVs (the IV length is only 64 bits). Aumasson, Dinur, Henzen, Meier, and Shamir [7] present a distinguishing attack for 81 rounds.

### 3.4.1 Description of Grain v1

The internal state of Grain v1 consists of 160 bits organized in two registers of length 80. The registers, denoted by \((s_0, \ldots, s_{79})\) and \((b_0, \ldots, b_{79})\), are initialized with the key \(k = (k_0, \ldots, k_{79})\) and the IV \(x = (x_0, \ldots, x_{63})\) as follows:

\[(s_0, \ldots, s_{79}) \leftarrow (x_0, \ldots, x_{63}, 1, \ldots, 1)\]
\[(b_0, \ldots, b_{79}) \leftarrow (k_0, \ldots, k_{79}).\]
The state is updated 160 times as follows (without producing keystream):

\[
\begin{align*}
    t_1 &\leftarrow s_{62} + s_{51} + s_{38} + s_{23} + s_{13} + s_0 \\
    t_2 &\leftarrow s_0 + b_{62} + b_{60} + b_{52} + b_{45} + b_{37} + b_{33} + b_{28} + b_{21} \\
        &+ b_{14} + b_9 + b_0 + b_{63}b_{60} + b_{37}b_{33} + b_{15}b_9 \\
        &+ b_{60}b_{52}b_{45} + b_{33}b_{28}b_{21} + b_{63}b_{45}b_{28}b_9 \\
        &+ b_{60}b_{52}b_{37}b_{33} + b_{63}b_{60}b_{21}b_{15} \\
        &+ b_{63}b_{60}b_{52}b_{45}b_{37} + b_{33}b_{28}b_{21}b_{15}b_9 \\
        &+ b_{52}b_{45}b_{37}b_{33}b_{28}b_{21} \\
    z &\leftarrow b_1 + b_2 + b_4 + b_{10} + b_{31} + b_{43} + b_{56} + s_{25} + s_{63} \\
        &+ s_3s_{64} + s_{46}s_{64} + s_{64}b_{63} + s_3s_{25}s_{46} \\
        &+ s_3s_{46}s_{64} + s_3s_{46}b_{63} + s_{25}s_{46}b_{63} + s_{46}s_{64}b_{63} \\
\end{align*}
\]

\[
\begin{align*}
    (s_0, \ldots, s_{79}) &\leftarrow (s_1, \ldots, s_{79}, t_1 + z) \\
    (b_0, \ldots, b_{79}) &\leftarrow (b_1, \ldots, b_{79}, t_2 + z).
\end{align*}
\]

After 160 rounds the initialization phase is done and the cipher starts generating keystream. In the subsequent rounds, \(t_1\), \(t_2\), and \(z\) are computed as before, \(z\) is output as a keystream bit and the state is updated as follows:

\[
\begin{align*}
    (s_0, \ldots, s_{79}) &\leftarrow (s_1, \ldots, s_{79}, t_1) \\
    (b_0, \ldots, b_{79}) &\leftarrow (b_1, \ldots, b_{79}, t_2).
\end{align*}
\]

### 3.4.2 Key Recovery up to 104 Rounds

A different strategy is used for Grain v1 than for KATAN/KTANTAN. Computing backwards from low-weight intermediate difference to obtain an IV difference does not work in general because differences are introduced into the constant part and into the key as well. The strategy will be used for the related-key attack below, however. Here, we start with a low-weight difference and prevent its propagation whenever possible. The following difference is chosen:

\[
\Delta x = 0x0000000200000000
\]

(single bit bit difference at \(x_{37}\)). This difference is chosen because it takes the maximum number of rounds until a difference propagates inevitably to the non-linear register (which has a much stronger diffusion and is more difficult to control than the linear register). The following conditions are imposed:
Round 12: \[ x_{15}x_{58} + x_{58}k_{75} + 1 = 0 \]

The condition is satisfied by \( x_{58} + 1 = 0 \) and \( x_{15} + K_1 = 0 \), where \( K_1 = k_{75} + 1 \).

Algorithm 3.1 (see Section 3.2) fails at around 35 rounds, because the state bits get too complex (compare Fig. 3.4). In order to derive more conditions we first derive them in terms of state bits and expand them in a second step.

Round 34: \( z \) depends on \( x_{37} \) via \( s_3 \):

\[
 z = b_1 + b_2 + b_4 + b_{10} + b_{31} + b_{43} + b_{56} + s_{25} + b_{63} + s_3s_{64} + s_{46}s_{64} + s_{64}b_{63} + s_3s_{25}s_{46} + s_3s_{46}s_{64} + s_3s_{46}b_{63} + s_{25}s_{46}b_{63} + s_{46}s_{64}b_{63}
\]

The propagation is prevented by \( s_{46} = 0 \) and \( s_{64} = 0 \), which is achieved by \( x_0 = 0, x_1 = 0, x_3 = 0, x_4 = 0, x_5 = 0, x_21 = 0, x_25 = 0, x_26 = 0, x_{27} = 0, x_{43} = 0, x_{46} = 0, x_{47} = 0, x_{48} = 0, x_{13} + x_{23} + x_{38} + x_{51} + x_{62} + K_2 = 0, \) and \( x_2 + x_{18} + x_{31} + x_{40} + x_{41} + x_{53} + x_{56} + K_3 = 0, \) where \( K_2 = k_1 + k_2 + k_4 + k_5 + k_6 + k_{10}k_{16}k_{22}k_{29}k_{34} + k_{10}k_{16} + k_{10}k_{29}k_{46}k_{64} + k_{10} + k_{11} + k_{12} + k_{15} + k_{16}k_{22}k_{61}k_{64} + k_{19} + k_{20} + k_{22}k_{29}k_{34}k_{38}k_{46}k_{53} + k_{22}k_{29}k_{34} + k_{28} + k_{29} + k_{32} + k_{33} + k_{34}k_{38}k_{53}k_{61} + k_{34}k_{38} + k_{34} + k_{38}k_{46}k_{53}k_{61}k_{64} + k_{38} + k_{44} + k_{45} + k_{46}k_{53}k_{61} + k_{46} + k_{49} + k_{53} + k_{57} + k_{58} + k_{61}k_{64} + k_{63} + k_{74} + k_{75} + 1. \)

Round 40: \( z \) depends on \( x_{37} \) via \( s_{64} \):

\[
 z = b_1 + b_2 + b_4 + b_{10} + b_{31} + b_{43} + b_{56} + s_{25} + b_{63} + s_3s_{64} + s_{46}s_{64} + s_{64}b_{63} + s_3s_{25}s_{46} + s_3s_{46}s_{64} + s_3s_{46}b_{63} + s_{25}s_{46}b_{63} + s_{46}s_{64}b_{63}
\]

The propagation is prevented by \( s_{43} = 0, s_{64} = 0, \) and \( b_{63} = 0, \) which is achieved by \( x_8 = 0, x_9 = 0, x_{10} = 0, x_{19} = 0, x_{28} = 0, x_{29} = 0, x_{31} = 0, x_{44} = 0, x_{49} = 0, x_{51} = 0, x_{52} = 0, x_{53} = 0, x_{57} = 0, x_6 + K_4 = 0, \) and \( x_7 + x_{20} + x_{23} + x_{32} + x_{45} + K_5 = 0, \) where \( K_4 = k_7 + k_8 + k_{10} + k_{16} + k_{37} + k_{49} + k_{62} + 1 \) and \( K_5 \) is a polynomial expression of degree 15 with 2365 monomials in 57 key variables.

Five expressions in key bits have to be guessed and 32 conditions are imposed in total. For “almost correct” guesses, a bias is detected in \( z \) after 97 rounds. More precisely, with samples of size \( N = 2^{22} \), a bias is detected at 4 of the \( 2^5 \) samples. It turns out that \( K_5 \) must be guessed correctly, but one of the other expressions can be wrong. This allows us to determine the correct guess in a Mastermind fashion.\(^{14}\) As an example, suppose that a bias is

\(^{14}\) See [http://en.wikipedia.org/wiki/Mastermind_(board_game)](http://en.wikipedia.org/wiki/Mastermind_(board_game)).
detected for $K \in \{(0, 1, 0, 0, 1), (0, 1, 0, 1, 1), (1, 1, 0, 1, 1), (0, 0, 1, 1)\}$. Then, the correct guess must be $(0, 1, 0, 1, 1)$, because it is the only candidate that differs by exactly one expression from the three other candidates. The principle works if less candidates are identified as long as the correct guess is among them. The attack requires $2^{27}$ chosen IVs and recovers five equations in key bits (however, one of them is quite large).

With samples of size $N = 2^{30}$ the bias can be detected after 104 rounds in about 50% of the cases on one of the samples. When a bias is detected, the guess for $K_1$ is always correct, but the other guesses might be wrong. The attack requires $2^{35}$ chosen IVs and recovers one bit of the key.

### 3.4.3 Related-Key Attack on 133 Rounds

The principle is the same as for the attacks on KATAN/KTANTAN in the single-key scenario. The initial difference and a corresponding characteristic are computed by linearized backward computation from an intermediate single bit difference after $q$ rounds. The intermediate difference is chosen at $s_{79}$ after $q = 20$ rounds, which gives the following initial difference in the IV and the key (no differences are allowed in the constant parts of the initial state, however):

$$(\Delta x, \Delta k) = (0x00000000000e810e, 0x000000000000000080456).$$

The following conditions are imposed in the first 20 rounds:

**Round 0:** $x_{46} + 1 = 0, x_{25} + k_{63} = 0$

**Round 1:** $k_{16}k_{22}k_{29}k_{34} + k_{16} + k_{29}k_{46}k_{64} = 0$

**Round 4:** $k_{13}k_{25}k_{32}k_{37} + k_{13} + k_{25}k_{64}k_{67} = 0$

**Round 5:** $x_{51} + 1 = 0, x_{30} + k_{68} = 0$

**Round 10:** $k_{25}k_{31}k_{38}k_{43} + k_{25} + k_{38}k_{55}k_{73} = 0$

**Round 12:** $x_{58} + 1 = 0, x_{37} + k_{75} = 0$

**Round 14:** $x_{60} + 1 = 0, x_{39} + k_{77} = 0$

**Round 15:** $x_{61} + 1 = 0, x_{40} + k_{78} = 0$

**Round 16:**

$$x_0 + x_{13} + x_{23} + x_{38} + k_1 + k_2 + k_4 + k_{10} + k_{31} + k_{43} + k_{56} + k_{63} = 0$$
The following condition additionally prevents the propagation at round 36:

**Round 36:** a difference is contained in $s_{46}$:

$$z = b_1 + b_2 + b_4 + b_{10} + b_{31} + b_{43} + b_{56} + s_{25} + b_{63} + s_3 s_{64} + s_{46} s_{64} + s_{64} b_{63} + s_3 s_{25} s_{46} + s_3 s_{46} s_{64} + s_3 s_{46} b_{63} + s_{25} s_{46} b_{63} + s_{46} s_{64} b_{63}$$

The propagation is prevented by $s_3 = 0$, $s_{64} = 0$, and $b_{63} = 0$, which is achieved by $x_1 = 0$, $x_4 = 0$, $x_5 = 0$, $x_{21} = 0$, $x_{26} = 0$, $x_{27} = 0$, $x_{38} = 0$, $x_{43} = 0$, $x_{47} = 0$, $x_{48} = 0$ $x_{14} + x_{24} + x_{52} + x_{63} + k_2 + k_3 + k_5 + k_{11} + k_{32} + k_{44} + k_{57} + k_{77} = 0$, and $x_2 + x_{13} + x_{15} + x_{18} + x_{23} + x_{53} + K = 0$, where $K$ is a polynomial expression of degree 10 with 163 monomials in 46 key variables.

Three conditions depend only on key bits. These conditions cannot be satisfied by the adversary and they define a class of weak keys. Eight expressions in key bits have to be guessed. A bias is detected for the correct guess after 133 rounds. With $N = 2^{27}$, the correct guess is always identified. The attack applies to $2^{77}$ weak keys, requires $2^{35}$ chosen IVs for two related keys and recovers eight equations in key bits.

### 3.5 Summary and Conclusion

Conditional differential cryptanalysis of NLFSR-based cryptosystems studies the propagation of differences through the first few updates of the NLFSR. Imposing conditions on the difference propagation yields explicit equations in key and IV variables. If these conditions are satisfied, many terms in the output difference virtually cancel out which leads to a detectable bias on IV samples that are derived from the conditions. Key dependent conditions can be exploited for key recovery.

The technique is quite effective against reduced-round Grain v1, where the best known attack in the single-key scenario is obtained. The attack works for 104 of the 160 rounds, requires $2^{35}$ chosen IVs, and recovers parts of the key. In the related-key scenario, our attack does not improve over [76].

The KATAN/KTANTAN family, on the other hand, showed a huge security margin against conditional differential attacks. This is mainly due to the following reasons:
Only very few rounds can be written out explicitly for condition analysis. Compared to the total number of rounds, the number of rounds that can be written out varies from about 7% for KATAN64 to about 10% for KATAN32 (compare Fig. 3.1 and 3.2).

Even if more rounds could be written out, one might not impose more conditions because, due to the small block length, not enough freedom degrees remain for choosing the samples.

After the controlled rounds, the differences spread freely. As any modern cipher, KATAN/KTANTAN is designed with sufficiently large security margin against plain differential attacks.

The framework is extended to higher order differential cryptanalysis in the next chapter, with applications to Grain-128, Grain-128a, and Trivium.
3.5 Summary and Conclusion

Figure 3.1: KATAN32: Evolution of degree and number of monomials.

Figure 3.2: KATAN64: Evolution of degree and number of monomials.
Figure 3.3: Trivium: Evolution of degree and number of monomials.

Figure 3.4: Grain v1: Evolution of degree and number of monomials.
3.5 Summary and Conclusion

Figure 3.5: Grain-128: Evolution of degree and number of monomials.

Figure 3.6: Grain-128a: Evolution of degree and number of monomials.
Conditional Differential Cryptanalysis of NLFSR
Chapter 4

High Order Differential Attacks on Stream Ciphers

In this chapter, conditional differential cryptanalysis is extended to higher order differential cryptanalysis. The resulting technique can be seen as a refinement and a generalization of previously proposed techniques such as the maximum degree test or cube testers. Before the technique is applied to Grain-128, Grain-128a, and Trivium, the most important chosen IV attacks on stream ciphers are reviewed — all of them have an elegant interpretation in terms of high order derivatives of Boolean functions.

4.1 Introduction

In the past, a variety of chosen IV attacks against stream ciphers were proposed. The keystream is modeled as a Boolean function $f : \{0, 1\}^\kappa \times \{0, 1\}^n \to \{0, 1\}$, mapping a $\kappa$-bit key and an $n$-bit IV to a keystream bit. The adversary is assumed to have black box access to $f(k, \cdot)$ for an unknown random key $k$. The techniques specify strategies to sum up the function’s output over specific IVs in order to distinguish $f(k, \cdot)$ from a random function or to recover information on the key. The most prominent example is the cube attack by Dinur and Shamir [41]. It recovers the full key of reduced-round Trivium with 767 rounds (66% of the full number of rounds). However, the cube attack exploits a rather specific property of Trivium and is very unlikely to apply to other ciphers (see Appendix A).
More general techniques are the maximum degree test by Englund, Johansson, and Turan [44], the derived functions approach for key recovery by Fischer, Khazaei, and Meier [48], and the cube testers by Aumasson, Dinur, Meier, and Shamir [8]. All of these techniques have a very elegant and natural description in terms of higher order derivatives of Boolean functions introduced by Lai [74] in 1994. This un masks them as high order differential attacks and suggests the use of conditional differential cryptanalysis in order to get more effective attacks. Condition analysis for higher order attacks, however, is less straightforward than in the first order case. One can think of several strategies to impose conditions. We propose one strategy that turned out to be effective in applications to the stream ciphers Grain-128, Grain-128a, and Trivium. The results are summarized in Table 4.1.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>$\kappa$</th>
<th>$n$</th>
<th>Rounds</th>
<th>Queries</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grain-128</td>
<td>128</td>
<td>96</td>
<td>197 (78%)</td>
<td>$2^{25}$</td>
<td>key recovery</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>207 (81%)</td>
<td>$2^{21}$</td>
<td>$2^{128}$ weak keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>215 (84%)</td>
<td>$2^{25}$</td>
<td>$2^{116}$ weak keys</td>
</tr>
<tr>
<td>Grain-128a</td>
<td>128</td>
<td>96</td>
<td>189 (74%)</td>
<td>$2^{26}$</td>
<td>$2^{122}$ weak keys</td>
</tr>
<tr>
<td>Trivium</td>
<td>80</td>
<td>80</td>
<td>798 (69%)</td>
<td>$2^{25}$</td>
<td>$2^{26}$ weak keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>961 (83%)</td>
<td>$2^{25}$</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.1:** Summary of results: $\kappa$ is the key length and $n$ is the length of the IV. If not stated otherwise, attacks are distinguishing attacks.

**Chapter Outline.** Section 4.2 introduces derivatives of Boolean functions and reviews previously proposed chosen IV attacks. Section 4.3 proposes conditional differential cryptanalysis for higher orders. In Sections 4.4 and 4.5 the framework is applied to the Grain-128, Grain-128a, and Trivium.

### 4.2 Review of High Order Differential Attacks

In this section, derivatives of Boolean functions are defined, following Lai [74]. Then, the terminology is used to review previous high order differential attacks that were originally presented in other terms.
4.2 Review of High Order Differential Attacks

4.2.1 High Order Derivatives of Boolean Functions

Lai [74] introduced derivatives of Boolean functions to formally describe high order differential cryptanalysis. Knudsen [71] applied the formalism to block cipher cryptanalysis. When modeling a stream cipher by Boolean functions, derivatives are the natural way to describe differential cryptanalysis.

**Definition.** For a Boolean function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \), the derivative of \( f \) with respect to \( v \in \{0, 1\}^n \) is defined as
\[
\Delta_v f(x) = f(x) + f(x + v).
\]

In this chapter, + denotes addition in the binary field and for vectors it is meant to be applied componentwise. The original definition in [74] is for discrete functions between Abelian groups, but for our applications, only the groups \((\{0, 1\}^n, +), n \geq 1 \), are of interest.

If \( V \subset \{0, 1\}^n \) is the linear subspace generated by \( v \), the definition of \( \Delta_v f(x) \) can be rewritten as \( f(x) + f(x + v) = \sum_{v \in V} f(x + v) \). This leads to the following generalization.

**Definition.** Let \( V \subset \{0, 1\}^n \) be a linear subspace of dimension \( d \). For a Boolean function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \), the \( d \)-th order derivative of \( f \) with respect to \( V \) is defined as
\[
\Delta_V f(x) = \sum_{v \in V} f(x + v).
\]

The output of a first order derivative \( \Delta_v f \) is the output difference of \( f \) for the input difference \( \Delta x = v \). Thus, the conditional differential attacks in the previous chapter detect a bias in the output of a derivative (on a well-chosen sample). The generalized attacks in this chapter, consider derivatives of order \( d > 1 \).

**General Properties of Derivatives**

In the following, it is always assumed that \( V \subset \{0, 1\}^n \) is a linear subspace of dimension \( d \) and that \( f : \{0, 1\}^n \rightarrow \{0, 1\} \) is a Boolean function.

**Property 4.1.** For any basis \( b_1, \ldots, b_d \) of \( V \), the derivative of \( f \) with respect to \( V \) can be written as \( \Delta_V f(x) = \Delta_{b_d} \cdots \Delta_{b_1} f(x) \).
Proof. This follows by writing out the definition. □

**Property 4.2.** If \( x, y \in \{0, 1\}^n \) with \( x + y \in V \), then \( \Delta_V f(x) = \Delta_V f(y) \).

Proof. The two affine spaces \( x + V \) and \( y + V \) are equal. Hence, \( \Delta_V f(x) \) and \( \Delta_V f(y) \) correspond to the same sum of outputs. □

**Property 4.3.** The degree of \( \Delta_V f \) is smaller than the degree of \( f \) by at least \( d \).

Proof. We argue that the statement is true in the first order case, then the result follows by induction. In the first order case, a monomial either cancels out completely (if it does not contain any variable affected by the difference) or it transforms into one are several monomials containing a strict subset of its original set of variables. □

**Notation for a Special Case**

Let \( v_i \in \{0, 1\}^n \) be the vector with a 1 at position \( i \) and 0s otherwise. The derivative \( \Delta_V f \) with respect to \( V = \langle v_i \rangle \) has the well-known interpretation of a partial derivative with respect to the variable \( x_i \) when \( f \) is considered as a multivariate polynomial over the real numbers. The notation \( v_i \) for single-bit differences will be used throughout the chapter.

### 4.2.2 Maximum Degree Test

Based on the works by Filiol [46] and Saarinen [102], Englund, Johansson, and Turan [44] present a general framework for chosen IV statistical attacks. The most effective attack of their framework is the maximum degree test.

Let \( f : \{0, 1\}^n \to \{0, 1\} \) be a Boolean function. The algebraic normal form of \( f \) is its representation as a multivariate polynomial. It has the form

\[
f(x_1, \ldots, x_n) = \sum_{u=0}^{2^n-1} a_u x^u
\]

where \( x^u = x_1^{u_1} \cdots x_n^{u_n} \) and \( u = (u_1, \ldots, u_n) \) is the binary representation. We say that the monomial \( x^u \) appears in \( f \) if \( a_u = 1 \). The degree of \( x^u \) is the Hamming weight of \( u \). The maximum degree test is based on the observation...
that the coefficient of the maximum degree monomial, that is $a_u$ with $u = 2^n - 1$, can be computed by summing $f$ over all inputs $x \in \{0, 1\}^n$:

$$a_{2^n - 1} = \sum_{x \in \{0, 1\}^n} f(x).$$

Englund, Johansson, and Turan propose to set $s \leq n$ IV variables to a random value and to check whether the maximum degree monomial is present in the resulting function $\{0, 1\}^{n-s} \rightarrow \{0, 1\}$. For a random polynomial $f$, the maximum degree polynomial is expected to appear with probability $1/2$.

In terms of derivatives, computing $a_{2^n - 1}$ is the same as evaluating the derivative $\Delta_V f$ for $V = \langle v_1, \ldots, v_n \rangle$ (not important at which point). Computing the maximum degree coefficient of a function that is derived by setting $s$ variables to a fixed value is equivalent to evaluating a derivative of order $n - s$ at some specific point. Testing for a bias in the presence of maximum degree coefficients for different assignments of the fixed variables turns out to be equivalent to testing for a bias in the output of a derivative.

### 4.2.3 Key Recovery with Derived Functions

Fischer, Khazaei, and Meier [48] propose the first approach to exploit the key dependency of the monomial coefficients for key recovery attacks. In terms of derivatives, they search for a subspace $V$ such that $\Delta_V f$ does not depend on all bits of the key. This is exploited for a divide-and-conquer key search strategy. The framework presented in [48] also allows more sophisticated scenarios where the influence of the key bits is not zero but only small.

### 4.2.4 Cube Attacks

The cube attacks are introduced by Dinur and Shamir [41]. A basic variant of the technique was independently proposed by Vielhaber [115].

The attack is described in ad hoc terminology which directly translates into higher order derivatives of Boolean functions. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a Boolean function. For an index set $I \subset \{1, \ldots, n\}$ define $t_I$ as the monomial containing all variables with index in $I$, these variables are called the cube variables and the size of $I$ is called the dimension of the cube. For each $I$ there is a unique polynomial $p$ such that $f$ can be written in the form

$$f(x) = t_I p(x) + q(x)$$
where \( p \) does not contain any cube variable and no monomial of \( q \) is divisible by \( t_I \). The polynomial \( p \) is called the superpoly of \( I \) in \( f \), and \( f \) is called the master polynomial. It is shown in [41, Theorem 1] that the superpoly of \( I \) in \( f \) can be evaluated by summing \( f \) over all possible configurations of the cube variables. As an example, let \( f \) be defined as

\[
f(x) = x_1x_5 + x_1x_2x_3x_6 + x_1x_2x_4 + x_3.
\]

The superpoly of \( I = \{1, 2\} \) in \( f \) computes as

\[
\sum_{x_1, x_2} f(x) = x_3x_6 + x_4,
\]

which is the polynomial that multiplies \( t_I = x_1x_2 \) in \( f \):

\[
f(x) = x_1x_2(x_3x_6 + x_4) + x_1x_5 + x_3.
\]

It follows immediately from this characterization that the superpoly of \( I \) in \( f \) is nothing else than the derivative of \( f \) with respect to \( V \), where \( V \) is generated by those \( v_i \) with \( i \in I \). The attack proposed by Dinur and Shamir works in two phases:

**Preprocessing:** Find \( \kappa \) derivatives \( \Delta_{V_1}f, \ldots, \Delta_{V_\kappa}f \) that are independent linear expressions in key bits. Note that the symbolic representation of \( f \) cannot be written out, but linearity of the derivative can be tested probabilistically by a test such as the one proposed by Blum, Luby, and Rubinfeld [22].

**Key recovery:** Evaluate the derivatives and solve the resulting system of linear equations in the key bits.

The attack is very elegant and efficient during the online phase. However, for real world ciphers it is quite unlikely that derivatives can be found which are linear expressions in key bits (see Appendix).

### 4.2.5 Cube Testers

Cube testers are introduced by Aumasson, Dinur, Meier, and Shamir [8]. Even though they build on the same terminology, they are conceptionally quite different from the original cube attack. The idea is to test a superpoly (that is, a derivative) for certain properties that can be detected by a relatively
small number of evaluations. The most effective test appears to be the test for balance, that is, the superpoly is tested probabilistically for a bias in the output. This is what Englund, Johansson, and Turan called the maximum degree test.

4.3 Higher Order Conditional Cryptanalysis

We have seen that the maximum degree test (and the most effective variant of a cube tester) essentially perform a bias test on the output of a derivative. Considering the technique as a differential attack suggests the use of ideas from the previous chapter. But first we consider a generalization that is not captured by the original interpretation.

4.3.1 Generalizing Previous Techniques

The maximum degree test is the extension of a simple bit-flip test. Only derivatives with respect to subspaces of the form \( V = \langle v_{i_1}, \ldots, v_{i_d} \rangle \) are considered. This does not include \( V = \langle v_1 + v_2, v_3 \rangle \), for example.

Considering more general derivatives turned out to be advantageous in the first order case (notably for KATAN/KTANTAN and for Grain v1 in the related-key scenario). For the higher order case however, it is open whether the generalization can lead to better attacks. Our applications will only use derivatives that fit into the special case of maximum degree tests — choosing the fixed and variable bits as well as the configuration of the fixed bits in a highly specific way, though.

4.3.2 Imposing Conditions for High Order Attacks

Recall the general framework of NLFSR-based constructions from Section 3.2 and consider a derivative \( \Delta_V f \) for \( V = \langle b_1, \ldots, b_d \rangle \).

A first idea could be to generalize the notion of a differential characteristic to a higher order differential characteristic. One would consider the propagations of higher order differences:

\[
\Delta_V t_i = \sum_{v \in V} t_i(k, x + v).
\]
Conditions of the form $\Delta V t_i = 0$ (or $\Delta V t_i = 1$) would be imposed. The problem with this approach is that $\Delta V t_i = 0$ holds trivially for the first few rounds, because the degree of $t_i$ is smaller than $d$ (see Property 4.3). This requires that much more rounds must be written out until non-trivial conditions can be derived.

Motivated by the interpretation of $\Delta V f$ as $\Delta b_d \ldots \Delta b_1 f$ (see Property 4.1) we propose the following heuristic strategy to impose conditions: consider each “basis difference” $b_i$ as a first order difference, control its propagation by conditions and merge the conditions for all differences to derive the sample.

Compared to the first order case, more conditions will be involved and there is quite some chance that conditions from one difference contradict conditions from another difference (which finally results in $X = \emptyset$). Algorithm 4.1 (an extension of Algorithm 3.1) turned out to be useful for finding conditions in the higher order case. It takes as input a basis of $V$ and a number $q$ and it returns the merged conditions from which the sample can be derived directly. For each difference, the propagation is prevented whenever possible for the first $q$ rounds. New conditions are considered modulo the previous conditions in order to avoid contradictions.

**Algorithm 4.1** Derive conditions for higher order attacks.

**Input:** $b_1, \ldots, b_d, q$

**Output:** a set of conditions

$$J \leftarrow \langle 0 \rangle$$

for $i = 1$ to $d$

for $j = 0$ to $q - 1$

$p \leftarrow \Delta b_i t_j$

if $p \neq 1 \mod J$ then

$J = J + \langle p \rangle$

end if

end for

end for

return a basis of $J$

**Note:** Algorithm 4.1 processes the differences sequentially and new conditions are imposed with respect to previous conditions. As a result, the order in which the differences are processed can make a difference. The conditions can also slightly vary for different bases.
4.4 Application to Grain-128 and Grain-128a

Grain-128 is a stream cipher proposed by Hell, Johansson, Maximov, and Meier [53] as a bigger version of Grain v1. It accepts a 128-bit key and a 96-bit IV. The general construction of the cipher is the same as for Grain v1, but the update functions are more sparse and have significantly lower degree.

Cryptanalytic Results. Grain-128 has received a lot of cryptanalytic attention. Englund, Johansson, and Turan [44] apply the maximum degree test for a distinguishing attack on 192 (of 256) rounds. De Cannière, Küçük, and Preneel [29] claim a first order differential attack on 192 rounds as well. The best key recovery attack is shown by Fischer, Khazaei, and Meier [48] using the derived functions approach. It works for 180 rounds and is able to speed up exhaustive key search by a factor $2^{4}$.

Our attacks significantly improve over these results, showing a distinguisher for up to 215 rounds, and key recovery attacks for only slightly less rounds. The results indicate that the security margin of Grain-128 is critically low. Indeed, our analysis got exceeded by a key recovery attack on the full Grain-128 by Dinur and Shamir [42]. Their attack is briefly discussed below. In response to these serious weaknesses, Ågren, Hell, Johansson, and Meier [2] recently proposed Grain-128a. We are not aware of previous cryptanalytic results on Grain-128a.

4.4.1 Description of Grain-128

The internal state of Grain-128 consists of 256 bits organized in two registers of length 128. The registers, denoted by $(s_0, \ldots, s_{127})$ and $(b_0, \ldots, b_{127})$, are initialized with the key $k = (k_0, \ldots, k_{127})$ and the IV $x = (x_0, \ldots, x_{95})$ as follows:

$$(s_0, \ldots, s_{127}) \leftarrow (x_0, \ldots, x_{95}, 1, \ldots, 1)$$
$$(b_0, \ldots, b_{127}) \leftarrow (k_0, \ldots, k_{127}).$$
The state is updated 256 times as follows (without producing keystream):

\[
\begin{align*}
t_1 &\leftarrow s_0 + s_7 + s_{38} + s_{70} + s_{81} + s_{96} \\
t_2 &\leftarrow s_0 + b_0 + b_{26} + b_{56} + b_{91} + b_{96} + b_{b_{67}} + b_{111}b_{13} \\
&\quad + b_{17}b_{18} + b_{27}b_{59} + b_{40}b_{48} + b_{61}b_{65} + b_{68}b_{84} \\
z &\leftarrow b_{12}b_{8} + s_{13}s_{20} + b_{95}b_{42} + s_{60}s_{79} + b_{12}b_{95}s_{95} \\
&\quad + b_2 + b_{15} + b_{36} + b_{45} + b_{64} + b_{73} + b_{89} + s_{93} \\
(s_0, \ldots, s_{127}) &\leftarrow (s_1, \ldots, s_{127}, t_1 + z) \\
(b_0, \ldots, b_{127}) &\leftarrow (b_1, \ldots, b_{127}, t_2 + z).
\end{align*}
\]

After 256 rounds the initialization phase is done and the cipher starts generating keystream; \(t_1\), \(t_2\), and \(z\) are computed as before, \(z\) is output as a keystream bit and the state is updated as follows:

\[
\begin{align*}
(s_0, \ldots, s_{127}) &\leftarrow (s_1, \ldots, s_{127}, t_1) \\
(b_0, \ldots, b_{127}) &\leftarrow (b_1, \ldots, b_{127}, t_2).
\end{align*}
\]

Reduced-round variants are attacked by considering \(z\) generated at round \(r < 256\).

### 4.4.2 First Order Attack on 161 Rounds

Let us consider a first order attack in the spirit of the attack against Grain v1 in Section 3.4 using the IV difference \(v_{69}\) (recall that this is the single-bit difference with difference in \(x_{69}\)). The difference is chosen by the same rationales as for Grain v1: bit \(x_{69}\) takes the most clocks until it spreads to the non-linear register. The following conditions control the difference propagation:

- **Round 9:** \(x_{88} = 0\)
- **Round 27:** \(k_{122} = 0\)
- **Round 49:** \(x_{62} = 0\)
- **Round 56:** \(x_{76} = 0\)
- **Round 61:** \(k_{73} = 0\)
4.4 Application to Grain-128 and Grain-128a

Round 64: \[ x_{31}k_{76} + x_{39}k_{43}k_{76} + x_{44}x_{51}k_{76} + x_{73}k_{76}k_{126} + x_{91}k_{76} + k_{31}k_{76} + k_{33}k_{76} + k_{34}k_{76}k_{98} + k_{42}k_{44}k_{76} + k_{43}k_{76}k_{126} + k_{46}k_{76} + k_{48}k_{49}k_{76} + k_{57}k_{76} + k_{58}k_{76}k_{90} + k_{67}k_{76} + k_{71}k_{76}k_{79} + k_{76}k_{87} + k_{76}k_{92}k_{96} + k_{76}k_{95} + k_{76}k_{99}k_{115} + k_{76}k_{104} + k_{76}k_{120} + k_{76}k_{127} = 0. \]

The bias is always detected for samples of size \( N = 2^{14} \) in \( \Delta_{V}z \) after 161 rounds for \( V = \langle v_{69} \rangle \). It turns out that the conditions at the rounds 61 and 64 do not significantly influence the bias, they can be ignored when deriving the samples. However, the condition \( k_{122} = 0 \) is essential. The attack only works for those \( 2^{127} \) weak keys that satisfy the condition.

4.4.3 Second Order Attack on 189 Rounds

The attack is now extended to the second order by choosing \( v_{68} \) as an additional difference, that is, by considering \( \Delta_{V}z \) for \( V = \langle v_{68}, v_{69} \rangle \). This difference is chosen for two reasons: first, because it spreads lately to the nonlinear register, and second, because propagation of \( v_{68} \) does not depend on \( x_{69} \) (at least not until round 80, as can be seen from the conditions). The following conditions control the propagation of \( v_{68} \):

**Until round 55:** \( x_{87} = 0, k_{121} = 0, x_{61} = 0, x_{75} = 0 \)

**Round 60:** \( k_{72} = 0 \)

**Round 63:** \[ x_{30}k_{75} + x_{38}k_{42}k_{75} + x_{43}x_{50}k_{75} + x_{72}k_{75}k_{125} + x_{90}k_{75} + k_{30}k_{75} + k_{32}k_{75} + k_{33}k_{75}k_{97} + k_{41}k_{43}k_{75} + k_{42}k_{75}k_{125} + k_{45}k_{75} + k_{47}k_{48}k_{75} + k_{56}k_{75} + k_{57}k_{75}k_{89} + k_{66}k_{75} + k_{70}k_{75}k_{78} + k_{75}k_{86} + k_{75}k_{91}k_{95} + k_{75}k_{94} + k_{75}k_{98}k_{114} + k_{75}k_{103} + k_{75}k_{119} + k_{75}k_{126} = 0, \]

Now, the conditions for both differences are considered when deriving the sample. For \( N = 2^{18} \), a bias is always detected after 189 rounds. Note that the attack requires \( 2^{22} \) queries, because each evaluation of \( \Delta_{V}z \) requires \( 2^2 \) queries. As in the first order case, it turns out that the conditions at the rounds 60, 61, 63, and 64 can be ignored, but \( k_{121} = 0 \) and \( k_{122} = 0 \) are essential for detecting the bias.
4.4.4 Higher Order Attacks up to 215 Rounds

Finding suitable differences for higher order attacks is not an exact science. The given heuristics (late diffusion to non-linear register and independent propagation from other differences) give some hints, but finally, differences have to be tested experimentally.

As an example, \(\langle v_{67}, v_{69}, v_{69} \rangle\) yields an attack on 207 rounds and the differences can be “shifted”: \(\langle v_{67-i}, v_{68-i}, v_{69-i} \rangle\) yields an attack on \(207 - i\) rounds, for \(0 \leq i \leq 7\), if the following conditions are satisfied:

- for \(v_{67-i}: x_{86-i} = 0, k_{120-i} = 0, x_{60-i} = 0, x_{74-i} = 0\)
- for \(v_{68-i}: x_{87-i} = 0, k_{121-i} = 0, x_{61-i} = 0, x_{75-i} = 0\)
- for \(v_{69-i}: x_{88-i} = 0, k_{122-i} = 0, x_{62-i} = 0, x_{76-i} = 0\)

However, adding one more consecutive difference such as \(\langle v_{66}, v_{67}, v_{69}, v_{69} \rangle\), does not further increase the number of rounds. A reason for this cannot be found in the propagations through the first 80 rounds.

Generally, increasing the order of the derivative does not increase the number of attackable rounds at will. Rather there is a limit around 210 rounds that is hard to break. Propagations of the differences heavily influence each other and conditions from one difference contradict those of other differences. Our best attack is for 215 rounds and uses

\[
V = \langle v_0, v_1, v_2, v_{34}, v_{35}, v_{36}, v_{37}, v_{65}, v_{66}, v_{67}, v_{68}, v_{69}, v_{95} \rangle.
\]

The following conditions are imposed:

- for \(v_0\): \(k_{45} = 0\)
- for \(v_1\): \(k_{46} = 0\)
- for \(v_2\): \(k_{47} = 0\)
- for \(v_{34}\): \(x_{27} = 0, x_{41} = 0, k_{38} = 0\)
- for \(v_{35}\): \(x_{28} = 0, x_{42} = 0, k_{39} = 0\)
- for \(v_{36}\): \(x_{29} = 0, x_{43} = 0, k_{40} = 0\)
- for \(v_{37}\): \(x_{30} = 0, x_{44} = 0, k_{41} = 0\)
4.4 Application to Grain-128 and Grain-128a

- for $v_{65}$: $x_{84} = 0$, $k_{118} = 0$, $x_{58} = 0$, $x_{72} = 0$
- for $v_{66}$: $x_{85} = 0$, $k_{119} = 0$, $x_{59} = 0$, $x_{73} = 0$
- for $v_{67}$: $x_{86} = 0$, $k_{120} = 0$, $x_{60} = 0$, $x_{74} = 0$
- for $v_{68}$: $x_{87} = 0$, $k_{121} = 0$, $x_{61} = 0$, $x_{75} = 0$
- for $v_{69}$: $x_{88} = 0$, $k_{122} = 0$, $x_{62} = 0$, $x_{76} = 0$
- for $v_{95}$: $x_{76} = 0$, $k_{47} = 0$

If all conditions are satisfied, a bias after 215 rounds is always detected for $N = 2^{12}$. The attack requires $2^{25}$ queries and works for $2^{116}$ weak keys. The bias can still be detected when only half of the key conditions are satisfied (it is not important which ones). However, the bias is only rarely detected for keys that satisfy less conditions.

### 4.4.5 Exploiting Biases for Key Recovery

Biases heavily depend on the conditions. As a result, conditions on the key can be exploited for key recovery with high reliability. The attack is explained by a prototypical example that recovers three bits for a round-reduced variant of Grain-128 with 197 rounds. Biases after 197, 199, and 200 rounds are exploited and bits are recovered correctly with high probability (close to 90%). The example can be extended to recover essentially the whole key and the same strategy can be applied for key recovery on 213 rounds, with lower success probability, however.

**Recovering $k_{40}$, $k_{119}$, and $k_{120}$**. Derivatives of order five are considered with respect to

$$V = \langle v_1, v_{36}, v_{66}, v_{67}, v_{68} \rangle$$

The chosen differences are a subset of the differences for the 215-round attack above and the same conditions are used for the individual differences. The key bits $k_{40}, k_{119}, k_{120}$, and $k_{121}$ are involved, but only $k_{40}, k_{119},$ and $k_{121}$ will be recovered. These bits have a strong influence on the bias after 199, 200, and 197 rounds, respectively.

Each bit is recovered individually, let us consider $k_{40}$ as an example. On a sample that satisfies all IV conditions, $\Delta_V z$ is evaluated after 199 rounds. Let $S$ be the number of 1s on a sample of size $N$. The value of $k_{40}$ is determined
by a binary hypothesis test. A threshold \( \pi \) \((0 \leq \pi \leq 1)\) is fixed in advance (determined by experiments). Then, a decision is taken as follows:

\[
k_{40} = \begin{cases} 
0 & \text{if } S/N < \pi, \\
1 & \text{otherwise.}
\end{cases}
\]

In comparison to just decide for \( k_{40} = 0 \) when a significant bias is detected, choosing the specific thresholds improves the success probabilities. By success probability we mean the probability of a correct decision. If \( \alpha \) and \( \beta \) are the probabilities of the type I and the type II error, respectively, the probability of a correct decision is given by \( 1 - (\alpha + \beta)/2 \). An optimal \( \pi \) that maximizes this probability is determined experimentally by computing histograms. Table 4.2 shows the resulting \( \pi \) and the corresponding success probabilities.

<table>
<thead>
<tr>
<th>( V )</th>
<th>Key bit</th>
<th>Round</th>
<th>Threshold</th>
<th>( \text{Pr[correct]} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle v_1, v_{36}, v_{66}, v_{67}, v_{68} \rangle )</td>
<td>( k_{40} )</td>
<td>199</td>
<td>0.494</td>
<td>0.801</td>
</tr>
<tr>
<td></td>
<td>( k_{119} )</td>
<td>200</td>
<td>0.492</td>
<td>0.682</td>
</tr>
<tr>
<td></td>
<td>( k_{121} )</td>
<td>197</td>
<td>0.486</td>
<td>0.867</td>
</tr>
</tbody>
</table>

**Table 4.2:** Key recovery by hypothesis testing: the success probabilities are computed for an adversary making \( 2^{25} \) queries.

**Recovering More Bits.** More bits can be recovered by choosing other differences. For example, eight more bits can be recovered by just shifting the differences. Table 4.3 shows the results. It turns out that the individual tests are not independent from each other and much better tests can be designed if parts of the bits are already known. Working out all the details is tedious, but it seems feasible to recover the whole key in this way.

<table>
<thead>
<tr>
<th>( V )</th>
<th>Key bit</th>
<th>Round</th>
<th>Threshold</th>
<th>( \text{Pr[correct]} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle v_2, v_{37}, v_{67}, v_{68}, v_{69} \rangle )</td>
<td>( k_{41} )</td>
<td>200</td>
<td>0.492</td>
<td>0.792</td>
</tr>
<tr>
<td></td>
<td>( k_{120} )</td>
<td>201</td>
<td>0.490</td>
<td>0.680</td>
</tr>
<tr>
<td></td>
<td>( k_{122} )</td>
<td>198</td>
<td>0.488</td>
<td>0.872</td>
</tr>
<tr>
<td>( \langle v_0, v_{35}, v_{65}, v_{66}, v_{67} \rangle )</td>
<td>( k_{39} )</td>
<td>198</td>
<td>0.492</td>
<td>0.795</td>
</tr>
<tr>
<td></td>
<td>( k_{118} )</td>
<td>199</td>
<td>0.492</td>
<td>0.681</td>
</tr>
<tr>
<td></td>
<td>( k_{120} )</td>
<td>196</td>
<td>0.492</td>
<td>0.868</td>
</tr>
</tbody>
</table>

**Table 4.3:** Recovering more bits by choosing other \( V \) (extension to Table 4.2).
Attacking More Rounds. The key recovery also works with higher order derivatives, for example with the one we used for the 215-round distinguishing attack. However, the more key bits that are involved, the less isolated dependencies can be found, which reduces the success probability. Table 4.4 shows the results. After 213 rounds, the key bits $k_{39}$ and $k_{122}$ can be recovered with success probability around 55% which is only slightly better than a random guess.

<table>
<thead>
<tr>
<th>$V$</th>
<th>Key bits</th>
<th>Round</th>
<th>Threshold</th>
<th>$\Pr[\text{correct}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle v_0, v_1, v_2, v_{34}, v_{35}, v_{36}, v_{37}, v_{65}, v_{66}, v_{67}, v_{68}, v_{69}, v_{95} \rangle$</td>
<td>$k_{39}$</td>
<td>213</td>
<td>0.490</td>
<td>0.591</td>
</tr>
<tr>
<td></td>
<td>$k_{72}$</td>
<td>213</td>
<td>0.488</td>
<td>0.566</td>
</tr>
<tr>
<td></td>
<td>$k_{119}$</td>
<td>206</td>
<td>0.356</td>
<td>0.830</td>
</tr>
<tr>
<td></td>
<td>$k_{120}$</td>
<td>207</td>
<td>0.486</td>
<td>0.807</td>
</tr>
<tr>
<td></td>
<td>$k_{120}$</td>
<td>211</td>
<td>0.484</td>
<td>0.592</td>
</tr>
<tr>
<td></td>
<td>$k_{122}$</td>
<td>213</td>
<td>0.478</td>
<td>0.581</td>
</tr>
</tbody>
</table>

Table 4.4: Key recovery up to 213 rounds with derivative of order 13.

4.4.6 Breaking Grain-128 with Dynamic Cube Attacks

Recently, Dinur and Shamir [42] presented a key recovery attack on the full Grain-128. Their technique, called dynamic cube attack, uses the output function (rather than the first few rounds) of Grain-128 to derive conditions and they dynamically adapt IV variables during the evaluation of derivatives. The attack recovers the full key about $2^{38}$ times faster than exhaustive search and is experimentally verified by Dinur, Güneysu, Paar, Shamir, and Zimmermann [40].

4.4.7 Description of Grain-128a

The internal state of Grain-128a is the same as for Grain-128 and we use the same notation. The registers are initialized as follows (changes to Grain-128 are in boldface):

$$(s_0, \ldots, s_{127}) \leftarrow (x_0, \ldots, x_{95}, 1, \ldots, 1, 0)$$

$$(b_0, \ldots, b_{127}) \leftarrow (k_0, \ldots, k_{127}).$$
The state is updated as follows:

\[
\begin{align*}
t_1 & \leftarrow s_0 + s_7 + s_{38} + s_{70} + s_{81} + s_{96} \\
t_2 & \leftarrow s_0 + b_0 + b_{26} + b_{56} + b_{91} + b_{96} + b_{67} + b_{11}b_{13} \\
& \quad + b_{17}b_{18} + b_{27}b_{59} + b_{40}b_{48} + b_{61}b_{65} + b_{68}b_{84} \\
& \quad + b_{88}b_{92}b_{93}b_{95} + b_{22}b_{24}b_{25} + b_{70}b_{78}b_{82} \\
z & \leftarrow b_{12}s_8 + s_{13}s_{20} + b_{95}s_{42} + s_{60}s_{79} + b_{12}b_{95}s_{94} \\
& \quad + b_2 + b_{15} + b_{36} + b_{45} + b_{64} + b_{73} + b_{89} + s_{93}
\end{align*}
\]

\((s_0, \ldots, s_{127}) \leftarrow (s_1, \ldots, s_{127}, t_1 + z)\)

\((b_0, \ldots, b_{127}) \leftarrow (b_1, \ldots, b_{127}, t_2 + z)\).

After 256 rounds the initialization phase is done and the cipher starts generating keystream; \(t_1\), \(t_2\), and \(z\) are computed as before, \(z\) is output as a keystream bit and the state is updated as follows:

\[(s_0, \ldots, s_{127}) \leftarrow (s_1, \ldots, s_{127}, t_1)\]

\[(b_0, \ldots, b_{127}) \leftarrow (b_1, \ldots, b_{127}, t_2).\]

Reduced-round variants are attacked when considering \(z\) generated at round \(r < 256\).

Note: Grain-128a has optional support for authentication. If authentication is used, the first 64 keystream bits are not available to the adversary.

### 4.4.8 Distinguishing Attacks up to 189 Rounds

For the first few rounds, the conditions for \(v_{69-i}, 0 \leq i \leq 9\), are the same as for Grain-128: \(x_{88-i} = 0, k_{122-i} = 0, x_{62-i} = 0, x_{76-i} = 0\). The attacks are basically the same as for Grain-128, but biases are detected at different rounds. Table 4.5 compares the number of rounds that can be attacked for Grain-128 and Grain-128a, respectively.

The attack of order thirteen which reaches 215 rounds on Grain-128 performs very badly on Grain-128a. Only around 170 rounds can be attacked (depending on which conditions are imposed). The reason is that the differences \(v_0, v_1, \text{ and } v_2\), in particular, but also \(v_{34}, v_{35}, v_{36}, \text{ and } v_{37}\) propagate quickly to the NLFSR and spread through the newly added high degree monomials. The attack of order 6 for 189 rounds (see Table 4.5) is the best attack that we could find.
4.5 Application to Trivium

Trivium [27] was designed by De Cannière and Preneel and was selected for the final eSTREAM portfolio [43]. It accepts a 80-bit key and a 80-bit IV.

Cryptanalytic Results. The maximum degree test by Englund, Johansson, and Turan [44] gives a distinguishing attack on 704 (of 1152) rounds. The derived functions approach by Fischer, Khazaei, and Meier is able to speed up exhaustive key search for 672 rounds by about a factor $2^{25}$. The most impressive attack is the cube attack by Dinur and Shamir [41]. It recovers the full key for 735 rounds in practical time, and for 767 rounds in close to practical time. The attack on the largest number of rounds is due to Aumasson, Dinur, Meier, and Shamir [8], it is a distinguisher for 790 rounds based on a cube tester.

Our main result further increases the number of attackable rounds by a distinguisher on 961 rounds, that works for a class of $2^{26}$ weak keys.

4.5.1 Description of Trivium

The internal state consists of 288 bits which are divided into three NLFSRs of lengths 93, 84 and 111, respectively. They are initialized with the key $k = (k_0, \ldots, k_{79})$ and the IV $x = (x_0, \ldots, x_{79})$ as follows:

\[
\begin{align*}
(s_1, \ldots, s_{93}) &\leftarrow (k_0, \ldots, k_{79}, 0, \ldots, 0) \\
(s_{94}, \ldots, s_{177}) &\leftarrow (x_0, \ldots, x_{79}, 0, 0, 0, 0) \\
(s_{178}, \ldots, s_{288}) &\leftarrow (0, \ldots, 0, 1, 1, 1).
\end{align*}
\]
The state is then updated iteratively by the following round transformation:

\[
\begin{align*}
t_1 & \leftarrow s_{66} + s_{93} \\
t_2 & \leftarrow s_{162} + s_{177} \\
t_3 & \leftarrow s_{243} + s_{288} \\
z & \leftarrow t_1 + t_2 + t_3 \\
t_1 & \leftarrow t_1 + s_{91}s_{92} + s_{171} \\
t_2 & \leftarrow t_2 + s_{175}s_{176} + s_{264} \\
t_3 & \leftarrow t_3 + s_{286}s_{287} + s_{69} \\
(s_1, \ldots, s_{93}) & \leftarrow (t_3, s_1, \ldots, s_{92}) \\
(s_{94}, \ldots, s_{177}) & \leftarrow (t_1, s_{94}, \ldots, s_{176}) \\
(s_{178}, \ldots, s_{288}) & \leftarrow (t_2, s_{178}, \ldots, s_{287}).
\end{align*}
\]

No output is produced during the first 1152 rounds, then the value of \(z\) is output as the keystream. Reduced-round variants are attacked by considering \(z\) generated at round \(r < 1152\).

### 4.5.2 Preliminary Study of Conditions

In comparison to the Grain family, Trivium has very sparse and low-degree update functions. This allows to write out explicitly more than 350 rounds (see Fig. 3.3). Another property is the strong symmetry in Trivium. The conditions look very much the same for all single bit differences. In the very early rounds, two propagations depend on key bits only. This makes that the adversary cannot control these propagations except by assuming weak keys. As a representative example, let us consider the propagation of \(v_3\):

**Round 0:** \(x_{68} + x_{77} + k_{65} + k_{68}\)

**Round 78:** \(x_4\)

**Round 79:** \(x_2\)

**Round 156:** \(k_{17}k_{18} + k_{19} + k_{61}\)

**Round 157:** \(k_{15}k_{16} + k_{17} + k_{59}\)

**Round 174:** \(x_{17}x_{18} + x_{19}\)

**Round 175:** \(x_{15}x_{16} + x_{17}\)
4.5 Application to Trivium

Round 189: $x_{67} + k_{55}$

Round 190: $x_0x_1 + x_{65} + k_{53} + k_{78}k_{79}$

4.5.3 Distinguishing Attacks up to 961 Rounds

Due to the particular form of the conditions (weak key conditions) and in view of very impressive attacks on Trivium with significantly reduced number of rounds, our attack goal was the following: attack the maximal number of rounds, accepting that the attack only works for a small class of weak keys.

The differences are chosen such that they propagate independently from each other. This precludes choosing differences at a distance one, because such neighboring differences influence each other in the very early rounds due to the quadratic monomials in the update functions ($x_4$ and $x_2$ control the propagation of $v_3$ at the rounds 78 and 78, respectively, see above). Motivated by an observation of Maximov and Biryukov [85] differences are chosen at a distance of three. Specifically, $V$ is chosen as

$$V = \langle v_0, v_3, v_6, v_9, \ldots, v_{60}, v_{63}, v_{66}, v_{69} \rangle.$$  

The dimension of $d = 24$ was fixed a posteriori in order to have a sufficient freedom degree to mount the distinguisher.

The conditions are derived by Algorithm 4.1 with $q = 200$. That is, each difference is processed sequentially and propagations are prevented whenever possible for the first 200 rounds.

The working of the algorithm is illustrated by showing the merged and simplified conditions at intermediate steps:

**Conditions for $v_0$:**

$$x_1 = 0, x_{12}x_{13} + x_{14} = 0, x_{14}x_{15} + x_{16} = 0, x_{77} + k_{65} = 0, x_{62} + x_{75}x_{76} + x_{75}k_{64} + x_{76}k_{63} + k_{50} + k_{63}k_{64} + k_{75}k_{76} + k_{77} = 0, x_{64} + k_{52} + k_{77}k_{78} + k_{79} = 0, k_{12}k_{13} + k_{14} + k_{56} = 0, k_{14}k_{15} + k_{16} + k_{58} = 0$$

**Conditions for $v_0$, $v_1$:**

$$x_1 = 0, x_2 = 0, x_4 = 0, x_{12}x_{13} + x_{14} = 0, x_{14}x_{15} + x_{16} = 0, x_{15}x_{16} + x_{17} = 0, x_{17}x_{18} + x_{19} = 0, x_{62} + x_{75}x_{76} + x_{75}k_{64} + x_{76}k_{63} + k_{50} + k_{63}k_{64} + k_{75}k_{76} + k_{77} = 0, x_{64} + k_{52} + k_{77}k_{78} + k_{79} = 0, x_{65} + k_{53} + k_{78}k_{79} = 0, x_{67} + k_{55} = 0, x_{68} + k_{68} = 0, x_{77} + k_{65} = 0, k_{12}k_{13} + k_{14} + k_{56} = 0, k_{14}k_{15} + k_{16} + k_{58} = 0, k_{15}k_{16} + k_{17} + k_{59} = 0, k_{17}k_{18} + k_{19} + k_{61} = 0$$
Conditions for $v_0, v_1, \ldots, v_{36}$:

\begin{align*}
x_1 &= 0, x_2 = 0, x_4 = 0, x_5 = 0, x_7 = 0, x_8 = 0, x_{10} = 0, x_{11} = 0, x_{13} = 0, \\
x_{14} &= 0, x_{16} = 0, x_{17} = 0, x_{19} = 0, x_{20} = 0, x_{22} = 0, x_{23} = 0, x_{25} = 0, \\
x_{26} &= 0, x_{28} = 0, x_{29} = 0, x_{31} = 0, x_{32} = 0, x_{34} = 0, x_{35} = 0, x_{37} = 0, \\
x_{38} &= 0, x_{40} = 0, x_{41} = 0, x_{43} = 0, x_{44} = 0, x_{46} = 0, x_{47} = 0, x_{49} = 0, \\
x_{50} &= 0, x_{52} = 0, x_{62} + k_{50} + k_{75}k_{76} + k_{77} = 0, x_{64} + k_{52} + k_{77}k_{78} + k_{79} = 0, x_{65} + k_{53} + k_{78}k_{79} = 0, x_{67} + k_{55} = 0, x_{68} + k_{68} = 0, x_{70} + k_{58} = 0, \\
x_{71} + k_{59} &= 0, x_{73} + k_{61} = 0, x_{74} + k_{62} = 0, x_{76} + k_{64} = 0, x_{77} + k_{65} = 0, \\
k_1 + k_{26}k_{27} + k_{28} &= 0, k_2 + k_{27}k_{28} = 0, k_4 = 0, k_5 = 0, k_7 = 0, k_8 = 0, \\
k_{10} &= 0, k_{11} = 0, k_{13} = 0, k_{14} + k_{68} = 0, k_{15}k_{43} + k_{42}k_{58} + k_{42}k_{68} + k_{44} + k_{59} = 0, k_{15}k_{68} + k_{42}k_{68} + k_{43} + k_{58} = 0, k_{16} + k_{42}k_{68} + k_{43} = 0, \\
k_{17} + k_{42}k_{43} + k_{44} &= 0, k_{18}k_{42}k_{43} + k_{18}k_{44} + k_{44}k_{45} + k_{46} + k_{61} = 0, \\
k_{18}k_{44}k_{45} + k_{18}k_{46} + k_{45}k_{46} + k_{47} + k_{62} &= 0, k_{19} + k_{44}k_{45} + k_{46} = 0, \\
k_{20} + k_{45}k_{46} + k_{47} &= 0, k_{21}k_{45}k_{46} + k_{21}k_{47} + k_{47}k_{48} + k_{49} + k_{64} = 0, \\
k_{21}k_{47}k_{48} + k_{21}k_{49} + k_{48}k_{49} &= 0, k_{50} + k_{65} = 0, k_{22} + k_{47}k_{48} + k_{49} = 0, \\
k_{23} + k_{48}k_{49} + k_{50} &= 0, k_{24}k_{48}k_{49} + k_{24}k_{50} + k_{50}k_{51} + k_{52} + k_{67} + 1 = 0, \\
k_{24}k_{50}k_{51} + k_{24}k_{52} + k_{26} + k_{68} &= 0, k_{25} + k_{50}k_{51} + k_{52} = 0, k_{29} = 0, \\
k_{31} &= 0, k_{32} = 0, k_{34} = 0, k_{35} = 0, k_{37} = 0, k_{38} = 0, k_{40} = 0, \\
k_{41} + k_{68} &= 0, k_{56} + k_{68} = 0
\end{align*}

...
After processing all differences, the conditions collapse into single bit conditions. From the IV, only the bits $x_{72}$, $x_{75}$, and $x_{78}$ are not fixed by conditions and not affected by a difference. Our attack tests for a “bias” in $\Delta V z$ on these bits. More precisely, each bit is tested separately by computing $\Delta v_i \Delta V z = \Delta V z(k, x) + \Delta V z(k, x + v_i)$ for $i = 72, 75, 78$. Table 4.6 shows the probability $p$ that the result is 1. Testing a single bit requires $2^{25}$ chosen IVs and gives a distinguisher with advantage $|1/2 - p|$. Two scenarios are considered: first, when only the conditions on the IV are satisfied, and second, when the conditions on the key are satisfied as well.

<table>
<thead>
<tr>
<th>Rounds</th>
<th>72</th>
<th>75</th>
<th>78</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>772</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IV conditions only</td>
</tr>
<tr>
<td>782</td>
<td>0.95</td>
<td>0.91</td>
<td>0.95</td>
<td></td>
</tr>
<tr>
<td>789</td>
<td>0.70</td>
<td>0.82</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>798</td>
<td>0.59</td>
<td>0.65</td>
<td>0.70</td>
<td></td>
</tr>
<tr>
<td>868</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>key and IV conditions, $2^{31}$ weak keys</td>
</tr>
<tr>
<td>953</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>key and IV conditions, $2^{26}$ weak keys</td>
</tr>
<tr>
<td>961</td>
<td>1</td>
<td>0.50</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6: Neutrality of the bits $x_{72}$, $x_{75}$, and $x_{78}$.

### 4.6 Summary and Conclusion

The framework of conditional differential cryptanalysis is extended to higher order differential cryptanalysis. The resulting technique refines and generalizes previous techniques such as the maximum degree test and cube testers, leading to more effective cryptanalytic attacks on reduced-round variants of Grain-128, Grain-128a, and Trivium. In addition to the derived functions approach from [48] and the cube attack from [41], conditional differential cryptanalysis provides a third approach for key recovery based on high order differential cryptanalysis.

Our main results are distinguishing and key recovery attacks up to 215 rounds of Grain-128 and a distinguishing attack for 961 rounds of Trivium. The latter applies only to a small class of weak keys, though. Grain-128a, which was designed to resist the dynamic cube attack, turned out to be slightly more resistant against conditional differential cryptanalysis than Grain-128. However, 189 rounds could still be attacked, which is 74% of the original number of rounds. A simple measure to increase the security further would
be to choose 320 instead of 256 initialization rounds (even when the authentication mode is not used).

It is interesting to compare Grain v1 and Trivium, because they follow diametrically opposite design rationales. Trivium uses very low-weight and sparse update functions, but performs as many as 1152 rounds. Grain v1 uses relatively dense update functions of degree 6 and performs only 160 rounds. It turned out that conditional differential cryptanalysis is less effective on Trivium than on Grain v1. This is mainly due to slower diffusion in Grain v1 than in Trivium. In the first 20% of the initialization rounds, there are less than 10 difference propagations in Grain v1, compared to more than 20 propagations in Trivium. On the other hand, the low-degree output function of Trivium make it suspicious to state recovery attacks and requires a relatively large state [85]. An interesting combination would be to use a small state with a Trivium-like initialization and a Grain-like output function.

The most heuristic part of high order differential attacks is the choice of the differences. An algorithm inspired by biological evolution was proposed in [7], but is not shown to significantly improve over random choice. In the first order case, linear differential characteristics can be computed in order to find specific input differences. Translating this strategy to the higher order case turned out to be very difficult, because the conditions for the many characteristics tend to contradict each other. A better understanding of “good” choices for differences in the higher order case is likely to give improved attacks. Other improvements can be expected from generalizations such as the dynamic cube attack which are beyond the framework of high order derivatives of Boolean functions.
Chapter 5

Analysis of the Knapsack Generator

The knapsack generator is an LFSR-based keystream generator proposed by Rueppel and Massey in 1985 as an alternative to non-linear Boolean filter functions. At that time this novel proposal received some cryptanalytic attention, and it turned out to be resistant against known techniques such as correlation attacks or algebraic attacks. Our analysis is quite specific and has not much in common with widespread techniques from symmetric cryptanalysis. The experimental results give a better understanding of the security level provided by the knapsack stream cipher.

5.1 Introduction

The main issue in the design of LFSR-based keystream generators consists in breaking the linearity of the driving LFSR. Rueppel and Massey addressed this problem in 1985 by proposing the knapsack generator as an alternative to Boolean filter functions [101]. The design goal is stated as follows: “The combination of nonlinear functions with LFSRs provides a good means for generating running keys. The major problem is the difficulty of implementing complex and truly random nonlinear functions. What we would like to have is a simple and fast implementation of a large set of highly nonlinear functions which is readily indexed by the key.” [100, Chapter 8]
The knapsack generator destroys the linearity of the LFSR by the use of integer addition. It avoids the tradeoff between high linear complexity and high correlation immunity which is inherent to non-linear Boolean functions, as shown by Siegenthaler [111]. A systematic analysis of the non-linearity of integer addition over the binary field is provided by Staffelbach and Meier [112]. Besides the knapsack generator, Rueppel also introduced the summation generator [99] to avoid this tradeoff, but the latter turned out to be vulnerable to correlation attacks [51, 86], algebraic attacks [75], and to attacks based on feedback with carry shift registers [64]. Apparently, the knapsack generator resisted these powerful techniques. It is given as Example 6.56 in [89] and due to its ease of implementation it is recommended in [33] for the use in RFID networks.

The name of this generator comes from the close relation to the knapsack problem (also known as the subset sum problem) which consists in finding a subset of given weights that add up to a given sum. The decisional version of this problem is known to be NP-complete [50] and several attempts have been made to use it in cryptography. A prominent example is the Merkle-Hellman knapsack cryptosystem [90] which was broken by Shamir [108]. In a different direction, Impagliazzo and Naor constructed provably secure pseudorandom generators and universal hash functions based on the knapsack problem [58]. Other than these cryptosystems, the security of the knapsack generator is not directly related to the hardness of the knapsack problem. The new generic algorithms for hard knapsack problems, recently presented by Howgrave-Graham and Joux in [57], have no implications on the security of the knapsack generator.

5.2 The Knapsack Generator

This section specifies the knapsack generator and reviews the most important security arguments given by Rueppel in [100]. In the last part, previous cryptanalytic results by Von zur Gathen and Shparlinski [116] are briefly commented.

5.2.1 Description

The knapsack generator uses an LFSR of length \( n \) with maximum period. We denote its state by by \( \mathbf{x} = (x_1, \ldots, x_n) \). The output filter is parameterized by
5.2 The Knapsack Generator

$n$ positive integers modulo $2^n$ written as a vector $w = (w_1, \ldots, w_n)$. At each step $n - \ell$ bits of keystream are generated as follows:

1. Compute the sum $s = \sum_{i=1}^{n} x_i w_i \mod 2^n$.
2. Discard the $\ell$ least significant bits of $s$ and output the remaining bits.
3. Clock the LFSR.

The $w_i$ are called the knapsack weights and $s$ is called the partial sum obtained by summing those weights which are selected by the binary vector $x$. Sometimes we refer to the $x_i$ as the control bits.

5.2.2 Key Length and Security Parameters

The secret key of the generator consists of $n + n^2$ bits: $n$ bits for the initial state of the LFSR and $n^2$ bits for the weights. The connection polynomial of the LFSR is assumed to be part of the known cipher specification and not part of the secret key. (Recall that it is must be primitive in order to guarantee maximum period.)

The cases $n = 32, 64$ are of particular interest because they are favorable for software implementation. In these cases the key consists of 1056 and 4160 bits, respectively. In view of these huge key lengths it is natural to ask about the effective security level provided by the knapsack generator. It is hard to believe that brute-force search over the full key space is the best way to recover the key. This was not claimed by Rueppel, but the question remained open. It turns out that besides of $n$, the number of discarded bits is an important security parameter. Intuitively, if few bits are discarded, the output reveals more information per step, whereas if many bits are discarded, the throughput of the generator gets low. A recommendation for $\ell$ is given by Rueppel based on linear complexity analysis (see Property 5.2 below). Our analysis will point out a new aspect on the influence of $\ell$.

5.2.3 Security Arguments by Rueppel

Rueppel provides an extensive study of the knapsack generator in his thesis [100]. He gives a complete description of the mapping $x \mapsto s = \sum_{i=1}^{n} x_i w_i$ in binary arithmetic and derives several properties from that. The two most interesting properties are given below. Let $s_i$ be the $i$-th bit of $s$ such that $s = \sum_{i=0}^{n-1} s_i 2^i$, and consider $s_i$ as a polynomial in the variables $x_1, \ldots, x_n$. 
Property 5.1. The degree of $s_i$ is bounded by $\min\{2^i, n\}$.

Justification. It is clear that the least significant bit, that is $s_0$, is linear (there is no carry) and that the degree of $s_i$ cannot be higher than $n$ (there are only $n$ variables). The $i$-th least significant bit depends on the carries of the previous positions which have at most degree $2^i$ (this can be seen from the description of the $s_i$ given in [100]).

As a security argument, Rueppel provides experimental evidence that the bound in Property 5.1 is typically quite tight if the weights are chosen uniformly at random.

The next property is on the linear complexity of the keystream when a single function $s_i$ is used as a filter function. Recall that the linear complexity of a sequence of bits is the length of the smallest LFSR that generates that sequence. The linear complexity can be computed by the Berlekamp-Massey algorithm [81].

Property 5.2. Let $\overline{s}_j$ be the sequence generated by the knapsack generator when at each step only $s_j$ is output as the keystream. The linear complexity of $\overline{s}_j$ is bounded by $\sum_{i=1}^{2^j} \binom{n}{i}$ if $j < \lceil \log n \rceil$, and by $2^n - 1$ otherwise. Equality or near-equality holds with high probability for uniformly chosen random weights.

Justification. Again, the result is clear for $\overline{s}_0$. We refer to [100] for a proof of the bound (tightness is suggested by empirical evidence).

The property says that the $\lceil \log n \rceil$ least significant bits of the partial sums do not achieve maximal linear complexity. As a consequence, Rueppel recommends to choose $\ell = \lceil \log n \rceil$.

5.2.4 Previous Third-party Cryptanalytic Results

Von zur Gathen and Shparlinski [116] consider scenarios where either the control bits or the weights are known. In both cases they translate the task of finding the unknown parts of the key into a short vector problem which they solve by LLL lattice reduction algorithms. In the known control bit scenario, they can predict the generator if $\ell$ is not too small using about $n^2 - n$ outputs. They only provide an asymptotic complexity analysis and it is difficult to estimate the practicality of their approach, in particular when it is extended to a guess and determine attack. No experimental results are provided.
5.3 Attack Description

This section describes an attack, that, given $N$ outputs of the knapsack generator and the corresponding control bits, partly recovers the weights such that subsequent outputs can be predicted. By guessing the control bits, this extends to an attack in a scenario where only the output of the generator can be observed.

Rueppel already considered the problem of recovering the weights from the partial sums when the control bits are known. He called this the unknown weight, but known input problem. However, his analysis only applies to a generator that does not discard any bits from the partial sums. In this case, the problem can be solved by elementary linear algebra. Starting from this, we show how to deal with the case when $\ell$ bits of each partial sum are discarded.

5.3.1 A System of Modular Equations

The problem of finding the unknown weights $w = (w_1, \ldots, w_n)$ from $N$ partial sums $s = (s_1, \ldots, s_N)$ when the control bits are known can be stated as a system of modular equations

$$s = Xw \mod 2^n \quad (5.1)$$

where $X$ is a $N \times n$ matrix whose coefficients are the control bits. We call $X$ the control matrix. Its rows, denoted by $x_i$, are the consecutive states of the LFSR. Hence, the control matrix is entirely determined by any of its rows. It is shown by Von zur Gathen and Shparlinksi [116] that $n$ consecutive rows $x_i, \ldots, x_{i+n}$ are linearly independent over the integers modulo $2^n$ if the LFSR has maximum period. Hence, the system (5.1) can be uniquely solved for $w$ if $N \geq n$. Namely if $N = n$, there is a unique inverse $X^{-1}$ with coefficients modulo $2^n$ such that $w = X^{-1}s \mod 2^n$. More generally, if $N \geq n$, any $n \times n$ matrix $T$ with $TX = I_n \mod 2^n$ can be used to compute the weights, where $I_n$ denotes the $n \times n$ identity matrix. The challenge is to deal with the discarded bits, that is, when only the $n - \ell$ most significant bits of each component of each $s_i$ are known to the attacker.

Let $z = (z_1, \ldots, z_N)$ be the truncated partial sums, that is, $z_i = s_i \gg \ell$ where $\gg$ denotes a right shift of $n$-bit integers. Left shift is denoted by $\ll$, and when used for vectors, the shifting is applied componentwise, for example $z = s \gg \ell$. System (5.1) rewrites as

$$(z \ll \ell) + d = Xw \mod 2^n \quad (5.2)$$
where \( d = (d_1, \ldots, d_N) \) are the discarded bits (hence, \( 0 \leq d_i < 2^\ell \)), and thus
\[
 w = T(z \ll \ell) + Td \mod 2^n.
\]

The problem is that \( d \) is not known. Guessing the \( N\ell \) bits of \( d \) is too expensive. The main observation for our attack is that for well chosen \( T \) one can obtain good approximations of \( w \) with completely wrong \( d \) (set to zero, for example). Concretely, we will compute approximate weights by
\[
 \tilde{w} = T(z \ll \ell) \mod 2^n.
\]
(5.3)

The matrix \( T \) is called an approximation matrix. Note that when \( T \) is computed by Gaussian elimination, the approximate weights obtained by (5.3) will be completely wrong in general. Roughly speaking, the approximation will be good if \( Td \) is small. This is made more precise in the following, and an algorithm to find good approximation matrices is proposed.

### 5.3.2 Weight Approximation and Prediction

We are going to characterize “good” approximation matrices by proving some probability bounds for weight approximations and output predictions. If not stated otherwise, numbers are assumed to be reduced modulo \( 2^n \) and represented as positive integers in the range 0, \ldots, \( 2^n - 1 \). The absolute value of \( a \) is defined as \( |a| = a \) if \( 0 \leq a \leq 2^{n-1} \), and \( |a| = 2^n - a \) otherwise. For example, \( |14| = 2 \) for \( n = 4 \). For vectors, the norm \( \|v\| = \sum_{i=1}^n |v_i| \) is used, where \( v = (v_1, \ldots, v_n) \).

We first prove a general result for further reference.

**Lemma 5.3.** Let \( m < n \) and assume that \( a \) and \( b \) are chosen uniformly at random, with \( |b| < 2^m \). Then, for \( 0 < \lambda \leq n - m \), the \( \lambda \) most significant bits of \( a \oplus (a + b) \) are zero with probability \( 1 - 2^{-c} \) where \( c = n - m - \lambda + 1 \).

**Proof.** The sum \( a + b \) can be recursively described by \( (a + b)_i = a_i \oplus b_i \oplus c_{i-1} \), \( c_i = a_i b_i \oplus a_i c_{i-1} \oplus b_i c_{i-1} \), where \( c_i \) denotes the carry bit, and \( c_{-1} = 0 \). If \( 0 \leq b < 2^m \), \( b_i = 0 \) for \( i \geq m \), and thus, \( (a + b)_i \oplus a_i = c_{i-1} \) for \( i \geq m \). Differences at bit positions higher than \( m \) only occur by carry propagation, and since \( c_i = a_i c_{i-1} \) for \( i \geq m \), the carry cancels out with probability \( 1/2 \) from one position to the next. The result follows under the assumption that \( c_{m-1} \) is uniformly distributed. It is shown in [87] that this is a reasonable assumption if \( m \) is not very small.
5.3 Attack Description

If \(2^n - 2^m \leq b < 2^n\), \(b_i = 1\) for \(i \geq m\), and thus, \((a + b)_i \oplus a_i = 1 \oplus c_i - 1\) for \(i \geq m\). Using that \(c_i = a_i \oplus a_i c_{i-1} \oplus c_{i-1}\), the result follows in a similar way as before.

Let us now analyze how well a single weight can be approximated by (5.3). For \(1 \leq i \leq n\), we have \(\tilde{w}_i = t_i(z \ll \ell) \mod 2^n\) where \(t_i\) is the \(i\)-th row of \(T\). To simplify the notation we omit the weight index \(i\), writing \(\tilde{w} = \tilde{w}_i\) and \(t = t_i\) for some \(i\).

**Lemma 5.4.** Assume that the weights are chosen uniformly at random, \(\ell > 0\) bits are discarded from each output, and \(\|t\| < 2^m\). Then, for \(0 < \lambda \leq n - m - \ell\), the \(\lambda\) most significant bits of \(\tilde{w}\) are correct with probability at least \(1 - 2^{-c}\) where \(c = n - m - \ell - \lambda + 1\).

**Proof.** Recall that \(d = (d_1, \ldots, d_N)\) are the discarded bits and \(0 \leq d_i < 2^\ell\) for \(1 \leq i \leq \ell\). It follows that \(\|td\| < 2^{m + \ell}\) by the assumption and Lemma 5.3 can be applied with \(a = t(z \ll \ell)\) and \(b = td\).

Lemma 5.4 is about the quality of the weight approximations given a certain matrix \(T\). An analogous statement can be made about the quality of predictions. In order to predict \(z_{N+1}\), one computes \(\tilde{z}_{N+1} = \tilde{s}_{N+1} \gg \ell\) where \(\tilde{s}_{N+1} = x_{N+1} \tilde{w} \mod 2^n\).

**Lemma 5.5.** Assume that the weights are chosen uniformly at random, \(\ell > 0\) bits are discarded from each output, and \(\|x_{N+1}T\| < 2^m\). Then, for \(0 < \lambda \leq n - m - \ell\), the \(\lambda\) most significant bits of \(\tilde{s}_{N+1}\) are correct with probability at least \(1 - 2^{-c}\) where \(c = n - m - \ell - \lambda + 1\).

**Proof.** Apply Lemma 5.3 with \(a = x_{N+1}T(z \ll \ell)\) and \(b = x_{N+1}Td\).

From Lemma 5.5, a sufficient condition for a successful prediction attack can be derived.

**Corollary 5.6.** Under the assumptions of Lemma 5.5 at least one bit of the knapsack generator can be predicted with probability higher than \(1/2\) if \(m \leq n - \ell - 2\).

**Proof.** It is required that \(1 - 2^{-c} > 1/2\), hence \(c = n - m - \ell - \lambda + 1 \geq 2\). Setting \(\lambda = 1\) yields the claim.
The condition on \( m \) can be reduced to a condition on the approximation matrix. From \( \| x_{N+1}T \| \leq \| T \| \), we derive

\[
\log \| T \| \leq n - \ell - 2,
\]

where \( \| T \| \) is defined as \( \| T \| = \sum_{i=1}^{n} \| t_i \| \). In general, approximation matrices with smaller norm will give better approximations, hence “good” essentially means small norm. In the next section we are going to describe a surprisingly effective algorithm. Nevertheless, the more bits are discarded, the better matrices are required, and our approach may fail if too many bits are discarded.

### 5.4 Finding Good Approximation Matrices

Let us briefly restate the problem. Given an \( N \times n \) control matrix \( X \) we aim at finding an \( n \times N \) integer matrix \( T \) such that \( TX = I_n \mod 2^n \) and such that \( T \) has small coefficients.

We are going to proceed row by row. Let \( t_i \) be the \( i \)-the row of \( T \) for \( 1 \leq i \leq n \). The problem of finding a suitable \( t_i \) with small norm is translated into a closest vector problem in a certain lattice which is approximately solved by a combination of basis reduction by Lenstra, Lenstra, and Lovász (LLL) [77] and the closest vertex algorithm by Babai [11].

#### 5.4.1 Using LLL Basis Reduction and Babai’s Algorithm

We consider the following problem, which is a variant of our problem of finding suitable rows of the approximation matrix.

**Problem.** Given an \( r \times t \) integer matrix \( A \) and an integer column vector \( b \) of length \( t \), find an integer row vector \( x \) of length \( r \) such that \( xA = b \) and such that the coefficients of \( x \) are small in absolute value.

Depending on \( A \) and \( b \), three cases can occur:

- No solution (the image of \( A \) contains \( b \)).
- A unique solution (image contains \( b \), kernel of \( A \) is trivial).
- Many solutions (image contains \( b \), kernel has dimension \( k > 0 \)).
5.4 Finding Good Approximation Matrices

We are only interested in the last case, where, among the many solutions, we want to pick a small one. This can be done as follows.

1. Find a solution \( x_0 \) (not especially small).
2. Compute a basis \( v_1, \ldots, v_k \) of the kernel of \( A \).
3. Find a LLL reduced basis \( v'_1, \ldots, v'_k \).
4. Use Babai’s closest vertex algorithm to find a vector \( v \) which is close to \( x_0 \) in the lattice spanned by \( v'_1, \ldots, v'_k \).
5. Return \( x = x_0 - v \) as a small solution.

Babai’s closest vertex algorithm computes real numbers \( \xi_1, \ldots, \xi_k \) such that \( x_0 = \sum_{i=1}^{k} \xi_i v'_i \) and it returns \( v = \sum_{i=1}^{k} \lfloor t_i \rfloor v'_i \). The vector \( v \) is close to \( x_0 \) if the basis vectors \( v'_i \) are reasonably orthogonal to one another (LLL reduced, for example) [56, Theorem 6.73].

5.4.2 Application to Our Problem

In our case, when computing \( t_i \), the \( i \)-th row of \( T \), we have \( A = X \) of dimension \( N \times n \) and \( b = e_i \) where \( e_i \) denotes the \( i \)-th column of \( I_n \). For \( N > n \), a solution is guaranteed to exist and the kernel of \( A \) is non-trivial. The kernel and its reduced basis have to be computed only once for all rows. This results in the following algorithm to find good approximation matrices:

1. Compute a basis \( v_1, \ldots, v_k \) of the kernel of \( X \).
2. Find a LLL reduced basis \( v'_1, \ldots, v'_k \).
3. For \( i \) from 1 to \( n \): Find \( t_i \) such that \( t_i X = e_i \) and such that \( t_i \) has small coefficients (Babai’s algorithm).

Our problem has specific properties compared to the general problem stated above. It turned out that the fact that \( X \) has only 0s and 1s as components is favorable. On the other hand, the particular structure of the control matrix (successive states of an LFSR) seems not to have a significant influence. In particular, the choice of the connection polynomial is not relevant.

The most important parameter is \( N \), the number of observed outputs. It basically determines the kernel dimension of \( X \). The performance of the algorithm is illustrated by several experiments in the next section.
5.5 Experimental Results

It is difficult to give theoretical estimates for the performance of our approach described below, instead we provide some experimental results. Our implementation uses the Number Theory Library by Shoup [110] providing LLL reduction and Babai’s algorithm (use the LatticeSolve method with the parameter reduced = 2).

Experiments are done for \( n = 32 \) and \( n = 64 \). The connection polynomials of the LFSR have been chosen as \( f(x) = x^{32} + x^{22} + x^2 + x + 1 \) and \( f(x) = x^{64} = x^{63} + x^{61} + x^{60} + 1 \), respectively (however, the results are the same for other choices).

5.5.1 Finding Approximation Matrices

Let us first analyze the norms of the approximation matrices. Figure 5.1 illustrates the results for \( \log \|T\| \) in function of \( N \) for \( n = 64 \). The samples are generated by choosing initial states of the LFSR uniformly at random.

![Figure 5.1: Norm of \( T \) in function of \( N \) for \( n = 64 \). Indicated are average, minimum, and maximum of \( \log \|T\| \) on random samples of size 100 for each \( N \).](image-url)
Recall that $\|T\| \leq n - \ell - 2$ has been derived as a sufficient condition for an attack. Assuming $\ell = \lceil \log n \rceil$ as recommended by Rueppel, the results in Fig. 5.1 show that much better approximation matrices can be found with surprisingly few outputs.

### 5.5.2 Weight Approximation

The subsequent results complement the theoretical analysis in Lemma 5.4. We experimentally determined how well the weights can be approximated given $N$ outputs. Figure 5.2 shows the results for $n = 32$ and $n = 64$ with $\ell = 5$ and 6, respectively.

### 5.5.3 Prediction Probability

Similar to the weight approximation we experimentally determined the quality of the output prediction depending on $N$. Figures look very much the same as Fig. 5.2, just shifted about $\log(n)$ bits to the left. Instead, we provide a slightly different benchmark. Table 5.1 shows the average number of consecutive most significant bits that are correctly predicted for $n = 32, 64, 128, 256$ with $\ell = \log n$ and different values of $N$. By random guessing we would achieve about $\sum_{i=1}^{n} i/2^{i+1} \approx 1$ bits per output.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$n = 32$</th>
<th>$n = 64$</th>
<th>$n = 128$</th>
<th>$n = 256$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n + 8$</td>
<td>20.6</td>
<td>42.9</td>
<td>85.3</td>
<td>164.6</td>
</tr>
<tr>
<td>$n + 16$</td>
<td>22.2</td>
<td>48.7</td>
<td>100.9</td>
<td>203.4</td>
</tr>
<tr>
<td>$n + 24$</td>
<td>22.6</td>
<td>50.3</td>
<td>105.9</td>
<td>216.4</td>
</tr>
<tr>
<td>$n + 32$</td>
<td>22.7</td>
<td>50.8</td>
<td>108.1</td>
<td>222.4</td>
</tr>
</tbody>
</table>

Table 5.1: Average number of correctly predicted bits per output.

### 5.5.4 Practical Attack for $n = 32$

In a scenario where the adversary does not know the control bits nor the weights (which is the principal scenario), our technique allows a surprisingly effective guess and determine attack: $n$ state bits of the LFSR have to be guessed. Testing the guesses requires only very few outputs (the number $N$ above): for each guess, an approximation matrix and candidate approximate
Figure 5.2: Weight approximation for \( n = 32 \) (top) and \( n = 64 \) (bottom). Reading example: a value of 81.3\% for \( \lambda = 16 \) on the curve \( N = 36 \) means that with 36 outputs the 16 most significant bits of each weight were correctly recovered in 81.3\% of the cases.
weights are computed. Making some predictions, the correct guess can be identified with very high reliability.

The most expensive part of the attack is computing the approximation matrices. Instead of computing \(2^n\) such matrices, we can check several guesses with the same matrix \(T\) by shifting the outputs used for the weight approximation. If \(T\) is computed for \(N\) outputs, but \(N + 2^t\) outputs are available, only \(2^{n-t}\) approximation matrices must be computed. Since this computation is independent of the observed outputs, it can even be done offline.

For \(n = 32\) and \(\ell = 5\) the attack is practical on a desktop computer. In our experiments we assumed that \(552 = 40 + 2^9\) outputs of 27 bits could be observed. We computed control matrices for \(N = 40\) (hence, \(2^{23}\) matrices). Each guess was checked with 20 predictions from which only the 5 most significant bits have been considered. A guess was accepted when more than 80 of the 100 predicted bits were correct. On a Intel Core 2 Duo E8400 3.0 GHz Processor with 4 GB of RAM it took about three days to identify the 32 correct initial control bits and about 870 bits of the weights.

5.6 Summary and Conclusion

We have shown a technique that allows to recover large parts of the knapsack weights if the control bits are known. This extends to guess-and-determine attack in an attack scenario where the adversary can observe \(N\) outputs for \(N\) only slightly larger than \(n\), where \(n\) is the length of the LFSR and the length of the weights.

It was not expected that the security level of the knapsack generator corresponds to the naive estimate based on its key length of \(n^2 + n\) bits. However, our attack shows that it is not essentially more than \(n\) bits. Preventing the attack would require to discard many more bits from each partial sum (at least 25 bits for \(n = 32\), and similar for larger \(n\)), which would drastically reduce the throughput of the generator. In contrast to previous cryptanalysis by Von zur Gathen and Shparlinski, our approach applies to all relevant parameters \(n\) and \(\ell\). Moreover, our attack can be easily implemented and verified by experiments.

On the other hand, we could not find a shortcut for guessing the control bits (for example by using ideas from fast correlation attacks) and hence, the knapsack generator indeed provides \(n\)-bit security. When comparing this to an LFSR-based construction with non-linear boolean filter function, the large key length is a major drawback of the knapsack generator.
Analysis of the Knapsack Generator
Chapter 6

New Preimage Attacks Against Reduced SHA-1

The SHA-1 hash function was specified in 1995 by the U.S. National Security Agency. Although its collision resistance has been formally broken in 2005, it is still widely used in practice and it is still believed preimage and second preimage resistant. We show new preimage attacks against reduced SHA-1 up to 57 steps. The best previous attack was for 48 steps finding a two-block preimage with incorrect padding at the cost of $2^{159.3}$ evaluations of the compression function. For the same variant our attacks find a one-block preimage at $2^{150.6}$ and a correctly padded two-block preimage at $2^{151.1}$ evaluations of the compression function. The improved results come out of a differential view on the meet-in-the-middle technique originally developed by Aoki and Sasaki. The new framework closely relates meet-in-the-middle attacks to differential cryptanalysis which turns out to be particularly useful for hash functions with linear message expansion and weak diffusion properties.

6.1 Introduction

In the past, collision resistance of hash functions has been studied much more intensively than preimage resistance. This can be attributed to differential cryptanalysis as a very powerful tool to accelerate the collision search [31].
No such tool was available for the preimage search and the few published attacks were based on ad hoc methods. Notable examples are the first attacks on GOST [88], MD4 [78], and reduced variants of SHA-0/1 [30]. The situation changed with the introduction of the meet-in-the-middle technique into hash function cryptanalysis. Originally, the technique was used for block ciphers, starting with Diffie and Hellman showing that double encryption under two different keys does not double the security level [39], and later by Chaum and Evertse for key recovery attacks on reduced DES [32].

Only recently, Aoki and Sasaki translated the approach to the hash function context. In a series of papers they developed and refined a framework that resulted in the first preimage attack on MD5 and the best results on reduced variants of SHA-0/1, the SHA-2 family, and similar hash functions [4–6, 104–107]. An application of the framework by other authors yielded improved results on MD4, SHA-2, and Tiger [52]. A notable technical contribution was made by Khovratovich, Rechberger, and Savelieva with the formalization of the initial structure technique as complete bipartite graphs (bicliques) [63]. The derived algorithms enhance meet-in-the-middle attacks using tools from differential cryptanalysis. Applications include key recovery attacks on AES [23] and preimage attacks on reduced variants of the SHA-3 finalist Skein and members of the SHA-2 family [63].

**Technical Contribution.** Most of the existing meet-in-the-middle framework has been developed for preimage attacks against hash functions with permutation based message expansions such as MD5. Even though the techniques have been generalized, notably to the linear message expansion of SHA-1 [6], the original terminology did not translate suitably to these algorithms. We carry on the simple and elegant differential view suggested by bicliques to other techniques such as partial matching, partial fixing, or indirect partial matching. Indeed, these techniques become quite natural from a differential perspective. Finding the attack parameters reveals to be equivalent to finding two sets of suitable differentials. Finding high probability differentials is a well studied problem from collision attacks and insights can be reused. Two algorithms are proposed to find and evaluate suitable attack parameters. They facilitate a systematic security evaluation while previous meet-in-the-middle attacks seem to heavily rely on elaborated by hand analysis and intuition. The new framework applies particularly well to hash functions with linear message expansion, which is demonstrated by various new attacks against reduced variants of SHA-1.
Table 6.1: Preimage attacks against reduced SHA-1. If not stated otherwise, proper preimages with a correct padding are computed.

<table>
<thead>
<tr>
<th>Steps</th>
<th>Complexity</th>
<th># Blocks</th>
<th>Ref</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>$2^{157.0}$</td>
<td>1</td>
<td>[30]</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>$2^{156.9}$</td>
<td>1</td>
<td>[6]</td>
<td>pseudo-preimage, no padding</td>
</tr>
<tr>
<td>48</td>
<td>$2^{159.3}$</td>
<td>2</td>
<td>[6]</td>
<td>no padding</td>
</tr>
<tr>
<td>44</td>
<td>$2^{146.2}$</td>
<td>1</td>
<td>6.3.2</td>
<td>no padding</td>
</tr>
<tr>
<td>48</td>
<td>$2^{150.6}$</td>
<td>1</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>56</td>
<td>$2^{157.9}$</td>
<td>1</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>57</td>
<td>$2^{158.7}$</td>
<td>1</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>48</td>
<td>$2^{149.2}$</td>
<td>1</td>
<td>6.3.4</td>
<td>pseudo-preimage</td>
</tr>
<tr>
<td>59</td>
<td>$2^{156.8}$</td>
<td>1</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>60</td>
<td>$2^{157.5}$</td>
<td>1</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>48</td>
<td>$2^{151.1}$</td>
<td>2</td>
<td>6.3.5</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>$2^{158.1}$</td>
<td>2</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>57</td>
<td>$2^{158.8}$</td>
<td>2</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

SHA-1 and Improved Results. The SHA-1 hash function was specified in 1995 by the U.S. National Security Agency as a successor of SHA-0. No weakness was found in the full SHA-1 until 2005, when Wang et al. presented astonishing collision attacks [118]. These attacks let the NIST recommend the use of SHA-2 instead of SHA-1. However, SHA-1 is still widely used in practice and it is still believed to be preimage and second preimage resistant. De Cannière and Rechberger [30] presented the first dedicated preimage attack on reduced SHA-1. They could break 44 steps with a complexity of $2^{157}$ using an approach that could not be extended so far. Aoki and Sasaki [6] presented an attack on 48 steps with a complexity of $2^{159.3}$ using the meet-in-the-middle technique, but their attack only finds messages without padding. Finding a correct padding makes the analysis more complicated and typically leads to higher attack complexities or to unpractically long messages. No progress has been made since 2009. Our results improve the existing results in several directions: variants with more steps can be attacked, significantly lower complexities are obtained for previously attacked variants, and correctly padded (short!) messages can be computed. The results are summarized in Table 6.1 and graphically illustrated in Fig. 6.4 in the summary of this chapter.
Chapter Outline. Section 6.2 describes the differential meet-in-the-middle framework. Section 6.3 applies the framework to SHA-1 leading to the new results. Section 6.4 discusses an optimization of the naive brute-force search and Section 6.5 concludes the chapter.

6.2 Differential Meet-in-the-Middle Framework

The compression function of SHA-1 and other dedicated hash functions can be seen as a block cipher in Davies-Meyer mode, albeit with unusual key and block length. Let $E : \{0,1\}^\kappa \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher with block length $n$ and key length $\kappa$ and assume $\kappa \gg n$ ($\kappa = 512$ and $n = 160$ for SHA-1). Finding a preimage for a given hash value $H$ is the problem of finding $M$ such that $H = E(M, IV) + IV$, where $IV$ is an initial vector. A generic algorithm tests $2^n$ messages such that, under reasonable assumptions, one preimage is expected to be found.

6.2.1 Differential View on the Meet-in-the-Middle Technique

This section describes the meet-in-the-middle technique from a new differential perspective. Suppose that $E$ is divided into two parts, $E = E_2 \circ E_1$. For a basic meet-in-the-middle attack, the cryptanalyst tries to find two linear spaces $D_1, D_2 \subset \{0,1\}^\kappa$ as follows (think of the elements in $D_1$ and $D_2$ as message differences):

- $D_1 \cap D_2 = \{0\}$.

- For each $\delta_1 \in D_1$ there is a $\Delta_1 \in \{0,1\}^n$ such that
  \[
  \Pr[\Delta_1 = E_1(M, IV) \oplus E_1(M \oplus \delta_1, IV)] = 1
  \] (6.1)
  for uniformly chosen $M$. In other terms, $(\delta_1, \Delta_1)$ is a differential for $E_1(\cdot, IV) : \{0,1\}^\kappa \rightarrow \{0,1\}^n$ with probability 1.

- For each $\delta_2 \in D_2$ there is a $\Delta_2 \in \{0,1\}^n$ such that
  \[
  \Pr[\Delta_2 = E_2^{-1}(M, H - IV) \oplus E_2^{-1}(M \oplus \delta_2, H - IV)] = 1
  \] (6.2)
  for uniformly chosen $M$. In other terms, $(\delta_2, \Delta_2)$ is a differential for $E_2^{-1}(\cdot, H - IV) : \{0,1\}^\kappa \rightarrow \{0,1\}^n$ with probability 1.
The first condition makes sure that the message space can be partitioned into affine sets of the form \( M \oplus D_1 \oplus D_2 \). Supposing that \( D_1 \) and \( D_2 \) both have dimension \( d \), these sets contain \( 2^{2d} \) different messages. The remaining two conditions effect that each such set can be searched for a preimage by computing \( 2^d \) times \( E_1 \) and \( 2^d \) times \( E_2^{-1} \) using Algorithm 6.1. The algorithm computes two lists \( L_1 \) and \( L_2 \). A match between the two lists identifies a preimage. The case \( d = 1 \) is illustrated by Fig. 6.1.

Algorithm 6.1 Testing \( M \oplus D_1 \oplus D_2 \)

Input: \( D_1, D_2 \subset \{0, 1\}^\kappa, M \in \{0, 1\}^\kappa, H \)

Output: A preimage of \( H \) if one is contained in \( M \oplus D_1 \oplus D_2 \).

for all \( \delta_2 \in D_2 \) do
  Compute \( L_1[\delta_2] = E_1(M \oplus \delta_2, IV) \oplus \Delta_2 \).
end for

for all \( \delta_1 \in D_1 \) do
  Compute \( L_2[\delta_1] = E_2^{-1}(M \oplus \delta_1, H - IV) \oplus \Delta_1 \).
end for

for all \( (\delta_1, \delta_2) \in D_1 \times D_2 \) do
  if \( L_1[\delta_2] = L_2[\delta_1] \) then
    return \( M \oplus \delta_1 \oplus \delta_2 \)
  end if
end for

return No preimage is contained in \( M \oplus D_1 \oplus D_2 \)

Correctness of Algorithm 6.1. The algorithm indeed finds a preimage if one is contained in \( M \oplus D_1 \oplus D_2 \), because

\[
L_1[\delta_2] = L_2[\delta_1] \\
\Leftrightarrow E_1(M \oplus \delta_2, IV) \oplus \Delta_2 = E_2^{-1}(M \oplus \delta_1, H - IV) \oplus \Delta_1 \\
\Leftrightarrow E_1(M \oplus \delta_1 \oplus \delta_2, IV) = E_2^{-1}(M \oplus \delta_1 \oplus \delta_2, H - IV) \quad \text{by (6.1) and (6.2)} \\
\Leftrightarrow E(M \oplus \delta_1 \oplus \delta_2, IV) + IV = H.
\]

Preimage Search. Using Algorithm 6.1 for the preimage search is straightforward. Affine subsets \( M \oplus D_1 \oplus D_2 \) are tested until a preimage is found. The messages \( M \) can be picked at random. The probability of testing the same message twice is very small for \( \kappa \gg n \). Alternatively, for example when \( \kappa = n \), the messages can be chosen pairwise different modulo \( D_1 \cup D_2 \).
New Preimage Attacks Against Reduced SHA-1

![Diagram of Algorithm 6.1](image)

**Figure 6.1:** Illustration of Algorithm 6.1. A match between the lists $L_1$ and $L_2$ identifies a preimage. Here, the $D_1$ and $D_2$ only have dimension $d = 1$ which allows to test four messages at the cost of two. In general, $2^{2d}$ messages are tested at the cost of $2^{d}$.

**Complexity Analysis.** In average, a total of $2^n$ messages have to be tested. This requires $2^{n-2d}$ executions of Algorithm 6.1. If we denote by $\Gamma_1$ and $\Gamma_2$ the cost of one evaluation of $E_1$ and $E_2$, respectively, this results in a total complexity of

$$2^{n-2d}(2^d\Gamma_1 + 2^d\Gamma_2) = 2^{n-d}\Gamma,$$

where $\Gamma$ is the cost of $E$. Depending on the implementation, memory is required to store the list $L_1$ and/or $L_2$. Both lists have length $2^d$ and entries of about $n + d$ bits. The cost for the matching (checking for common entry in $L_1$ and $L_2$) is ignored. It can be done logarithmically in $2^d$, for example by sorting one of the two lists.

### 6.2.2 Using Probabilistic and Truncated Differentials

In order to reach more rounds, probabilistic and truncated differentials can be used instead of the deterministic differentials on the full state. The following notation is used for the truncation. For $T \in \{0, 1\}^n$ we write "$=_T$" for equality on those bits of $T$ which are 1, that is, $A =_T B$ if and only if $A \wedge T = B \wedge T$, where $\wedge$ denotes bitwise and. We will call $T$ a truncation mask.

The cryptanalyst tries to find $D_1, D_2 \subset \{0, 1\}^\kappa$ and $T \in \{0, 1\}^n$ such that $D_1$ and $D_2$ satisfy the conditions in the previous section but with (6.1) and
(6.2) replaced by

\[
\begin{align*}
\Pr[\Delta_1 = T \ E_1(M, IV) \oplus E_1(M \oplus \delta_1, IV)] &= p_1, \\
\Pr[\Delta_2 = T \ E_2^{-1}(M, H - IV) \oplus E_2^{-1}(M \oplus \delta_2, H - IV)] &= p_2,
\end{align*}
\]

(6.3)

for uniformly chosen \(M\). Then,

\[
E_1(M \oplus \delta_2, IV) \oplus \Delta_2 = T \ E_2^{-1}(M \oplus \delta_1, H - IV) \oplus \Delta_1
\]

(6.5)
is not anymore equivalent to \(E(M \oplus \delta_1 \oplus \delta_2, IV) + IV = H\). Testing whether \(M \oplus \delta_1 \oplus \delta_2\) is a preimage by Eq. (6.5) is best formulated as a hypothesis test with null hypothesis

\[H_0: M \oplus \delta_1 \oplus \delta_2\] is a preimage.

\(H_0\) is rejected if Eq. (6.5) does not hold, otherwise it is kept. \(H_0\) is falsely rejected with probability \(\alpha = 1 - p_1p_2\). On the other hand, \(H_0\) is falsely not rejected with probability \(\beta = 2^{-r}\), where \(r\) is the Hamming weight of \(T\).

Algorithm 6.2 is a modified variant of Algorithm 6.1 using this hypothesis test in the last loop to detect candidate preimages. Candidate preimages have to be rechecked, for example by computing \(E(M \oplus \delta_1 \oplus \delta_2, IV)\).

**Algorithm 6.2** Testing \(M \oplus D_1 \oplus D_2\) using truncated differentials

**Input:** \(D_1, D_2 \subset \{0, 1\}^k, T, M \in \{0, 1\}^k\)

**Output:** Eventually a preimage if one is contained in \(M \oplus D_1 \oplus D_2\).

for all \(\delta_2 \in D_2\) do

Compute \(L_1[\delta_2] = E_1(M \oplus \delta_2, IV) \oplus \Delta_2\).

end for

for all \(\delta_1 \in D_1\) do

Compute \(L_2[\delta_1] = E_2^{-1}(M \oplus \delta_1, H - IV) \oplus \Delta_1\).

end for

for all \((\delta_1, \delta_2) \in D_1 \times D_2\) do

if \(L_1[\delta_2] = T \ L_2[\delta_1]\) then

Check whether \(M \oplus \delta_1 \oplus \delta_2\) is indeed a preimage, otherwise go on.

end if

end for

return No preimage found in \(M \oplus D_1 \oplus D_2\)

**Complexity Analysis.** Using Algorithm 6.2 for the preimage search increases the complexity of the attack for two reasons. First, because candidate preimages have to be rechecked in the last loop. This concerns a fraction of \(2^{-r}\)
messages. Second, because some preimages fail to be detected. The probability that an actual preimage is not detected is denoted by $\alpha$, the average of $\alpha$ over all $(\delta_1, \delta_2) \in D_1 \times D_2$. In order to compensate non-detected preimages, the number of tested messages has to be increased by a factor $1/(1-\alpha)$. This results in a total complexity of

$$ (2^{n-d}\Gamma_1 + 2^{n-d}\Gamma_2 + 2^{n-r}\Gamma_{re})/(1-\alpha), \quad (6.6) $$

where $\Gamma_{re}$ is the cost of rechecking a candidate preimage.

### 6.2.3 Algorithms to Evaluate $D_1$ and $D_2$

Given a separation $E = E_2 \circ E_1$, the crucial task for the cryptanalyst is to find the subspaces $D_1$ and $D_2$ and a truncation mask $T$ that minimize (6.6). Minimizing (6.6) means that $d$ (the dimension of $D_1$ and $D_2$) and $r$ (the Hamming weight of $T$) should be large and $\alpha$ small. Finding a good configuration is very tedious by hand and two algorithms are proposed that turned out to be very useful to explore the many tradeoffs.

Given $D_1$, $D_2$, and $r$, Algorithm 6.3 finds a truncation mask $T$ of Hamming weight $r$. It performs a bitwise ranking of difference probabilities in order to choose a subset of $r$ bit positions with the highest matching probability. These bits are set to 1 in $T$. Then, given $D_1$, $D_2$, and $T$, Algorithm 6.4 estimates the corresponding $\alpha$ (that is, the average probability of falsely rejecting a preimage over all $(\delta_1, \delta_2) \in D_1 \times D_2$).

### 6.2.4 Special Case: Linear Message Expansion

Suppose that the message expansion can be described by $N$ linear maps $\varphi_i : \{0,1\}^\kappa \rightarrow \{0,1\}^w$, for $i = 0, \ldots, N - 1$, such that $\varphi_i(M)$ is the $w$-bit message word used at round $i$. Then, a set of candidates for $D_1$ and $D_2$ can be obtained by linear algebra. If $\delta$ lies in the kernel of $\varphi_i$ it does not introduce differences at round $i$. Hence, if $D_1$ is a subspace of $\bigcap_{i=0}^{k-1} \ker(\varphi_i)$ for some $k > 0$, $\delta_1 \in D_1$ does not introduce differences for the first $k$ rounds. The idea is to choose $k$ as large as possible. Similar, $D_2$ should be chosen as a subspace of $\bigcap_{i=N-k'}^{N-1} \ker(\varphi_i)$ for $k'$ as large as possible.
6.2 Differential Meet-in-the-Middle Framework

Algorithm 6.3 Find truncation mask $T$

Input: $D_1, D_2 \subset \{0, 1\}^\kappa, r$
Output: A truncation mask $T \in \{0, 1\}^n$ of Hamming weight $r$.
$c = \text{an array of } n \text{ counters set to 0}$
for $q = 1$ to $Q$ do
    Choose $M \in \{0, 1\}^\kappa$ at random
    $H \leftarrow E(M, IV) + IV$
    Choose $(\delta_1, \delta_2) \in D_1 \times D_2$ at random
    $\nabla \leftarrow E_1(M \oplus \delta_1, IV) \oplus \Delta_1 \oplus E_2^{-1}(M \oplus \delta_2, H - IV) \oplus \Delta_2$
    for $i = 0$ to $n - 1$ do
        if the $i$-th bit of $\nabla$ is 1 then
            $c[i] \leftarrow c[i] + 1$
        end if
    end for
end for
Set those $r$ bits of $T$ to 1 which have the lowest counters.
return $T$

Algorithm 6.4 Estimate type I error probability

Input: $D_1, D_2 \subset \{0, 1\}^\kappa, T \in \{0, 1\}^n$
Output: Average type I error probability $\alpha$.
$c = \text{a counter set to 0}$
for $q = 1$ to $Q$ do
    Choose $M \in \{0, 1\}^\kappa$ at random
    $H \leftarrow F(M, IV) + IV$
    Choose $(\delta_1, \delta_2) \in D_1 \times D_2$ at random
    if $E_1(M \oplus \delta_1, IV) \oplus \Delta_1 \neq_T E_2^{-1}(M \oplus \delta_2, H - IV) \oplus \Delta_2$ then
        $c \leftarrow c + 1$ // a false rejection occurred
    end if
end for
return $c/Q$

6.2.5 Splice and Cut

The splice and cut technique was introduced in [5] and became a core technique for all meet-in-the-middle attacks by Aoki and Sasaki. The idea is to start the computations at an intermediate state, connecting the last and the first step via the feed-forward of the Davies-Meyer mode.
The technique translates as follows to our framework. The computation of $E$ is cut into $E = E_2 \circ E_1$ and a new function $E'$ is defined as

$$E'(M, X) = E_1(M, H - E_2(M, X))$$

(this is the splicing). Then, meet-in-the-middle is used to find $M$ such that $E'(M, X) = X$ for some $X$. Such an $M$ is a pseudo-preimage of $H$. (A pseudo-preimage of $H$ is a message $M$ such that $E(M, X) + X = H$ for some $X \neq IV$.) To see this, note that $E^{-1}_1(M, X) = H - E_2(M, X)$ by assumption.

Then,

$$E(M, H - E_2(M, X)) + H - E_2(M, X)$$

$$= E_2(M, E_1(H - E_2(M, X))) + H - E_2(M, X)$$

$$= E_2(M, E_1(E^{-1}_1(M, X))) + H - E_2(M, X)$$

$$= E_2(M, X) + H - E_2(M, X)$$

$$= H.$$

Hence, the “IV” of the pseudo-preimage is $H - E_2(M, X)$. The resulting pseudo-preimage attack can be generically transformed into a preimage attack using [89, Fact 9.99], for example.

### 6.2.6 Bicliques

Bicliques [63] are a formalization of initial structures [107] and are often used in combination with the splice and cut technique. The idea is to start the computations not from a single state, but from a precomputed structure of states that covers several rounds. This increases the number of attackable rounds without actually increasing the complexity of the attack if the amortized cost of constructing the bicliques is negligible.

For an attack with bicliques, the cipher is divided into three parts, $E = E_3 \circ E_2 \circ E_1$, and bicliques are constructed for one part, say for $E_3$. A biclique for $E_3$ is a tuple $\{M, D_1, D_2, Q_1, Q_2\}$ where $M \in \{0, 1\}^\kappa$ is a message, $D_1$ and $D_2$ are linear spaces of differences as above, and $Q_1$ and $Q_2$ are two lists of states indexed by $\delta_1 \in D_1$ and $\delta_2 \in D_2$, respectively, such that for all $(\delta_1, \delta_2) \in D_1 \times D_2$:

$$Q_2[\delta_2] = E_3(M \oplus \delta_1 \oplus \delta_2, Q_1[\delta_1]).$$

If probabilistic and truncated differentials are used as in section 6.2.2, a candidate pseudo-preimage (!) is detected by testing

$$E_1(M \oplus \delta_2, H - Q_2[\delta_2]) \oplus \Delta_2 = E_2^{-1}(M \oplus \delta_1, Q_1[\delta_1]) \oplus \Delta_1.$$
Only pseudo-preimages are found, because in general, \( IV \neq H - Q_2[\delta_2] \). Algorithm 6.5 is a modified variant of Algorithm 6.2 for using bicliques.

**Algorithm 6.5 Testing \( M \oplus D_1 \oplus D_2 \) using bicliques**

**Input:** \( D_1, D_2 \subset \{0, 1\}^\kappa, T, M \in \{0, 1\}^\kappa, Q_1, Q_2 \)

**Output:** Eventually a preimage if one is contained in \( M \oplus D_1 \oplus D_2 \).

for all \( \delta_2 \in D_2 \) do
    Compute \( L_1[\delta_2] = E_1(M \oplus \delta_2, H - Q_2[\delta_2]) \oplus \Delta_2 \).
end for

for all \( \delta_1 \in D_1 \) do
    Compute \( L_2[\delta_1] = E_2^{-1}(M \oplus \delta_1, Q_1[\delta_1]) \oplus \Delta_1 \).
end for

for all \( (\delta_1, \delta_2) \in D_1 \times D_2 \) do
    if \( L_1[\delta_2] =_T L_2[\delta_1] \) then
        Check whether \( M \oplus \delta_1 \oplus \delta_2 \) is indeed a preimage, otherwise go on.
    end if
end for

return No preimage found in \( M \oplus D_1 \oplus D_2 \)

If the amortized cost of constructing the bicliques is negligible, the complexity of an attack with Algorithm 6.5 is computed as in (6.6).

Bicliques and the splice and cut technique are used in our application to SHA-1, but not in combination with [89, Fact 9.99]. A dedicated approach will be used to find correctly padded two-block preimages at essentially the same cost as one-block preimages without padding.

### 6.3 Application to SHA-1

The differential meet-in-the-middle framework is now applied to reduced variants of SHA-1. Different types of “preimages” are obtained. A one-block preimage without padding is the straightforward application of the framework. A one-block preimage with padding is slightly more costly and a dedicated approach is proposed to obtain correctly padded two-block preimages at essentially the same cost as one-block preimages without padding. First, a description of SHA-1 is given.
6.3.1 Description of SHA-1

The hash function SHA-1 was designed by the U.S. National Security Agency. A full specification can be found in [92]. The construction follows the Merkle-Damgård principle with a block cipher based compression function in Davies-Meyer mode. A message is padded to a length which is a multiple of 512 bits. This is done by appending a 1, a variable number of 0s, and the length in bits as a 64-bit integer. After the padding, the message is split into 512-bit blocks $M = (M^0, \ldots, M^{L-1})$ which are iteratively processed according to

$$H^0 = IV, \quad H^{l+1} = F(M^l, H^l) + H^l, \quad 0 \leq l \leq L - 1.$$  

The chaining values $H^l$ consist of five 32-bit words. The last chaining value is the output of the hash function.

The function $F$ can be considered as a block cipher with key length $\kappa = 512$ and block length $n = 160$. The computation has two parts: First, the message block $M^l = (M_0, \ldots, M_{15})$ is expanded to 80 round keys $W = (W_0, \ldots, W_{79})$ as follows:

$$W_i = M_i, \quad 0 \leq i \leq 15,$$

$$W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \ll 1, \quad 16 \leq i \leq 79. \quad (6.7)$$

Second, the chaining value is loaded into the registers $(A, B, C, D, E)$ and updated through 80 steps according to Fig. 6.2. At each step, an expanded message word $W_i$, a bitwise boolean function $f_i$, and a constant $K_i$ are used. The final content of the registers is the output of $F$.

![Figure 6.2: The step transformation of SHA-1.](image-url)
6.3 Application to SHA-1

6.3.2 One-Block Preimages Without Padding

When searching for a one-block preimage, we are given \( H \), and we want to find \( M \) such that \( H = F(M, IV) + IV \). The presented framework applies one-to-one to this problem. In this section we ignore the padding of \( M \).

Our attack parameters have been found by extensive experiments using Algorithms 6.3 and 6.4 for different configurations. The following guidelines have been considered for the choice of candidate configurations:

**Kernels of message expansion.** The message expansion of SHA-1 is linear and the kernel approach from Section 6.2.4 can be used. It turns out that the kernel of any \( k \) consecutive steps has dimension \((16 - k) \times 32\) (dimension is 0 for \( k > 16 \)).

**Choosing \( D_1 \) and \( D_2 \).** According to Section 6.2.4 \( D_1 \) and \( D_2 \) are chosen as kernel subspaces for the first 15 steps and the last 15 steps, respectively. Both kernels have dimension 32, and a basis can be derived using the message expansion. For the kernel of the first 15 steps, set \( W_0, \ldots, W_{14} \) to 0 and \( W_{15} \) to \((1 \ll i)\) for \( i = 0, \ldots, 31 \) where \( \ll \) denotes left rotation of 32-bit words. For the kernel of the last 15 steps, set \( W_{N-1}, \ldots, W_{N-15} \) to 0 and \( W_{N-16} \) to \((1 \ll i)\) for \( i = 0, \ldots, 31 \), and expand the message in the backward direction in order to obtain \( W_0, \ldots, W_{15} \). The resulting bases have lowest possible Hamming weight which is a good starting point to find differentials with high probabilities. Due to the limited diffusion of the SHA-1 step transformation, the best configurations are obtained for \( D_1 \) and \( D_2 \) containing “consecutive” basis vectors, meaning basis vectors that are generated with the \( W_{15} \) (resp. the \( W_{N-16} \)) chosen as \((1 \ll i)\) with \( i = t, \ldots, t + d - 1 \) for some \( t \). There are only \( 32^2 \) such combinations which can be searched exhaustively.

**Weak backward diffusion of the step transformation.** The diffusion of the SHA-1 step transformation turns out to be much weaker in the backward direction than in the forward direction. This is relevant for the choice of a suitable separation \( F = F_2 \circ F_1 \). If \( n_1 \) and \( n_2 \) are the steps computed by \( F_1 \) and \( F_2 \), respectively, \((n_1 - 15)/(n_2 - 15) \approx 1/3\) turned out to be a good ratio.

**Output differences \( \Delta_1 \) and \( \Delta_2 \).** It is well known that good differentials for SHA-1 can be obtained by linearization [31]. For our attacks we use \( \Delta_1 = \bar{F}_1(\delta_1, 0) \) and \( \Delta_2 = \bar{F}_2^{-1}(\delta_2, 0) \), where \( \bar{F}_1 \) and \( \bar{F}_2 \) are linearized
versions of $F_1$ and $F_2$, obtained by replacing the $f_i$ by $f(X,Y,Z) = X \oplus Y \oplus Z$, “+” by “⊕”, and $K_i$ by 0.

Table 6.2 shows the results for the best attack parameters we could find. The detailed parameters are given in the Appendix. The complexities are computed according to (6.6).

<table>
<thead>
<tr>
<th>$N$</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$d$</th>
<th>$\bar{\alpha}$</th>
<th>Complexity</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>18</td>
<td>26</td>
<td>15</td>
<td>0.428</td>
<td>$2^{146.21}$</td>
<td>no padding</td>
</tr>
<tr>
<td>48</td>
<td>19</td>
<td>29</td>
<td>11</td>
<td>0.552</td>
<td>$2^{150.62}$</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>19</td>
<td>30</td>
<td>10</td>
<td>0.593</td>
<td>$2^{151.78}$</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>19</td>
<td>31</td>
<td>10</td>
<td>0.811</td>
<td>$2^{152.89}$</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>19</td>
<td>32</td>
<td>9</td>
<td>0.842</td>
<td>$2^{154.16}$</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>21</td>
<td>31</td>
<td>7</td>
<td>0.631</td>
<td>$2^{154.95}$</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>21</td>
<td>32</td>
<td>6</td>
<td>0.577</td>
<td>$2^{155.76}$</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>21</td>
<td>33</td>
<td>5</td>
<td>0.547</td>
<td>$2^{156.67}$</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>22</td>
<td>33</td>
<td>5</td>
<td>0.699</td>
<td>$2^{157.27}$</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>22</td>
<td>34</td>
<td>4</td>
<td>0.620</td>
<td>$2^{157.94}$</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>26</td>
<td>31</td>
<td>3</td>
<td>0.541</td>
<td>$2^{158.68}$</td>
<td></td>
</tr>
</tbody>
</table>

**Table 6.2**: One-block preimages without padding: $N$ is the number of attacked steps, $n_1$ and $n_2$ the number of steps computed by $F_1$ and $F_2$, respectively, $d$ the dimension of $D_1$ and $D_2$, and $\bar{\alpha}$ the average type I error probability of matching.

### 6.3.3 One-Block Preimages with Padding

So far, we did not care about the padding of our preimage. If $M$ is required to have correct padding, the least significant bit of $M_{13}$ must be 1, $M_{14} = 0$, and $M_{15} = 447$ (assuming a message length of 447 bits) and the differences in $D_1$ and $D_2$ must be zero at the corresponding bits. This imposes 65 bit conditions, restricting the choice of $D_1$ and $D_2$. In general, the kernels of the first 15 steps and the last 15 steps do not contain any non-trivial differences satisfying these conditions and we have to settle for 13 steps. Table 6.3 summarizes the results and the detailed attack parameters are given in the Appendix. It turns out that we loose about five steps, one step can be attributed to the fact that differences in $D_2$ tend to have higher Hamming weight.
6.3 Application to SHA-1

Table 6.3: One-block preimages with padding: $N$ is the number of attacked steps, $n_1$ and $n_2$ the number of steps computed by $F_1$ and $F_2$, respectively, $d$ the dimension of $D_1$ and $D_2$, and $\bar{\alpha}$ the average type I error probability of matching.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$d$</th>
<th>$\bar{\alpha}$</th>
<th>Complexity</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>17</td>
<td>27</td>
<td>10</td>
<td>0.546</td>
<td>$2^{151.54}$</td>
<td>with padding</td>
</tr>
<tr>
<td>48</td>
<td>18</td>
<td>30</td>
<td>7</td>
<td>0.484</td>
<td>$2^{154.41}$</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>19</td>
<td>31</td>
<td>5</td>
<td>0.598</td>
<td>$2^{156.80}$</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>19</td>
<td>32</td>
<td>4</td>
<td>0.590</td>
<td>$2^{157.78}$</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>19</td>
<td>33</td>
<td>4</td>
<td>0.738</td>
<td>$2^{158.44}$</td>
<td></td>
</tr>
</tbody>
</table>

6.3.4 One-Block Pseudo-Preimages

A pseudo-preimage is a preimage that allows for a different IV than specified for SHA-1. Given $H$, we want to find $M$ and $H'$ such that $H = F(M, H') + H'$. The freedom of choosing $H'$ allows to use bicliques and the splice and cut technique.

We separate $F$ into three parts as shown in Fig. 6.3. The bicliques are constructed for $F_3$ computing the steps $27 - n_3$ to $26$ ($n_3$ steps). $F_1$ computes the steps $27$ to $26 + n_1$ ($n_1$ steps) and $F_2$ computes the steps $27 + n_1$ to $N - 1$ and $0$ to $26 - n_3$ ($n_1$ steps) using the splice and cut technique. With this choice, the elements of the kernel of the steps $27$ to $41$ (15 steps) and the elements of the kernel of the steps $11 - n_3$ to $26 - n_3$ (15 steps) automatically satisfy the padding conditions if $0 \leq n_3 \leq 11$.

Table 6.4 shows the best results we could find using suitable variants of Algorithm 6.3 and 6.4. The detailed attack parameters and a sample biclique for each configuration are given in the Appendix. The bicliques have been found within a few seconds using brute-force. Since $n_3 < 16$ one can build many bicliques from a single one by modifying those message words outside the bicliques. This makes that the amortized cost to construct bicliques is
negligible. The complexity is computed as $2^{n-d}(\Gamma_1 + \Gamma_2) + 2^{n-d}\Gamma_{re}$, where $\Gamma_1 + \Gamma_2 = (n_1 + n_2)/N$ and $\Gamma_{re} = (N - n_3 - 30)/N$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$d$</th>
<th>$\bar{\alpha}$</th>
<th>Complexity</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>18</td>
<td>28</td>
<td>2</td>
<td>12</td>
<td>0.447</td>
<td>$2^{149.22}$</td>
<td>with padding</td>
</tr>
<tr>
<td>49</td>
<td>19</td>
<td>28</td>
<td>2</td>
<td>11</td>
<td>0.398</td>
<td>$2^{150.12}$</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>19</td>
<td>29</td>
<td>2</td>
<td>10</td>
<td>0.404</td>
<td>$2^{151.15}$</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>17</td>
<td>30</td>
<td>4</td>
<td>9</td>
<td>0.381</td>
<td>$2^{152.02}$</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>18</td>
<td>30</td>
<td>4</td>
<td>8</td>
<td>0.332</td>
<td>$2^{152.93}$</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>18</td>
<td>29</td>
<td>6</td>
<td>7</td>
<td>0.128</td>
<td>$2^{153.46}$</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>18</td>
<td>30</td>
<td>6</td>
<td>8</td>
<td>0.569</td>
<td>$2^{153.50}$</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>18</td>
<td>30</td>
<td>7</td>
<td>6</td>
<td>0.208</td>
<td>$2^{154.60}$</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>19</td>
<td>31</td>
<td>6</td>
<td>8</td>
<td>0.765</td>
<td>$2^{154.41}$</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>19</td>
<td>32</td>
<td>6</td>
<td>7</td>
<td>0.675</td>
<td>$2^{154.96}$</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>21</td>
<td>30</td>
<td>7</td>
<td>5</td>
<td>0.478</td>
<td>$2^{156.25}$</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>19</td>
<td>34</td>
<td>6</td>
<td>5</td>
<td>0.626</td>
<td>$2^{156.78}$</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>20</td>
<td>34</td>
<td>6</td>
<td>4</td>
<td>0.524</td>
<td>$2^{157.45}$</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.4: One-block pseudo-preimages: $N$ is the number of attacked steps, $n_1$ and $n_2$ the number of steps computed by $F_1$ and $F_2$, respectively, $n_3$ is the length of the bicliques, $d$ the dimension of $D_1$ and $D_2$, and $\bar{\alpha}$ the average type I error probability of matching.

### 6.3.5 Two-Block Preimages

Given $H$, we want to find $M = (M^0, M^1)$ with a correct padding and such that $H = F(M^1, H^1) + H^1$, where $H^1 = F(M^0, IV) + IV$. The problem can be separated into two steps:

1. Find $M^1$ such that $H = F(H^1, M^1) + H^1$ for some $H^1$ and such that $M^1$ has a correct padding (a “one-block pseudo-preimage with padding”).

2. For the $H^1$ obtained in the first step, find $M_0$ such that $H^1 = F(M^0, IV) + IV$ (a “one-block preimage without padding”).

The two steps can be solved using the attacks from Section 6.3.2 and 6.3.4, respectively. The total complexity is the sum of the complexities of both steps, which is dominated by the second step (computing the first block). As an
example, for $N = 48$ we can compute a correctly padded two-block preimage with complexity $2^{150.62} + 2^{149.22} = 2^{151.08}$. For $N = 57$ the complexity is $2^{158.79}$.

For $N \geq 58$ we lack a method to compute the first block faster than by brute-force and we would have to use the generic method from [89, Fact 9.99] to convert a pseudo-preimage attack into a preimage attack. In fact, this is the procedure of most meet-in-the-middle preimage attacks. A pseudo-preimage attack with complexity $2^m$ can be converted to a preimage attack with complexity $2^{1+(m+n)/2}$. In our case, the resulting speed-up is less than a factor two for all results with $N \geq 58$ given in Table 6.4.

### 6.4 Accelerated Brute-Force Search

In this last section we briefly describe a generic optimization of the brute-force search which comes out of the meet-in-the-middle approach. It applies to any number of rounds, but the speed-up is very small. The idea is to avoid recomputing parts of $E$ for many trials. This has been previously applied to MD5 [5] and HAVAL-5 [105], and the same idea is called “matching with precomputation” and lead to the first key recovery attacks on full AES [23].

Suppose that $E$ can be separated into three parts, $E = E_2 \circ E_3 \circ E_1$, and let $D_1$ and $D_2$ be as in Section 6.2 with zero output differences ($\Delta_1, \Delta_2 = 0$ for all $\delta_1 \in D_1$ and $\delta_2 \in D_2$). Then, Algorithm 6.6 applies. The difference to Algorithm 6.1 is in the last loop where $E_3$ is computed each time. Testing $2^n$ keys in this way has complexity $2^{n-d}(\Gamma_1 + \Gamma_2) + 2^n \Gamma_3$. Thus, for reasonably large $d$, the complexity of brute-force search is reduced by the factor $\Gamma_1/\Gamma_3$.

**Application to SHA-1.** For SHA-1 with $N$ steps, the speed-up factor is about $N/(N - 30)$. Combining accelerated search with advanced matching techniques, the speed-up can be slightly increased. For $N = 80$ not more than a factor two is obtained, however. Such optimizations might not be considered as attacks, but they provide a minimal benchmark for attacks which is more accurate than just naive brute-force search.

### 6.5 Summary and Conclusion

We proposed a differential view on the meet-in-the-middle framework originally introduced by Aoki and Sasaki. Advanced matching techniques such
Algorithm 6.6 Testing $K \oplus D_1 \oplus D_2$ for a right key (accelerated search)

Input: $D_1, D_2 \subset \{0, 1\}^\kappa$, $K \in \{0, 1\}^\kappa$
Output: A right key if one is contained in $K \oplus D_1 \oplus D_2$.

for all $\delta_2 \in D_2$ do
    Compute $L_1[\delta_2] = E_1(K \oplus \delta_2, P)$.
end for

for all $\delta_1 \in D_1$ do
    Compute $L_2[\delta_1] = E_2^{-1}(K \oplus \delta_1, C)$.
end for

for all $(\delta_1, \delta_2) \in D_1 \times D_2$ do
    if $E_3(K \oplus \delta_1 \oplus \delta_2, L_1[\delta_2]) = L_2[\delta_1]$ then
        return $K \oplus \delta_1 \oplus \delta_2$
    end if
end for

return No right key in $K \oplus D_1 \oplus D_2$

as partial matching, indirect partial matching, partial fixing, and probabilis-
tic matching appear very natural from this perspective. For block cipher
based hash functions in Davies-Meyer mode, the principal attack parame-
ters are two sets of suitable related-key differentials. Tools are proposed that
facilitate a systematic search for these sets.

Applied to SHA-1, our framework leads to significantly better preimage
attacks up to 57 out of 80 steps. The results are illustrated in Fig. 6.4 and
compared to the best previous attack as well as to accelerated brute-force
search. The improvements essentially come from a more systematic use of
probabilistic matching. It is remarkable that we do not rely on the generic
conversion of pseudo-preimage attacks into preimage attacks. This allows
us to obtain speed-up factors that would be hard to achieve with the generic
conversion.

Application of the framework to the SHA-2 family seems more compi-
lcated, namely due to the non-linear message expansion. Nevertheless, it can
be expected that the differential perspective on meet-in-the-middle attacks
leads to improved results on other primitives as well.
Figure 6.4: Preimage attacks against reduced SHA-1: Illustration of the new results and comparison to accelerated brute-force search.
Appendix A

On the Applicability of the Cube Attack

The cube attack was introduced by Dinur and Shamir [41] as a very general attack, applicable to a broad class of cryptographic algorithms and provably successful on random polynomials. But so far, the only relevant application is against reduced variants of Trivium [41]. Let us briefly recall the attack. Consider a Boolean function $f : \{0, 1\}^\kappa \times \{0, 1\}^n \to \{0, 1\}$ modeling the output of a cipher. The cube attack works in two phases:

Preprocessing: Find $\kappa$ derivatives $\Delta_{V_1} f, \ldots, \Delta_{V_\kappa} f$ that are independent linear expressions in key bits.

Key recovery: Evaluate the derivatives and solve the resulting system of linear equations in the key bits.

Preprocessing Random Polynomials

It is shown in [41] that the cube attack succeeds with overwhelming probability if $f$ is a random polynomial of degree $\delta \leq n - \log_2 \kappa$. This follows from the fact that a derivative of order $d$ has degree at most $\delta - d$ (Property 4.3). Hence, if $d = \delta - 1$, the derivative is a priori linear, and it is a non-trivial linear expression in key bits with probability $1 - 2^{-\kappa}$. However, by a very similar reasoning one can see that if $f$ is a random polynomial of degree $\delta > n$, the cube attack fails with overwhelming probability. The order of the derivative
is limited to \( n \) and the derivatives are not a priori linear anymore. They are linear in the key variables only if none of the higher degree monomials in key bits appear. This probability is very small. As pointed out by Bernstein [16], designers are aware that the output of a cipher should achieve high degree. For modern ciphers it is much more reasonable to assume that \( f \) is a random polynomial of degree \( \delta = n + \kappa \) than a random polynomial of degree \( \delta < n \).

**How to Prevent the Cube Attack**

Modern ciphers are immune against the cube attack, because their outputs have high degree. However, even low degree ciphers can easily be protected against the cube attack as it is shown by the following observation on Trivium (we refer to Section 4.5 for the description of Trivium). It turns out that some bits of the key never appear in a linear superpoly. Namely, these are the bits \( k_{69}, \ldots, k_{79} \) (see [41, Appendix B]). This can be explained by analyzing the first few initialization rounds. Let’s consider \( k_{69} \) as an example. It enters the mixing process via \( t_1 \) at the rounds 21, 22, and 23. After 24 rounds it appears in the state exclusively as

\[
\begin{align*}
s_{94} &= x_{54} + k_{42} + k_{67}k_{68} + k_{89}, \\
s_{95} &= x_{55} + k_{43} + k_{68}k_{69} + k_{70}, \\
s_{96} &= x_{56} + k_{44} + k_{69}k_{70} + k_{71}.
\end{align*}
\]

The only possibility for \( k_{69} \) to appear in a linear superpoly is when the quadratic term in \( s_{94} \) cancels out during the initialization process. This is not excluded, but it is very unlikely. In general, a non-linear mixing of the key bits at the very beginning of the initialization effectively prevents the cube attack. This explains why the Grain family is completely immune against the cube attack — even for a very reduced number of rounds.\(^{15}\)

\(^{15}\)This does not hold for the dynamic cube attack [42] which is quite different from the original cube attack considered here.
Appendix B

Collisions for CubeHash Variants

CubeHash is a hash function designed by Bernstein and submitted to the SHA-3 competition in 2008 [15]. It has been selected as one of sixteen second-round candidates. In [62] we presented improved collision attacks on reduced variants. Our results have been awarded with a prize of 500 € by the designer and are still the best so far.

Description of CubeHash. The hash function is designed with two parameters $r$ and $b$ which are the number of rounds and the number of bytes per message block. CubeHash-$r/b$ becomes stronger (but slower) for increasing $r$ and / or decreasing $b$. Initially, CubeHash-8/1 was proposed, later it was tweaked to CubeHash-16/32. All output lengths required by the NIST can be generated: 224, 256, 384, and 512 bits.

CubeHash operates on 32-bit words. It maintains a 1024-bit internal state $X$ which is composed of 32 words $X_0, \ldots, X_{31}$. The algorithm is composed of five steps ($h$ denotes the output length):

1. Initialize the state $X$ to a value that depends on $r$, $b$, and $h$.
2. Pad the message to a sequence of $b$-byte input blocks.
3. For every $b$-byte input block:
- XOR the block into the first $b$-bytes of the state.
- Transform the state through $r$ identical rounds.

4. Finalize the state: XOR 1 into $X_{31}$ and transform the state through $10r$ identical rounds.

5. Output the first $h$ bits of the state.

A round consists of the following steps:
- Add $X_i$ into $X_{i\oplus 16}$, for $0 \leq i \leq 15$.
- Rotate $X_i$ to the left by seven bits, for $0 \leq i \leq 15$.
- Swap $X_i$ and $X_{i\oplus 8}$, for $0 \leq i \leq 7$.
- XOR $X_{i\oplus 16}$ into $X_i$, for $0 \leq i \leq 15$.
- Swap $X_i$ and $X_{i\oplus 2}$, for $i \in \{16, 17, 20, 21, 24, 25, 28, 29\}$.
- Add $X_i$ into $X_{i\oplus 16}$, for $0 \leq i \leq 15$.
- Rotate $X_i$ to the left by eleven bits, for $0 \leq i \leq 15$.
- Swap $X_i$ and $X_{i\oplus 4}$, for $i \in \{0, 1, 2, 3, 8, 9, 10, 11\}$.
- XOR $X_{i\oplus 16}$ into $X_i$, for $0 \leq i \leq 15$.
- Swap $X_i$ and $X_{i\oplus 1}$, for $i \in \{16, 18, 20, 22, 24, 26, 28, 30\}$.

**Linearization Framework and Condition Function.** The results have been obtained using a framework for collision attacks proposed by Brier, Khazaei, Meier, and Peyrin in [26]. The principal tool of the framework is the so-called condition function and a tree-based backtracking algorithm to find a preimage of zero for this function. Finding such a preimage is equivalent to finding a message pair $(M, M \oplus \Delta)$ that follows a particular linear differential characteristic. The algorithms also allows to estimate the complexity of attacks that are not feasible in practice. We refer to [26] and to Khazaei’s thesis [61] for a detailed description of the framework.

The framework has been applied already in [26] to CubeHash. Our improved results are due to better techniques for finding suitable linear differential characteristics. First, we noticed that not only the Hamming weight of the characteristics but also the weight distribution is important for obtaining low-complexity attacks. Second, we applied a pragmatic randomized search algorithm inspired by Stern’s algorithm for finding low-weight codewords of linear codes [93, 114].
New Attacks up to 8 Rounds. For CubeHash-$r/b$ there is a generic collision attack with complexity of about $2^{512-4b}$ (see [15]). For $b > 64$ this is faster than the generic birthday attack on hash functions with output length $h = 512$. For $b = 96$, specifically, the generic attack has a complexity of about $2^{128}$ evaluations of the compression function. Our attacks clearly improve over this bound.

The previously best collision attacks on CubeHash-$r/b$ were presented in [26]. For $b = 32$ they found attacks of complexity $2^{54}$ and $2^{182}$ evaluations of the compression function for four and six rounds, respectively. For $b = 64$ an attack of complexity $2^{203}$ for seven rounds is given. For six rounds, our attacks have complexities $2^{180}$, $2^{132}$, and $2^{51}$ for $b = 32, 64$, and 96, respectively. For eight rounds, we found an attack with complexity $2^{80}$ for $b = 96$. We refer to [61] for a complete survey of collision attacks on CubeHash variants.

Collision for CubeHash-5/96. The following two-block difference allowed us to practically find three-block collisions:

$$\Delta^1 = 0x08000208 \ 0x08000208 \ 0x00000000 \ 0x00000000 \ 0x40000100$$
$$0x00000000 \ 0x00400110 \ 0x00000000 \ 0x00400000 \ 0x08000000$$
$$0x00000000 \ 0x08000888 \ 0x08000208 \ 0x00000000 \ 0x00000000$$
$$0x40011000 \ 0x00000000 \ 0x00451040 \ 0x00000000.$$  

$$\Delta^2 = 0x80000000 \ 0x80000000 \ 0x80000000 \ 0x80000000 \ 0x00000000$$
$$0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000$$
$$0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000$$
$$0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000$$

The difference in the first block, that is $\Delta^1$, results in the difference $\Delta^2$ after 5 rounds if a linear characteristic is followed. No differences appear in the last $128 - 96$ bytes of the state, and all differences can be cancelled by the difference in the second block. The crucial property of $\Delta^1$ is that its linear characteristic is relatively sparse in the last rounds.

Using Khazaei’s framework, the complexity of finding a collision with this difference is estimated about $2^{32}$. In practice, we found several collisions after $2^{22.5}$ to $2^{32.25}$ evaluations of the compression functions. An example is the following message $M = M^0 \| M^1 \| M^2$: 
$M^0 = 0xf06bb068 \ 0x487c5fe1 \ 0xcccaba70 \ 0xa989262 \ 0x801edc3a
0x69292196 \ 0x8848f445 \ 0xb8608777 \ 0xc037795a \ 0x10d5d799
0xfd16c037 \ 0xa52d0b51 \ 0x63a74c97 \ 0xdf858eef \ 0x7809480f
0x43eb264c \ 0xdb61863 \ 0x2a8ccfe2 \ 0x9ea22b139 \ 0xd99e4888
0x8ca844fb \ 0xece3295 \ 0x150ca98e \ 0xb16b0b92,$

$M^1 = 0x3db4d4ee \ 0x02958f57 \ 0x8eff307a \ 0x5be9975b \ 0x4d0a669e
0xe6025663 \ 0x8db6421 \ 0xbad8f1e4 \ 0x384fe128 \ 0x4ebb7e2a
0x72e45678 \ 0x1e44c51b \ 0xda607fd9 \ 0x1ddad41f \ 0x4180297a
0x1607f902 \ 0x2463d259 \ 0x2b73f829 \ 0xc79e766d \ 0x0f672ecc
0x084e841b \ 0xfc7020f5 \ 0x3095e865 \ 0x88eeb85d5,$

$M^2 = $ arbitrary message block of 96 bytes.

Using $M^2 = 0$, the messages $M^0 || M^1 || M^2$ and $M^0 || M^1 \oplus \Delta^1 || M^2 \oplus \Delta^2$ collide to the following hash value:

$H = 0xc2e51517 \ 0xc503746e \ 0x46ecd6ad \ 0x5936ec9b \ 0xff9b74f9
0x2ce4506 \ 0x624f2b0b \ 0xfe584d2c \ 0x56cd3e0e \ 0x18853ba8
0x4a9d6d38 \ 0xf1f8e45f \ 0x2129c678 \ 0xcb3636d4 \ 0xd865de13
0x410e96c.$

**Conclusion.** Combining known techniques (tools from coding theory and Khazaei’s framework for collision attacks) we found collision attacks up to 8 rounds of CubeHash (with very large $b = 96$, however). For CubeHash-5/96 we could find collisions in a few seconds on a PC. The experiments provide some evidence that the theoretical complexity estimates are quite accurate. Our results do not reveal any weakness in the official CubeHash-16/32. In contrary, they confirm that the tweak of initial CubeHash-8/1 was a good decision.\(^\text{16}\)

\(^{16}\)Nevertheless, CubeHash did not make it into the final round of the SHA-3 competition.
Appendix C

Attack Parameters for SHA-1

The attack complexities given in Chapter 6 rely on experiments, namely in an estimated value of the average type I error probability $\alpha$. In the following we provide all the details which are required to reproduce the results.

Description of Attack Parameters

The attack parameters always include the separation of $F$ (given by $n_1$ and $n_2$), two linear difference spaces $D_1$ and $D_2$ of dimension $d$, and a truncation mask $T$. Additionally, when using bicliques and the splice and cut technique, the length of the bicliques is required (given by $n_3$) and a sample biclique is given in order to demonstrate that suitable bicliques indeed exist. We use the following notations and conventions:

- C-style hex notation for 32-bit words: for example 0x5a827999 is the round constant $K_0$ in SHA-1.

- The linear spaces $D_1$ and $D_2$ have a particular form which allows them to be represented by a single element and the dimension $d$. If the difference $x_0||\ldots||x_{15}$ is given, a basis of the corresponding subspace is obtained by word-wise rotation as follows: $(x_0 \ll i)||\ldots||(x_{15} \ll i)$ for $i = 0, \ldots, d - 1$.

- A biclique is specified by one of its states, say $Q_1[0]$, a message $M$, and the subspaces $D_1$ and $D_2$. The remaining states of the biclique can be computed using the definition (see Section 6.2.6).
Parameters for Results in Table 6.2 (one-block, no padding)

$N = 44$, $n_1 = 18$, $n_2 = 26$, $d = 15$

$D_1$: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000008

$D_2$: 0xc0000000 0x00000000 0x80000000 0x80000000 0x40000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

$T$: 0xe0000007 0xe0000007 0xc0000001 0x00000000 0x00000000

$N = 48$, $n_1 = 19$, $n_2 = 29$, $d = 11$

$D_1$: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000008

$D_2$: 0x50000000 0x00000000 0x40000000 0x00000000 0x60000000 0x00000000 0x40000000 0x40000000 0x20000000 0x00000000 0x00000000 0x40000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

$T$: 0xc0000001 0xc0000005 0x60000001 0x00000000 0x00000000

$N = 49$, $n_1 = 19$, $n_2 = 30$, $d = 10$

$D_1$: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000008

$D_2$: 0x40000000 0x50000000 0x00000000 0x40000000 0x00000000 0x00000000 0x40000000 0x40000000 0x20000000 0x00000000 0x00000000 0x40000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

$T$: 0xc0000001 0xc0000005 0x40000001 0x00000000 0x00000000
\[ N = 50, \ n_1 = 19, \ n_2 = 31, \ d = 10 \]

\[
D_1: \begin{align*}
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
\end{align*}
\]

\[
D_2: \begin{align*}
0x50000000 \ 0x40000000 \ 0x50000000 \ 0x00000000 \ 0x40000000 \\
0x00000000 \ 0x60000000 \ 0x00000000 \ 0x40000000 \ 0x40000000 \\
0x20000000 \ 0x00000000 \ 0x40000000 \ 0x40000000 \ 0x20000000 \\
0x00000000 \\
\end{align*}
\]

\[
T: \ 0xe0000001 \ 0x80000007 \ 0x00000001 \ 0x00000001 \ 0x00000000 \\
\]

\[ N = 51, \ n_1 = 19, \ n_2 = 32, \ d = 9 \]

\[
D_1: \begin{align*}
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000020 \\
\end{align*}
\]

\[
D_2: \begin{align*}
0x00000000 \ 0x50000000 \ 0x40000000 \ 0x50000000 \ 0x00000000 \\
0x40000000 \ 0x00000000 \ 0x60000000 \ 0x00000000 \ 0x40000000 \\
0x40000000 \ 0x20000000 \ 0x00000000 \ 0x40000000 \ 0x40000000 \\
0x20000000 \\
\end{align*}
\]

\[
T: \ 0x80000007 \ 0x00000007 \ 0x00000001 \ 0x00000001 \ 0x00000000 \\
\]

\[ N = 52, \ n_1 = 21, \ n_2 = 31, \ d = 7 \]

\[
D_1: \begin{align*}
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000020 \\
\end{align*}
\]

\[
D_2: \begin{align*}
0x40000000 \ 0x00000000 \ 0xa0000000 \ 0x80000000 \ 0xa0000000 \\
0x00000000 \ 0x80000000 \ 0x00000000 \ 0xc0000000 \ 0x00000000 \\
0x80000000 \ 0x80000000 \ 0x40000000 \ 0x00000000 \ 0x80000000 \\
0x80000000 \\
\end{align*}
\]

\[
T: \ 0x40000000 \ 0x00000006 \ 0xc0000001 \ 0x00000001 \ 0x00000000 \\
\]
$N = 53, n_1 = 21, n_2 = 32, d = 6$

$D_1: \begin{align*}
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000080 & \\
\end{align*}$

$D_2: \begin{align*}
0x00000000 & 0x20000000 \\
0x50000000 & 0x40000000 \\
0x00000000 & 0x40000000 \\
0x40000000 & 0x60000000 \\
0x40000000 & \\
\end{align*}$

$T: \begin{align*}
0x40000000 & 0x000000005 \\
0x40000000 & 0x00000001 \\
0x00000000 & \\
\end{align*}$

$N = 54, n_1 = 21, n_2 = 33, d = 5$

$D_1: \begin{align*}
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000040 & \\
\end{align*}$

$D_2: \begin{align*}
0x00000000 & 0x20000000 \\
0x40000000 & 0x50000000 \\
0x40000000 & 0x40000000 \\
0x60000000 & 0x40000000 \\
0x40000000 & \\
\end{align*}$

$T: \begin{align*}
0xc0000000 & 0x00000004 \\
0x00000000 & 0x00000001 \\
0x00000000 & \\
\end{align*}$

$N = 55, n_1 = 22, n_2 = 33, d = 5$

$D_1: \begin{align*}
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 \\
0x00000080 & \\
\end{align*}$

$D_2: \begin{align*}
0x40000000 & 0x00000000 \\
0x50000000 & 0x40000000 \\
0x00000000 & 0x20000000 \\
0x40000000 & 0x40000000 \\
0x20000000 & \\
\end{align*}$

$T: \begin{align*}
0x00000004 & 0x80000001 \\
0x00000000 & 0x00000001 \\
0x00000000 & \\
\end{align*}$
\[ N = 56, \ n_1 = 22, \ n_2 = 34, \ d = 4 \]

\[ D_1: \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000080 \]

\[ D_2: \ 0x40000000 \ 0x40000000 \ 0x00000000 \ 0x00000000 \ 0x20000000 \\
0x00000000 \ 0x50000000 \ 0x40000000 \ 0x50000000 \ 0x00000000 \\
0x40000000 \ 0x00000000 \ 0x60000000 \ 0x00000000 \ 0x40000000 \\
0x40000000 \]

\[ T: \ 0x00000004 \ 0x80000001 \ 0x00000001 \ 0x00000000 \ 0x00000000 \]

\[ N = 57, \ n_1 = 26, \ n_2 = 31, \ d = 3 \]

\[ D_1: \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
0x00000080 \]

\[ D_2: \ 0x80000000 \ 0x80000000 \ 0x80000000 \ 0x00000000 \ 0x00000000 \\
0x40000000 \ 0x00000000 \ 0xa0000000 \ 0x80000000 \ 0xa0000000 \\
0x00000000 \ 0x80000000 \ 0x00000000 \ 0xc0000000 \ 0x00000000 \\
0x80000000 \]

\[ T: \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0xc0000001 \]
Parameters for Results in Table 6.3 (one-block, with padding)

$N = 44$, $n_1 = 17$, $n_2 = 27$, $d = 10$

$D_1$: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

$D_2$: 0x0000000004 0x0000000014 0x0000000000 0x0000000000 0x0000000004 0x000000001c 0x0000000010 0x0000000000 0x0000000004 0x000000001c 0x0000000008 0x0000000018 0x000000000c 0x0000000004 0x0000000000

$T$: 0x0000003c 0x0000000006 0x0000000000 0x0000000000

$N = 48$, $n_1 = 18$, $n_2 = 30$, $d = 7$

$D_1$: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

$D_2$: 0x0000000004 0x0000000005 0x0000000000 0x0000000004 0x0000000000 0x0000000006 0x0000000000 0x0000000004 0x0000000004 0x0000000002 0x0000000000 0x0000000004 0x0000000004 0x0000000002 0x0000000000

$T$: 0x00000018 0x00000001c 0x0000000006 0x0000000000 0x0000000000

$N = 50$, $n_1 = 19$, $n_2 = 31$, $d = 5$

$D_1$: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

$D_2$: 0x80000003 0x40000002 0x60000002 0xc0000001 0xa0000003 0x00000001 0x00000002 0x80000001 0xc0000003 0x00000002 0x00000000 0x00000000 0x40000003 0x00000002 0x00000000 0x00000000

$T$: 0x00000007 0x00000000 0x80000001 0x00000000 0x00000000
\[ N = 51, \ n_1 = 19, \ n_2 = 32, \ d = 4 \]

\[ D_1: \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
    0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
    0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
    0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
    0x00000000 \]

\[ D_2: \ 0x08000000 \ 0x18000000 \ 0x04000000 \ 0x1a000000 \ 0x0c000000 \\
    0x0e000000 \ 0x10000000 \ 0x00000000 \ 0x18000000 \ 0x04000000 \\
    0x00000000 \ 0x00000000 \ 0x10000000 \ 0x1c000000 \ 0x00000000 \\
    0x00000000 \ 0x00000000 \]

\[ T: \ 0x38000000 \ 0x20000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \]

\[ N = 52, \ n_1 = 19, \ n_2 = 33, \ d = 4 \]

\[ D_1: \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
    0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
    0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000020 \\
    0x00000000 \ 0x00000000 \]

\[ D_2: \ 0x04000000 \ 0x04000000 \ 0x02000000 \ 0x02000000 \ 0x16000000 \\
    0x1a000000 \ 0x1c000000 \ 0x08000000 \ 0x1c000000 \ 0x0c000000 \\
    0x10000000 \ 0x00000000 \ 0x1c000000 \ 0x14000000 \ 0x00000000 \\
    0x00000000 \ 0x00000000 \]

\[ T: \ 0x78000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \]
Parameters for Results in Table 6.4 (pseudo, with padding)

$N = 48, n_1 = 18, n_2 = 28, n_3 = 2, d = 12$

$D_1: \begin{align*}
0x00000040 & 0x00000040 0x00000020 0x00000000 0x00000040 \\
0x00000040 & 0x00000020 0x00000000 0x00000000 0x00000040 \\
0x00000060 & 0x00000000 0x00000040 0x00000040 0x00000000 \\
0x00000000 & 
\end{align*}$

$D_2: \begin{align*}
0x00000000 & 0x00000000 0x00000000 0x00000020 0x00000000 \\
0x00000000 & 0x00000020 0x00000000 0x00000020 0x00000000 \\
0x00000000 & 0x00000000 0x00000000 0x00000000 0x00000000 \\
0x00000000 & 
\end{align*}$

$T: \begin{align*}
0x0000007c & 0x00000078 0x00000018 0x00000010 0x00000000 \\
\end{align*}$

$Q_1[0]: \begin{align*}
0x9ee4eb7b & 0xfaf60c40 0x3d9f5367 0x4faed2df 0x102cdef0 \\
\end{align*}$

$M: \begin{align*}
0x1e45706f & 0x23072fb4 0x83ca0391 0xd019a507 0xfaf436fe \\
0x1a5323a & 0xd5b2372d 0xe3ab26bc 0x60d820ff 0x823704a1 \\
0x52a9a83b & 0xd251e88b 0x5efac42d 0xa156a969 0x00000000 \\
0x00000000 & 
\end{align*}$

$N = 49, n_1 = 19, n_2 = 28, n_3 = 2, d = 11$

$D_1: \begin{align*}
0x00000040 & 0x00000040 0x00000020 0x00000000 0x00000040 \\
0x00000040 & 0x00000020 0x00000000 0x00000000 0x00000040 \\
0x00000060 & 0x00000000 0x00000040 0x00000000 0x00000000 \\
0x00000000 & 
\end{align*}$

$D_2: \begin{align*}
0x00000000 & 0x00000000 0x00000000 0x00000020 0x00000000 \\
0x00000000 & 0x00000020 0x00000000 0x00000020 0x00000000 \\
0x00000000 & 0x00000000 0x00000000 0x00000000 0x00000000 \\
0x00000000 & 
\end{align*}$

$T: \begin{align*}
0x00000000 & 0x00000000 0x00000000 0x00000000 0x00000000 \\
\end{align*}$

$Q_1[0]: \begin{align*}
0xa1dc712e & 0x95d6262d 0x92df251f 0x7bcb2b2c6 0x2fe92f9b \\
\end{align*}$

$M: \begin{align*}
0xfc9adc5d & 0x3e808f27 0x88c2d680 0x8d1369d6 0x344e2afa \\
0x3b33e529 & 0x274b7f9d 0x8820b297 0x4f2d897d 0x3b56b074 \\
0x95135463 & 0xed2aa206 0x2e889c55 0xcba1d561 0x00000000 \\
0x00000000 & 
\end{align*}$
\[ N = 50, n_1 = 19, n_2 = 29, n_3 = 2, d = 10 \]

\[ D_1: \]
\begin{verbatim}
0x00000040 0x00000040 0x00000200 0x00000000 0x00000040
0x00000040 0x00000200 0x00000000 0x00000000 0x00000040
0x00000060 0x00000000 0x00000040 0x00000000 0x00000000
0x00000000
\end{verbatim}

\[ D_2: \]
\begin{verbatim}
0x00000000 0x00000000 0x00000010 0x00000000 0x00000000
0x00000010 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000
\end{verbatim}

\[ T: \]
\begin{verbatim}
0x0000001c 0x0000003c 0x0000001c 0x00000000 0x00000000
\end{verbatim}

\[ Q_1[0]: \]
\begin{verbatim}
xdf32bba1 0xec891b4d 0x187d32d3 0xf4393c27 0x46b65b83
\end{verbatim}

\[ M: \]
\begin{verbatim}
oxafe5cd01 0xca7ff558 0x2b762e13 0x3b5ac985 0x76cc73cf
0x435e511b 0xb97d9f39 0xecb73848 0x2c594bfd 0x39f075e6
0x891cf868 0x8e5fd6a3 0x4aceb64c 0x54132315 0x00000000
0x000003bf
\end{verbatim}

\[ N = 51, n_1 = 17, n_2 = 30, n_3 = 4, d = 9 \]

\[ D_1: \]
\begin{verbatim}
0x00010000 0x00010000 0x00008000 0x00000000 0x00010000
0x00010000 0x00008000 0x00000000 0x00000000 0x00010000
0x00018000 0x00000000 0x00010000 0x00010000 0x00000000
0x00000000
\end{verbatim}

\[ D_2: \]
\begin{verbatim}
0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000
\end{verbatim}

\[ T: \]
\begin{verbatim}
0x00000003c 0x00000007c 0x00000000 0x00000000 0x00000000
\end{verbatim}

\[ Q_1[0]: \]
\begin{verbatim}
x1fb4844b 0xc4eb4d4c 0xffccc87a 0x74f112bd 0xb33d94f1
\end{verbatim}

\[ M: \]
\begin{verbatim}
ox3b2701c3 0x532837db 0xe9f9477f 0x4f8ab29f 0xf05433ee
0xe9bad6bc4 0x1b9933e5 0xf07f0a3d 0x536d687b 0x315d017c
0x87bb41d2 0xd7cf7647 0x8fe9fffa 0x4f9a2687 0x00000000
0x000003bf
\end{verbatim}
$N = 52$, $n_1 = 18$, $n_2 = 30$, $n_3 = 4$, $d = 8$

$D_1$: 0x00002000 0x00002000 0x00001000 0x00000000 0x00002000
        0x00002000 0x00001000 0x00000000 0x00000000 0x00002000
        0x00003000 0x00000000 0x00002000 0x00002000 0x00000000
        0x00000000

$D_2$: 0x00000000 0x00000008 0x00000000 0x00000008 0x00000000
        0x00000008 0x00000000 0x00000008 0x00000000 0x00000000
        0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
        0x00000000

$T$: 0x0000001c 0x00000003c 0x00000004 0x00000000 0x00000000

$Q_1[0]$: 0xf6b1c9fd 0xe30107b 0x9308c95d 0xc955f6e3 0x656e0e53

$M$: 0x18d3ab4b 0xf0e83376 0x80a8fa1a 0x286eaa4c 0xe947f6f2
     0x3a974781 0x99f70ef6 0x339ad3c3 0x2e688d27 0xe5f6533b
     0x3e9e7eca 0xa5a21b0c 0x24b2b404 0x2373a2d7 0x00000000
     0x000003bf

$N = 53$, $n_1 = 18$, $n_2 = 29$, $n_3 = 6$, $d = 7$

$D_1$: 0x00800000 0x00800000 0x00400000 0x00000000 0x00800000
        0x00800000 0x00400000 0x00000000 0x00000000 0x00800000
        0x00c00000 0x00000000 0x00800000 0x00800000 0x00000000
        0x00000000

$D_2$: 0x00000000 0x00000008 0x00000000 0x00000008 0x00000000
        0x00000008 0x00000000 0x00000008 0x00000000 0x00000000
        0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
        0x00000000

$T$: 0x00000007 0x00000000 0x00000007 0x00000004 0x00000000

$Q_1[0]$: 0xf405ff03 0x6749a21d 0x43681806 0x4190a0cf 0x51d242ee

$M$: 0x9c34e7cf 0x88c0c05b 0xf004c8ed 0xe80233b3 0xdf105a66
     0x846f1ff3 0xf948b1ee 0xd43fa31e 0x35dd01db 0x38e640bf
     0x4cd2dd3e 0xff42a038 0xd64f98d3 0x734e7feb 0x00000000
     0x000003bf
\[ N = 54, n_1 = 18, n_2 = 30, n_3 = 6, d = 8 \]

\[
D_1: \quad \begin{array}{cccccc}
0x00200000 & 0x00200000 & 0x00100000 & 0x00000000 & 0x00200000 \\
0x00200000 & 0x00100000 & 0x00000000 & 0x00000000 & 0x00200000 \\
0x00300000 & 0x00000000 & 0x00200000 & 0x00200000 & 0x00000000 \\
0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 \\
\end{array}
\]

\[
D_2: \quad \begin{array}{cccccc}
0x00000000 & 0x00000000 & 0x00000004 & 0x00000000 & 0x00000000 \\
0x00000000 & 0x00000004 & 0x00000000 & 0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 \\
\end{array}
\]

\[
T: \quad \begin{array}{cccccc}
0x00000001 & 0x00000010 & 0x0000000f & 0x0000000c & 0x00000000 \\
\end{array}
\]

\[
Q_1[0]: \quad \begin{array}{cccccc}
0x7d3eebf0 & 0xf8c333c5 & 0xce413a03 & 0xbb70c7c5 & 0x8f1034a9 \\
\end{array}
\]

\[
M: \quad \begin{array}{cccccc}
0x32fb067d & 0x54eb75ba & 0xdbe304c3 & 0x2389c4bc & 0x679195c1 \\
0x188df1ff & 0x5aca2211 & 0x36ed7d12 & 0x34ee1e98 & 0x5f07d210 \\
0x28a18e32 & 0x1cc6315 & 0x139d4efd & 0x888124e7 & 0x00000000 \\
0x0000003bf & \end{array}
\]

\[ N = 55, n_1 = 18, n_2 = 30, n_3 = 7, d = 6 \]

\[
D_1: \quad \begin{array}{cccccc}
0x00800000 & 0x00800000 & 0x00400000 & 0x00000000 & 0x00800000 \\
0x00800000 & 0x00400000 & 0x00000000 & 0x00000000 & 0x00800000 \\
0x00c00000 & 0x00000000 & 0x00800000 & 0x00800000 & 0x00000000 \\
0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 \\
\end{array}
\]

\[
D_2: \quad \begin{array}{cccccc}
0x00000002 & 0x00000000 & 0x00000002 & 0x00000000 & 0x00000002 \\
0x00000000 & 0x00000002 & 0x00000000 & 0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 \\
0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 & 0x00000000 \\
\end{array}
\]

\[
T: \quad \begin{array}{cccccc}
0x00000007 & 0x00000000 & 0x80000003 & 0x00000000 & 0x00000000 \\
\end{array}
\]

\[
Q_1[0]: \quad \begin{array}{cccccc}
0x67d71ad3 & 0xd421573b & 0x06575dab & 0x32448a3a & 0x290c7720 \\
\end{array}
\]

\[
M: \quad \begin{array}{cccccc}
0xf695e1ae & 0x21db1192 & 0x248c67ba & 0x13349385 & 0x90f957c5 \\
0x9f48780f & 0xa0574385 & 0x46a7df4 & 0x75dc07e0 & 0x14ec69fe \\
0xa429beb8 & 0x61c7b895 & 0x21b1114a & 0xa7c8eb73 & 0x00000000 \\
0x000003bf & \end{array}
\]
$N = 56, n_1 = 19, n_2 = 31, n_3 = 6, d = 8$

$D_1: \begin{array}{c}
0x00080000 0x00080000 0x00040000 0x00000000 0x00080000 \\
0x00080000 0x00040000 0x00000000 0x00000000 0x00080000 \\
0x000c0000 0x00000000 0x00080000 0x00080000 0x00000000 \\
0x00000000
\end{array}$

$D_2: \begin{array}{c}
0x00000000 0x00000001 0x00000000 0x00000000 0x00000000 \\
0x00000001 0x00000000 0x00000000 0x00000000 0x00000000 \\
0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 \\
0x00000000
\end{array}$

$T: \begin{array}{c}
0x0000000c 0x00000000 0x00000007 0x00000007 0x00000000
\end{array}$

$Q_1[0]: \begin{array}{c}
0xb5a23a76 0xc16ccdbf 0xe24aa7f9 0x5d989435 0x8af32e32
\end{array}$

$M: \begin{array}{c}
0xdc467f36 0xa7de388e 0x96ff6b6f 0x83a9070d 0xb96959e7 \\
0xef872e08 0xc985d88e 0x6faa299e 0xb41b3686 0xc1635fbc \\
0xb6dd33ae 0x4f10520b 0x7100ff03 0x9fe99105 0x00000000 \\
0x000003bf
\end{array}$

$N = 57, n_1 = 19, n_2 = 32, n_3 = 6, d = 7$

$D_1: \begin{array}{c}
0x00020000 0x00020000 0x00010000 0x00000000 0x00020000 \\
0x00020000 0x00010000 0x00000000 0x00000000 0x00020000 \\
0x00030000 0x00000000 0x00020000 0x00020000 0x00000000 \\
0x00000000
\end{array}$

$D_2: \begin{array}{c}
0x00000000 0x80000000 0x00000000 0x80000000 0x00000000 \\
0x80000000 0x00000000 0x00000000 0x00000000 0x00000000 \\
0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 \\
0x00000000
\end{array}$

$T: \begin{array}{c}
0x00000003 0x00000000 0x80000003 0x00000003 0x00000000
\end{array}$

$Q_1[0]: \begin{array}{c}
0x1c2652fe 0x53eb4c0a 0x57e9168f 0xf65b3a56 0x7c428e01
\end{array}$

$M: \begin{array}{c}
0xce369809 0x3ea1797b 0x1ab39a0d 0x96d1d5e0 0x7a550f31 \\
0xad4da4dd 0xf72712f 0x17d8a5e8 0x96d2d21d 0x3b0faff80 \\
0x7cc259ff 0xb27a9d25 0x22a2a94 0x88bbfd35 0x00000000 \\
0x000003bf
\end{array}$
\( N = 58, n_1 = 21, n_2 = 30, n_3 = 7, d = 5 \)

\[ D_1: \]
0x00800000 0x00800000 0x00400000 0x00000000 0x00800000
0x00800000 0x00400000 0x00000000 0x00000000 0x00800000
0x00c00000 0x00000000 0x00800000 0x00800000 0x00000000
0x00000000

\[ D_2: \]
0x80000000 0x00000000 0x80000000 0x00000000 0x80000000
0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000

\[ T: \]
0x00000000 0x00000000 0x00000000 0xf0000000 0x00000000

\[ Q_1[0]: \]
0x77f88fc0 0xef5f9fab 0xa53354bc 0x7ebfcd0f 0xe9bdf64c

\[ M: \]
0x9c8db6ac 0xf00b105f 0x9bc20b30 0xd61c24b4 0xb7c07ff0
0xf0715fbb 0xabc44099 0x2d29be5a 0x0a49f67b 0x7befcf1b
0xc8b7bbd6 0x9800488d 0xf49d23e2 0x423af681 0x00000000
0x000003bf

\( N = 59, n_1 = 19, n_2 = 34, n_3 = 6, d = 5 \)

\[ D_1: \]
0x00400000 0x00400000 0x00200000 0x00000000 0x00400000
0x00400000 0x00200000 0x00000000 0x00000000 0x00400000
0x00600000 0x00000000 0x00400000 0x00400000 0x00000000
0x00000000

\[ D_2: \]
0x00000000 0x00000004 0x00000000 0x00000004 0x00000000
0x00000004 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000

\[ T: \]
0x0000007c 0x00000000 0x00000000 0x00000000 0x00000000

\[ Q_1[0]: \]
0xc7777a02 0x4affb4ec 0xe246ecfb 0x8a184904 0x63d257f6

\[ M: \]
0xff725191 0xf85050c3 0x6076def2 0x8e86d6c2 0x158d7a79
0x6220da96 0x998f05a6 0xc0a7591a 0x0feaaafdec 0xd0ef7a21
0x3cc64d89 0x55b76464 0x5e0611c2 0xc226ac03 0x00000000
0x000003bf
$N = 60, n_1 = 20, n_2 = 34, n_3 = 6, d = 4$

\begin{align*}
\text{D}_1: & \quad 0x00010000 \ 0x00010000 \ 0x00008000 \ 0x00000000 \ 0x00010000 \\
& \quad 0x00010000 \ 0x00008000 \ 0x00000000 \ 0x00000000 \ 0x00010000 \\
& \quad 0x00010000 \ 0x00008000 \ 0x00000000 \ 0x00000000 \ 0x00010000 \\
& \quad 0x00000000
\end{align*}

\begin{align*}
\text{D}_2: & \quad 0x00000000 \ 0x00000002 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
& \quad 0x00000002 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
& \quad 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
& \quad 0x00000000
\end{align*}

\begin{align*}
\text{T}: & \quad 0x0000003c \ 0x00000000 \ 0x00000000 \ 0x00000000 \ 0x00000000 \\
\text{Q}_1[0]: & \quad 0x1cea526c \ 0xeecf2c58 \ 0xdb83824f \ 0x0383abe3 \ 0xa00c961e
\end{align*}

\begin{align*}
\text{M}: & \quad 0x05c7f6fc \ 0x9287f2ac \ 0xb1984927 \ 0x00b33e60 \ 0x1d2d7155 \\
& \quad 0x39213da5 \ 0x0dbd7a04 \ 0x57bfe02e \ 0x6b635b2b \ 0xb48a2cbf \\
& \quad 0xfa4d9d73 \ 0x1abb640f \ 0xfce52de4 \ 0xe2219ee5 \ 0x00000000 \\
& \quad 0x000003bf
Bibliography


[27] Christophe De Cannière. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In Sokratis K. Katsikas,


[43] ECRYPT. The eSTREAM project. www.ecrypt.eu.org/stream/.


Curriculum Vitae

Personal Details

Name: Simon Knellwolf
Address: Brackenweg 8, 5200 Brugg, Switzerland
Nationality: Swiss
Birthday: May 25, 1982

Education

Jul 09 – Jun 12 PhD candidate, ETH Zurich
Oct 06 – Jan 11 Diploma of Didactics of Mathematics, ETH Zurich
Oct 03 – Sep 08 BSc and MSc in Mathematics, EPF Lausanne and ETH Zurich
Feb 02 – Oct 03 Military service
Aug 93 – Dec 01 Matura Type B (linguistic), Solothurn