Cryptanalysis of Ciphers and Protocols

Elad Pinhas Barkan

Technion - Computer Science Department - Ph.D. Thesis PHD-2006-04 - 2006

Cryptanalysis of Ciphers and Protocols

Research Thesis

Submitted in partial fulfillment of the
Requirements for the

Degree of Doctor of Philosophy

Elad Pinhas Barkan

Submitted to the Senate of
the Technion — Israel Institute of Technology
Adar 5766 Haifa March 2006

The research thesis was done under the supervision of Prof. Eli Biham in the Faculty of Computer Science.

It is my privilege to thank Eli Biham for his insightful support that made this work possible, and for bringing me up as a scientist and researcher. I especially acknowledge Eli for his respect and trust, and for providing me with a very high degree of independence. Eli found the golden path among education, rigorousness, and care. His unique ability to quickly communicate anything in a personal (and sometimes playful) way always leaves me with a smile on my face.

I am thankful to Adi Shamir for our fruitful collaboration, for being highly available around the clock (and around the globe), and for his patience and his wisdom. I acknowledge Nathan Keller for his wonderful and helpful curiosity, and for being an amazing brainmaker. It is a pleasure to thank my colleagues at the Technion, Orr Dunkelman and Rafi Chen, for fruitful discussions and for the wonderful time we had together.

I feel that no words can express my deep gratitude to my loving family, which gave me unconditional love, support, and understanding through the better and worse times of my research. Special thanks goes to my future wife Tamar Kashti for listening to all my research ideas during our many speed-walking, for her love, for her good advices and encouragement, and for being a great companion.

The generous financial help of Technion is gratefully acknowledged.

Contents

List of Tables

List of Figures

Abstract

Cryptography is a major enabler of the modern way of life. It provides secure electronic commerce, digital signatures, secure protocols, secure satellite set-top boxes, secure phone calls, electronic voting, and much more. *Cryptanalysis* verifies the promises of the cryptography. For example, we are interested in verifying that encryption algorithms are indeed secure, as well as the protocols in which they are embedded. In many cases, it is better having a system without any security claims than having a poorly designed cipher in a badly designed system, as the user of a system with no security is aware of the fact that it is insecure, while users of a compromised communications system might believe that the system is secure, and trust it with their secrets.

This thesis contains four independent contributions in the field of cryptanalysis. In the first contribution we consider the cipher Rijndael, which was recently chosen as the United-States' Advanced Encryption Standard (AES). Like many other ciphers, Rijndael has constant values that are used during the encryption process. We ask what happens when we replace all the constant in the cipher. We show that such replacements can create many *dual ciphers* which are isomorphic to the original one. Dual ciphers have several possible applications, including insight for cryptanalysis, protection against *side-channel attacks* (such as measuring the power used during the encryption process to recover the encryption key), and finding faster implementations of existing ciphers. As a result of our work, researchers used our dual ciphers to construct a very efficient implementation of Rijndael in hardware.

In the second contribution, we consider the most deployed cellular system — the Global System for Mobile communications (GSM). We present a very practical *ciphertext-only attack* (an attack that can recover the encryption key given just some encrypted information) on encrypted GSM communications that works whenever the "weaker" cipher A5/2 is used. The attack takes less than a second to complete on a personal computer. Then, we adapt the attack to a more complicated and slower passive attack on the stronger cipher A5/1. We also describe a fast attack on networks using A5/1. This attack is an active attack, i.e., the attacker is required to

transmit. We stress that the attack is on the protocol of GSM, and it works whenever the mobile phone supports the weaker cipher A5/2. This attack can also be used to attack the newest and strongest GSM cipher A5/3, by breaking A5/1 or A5/2, and can be used to attack GPRS (an internet-like service on GSM) in a similar way. We have provided early warnings to the GSM authorities about these attacks, and the authorities are working to correct the flaws.

Our third contribution is a new attack that we present on stream ciphers that use Linear Feedback Shift Registers (LFSRs) in a certain way that is called irregular clocking. The attack uses a new method called *conditional estimators* that can overcome some of the cryptanalytic difficulties induced by the irregular clocking. We apply the attack to GSM's A5/1, and achieve the best known-plaintext attack on A5/1 so far. With 1500–2000 frames of known keystream, i.e., about 6.9–9.2 seconds of communication, the attack can find the encryption key within a couple of tens of seconds to a couple of minutes of computation on a personal computer.

Our fourth contribution relates to generic attacks on ciphers, in particular, we prove bounds on cryptanalytic time/memory tradeoffs. In generic attacks, the cipher is treated as a *black-box* function $f : \{0, 1, \ldots, N\} \mapsto \{0, 1, \ldots, N\}$, and the goal of the attack is to invert f on a value y, i.e., to find an x such that $f(x) = y$. Two extreme generic attacks are the exhaustive search attack which goes over all the values x in search for a pre-image of y, and the table lookup attack which uses a huge table that stores for each image y a preimage x. In 1980, Hellman presented the best known cryptanalytic time/memory tradeoff, which can be seen as a compromise between exhaustive search and table lookup. In a time/memory tradeoff, the attacker uses several tables which together consume significantly less memory compared to the table needed for table lookup, but the attack also works in a significantly shorter time than exhaustive search. Since Hellman's discovery, many improvements to time/memory tradeoff followed, including a new scheme from 2003, called the Rainbow scheme, which claims to save a factor two in the worst-case time complexity. In our work, we set a general model for cryptanalytic time/memory tradeoffs, which includes all the existing schemes as special cases. The model is based on a new notion of *stateful random graphs*, in which the evolution of paths depends on a *hidden state*. Through a rigorous combinatorial analysis, we prove an upper bound on the number of images $y = f(x)$ for which f can be inverted using a tradeoff scheme, and derive from it a lower bound on the number of hidden states. These bounds hold with an overwhelming probability over the random choice of the function f. With some additional natural assumptions on the behavior of the *online phase* of the algorithm, we prove a tight lower bound on its worst-case time complexity $T = \Omega(\frac{N^2}{M^2 \ln N})$, where M is the memory complexity. We describe

new variants of existing schemes, including a method that can improve the time complexity of the online phase (by a small factor) by performing deeper analysis during the preprocessing phase.

Chapter 1

Introduction

While historically cryptography was used mostly in military settings, it has become a leading enabler in everyday modern life. From secure purchasing over the Internet to digital satellite decoders, from encrypting cellular conversations to automatic toll collection, from smart cards to electronic voting, cryptography is an essential component.

Cryptanalysis is the science that evaluates the promises of cryptography: When we make a purchase over the Internet, we would like our transaction to be "secure". We wish that attackers would not be able to tap to our personal information, change the results of electronic voting without being detected, make fraudulent calls on our account, etc. Cryptanalysis focuses in evaluating the strength of cryptographic primitives and protocols.

It is commonly believed that the fate of entire nations was affected by cryptography. One example is the German Enigma machine. It started as a commercial cipher, and later continuously improved and used by the German army. The Polish broke the Enigma in the 1930's, and improved their methods side by side with the German improvement of the Enigma. A few weeks before the break of World War II the Polish transferred their knowledge to the French and to the British. The British further improved the Polish methods, and created a huge intelligence organization of deciphering German encrypted communications. Later, the British shared their information with the US forces. Although the French had an active military unit for decrypting German Enigma communications even during the German occupation, and although the British used decrypted information in battles and to sink German submarines, the fact that Enigma could be broken did not leak to the Germans. Many believe that the allies' ability to decrypt German communication had an overwhelming role in the result of the war.

Cryptography has greatly evolved since. The Data Encryption Standard [62] (DES) is probably the best known and studied modern cipher. It was designed in the 1970's, adopted as a standard in 1977, and later became the most widely used cryptosystem. Although it has short keys of 56 bits, and although there are several attacks on DES, it is still used today in some applications. It is the first modern cipher that was designed as a standard, with open specifications, and was widely accepted.[1]

Shortly after DES was published (and just before it was adopted as a standard) the complementation property of DES was discovered, namely, given the ciphertext T of a plaintext P, encrypted under a key K, i.e., $T = DES_K(P)$, we know that $\bar{T} = DES_{\bar{K}}\bar{P}$, where $\bar{X}$ denotes the 1-complement of X. This property can be used to reduce the complexity of exhaustive search by a factor of two.

The first attack on DES was a generic method of cryptanalysis, called a *cryptanalytic time/memory tradeoff*, and it was introduced by Hellman [48] in 1980. The basic idea of the attack is to choose a fixed plaintext P and treat the function $f(x) = DES_x(P)$ from the key to the ciphertext as a random function. Success in inverting f is equivalent to finding the secret key. In a preprocessing phase the whole key space is explored, and relevant data is stored in many tables (each table covers only a small fraction of the images of f). In the inversion (online) phase the ciphertext is processed, and the tables are searched in order to invert f on the given ciphertext. As the ciphertext must be the encryption of the fixed plaintext P, this method is generally considered a chosen plaintext attack. However, in many settings it can be applied as a known-plaintext attack or a ciphertext-only attack, e.g., when a fixed message is expected to be encrypted (like "login:"). The tradeoff curve of Hellman's time/memory tradeoff is $\sqrt{T}M = N$, where M is the memory complexity (which corresponds to the total number of rows in the tables), T is the time complexity of the inversion phase, and N is the size of the key space. Hellman's method was improved over the years. With an additional idea due to Rivest, the number of memory accesses in the inversion phase can be reduced to $\sqrt{T}$ (from T in Hellman's original method). Golic [46] and Babbage [5] independently discovered that a better time/memory tradeoff exists for stream ciphers. Later, Biryukov and Shamir [20] presented an improved time/memory/data tradeoff for stream ciphers reaching a tradeoff curve of $TM^2D^2 = N^2$, where D is the data that is available for the attacker. Recently, Oechslin [67] presented a new scheme for cryptanalytic time/memory tradeoff, whose tradeoff curve is $\sqrt{2T}M = N$.

[1] Not all standards set their cipher specification open. For example, the internal design of many of the encryption algorithms of the GSM cellular standard, which was designed in the late 1980's, were never officially published.

A major development in cryptanalysis of stream ciphers occurred in 1985, when Siegenthaler [75] introduced correlation attacks on stream ciphers (which at that time were commonly based on Linear Feedback Shift Registers — LFSRs). Siegenthaler observed that there is a correlation between the output of the cipher and the internal state of the registers. If the cipher is poorly designed (and most stream ciphers of the time were in fact poorly designed), an attacker can reconstruct some of the internal state of the cipher given enough bits of the output of the cipher. In the following years, there were many improvements to correlation attacks, which reduced the time complexities of the attacks, and extended them to situations where the basic attack could not work.

In 1990, Biham and Shamir introduced *differential cryptanalysis* [16], which marked a breakthrough in the cryptanalysis of block ciphers and hash functions. Differential cryptanalysis was the first general method of analysis of block ciphers that could, in principle, be applied to any iterated block cipher (although the resulting attack might be worse than exhaustive search). Being a general framework for an attack, differential cryptanalysis revolutionized the science of cryptanalysis. When applied to DES, differential cryptanalysis reduces the complexity of key recovery of DES to an equivalent of 2^{37} encryptions, given 2^{47} chosen plaintexts.

In 1993, Matsui presented *linear cryptanalysis* [57]. Unlike differential cryptanalysis, which is in its core a chosen-plaintext attack, linear cryptanalysis is a known plaintext attack. Similarly to differential cryptanalysis, it is a statistical method that can be applied, in principle, to any cipher. When applied to DES, it can recover a DES key given only 2^{43} known plaintexts.

Although theoretically DES is considered "broken", differential and linear attacks require a considerable amount of plaintext and ciphertext pairs. The Internet gave a new chance for a collaborative work. RSA Security published a series of "DES Challenges": each contains a plaintext and its DES encryption under a secret key, and offered a prize for the first person to recover each secret key. A collaborative effort of tens of thousands of computers was formed over the Internet to solve the challenges. Later, the Electronic Frontier Foundation developed a US$ 210,000 DES cracking machine, specially designed to perform exhaustive search. In 1998 it recovered a DES key in 56 hours. Consequently, *Triple-DES*, which encrypts a plaintext three times under three different keys, replaced DES as the de-facto standard. However, the need for a new encryption standard was already clear.

In 1997 NIST initiated an open contest [63] for the Advanced Encryption Standard — the AES. The intent was to choose a block cipher which will be secure well into the 21^{st} century. The requirements were a secure block cipher with 128-bit block size, and key sizes of 128, 196, and 256 bits. The algorithms for encryption and de-

cryption had to be efficient both in software and in hardware on various devices. Five ciphers made it to the second round of the contest: MARS [27], RC6 [69], Rijndael [65], Serpent [3], and Twofish [73]. Rijndael was selected in October 2000 as the AES, and was officially declared the AES [65] in November 2001.

Rijndael was suggested by a research group that studied and promoted ciphers that operate over the algebraic Galois field $GF(2^n)$. Using this kind of ciphers is now a trend. The researchers' motivations for using this kind of ciphers were computational efficiency and mathematical simplicity: Modern computers process byte operations very fast, while bit operations require more time. Thus, by representing an element of $GF(2^8)$ as a byte, these ciphers are very fast in software. In addition, the non-linear component of the cipher is often chosen to be the multiplicative inversion in $GF(2^8)$, as it has simple mathematical representation and optimal resistance to linear and differential cryptanalysis. The drawback is that the resulting cipher has a clear and simple algebraic structure. Algebraic structures may be used to develop cryptographic attacks that exploit the simple algebraic description of the cipher. Rijndael, being algebraic in nature, and being designed to resist linear and differential cryptanalysis, has motivated new kinds of algebraic attacks.

In recent years there has been an increasing interest in algebraic attacks, both in developing algorithms to efficiently solve the kind of systems of equations that arise in cryptology, and also in developing attacks against specific cryptosystems. In 1999 Kipnis and Shamir introduced the *relinearization* [52] algorithm, which is focused at solving overdefined systems of quadratic equations, and used it in an attempt to attack the HFE public key cryptosystem. Later Courtois, Klimov, Patarin, and Shamir presented the *XL* [30] algorithm that can be seen as an improvement of relinearization. In 2002 Courtois and Pieprzyk developed the *XSL* [31] algorithm, which is focused at solving sparse systems, in an attempt to attack block ciphers in general, and Rijndael in particular. They claim that XSL can attack Rijndael faster than exhaustive search. However, it appears difficult to estimate XSL's time complexity, so its time complexity remains in debate. In 2003 Courtois and Meier successfully mounted an algebraic attack against the stream cipher Toyocrypt [29] in a time complexity of about 2^{49} CPU cycles, given 20 Kilobytes of keystream. In spite of these advances, there is no known attack on Rijndael that can provably work in a time faster than exhaustive key search.

Our motivation is to increase the knowledge and understanding of cryptanalysis. Our main focus in this thesis is set on evaluating the strength of symmetric cryptographic primitives, and the way these primitives are embedded in communication protocols. In particular, we study structures of ciphers and the protocols in which they are embedded, and develop methods to exploit these structures for cryptanaly-

sis. The lessons learned from this thesis can serve designers in their mission to build stronger systems and to improve the security of existing ones.

In our first contribution, we research into the importance of the specific choice of constants in a cipher. We take Rijndael as an example, and ask what happens if we replace all the constants in Rijndael (including the irreducible polynomial, the coefficients of the MixColumns operation, the affine transformation in the S box, etc). We show that such replacements can create new *dual ciphers*, which are *isomorphic* to the original cipher. We present several such dual ciphers of Rijndael, such as the square of Rijndael, and dual ciphers with the irreducible polynomial replaced by primitive polynomials. We also describe another family of dual ciphers consisting of the logarithms of Rijndael. Then, we discuss self-dual ciphers, and show that they can be attacked in a time faster than exhaustive search. We conclude this contribution by discussing possible applications of dual ciphers, including insight for cryptanalysis, protection against side-channel attacks, and finding faster implementations of existing ciphers. As a result of our work, [78] used our dual ciphers to construct a very efficient implementation of Rijndael in hardware.

In our second contribution, we present a very practical ciphertext-only cryptanalysis of communication encrypted in the most deployed cellular technology — GSM (Global System for Mobile communication), and various active attacks on the GSM protocols. These attacks can even break into GSM networks that use "unbreakable" ciphers. We first describe a ciphertext-only attack on A5/2 which is the "weak" cipher of GSM A5/2. The attack is an algebraic attack in its nature, and given a few dozen milliseconds of encrypted off-the-air cellular conversation, it finds the correct key in less than a second on a personal computer. We extend this attack to a (much more complex) ciphertext-only attack on the stronger A5/1 cipher. We then describe new (active) attacks on the protocols of networks that use A5/1, the newest GSM cipher A5/3, or even the GPRS cipher (General Packet Radio Service, which is a technology for implementing internet connectivity over GSM). These attacks exploit flaws in the GSM protocols, and they work whenever the mobile phone supports a weak cipher such as A5/2. We emphasize that these attacks are on the protocols, and are thus applicable whenever the cellular phone supports a weak cipher, for example, they are also applicable for attacking A5/3 networks using the cryptanalysis of the weaker A5/1. Unlike previous attacks on GSM that require unrealistic information, like long known plaintext periods, our attacks are very practical and do not require any knowledge of the content of the conversation. Furthermore, we describe how to fortify the attacks to withstand reception errors. As a result, our attacks allow attackers to tap conversations and decrypt them either in real-time, or at any later time. We discuss several attack scenarios such as call hijacking, altering of

data messages and call theft. We have warned the GSM authorities of the securities flaws prior to the publication, and they are changing the way GSM works in order to overcome these flaws.

In our third contribution, we research into irregularly-clocked linear feedback shift registers (LFSRs), which are commonly used in stream ciphers. The irregular clocking of the LFSRs causes an obfuscation effect by hiding the clockings of the registers. We present a new attack method called *conditional estimators*, and harness their cryptanalytic strength to mount correlation attacks on these ciphers. Conditional estimators compensate for some of the obfuscating effects of the irregular clocking, resulting in a correlation with a considerably higher bias. On GSM's cipher A5/1, a factor two is gained in the correlation bias compared to previous correlation attacks. We mount an attack on A5/1 using conditional estimators and using three weaknesses that we observe in one of A5/1's LFSRs (known as $R2$). The weaknesses imply a new criterion that should be taken into account by cipher designers. Given 1500–2000 known-frames (about 6.9–9.2 conversation seconds of known keystream), our attack completes within a few tens of seconds to a few minutes on a personal computer, with a success rate of about 91%. To complete our attack, we present a source of known-keystream in GSM that can provide the keystream for our attack out of 3–4 minutes of GSM ciphertext, thus transforming our attack to a ciphertext-only attack.

In our fourth contribution, we formally define a general model of cryptanalytic time/memory tradeoffs for the inversion of a random function $f : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$. The model contains all the known tradeoff techniques as special cases. It is based on a new notion of *stateful random graphs*. The evolution of paths in the stateful random graph depends on a *hidden state* such as the color in the Rainbow scheme or the table number in the classical Hellman scheme. We prove an upper bound on the number of images $y = f(x)$ for which f can be inverted using a tradeoff scheme with S hidden states, and derive from it a lower bound on the number of hidden states. These bounds hold with an overwhelming probability over the random choice of the function f, and their proofs are based on a rigorous combinatorial analysis. With some additional natural assumptions on the behavior of the *online phase* of the algorithm, we prove a lower bound on its worst-case time complexity $T = \Omega(\frac{N^2}{M^2 \ln N})$, where M is the memory complexity. We describe several new variants of existing schemes, including a method that can improve the time complexity of the online phase (by a small factor) by performing deeper analysis during the preprocessing phase, and adaptations of the Rainbow scheme to time/memory/data tradeoffs.

The four independent contributions are detailed in the following four chapters,

where each chapter is self contained. For completeness and general background, we include a short introductions to some of the modern methods of cryptanalysis in Appendix A.

Chapter 2

In How Many Ways Can You Write Rijndael?

In this chapter, we ask the question what happens if we replace all the constants in Rijndael, including the irreducible polynomial, the coefficients of the MixColumns operation, the affine transformation in the S box, etc. We show that such replacements can create new *dual ciphers*, which are *isomorphic* to the original cipher. We present several such dual ciphers of Rijndael, such as the square of Rijndael, and dual ciphers with the irreducible polynomial replaced by primitive polynomials. We also describe another family of dual ciphers consisting of the logarithms of Rijndael. Then, we discuss self-dual ciphers, and show that they can be attacked in time faster than exhaustive search. Finally, we discuss possible applications for dual ciphers, including insight for cryptanalysis, protection against side-channel attacks, and finding faster implementations of existing ciphers.

The work described in this chapter is a joint work with Prof. Eli Biham. It was originally published in [7].

2.1 Introduction

In 2000, the cipher Rijndael [33] was selected as the Advanced Encryption Standard (AES) [65]. Rijndael was designed to withstand known attacks such as differential [16] and linear [57] attacks. The cipher structure is (mostly) specified in terms of algebraic operations of the Galois field $GF(2^8)$. The motivation behind this structure is computational efficiency, as $GF(2^8)$ elements can be represented by bytes, which can be very efficiently processed by modern computers, unlike bit-level operations that are usually more expensive in computer power. The drawback is

that algebraic structures are inherited from the simple $GF(2^8)$ operations (see for example [42, 77]), and the fear is that cryptographic attacks exploiting these elegant algebraic structures would be developed. An example for such attacks are interpolation attacks [50]. In an attempt to avoid some of these drawbacks, other mechanisms are introduced to Rijndael, such as $GF(2)$ affine transformations. However, an affine transformation can be expressed as a (linear) polynomial over $GF(2^8)$, thus, all the operations of Rijndael are expressed in $GF(2^8)$ (Rijndael with this representation is called Rijndael-GF [33, Appendix A.5]). Newer attacks [31] claim to be successful at breaking Rijndael, however, the complexity of these attacks is not well understood.

Rijndael's operations, like most secret-key ciphers, involve constants. These constants include the irreducible polynomial over which $GF(2^8)$ multiplications are performed, coefficients in the MixColumns operation, the affine transformation in the S box, etc. The choice of the specific constants raises some natural questions for the cryptanalyst: does this choice of constants provides the highest level of security? Is there another choice of constants that provides the same or higher level of security? Does the choice of constants have any relevance to security of the cipher, i.e., is there a choice of constants that provides a lower level of security?

In this chapter we ask the question what happens if we replace all the constants in Rijndael, including the irreducible polynomial, the coefficients of the MixColumns operation, the affine transformation in the S box, etc. We show that such replacements can create new *dual ciphers*, which are *isomorphic* to Rijndael. Although in the dual ciphers the intermediate values during encryption are different than Rijndael's, we show that they are *isomorphic* to Rijndael. Examples of such ciphers include ciphers with a primitive polynomial (replacing the irreducible polynomial of Rijndael), the cipher *Square of Rijndael* that encrypts the square of the plaintext under the square of the key to the square of the ciphertext, and a cipher with a triangular affine matrix in the S box.

The following definition stands in the center of this chapter:

Definition 1 *Two ciphers E and E' are called Dual Ciphers, if they are isomorphic, i.e., if there exist invertible transformations $h_k(\cdot)$, $h_p(\cdot)$ and $h_c(\cdot)$ such that*

$$\forall P, K \qquad h_c(E_K(P)) = {E'}_{h_k(K)}(h_p(P)).$$

Trivial dual ciphers are very easy to find for all ciphers. For example, every cipher is dual to itself with the identity transformations. Also, for any cipher, the addition of non-cryptographic invertible initial and final transformations creates a trivial dual cipher. We are not interested in these kinds of dual ciphers. The interesting question is whether there exist non-trivial dual ciphers of widely-used or well-known ciphers.

We define a class of ciphers, which includes many stream ciphers and block ciphers. We call the ciphers in the class $EGF(2^8)$ ciphers. Examples of ciphers in the class are: Shark [23], Square [32], Scream [47], Crypton [56], Anubis [11], and Khazad [12]. Rijndael (AES) [33], which is the focus of this chapter, is also included in this class. We show that any cipher in the class has a family of at least 239 dual ciphers in addition to the original cipher. These dual ciphers can be generated by careful replacements of the constants in the cipher. We note, however, that the existence of dual ciphers for a specific cipher does not necessarily mean that the security of the cipher is compromised, and in fact, dual ciphers can be used to strengthen the cipher against side-channel attacks.

We present another family of *Log Dual Ciphers* for $EGF(2^8)$ ciphers. In a log dual cipher, the logarithm of the plaintext is encrypted by the logarithm of the key to the logarithm of the ciphertext. We show that Rijndael has a family of log dual ciphers.

An interesting extension of dual ciphers, are semi-dual ciphers:

Definition 2 *A cipher E' is called a semi-dual cipher of E, if there exist transformations $h_k(\cdot)$, $h_p(\cdot)$ and $h_c(\cdot)$ such that*

$$\forall P, K \qquad h_c(E_K(P)) = E'_{h_k(K)}(h_p(P)).$$

where h_k,h_p and h_c are not necessarily invertible (and even not necessarily length-preserving).

Semi-dual ciphers potentially reduce the plaintext, the ciphertext, and the key spaces, and thus may allow to develop efficient attacks on their original cipher.

A special case of dual ciphers is the case of self-duality, i.e., the case where a cipher is a (non-trivial) dual of itself. We study this case and show that such ciphers can be attacked faster than exhaustive search. We discuss what change in the constants of Rijndael would lead to a self-dual cipher. In the context of self-dual ciphers, it is interesting to mention that RSA [70] is an example of a self-dual public key cipher. Let e and n be the RSA public key, and let $c = m^e \pmod n$, where m is the plaintext and c is the ciphertext. Then it follows that RSA is a dual of itself, e.g., $c^3 = (m^3)^e \pmod n$. Another known example of self-duality, is the complementation property of DES, i.e., $\overline{DES_k(p)} = DES_{\overline{k}}(\overline{p})$, where k is the key, p is the plaintext, and $\overline{x}$ denotes the 1-complement of x.

We indicate a variety of possible applications for dual ciphers: On the analysis side, they might provide insight to new attacks; on the protection side, they could protect against side-channel attacks; and on the implementation side, dual ciphers could be used to find more efficient implementations of existing ciphers.

This chapter is organized as follows: We begin with a short description of Rijndael, which is given in Section 2.2. In Section 2.3, we present the square dual cipher. Then, we change the irreducible polynomial in Section 2.4, and show the family of 240 dual ciphers for each $EGF(2^8)$ cipher. Section 2.5 shows how to define log dual ciphers. We discuss the special case of self-duality, and show how to mount an attack on self-dual ciphers in Section 2.6. We discuss applications of dual cipher in Section 2.7. The chapter is summarized in Section 2.8. Appendix 2.9 gives the affine matrix of the square of Rijndael. In Appendix 2.10 we describe the relation between dual ciphers of different constants but with the same irreducible polynomial, while in Appendix 2.11 we describe the relation where the irreducible polynomial is also replaced. Appendix 2.12 details properties related to log dual ciphers. Some of the details of the attack on self-dual ciphers are detailed in Appendix 2.13.

2.2 Description of Rijndael

In this section we give a short description of Rijndael. For a full description of Rijndael the reader may consult [33]. The AES [65] consists of Rijndael with 128-bit blocks, and three key sizes of 128, 192 and 256 bits. For simplicity, in the rest of this section we describe Rijndael with a key length of 128 bits (though our results hold for all variants of Rijndael). The 128-bit blocks are viewed as either 16 bytes or as four 32-bit words. The bytes are organized in a square form:

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

where b_i notes the i'th byte of the block.

Each column in this representation can be viewed as a 4-byte word. Rijndael has operations that work on columns, operations that work on rows, and operations that work on each byte separately. The combination of these operations ensures a complete mixture of all data bits after several rounds, i.e., every input bit affects every output bit (note however, that this property does not necessarily mean that the cipher is secure).

Rijndael encryption is performed as follows: The plaintext is XORed with an initial subkey (via the AddRoundKey operation mentioned later) to form the input to the first round. Then, ten rounds are performed. Their final output is the ciphertext.

The round function is composed of 4 consecutive operations:

1. SubBytes: An S box is applied in parallel to each of the 16 bytes of the data.
2. ShiftRows: Byte-wise rotation of bytes in each row.
3. MixColumns: Every column is mixed by a linear operation. MixColumns is not perform in the last round.
4. AddRoundKey: The data is XORed with a 128-bit subkey.

The S box of Rijndael first calculates the multiplicative inverse of the input in $GF(2^8)$ (modulo the irreducible polynomial of Rijndael $z^8 + z^4 + z^3 + z + 1$, which is denoted in binary notation by $11B_x$ (the coefficient of the polynomial in a binary notation); for the purpose of inversion the inverse of 00_x is defined to be 00_x). The resulting inverse x is transformed by the affine transformation to produce the output y:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

where the x_i's and the y_i's are coefficients of x and y (i.e., the bits of the bytes), and x_0 and y_0 are the least significant bits.

The ShiftRows operation is a byte-wise rotation of the bytes as as follows:

- Leaving the first first row unchanged.
- Shifting the second row by one byte to the left (cyclically).
- Shifting the third row by two bytes to the left (cyclically).
- Shifting the fourth row by three bytes to the left (cyclically).

Taking the square form as the input of the ShiftRows operation, the ShiftRows operation has the following effect:

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

$\xrightarrow[\text{ShiftRows}]{}$

b_0	b_4	b_8	b_{12}
b_5	b_9	b_{13}	b_1
b_{10}	b_{14}	b_2	b_6
b_{15}	b_3	b_7	b_{11}

The MixColumns operation mixes each column independently. The new state is defined by

$$\begin{array}{|c|c|c|c|} \hline b_0 & b_4 & b_8 & b_{12} \\ \hline b_1 & b_5 & b_9 & b_{13} \\ \hline b_2 & b_6 & b_{10} & b_{14} \\ \hline b_3 & b_7 & b_{11} & b_{15} \\ \hline \end{array} \xrightarrow[\text{MixColumns}]{} \begin{array}{|c|c|c|c|} \hline b'_0 & b'_4 & b'_8 & b'_{12} \\ \hline b'_1 & b'_5 & b'_9 & b'_{13} \\ \hline b'_2 & b'_6 & b'_{10} & b'_{14} \\ \hline b'_3 & b'_7 & b'_{11} & b'_{15} \\ \hline \end{array}$$

where each column $(b_i, b_{i+1}, b_{i+2}, b_{i+3})$, $i \in \{0, 4, 8, 12\}$ is mixed by:

$$\begin{pmatrix} b'_i \\ b'_{i+1} \\ b'_{i+2} \\ b'_{i+3} \end{pmatrix} = \begin{pmatrix} 02_x & 03_x & 01_x & 01_x \\ 01_x & 02_x & 03_x & 01_x \\ 01_x & 01_x & 02_x & 03_x \\ 03_x & 01_x & 01_x & 02_x \end{pmatrix} \begin{pmatrix} b_i \\ b_{i+1} \\ b_{i+2} \\ b_{i+3} \end{pmatrix}.$$

Note that the multiplication and additions are performed over $GF(2^8)$. A mixing of a column can also be seen as a multiplication of the column by the polynomial $c(x) = 03_x x^3 + 01_x x^2 + 01_x x + 02_x$ in $GF(2^8)^4$ modulo the polynomial $x^4 + 1$.

The AddRoundKey simply XORs the 128-bit subkey to the data. The subkey is generated by the key expansion.

When the key size is 128 bits the round-function is repeated 10 times. The number of rounds is higher when longer keys or blocks are used: there are 12 rounds if the key or block size is 192 bits, and 14 rounds if the key or block size is 256 bits.

The key expansion of Rijndael generates the subkeys from the key using a blend of the above operations, and using the round constants $Rcon[i] = (02_x)^{i-1}$ (i starts at 1). The input to the key expansion is the 128-bit key, and the output are the eleven 128-bit subkeys. The first subkey K_0 is equal to the key. Each one of the rest of the subkeys K_i is defined as a function of the previous round's subkey K_{i-1} and the round number $i \in \{1, \ldots, 10\}$. Let $K_{i,j}$ be the j^{th} byte of the subkey of round i, $j \in \{0, \ldots, 15\}$. Then, $K_{i,0} = Rcon[i] \oplus S[K_{i-1,13}]$, $K_{i,1} = S[K_{i-1,14}]$, $K_{i,2} = S[K_{i-1,15}]$, $K_{i,3} = S[K_{i-1,12}]$. For every $j > 3$, $K_{i,j} = K_{i-1,j} \oplus K_{i,j-4}$. The key schedule is slightly different for the 192-bit and 256-bit keys, although it follows the same operations.

2.3 Square Dual Ciphers

We begin with defining the $EGF(2^8)$ class of ciphers.

Definition 3 *Consider the operations:*

- *Basic field operations in $GF(2^8)$:*
  1. *Addition (i.e., XOR: $f(x,y) = x \oplus y$).*
  2. *XOR with a constant (e.g., $f(x) = x \oplus 3F_x$).*
  3. *Multiplication ($f(x,y) = x \cdot y$).*
  4. *Multiply by a constant (e.g., $f(x) = 03_x \cdot x$).*
  5. *Raise to any power (i.e., $f(x) = x^c$, for any integer c). This includes the inverse of x: x^{-1}.*
  6. *Any replacement of the order of elements (e.g., taking a vector containing the elements $[a,b,c,d]$, and changing the order to $[d,c,a,b]$).*
- *Complex (with respect to $GF(2^8)$) 8-bit operations:*
  7. *Linear transformations $f(x) = Ax$, for any boolean matrix A.*
  8. *Any unary operation over elements in $GF(2^8)$. (i.e., a look-up table, $S(x) = LookUpTable[x]$ or $F(x) : \{0,1\}^8 \longrightarrow \{0,1\}^8$).*

We call these operations $EGF(2^8)$ operations. If a cipher is specified in terms of operations in $EGF(2^8)$ we call it a cipher in $EGF(2^8)$, or an $EGF(2^8)$ cipher.

Note that our notation implies that in item 7 of Definition 3, the variable x, which is an element in $GF(2^8)$, is converted to an 8-bit vector (in $GF(2)^8$) before being multiplied by the matrix A. The result is converted back to be an element of $GF(2^8)$. It should be noted that since XOR with a constant is also allowed in item 2, any affine transformation is included in the operations we consider (i.e., $F(x) = Ax \oplus b$).

It is important to understand that any operation covered by item 7 or item 8 can be expressed as a polynomial in $GF(2^8)$ (thus, "covered" by previous operations). In fact, practically all ciphers can be translated to operations in $EGF(2^8)$, but the resulting specification would be unnatural and complex. As our main motivation is to gain insight to the specific design of the cipher and the choice of constants in the cipher, we limit our discussion to ciphers specified only in terms of the above operations (rather than equivalent representations that result by translating the cipher's operations to $EGF(2^8)$ operations).

We now show the existence of square dual ciphers. Given a cipher E that uses only operations of $EGF(2^8)$, we define the cipher E^2 by modifying the constants of E. In the terms of Definition 1, we set $h_k(x) = h_p(x) = h_c(x) = x^2$, where x^2 is squaring each byte of x, independently, in $GF(2^8)$. The notation K^2, and P^2

denote the square operation of each byte of K and P (and similarly for any other byte vector). We define E^2 such that $E^2_{K^2}(P^2) = (E_K(P))^2$.

All the operations that do not involve constants remain unchanged. There are only four operations that involve constants:

1. $f(x) = c \cdot x$.
2. $f(x) = c \oplus x$.
3. $f(x) = Ax$, where A is a constant matrix.
4. $S(x) = LookUpTable[x]$, where the look-up table is constant.

In the first two operations we change the constant c in E to be c^2 in E^2, where c^2 is the result of squaring c in $GF(2^8)$. In the affine transformation, A is replaced by QAQ^{-1}, where in the case of Rijndael Q and Q^{-1} are:

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \qquad Q^{-1} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad (2.1)$$

We show later that given an element x the value of Qx is x^2, i.e., multiplying by the boolean matrix Q is actually squaring.

From now on we denote QAQ^{-1} by A^2, as for any x, $QAQ^{-1}x^2 = QAx = (Ax)^2$. A^2 of Rijndael is given in Appendix 2.9. The matrices Q and Q^{-1} depend on the irreducible polynomial of $GF(2^8)$. The matrices above suit Rijndael's irreducible polynomial $z^8 + z^4 + z^3 + z + 1$.

Finally, we replace look-up tables of the form $S(x)$ with $S^2(x)$, where $S^2(x)$ is defined as $S^2(x) = QS(Q^{-1}x)$.

Remark: To make it clear, in our notation, E^2 is not $E(E(\cdot))$ nor $(E(\cdot))^2$, A^2 is not the matrix A multiplied with itself, and $S^2(x)$ is not $(S(x))^2$, nor $S(S(x))$.

We can now define the dual cipher E^2 of a cipher E: we take the specifications of the cipher E, raise all the constants in the cipher to their second power, replace matrices A by $A^2 = QAQ^{-1}$, and replace look-up tables $S(x)$ by $S^2(x) = QS(Q^{-1}x)$. If we take Rijndael as an example of E, the polynomial $03_x x^3 + 01_x x^2 + 01_x x + 02_x$ of the mix column operation is replaced by $05_x x^3 + 01_x x^2 + 01_x x + 04_x$.[1] The affine

[1] In $GF(2^8)$, $03_x^2 = 05_x$.

transformation $Ax+b$ is replaced by the affine transformation $A^2x+b^2 = QAQ^{-1}x+b^2$.

The key expansion consists of S boxes, XORs, and XORs with constants in $GF(2^8)$ (called *Rcon*) which are powers of 02_x. These operations are replaced by the replacement operations as mentioned above, with the *Rcon* constants being replaced by their squares.

We now show that E and E^2 are dual ciphers, with $f(x) = x^2$:

Theorem 1 *For any K and P, ${E^2}_{K^2}(P^2) = (E_K(P))^2$.*

This theorem states that if P is the plaintext, K is the key and the result of encryption with cipher E is C, then the result of encrypting P^2 under the key K^2 with the cipher E^2 is necessarily C^2.

Proof Any Galois field is congruent to a Galois field of the form of $GF(q^m)$, where q is a prime. The number q is called the characteristic of the field. It is well known that for any $a, b \in GF(q^m)$ it follows that: $(a+b)^q = a^q+b^q$. In $GF(2^8)$: $(a+b)^2 = a^2+b^2$. That actually means that squaring an element in $GF(2^8)$ is a linear operation, which can be applied by a multiplication by a binary matrix Q of size 8×8. Eq. (2.1) gives the Q matrix of Rijndael. It follows that Q^{-1} is the matrix that takes out the square root of an element in $GF(2^8)$. In Appendix 2.10, we give a brief proof of the fact that $(a + b)^2 = a^2 + b^2$ implies that multiplication can be performed by a multiplication by a matrix Q, and we show how to compute Q in other representations and other Galois fields.

To complete the proof of the theorem, it suffices to show that for each operation $f(\cdot)$ in E, the corresponding operation $f^2(\cdot)$ in E^2 satisfies $f^2(x^2) = (f(x))^2$:

1. $f(x, y) = x \oplus y$. In this case $f^2(x^2, y^2) = x^2 \oplus y^2 = (x \oplus y)^2 = (f(x, y))^2$.
2. $f(x) = x \oplus c$. By definition $f^2(x^2) = x^2 \oplus c^2 = (x \oplus c)^2 = (f(x))^2$.
3. $f(x, y) = x \cdot y$. In this case $f^2(x^2, y^2) = x^2 \cdot y^2 = (x \cdot y)^2 = (f(x, y))^2$.
4. $f(x) = x \cdot c$. By definition $f^2(x^2) = x^2 \cdot c^2 = (x \cdot c)^2 = (f(x))^2$.
5. $f(x) = x^c$. In this case $f^2(x^2) = (x^2)^c = (x^c)^2 = (f(x))^2$.
6. It is clear that replacing the order of elements after they are raised to their second power is equal to raising elements to their second power, and then replacing their order.

7. $f(x) = Ax$. By definition $f^2(x^2) = A^2x^2 = QAQ^{-1}x^2 = QAx = (Ax)^2 = (f(x))^2$, as Q is the matrix which corresponds to the squaring operation in $GF(2^8)$.

8. $f(x) = S(x) = LookUpTable[x]$. By definition

$$f^2(x^2) = S^2(x^2) = QS(Q^{-1}x^2) = QS(x) = (S(x))^2 = (f(x))^2.$$

■

The cipher $E^4 = (E^2)^2$ is a dual cipher of E^2, and thus also of E. Moreover, all ciphers E^{2^i} (for all i), i.e., E, E^2, E^4, E^8, E^{16}, E^{32}, E^{64} and E^{128}, are all dual ciphers of each other (there are 8 such ciphers as $E^{2^8} = E$).

It is interesting to note that Rijndael has these dual ciphers, independently of the key size, the block size, the number of rounds, and even the arrangement of operations in the cipher. These dual ciphers exist for any cipher whose all operations are $EGF(2^8)$ operations.

2.4 Modifying the Polynomial

An $EGF(2^8)$ cipher E can include multiplication modulo an irreducible polynomial. The irreducible polynomial in Rijndael is used for the inverse computation in the S box and also in the multiplications in the MixColumns operation. Several researchers asked why the irreducible polynomial of Rijndael was not selected to be primitive (there are 30 irreducible polynomials of degree 8, of which 16 are primitive, and any one of these 30 polynomial could have been used in Rijndael). We show that it is irrelevant if the irreducible polynomial is primitive or not, due to existence of dual ciphers of Rijndael with any of the above irreducible polynomials.

In Appendix 2.11, we show that replacing the irreducible polynomial creates an isomorphic $GF(2^8)$ field, and that the isomorphism function is linear. We denote this linear function by R. Let x be a binary vector representing an element under Rijndael's irreducible polynomial $g(x)$. The representation of x under another irreducible polynomial $\hat{g}(x)$ is given by $R \cdot x$, where R is an 8×8 binary matrix. In Appendix 2.11, we further show that the matrix R is always of the form $R = (1, a, a^2, a^3, a^5, a^6, a^7)$, where the columns a^i are computed modulo the irreducible polynomial $\hat{g}(x)$.

We define a new cipher E^R using the new irreducible polynomial $\hat{g}(x)$, such that E^R is a dual cipher of E, with $h_k(x) = h_p(x) = h_c(x) = R \cdot x$. We define E^R using the matrix R in the same way that we used the matrix Q to define the square dual

cipher. As a result, the operations in E^R are identical to the operations in E, upto a change of constants. To fully specify E^R replace Q with R in Section 2.3, and replace x^2 with $R \cdot x$. The proof of duality follows.

Note that the Q matrix (Appendix 2.10) is actually a special case of the R matrix, where $\hat{g}(x) = g(x)$. For each irreducible polynomial we can define its eight square dual ciphers. Since there are 30 irreducible polynomials, we get that there are 240 dual ciphers for each $EGF(2^8)$ cipher. As there can be no other R matrices (as the remaining $256 - 240 = 16$ R matrices are singular), it follows that there are exactly 240 dual ciphers for each $EGF(2^8)$ ciphers under our constraints.

For example, we describe one of these 240 dual ciphers of Rijndael: the irreducible polynomial of Rijndael is replaced by the primitive polynomial $z^8 + z^4 + z^3 + z^2 + 1$ (denoted in binary notation by $11D_x$). In this example, the R matrix is

$$R = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad R^{-1} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The inverse matrix R^{-1} takes an element of the dual cipher to Rijndael's representation. It is interesting to note that in this particular example the affine matrix of the S box becomes lower triangular:

$$\hat{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Also, the constant 63_x in the S box becomes 64_x, and the coefficients 03_x, 02_x of the MixColumns operation are interchanged (i.e., to 02_x, 03_x). The coefficients $0B_x$, $0D_x$, 09_x, $0E_x$ are also interchanged in pairs to $0D_x$, $0B_x$, $0E_x$, 09_x. The *Rcon* constants $(02_x)^{i-1}$ are replaced by $(03_x)^{i-1}$. The full description of these 240 dual ciphers of Rijndael can be found in *The Book of Rijndaels* [9].

Due to the existence of a dual cipher with any irreducible polynomial, we conclude that the choice of the irreducible polynomial of Rijndael can be chosen arbitrarily. In particular, there is no advantage to selecting a primitive polynomial over the current polynomial of Rijndael.

2.5 Log Dual Ciphers

In this section we discuss another family of dual ciphers for $EGF(2^8)$ ciphers. We we call this family *log dual ciphers*.

Let g be a generator of the multiplicative group of $GF(2^8)$. Since the cipher works on elements of $GF(2^8)$ we can write any element x as an exponent of g, i.e., $x = g^i$, except for $x = 0$, which we define as $g^{-\infty}$. In a logarithmic notation we write: $\log_g x = i$, where $log_g 0 = -\infty$. In the log cipher we use the logarithm representation of the elements, instead of the polynomial representation used in the original description of the cipher.

Let x and y be elements of $GF(2^8)$, and let $i = \log_g x$, $j = \log_g y$.

We use the notation $E^{\log_g}$, or shortly $E^{\log}$, to denote the log dual cipher. We show that $E^{\log}$ is a dual cipher of E, where $h_k(x) = h_p(x) = h_c(x) = \log_g x$. The log dual cipher is defined by taking the specifications of the cipher, and replacing the following operations:

1. The operation $f(x, y) = x \oplus y$ is replaced by the operation

$$f^{\log}(i, j) = j + T(i - j) \pmod{255} \tag{2.2}$$

   or by

$$f^{\log}(i, j) = i + T(j - i) \pmod{255}, \tag{2.3}$$

   where the Zech logarithm [71] $T(i)$ is defined as $T(i) = \log_g(g^i \oplus 1)$. In cases where $-\infty$ appears in $f^{\log}$, we define $f^{\log}(-\infty, j) = j$, and $f^{\log}(i, -\infty) = i$. Note that an alternative solution for the case that $-\infty$ is an argument is a careful definition of $T(\cdot)$ for cases that involve $-\infty$. This alternative definition preserves consistency with Equations (2.2) and (2.3). $f^{\log}(i, j) = f^{\log}(j, i) = f^{\log}(j, i) = j + T(i - j) \pmod{255}$.

2. The operation $f(x) = x \oplus c$ is replaced by the operation $f^{\log}(i) = k + T(k - i) \pmod{255}$ where $k = \log_g c$.

3. The operation $f(x, y) = x \cdot y$ is replaced by the operation $f^{\log}(i, j) = i + j \pmod{255}$. If either x or y is $-\infty$, then the result is $-\infty$.

4. The operation $f(x) = x \cdot c$ is replaced by the operation $f^{\log}(i) = i + k \pmod{255}$, where $k = \log_g c$.

5. The operation $f(x) = x^m$ is replaced by the operation $f^{\log}(i) = i \cdot m \pmod{255}$. If $i = -\infty$ then the result is $-\infty$.

6. Changing the arrangement of elements.

7. The operation $S(x) = LookUpTable[x]$ is replaced by the operation $S^{\log}(i) = \log_g(S(g^i))$.

8. The linear transformation $L(x) = Ax$ is written as a polynomial $\sum a_i \cdot x^{2^i}$, and treated as a combination of exponentiations, multiplications, and additions.

The following theorem suggests that if P is the plaintext, K is the key, and C is the result of encrypting P under the key K with cipher E, then the result of encrypting $\log_g(P)$ under the key $\log_g(K)$ with the cipher $E^{\log}$ is necessarily $\log_g(C)$.

Theorem 2 *Let g be a generator in $GF(2^8)$. For any K and P:*

$$E^{\log}_{\log_g K}(\log_g P) = log_g(E_K(P)).$$

In the context of this chapter $\log_g X$ denotes the logarithm of each byte of X, where X is one of P, C, or K.

Proof It suffices to show that for each operation $f(x)$ in E, and the corresponding operation in $E^{\log}$, which we denote by $f^{log}(x)$, it follows that $f^{\log}(\log_g x) = \log_g(f(x))$.

1. $f(x, y) = x \oplus y$. By definition $f^{\log}(i, j) = j + T(i - j) = j + \log_g(g^{i-j} \oplus 1) = \log_g(g^j \cdot (g^{i-j} \oplus 1)) = \log_g(g^i \oplus g^j) = \log_g(x \oplus y) = \log_g(f(x, y))$. The proof is trivial for cases that involve $-\infty$.

2. $f(x) = x \oplus c$, in the same way as the previous item.

3. $f(x, y) = x \cdot y$. In this case $f^{\log}(i, j) = i + j = \log_g(g^{i+j}) = \log_g(x \cdot y) = \log_g(f(x, y))$.

4. $f(x) = x \cdot c$, in the same way as the previous item.

5. $f(x) = x^c$. In this case $f^{\log}(i) = i \cdot c = log_g(x^c) = \log_g(f(x))$.

6. It is clear that changing the arrangement of elements after their log-value is taken is equal to first changing the arrangement of elements and then taking the log-value of the elements.

7. $f(x) = S(x) = LookUpTable[x]$. In this case, by definition of $f^{\log}$ it follows that: $f^{\log}(i) = S^{\log}(i) = \log_g(S(g^i)) = \log_g(S(x)) = \log_g(f(x))$.

8. $L(x) = Ax$. This follows from items 1,2,4, and 5.

The above equations hold also in the case that $-\infty$ is an argument. ■

Note that the non-linear part of the SubBytes transformation of Rijndael in the log dual cipher, i.e., finding the multiplicative inverse of an element, becomes very simple (and linear). This operation is replaced by negation in the log dual cipher:

$$x \longrightarrow i \Leftrightarrow x^{-1} \longrightarrow -i.$$

The T transformation is non-linear. It has interesting properties. Here are some of the properties of the T transformation:

1. $T(x) - T(-x) = x$

2. $T(2x) = 2T(x)$

3. $T(T(x)) = x$

Additional properties of $T(x)$ can be found in Appendix 2.12.

How does the 240 mentioned representations of Rijndael affect the number of log dual ciphers? The group of 240 representations of Rijndael has a single group of 128 log dual ciphers. Choosing a generator g in Rijndael's representation generates the same dual cipher as choosing the generator $R \cdot g$ in another dual cipher. Therefore, the number of log dual cipher is the same as the number of generators, i.e., there are only 128 log dual ciphers.

2.6 Self-Dual Ciphers

We mention that any cipher is trivially dual to itself. However, it is possible to find ciphers that are self-dual in a non-trivial way. One such interesting case of self-dual ciphers can be derived from square dual ciphers. Let E be a square self-dual cipher. It follows that:

$$(E_K(P))^2 = E_{K^2}(P^2),$$

i.e., by encrypting the square of P by the square of K under the cipher, we get the square of the original ciphertext. For that, we require that each constant is the square of itself. In $GF(2^8)$ it means that the constants are either 0 or 1.

We take Rijndael as an example and modify it to become a self-dual cipher. We need to change the constant 63_x in the affine transformation in the S box to either 00_x or 01_x. We also need to change the constants of the MixColumns operation to either 00_x or 01_x. A possible alternative matrix for the mix column operation, whose entries consist of only 00_x's and 01_x's is:

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}.$$

Note that in this case $M^{-1} = M$, but there are other possible matrices for which this is not the case.

In the key expansion we need to change the round constant. Any selection of values from $\{00_x, 01_x\}$ can be made for the *Rcon* constants. There are various such selections that can still prevent related key attacks [14].

We replace the affine transformation by a self-dual one. We can easily find eight affine transformations that are self-squares (i.e., $QAQ^{-1} = A$): The matrix Q (shown in Eq. (2.1)) is the square of itself under our definition, since $Q^2 = Q(Q)Q^{-1} = Q$ (remember that the notation Q^2 is not $Q \cdot Q$, but rather Q^2 is what the matrix Q is transformed to in the square dual cipher). The order of Q is eight, therefore, the following eight transformations are self-square transformations:

1. Q
2. $Q \cdot Q$
3. $Q \cdot Q \cdot Q$
4. $Q \cdot Q \cdot Q \cdot Q$
5. $Q \cdot Q \cdot Q \cdot Q \cdot Q$
6. $Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q$
7. $Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q$
8. $Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q = I$

Notice that all the linear combinations with coefficients from $\{0,1\}$ of these matrices, are also self-squares matrices. Therefore, there are at least 256 such self-square matrices. Detailed analysis shows that these are all the self-square matrices. Of these 256 matrices, only 128 matrices are involutions.

A property of such self-dual ciphers is that if all the bytes of the key and all the bytes of the plaintext are in $\{00_x, 01_x\}$, then so are all the bytes of the ciphertext.

Note that the notion of simple relations, presented by Knudsen in [53] in another context, is related to self-dual ciphers. In fact, the property of a cipher being dual to itself is a simple relation in the terms of [53] (given that $h_k(\cdot)$, $h_p(\cdot)$ and $h_c(\cdot)$ are easy to evaluate).

2.6.1 Higher-Order Self-Dual Cipher

We define the 4'th power self-dual cipher as follows: E is a 4'th power self-dual cipher if:

$$(E_K(P))^4 = E_{K^4}(P^4).$$

We take Rijndael for example, and modify it so it becomes a 4'th power self-dual cipher. We require that each constant in the cipher is the 4'th power of itself. There are four such values: 00_x, 01_x, and the two elements of order 3, BC_x and BD_x (in Rijndael's representation), which are g^{85} and g^{170}, where g is a generator.

We modify the affine transformation in the S box to a 4'th power self-dual affine matrix, i.e., that $A^4 = Q \cdot Q \cdot (A) \cdot Q^{-1} \cdot Q^{-1} = A$. We can see that:

1. Q
2. $Q \cdot Q$
3. $Q \cdot Q \cdot Q$
4. $Q \cdot Q \cdot Q \cdot Q$
5. $Q \cdot Q \cdot Q \cdot Q \cdot Q$
6. $Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q$
7. $Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q$
8. $Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q \cdot Q = I$

satisfy the requirement for the affine transformation in a similar way to satisfying the requirements for the affine transformation in the square self-dual case. All the linear combinations of the above eight matrices with coefficients from $\{00_x, 01_x, g^{85}, g^{170}\}$ (which is $GF(2^2)$) also satisfy the requirements for the affine transformation. The total number of linear combinations is 2^{16}, of which $3 \cdot 2^{13}$ are involutions.

The resulting ciphers are 4'th power self-dual cipher. In addition, to the self-duality property, these ciphers have the property that if all the bytes of the plaintext and key are chosen from the set of the above four elements $\{00_x, 01_x, g^{85}, g^{170}\}$, then the bytes of the ciphertext also belong to this set of four elements.

For the 16'th power self-dual cipher, we require that all the constants are of 00_x, 01_x, and the 14 elements of orders 3, 5, and 15. The 16'th power self-dual matrices are all the linear combinations of the Q^i matrices, with coefficients from the above constants. The total number of 16'th power self-dual matrices is 2^{32}, of which $7 \cdot 5 \cdot 3^2 \cdot 2^{22}$ matrices are involutions. Fortunately, Rijndael's A matrix is none of these matrices. An additional property for 16'th power self-dual cipher is that if all the bytes of the plaintext and key are chosen from the set of the above 16 constants, then the bytes of the resulting ciphertexts are also from this set.

2.6.2 Cryptanalysis of Self-Dual Ciphers

The self-duality property of a cipher can be used to mount an attack, which reduces the complexity of exhaustive search by a factor of about 8 for a square dual cipher the case above (or by a factor of the number of the self-duals in the more general case). For example, if the key size is 128 bit, exhaustive search requires 2^{128} applications of the cipher E, and the attack we propose requires about 2^{125} applications of E using 8 chosen plaintexts. If we consider the expected time to complete the attack, exhaustive search takes about 2^{127} applications of E, and our attack takes about 2^{124} applications of E.

The attack takes advantage of cycles of keys under the squaring operation: A cycle is a set of keys where each key is the square of its predecessor, i.e., $\{K', K'^2, \ldots, K'^{2^7}\}$, and where the square of the last element equals the first element : $K' = K'^{2^8}$. Note that the possible cycle lengths are 8, 4, 2, and 1. The attacker's algorithm is as follows.

1. Choose a plaintext P, and compute $P_i = P^{2^i}$, for $i = 0, \ldots, 7$.

2. Ask for the encryption of $P_0, \ldots, P_7$, and denote the corresponding ciphertexts by $C_0, \ldots, C_7$. For every i, compute $\hat{C}_i = (C_i)^{2^{-i}}$, where the square root is

defined to be the operation that finds for every byte its square root in $GF(2^8)$ (there is only one square root for each value).

3. Choose one key K' in each cycle, and compute $C = E_{K'}(P)$. If $C = \hat{C}_i$ for some $i \in \{0, \ldots, 7\}$, K'^{2^i} is a candidate to be K. Otherwise, K is not one of $\{K'^{2^i}\}$.

An equality $C = \hat{C}_i$ in step 3 ensures that encryption of P_i under the key K'^{2^i} gives C_i: If $C = \hat{C}_i$, then $C^{2^i} = \hat{C}_i^{2^i} = C_i$. Therefore, $C^{2^i} = (E_{K'}(P_0))^{2^i} = C_i = E_K(P_0^{2^i}) = E^{2^i}{}_K(P_0^{2^i})$. From the self-duality property it follows that: $K = K'^{2^i}$ (unless this is a false-alarm, which can then be easily checked with another block).

Note that the correct key is always found by this method: divide the key space to cycles, and therefore, the correct key K must be in some cycle $\{K', K'^2, \ldots, K'^{2^7}\}$. Let j be the index in the above cycle such that $K = K'^{2^j}$. Since K'^{2^j} is the correct key it holds that $E_{K'^{2^j}}(P_j) = C_j$. Using the self-duality property, it follows that $E_{K'}(P_0) = C_j^{2^{-j}}$. This equality is detected in step 3, and thus the key is found. Thus, by encrypting one key of every cycle, we cover all the keys.

We test about eight keys by every trial encryption. It is easy to choose the keys K' in such a way that we choose only one key out of each cycle of keys. Therefore, this attack finds the key in about 2^{125} applications of E. In Appendix 2.13, we show how to enumerate the keys (choosing only one key of each cycle), and show that the total number of cycles, and thus, the maximal complexity of this attack, is $2^{125} + 2^{61} + 2^{30} + 2^{15}$, using 8 chosen plaintexts. The average case complexity is $2^{124} + \varepsilon$ where $\varepsilon = 2^{-4} + 2^{-67} + 2^{-98}$.

We note that a similar attack can be designed for higher-order self-dual ciphers. It is also interesting to note that the number of rounds of the cipher does not affect the complexity of this attack (in terms of the number of applications of E).

2.6.3 Application to BES

BES [61] (Big Encryption System) was proposed by Murphy and Robshaw. BES is basically an 8-fold Rijndael, with a special relation between the 8 applications of Rijndael. In this design, the affine operation in Rijndael's S box is replaced by its interpolation polynomial in $GF(2^8)$ (with some more details that we describe below). The result is a cipher with a $128 \cdot 8 = 1024$-bit keys and blocks, as follows:

$$\hat{C} = BES_{\hat{K}}(\hat{P}),$$

where $\hat{P} = (P_1, P_2, \ldots, P_8)$, $\hat{K} = (K_1, K_2, \ldots, K_8)$, $\hat{C} = (C_1, C_2, \ldots, C_8)$, where P_i, K_i and C_i are 128-bit blocks.

BES is defined in such a way that encryption with Rijndael, $C = Rijndael_K(P)$, can be performed as follows:

$$(C, C^2, C^4, C^8, C^{16}, ..., C^{128}) = \\ BES_{(K,K^2,K^4,K^8,K^{16},...,K^{128})}(P, P^2, P^4, P^8, P^{16}, ..., P^{128}),$$

where K and P are the key and plaintext of Rijndael, and C is the ciphertext.

The internal structure of BES is quite similar to parallel applications of $Rijndael$ and its dual-ciphers: $Rijndael^2$, $Rijndael^4$, $Rijndael^8$, $Rijndael^{16}$, $Rijndael^{32}$, $Rijndael^{64}$, $Rijndael^{128}$. The difference is that the affine matrix is replaced by its interpolation polynomial $\sum a_i \cdot x^{2^i}$. The values x^{2^i} are not computed directly, but are taken instead from a parallel value in one of the parallel applications (since if we have an intermediate value x in a parallel application of $Rijndael$ then we have the value x^{2^i} in the same location in the application of $Rijndael^{2^i}$).

We observe that BES is a self-dual cipher with $h(\cdot) = h_k(\cdot) = h_p(\cdot) = h_c(\cdot)$

$$h(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8) = (X_8^2, X_1^2, X_2^2, X_3^2, X_4^2, X_5^2, X_6^2, X_7^2).$$

Therefore, a variant of the attack we present for self-dual ciphers applies to BES. The expected time complexity is 8 times faster than exhaustive search.

However, the fact that BES is a self-dual cipher does not seem to have consequences on the security of Rijndael: Observe that when limiting the plaintext, ciphertext, and key space of BES so it performs Rijndael encryption, the self-duality function $h(\cdot)$ (that rotates to the right and squares) does not change the data blocks, as performing rotation to the right and squaring of a tuple $(X, X^2, X^4, X^8, ..., X^{128})$ leaves us with the same tuple $(X, X^2, X^4, X^8, ..., X^{128})$ (recall that in $GF(2^8)$, $X^{256} = X$).

2.7 Applications of Dual Ciphers

Dual ciphers might be used to gain insight to linear and differential cryptanalysis, and provide insight for developing new attacks. In such cases the insight gained from the dual ciphers could be used to attack the dual cipher, an attack which can be easily transformed to the original (as the original and dual cipher are isomorphic they are of the same strength against attacks). A possible example for such insight might be the simplification of the affine transformation in the S box to a triangular matrix (see Section 2.4), which reduces the effect of modifying bits in the input on the resultant

output of this transformation (e.g., the first bit is determined only by a single bit). It should be noted that dual ciphers do not change the resistance of ciphers to linear and differential cryptanalysis. It is actually possible to study the linear and differential properties of Rijndael in $GF(2^8)$ regardless of the representation of $GF(2^8)$ that is chosen, as is shown in Appendix A of [33].

The existence of dual ciphers can also be used to protect implementation against side channel attacks, such as fault-analysis [17] and power-analysis [54], by selecting a different dual cipher at random each time an encryption or decryption is desired. Alternatively, different rounds of the encryption process can be performed using different dual ciphers, with a conversion layer between them. The conversation layer converts the data from one dual to another.

An interesting application of dual ciphers might be an optimization of the speed of the cipher, as in some cases the dual cipher might actually be faster to compute than the original cipher. For example, many ciphers include multiplications by constants. The Hamming weight and the size of the constant has implications on the implementation efficiency. Thus, finding a more efficient dual cipher might be a good optimization strategy. Also, in some cases encryption might be most efficient using one dual cipher, and decryption be most efficient using another dual cipher. In [72] a resembling approach is taken, representing elements of $GF(2^8)$ as the composite field $GF(2^4)^2$, achieving a more efficient implementation. A more sophisticated approach combining our dual ciphers with composite fields was taken in [78], reaching a faster implementation of AES in hardware using a fewer number of gates.

2.8 Summary

In this chapter, we show how to write many different implementations of Rijndael using its various dual ciphers. We describe hundreds of non-trivial dual ciphers of Rijndael, many of them differ from Rijndael only by the replacement of constants. Thus, a program implementing a dual cipher would differ only in the constants. We discuss a special class of ciphers — self-dual ciphers — and mount an attack on these ciphers. Finally, we indicate several applications for dual ciphers, including insight for cryptanalysis, protection from side-channel attacks, and finding faster implementations of existing ciphers.

One result of this chapter is that the irreducible polynomial of Rijndael can be chosen arbitrarily, and that it is in fact possible to replace the irreducible polynomial of Rijndael by any other irreducible or primitive polynomial (of degree 8) without changing the strength of cipher, and even without changing the cipher itself.

We would also like to mention that there are other kinds of dual ciphers of

Rijndael that are not described in the chapter. For example, in [19] the following dual ciphers are suggested: Given a non-zero $\alpha \in GF(2^8)$, create a dual cipher with $h(x) = \alpha \cdot x$, where the multiplication by $\alpha \neq 0$ is byte-wise. Multiplication by α in $GF(2^8)$ can be performed by a multiplication by boolean matrix which is denoted by $[\alpha]$. Therefore, the affine transformation of the S box of the first round can be modified to cancel the multiplication, thus the result of encryption by the dual cipher E' is equal to the result of encryption by the original cipher E. I.e., the S box in E' is $S'(x) = A' \cdot x^{-1} \oplus b = (A \cdot [\alpha])x^{-1} \oplus b$. Many variations are also possible.

Acknowledgements

We are pleased to thank Ronny Roth for the various discussions which helped improving the results of this work and to John Kelsey for observing that dual ciphers may be used to prevent side-channel attacks.

The work described in this chapter has been supported by the European Commission through the IST Programme under Contract IST-1999-12324.

2.9 Appendix: The Affine Transformation of Rijndael and Rijndael^2

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad A^2 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

2.10 Appendix: The Matrix Q

We prove that from the equation $(a+b)^2 = a^2+b^2$ it follows that the square operation can be done by multiplication by a matrix. While doing it we discover how to compute such a matrix Q for any irreducible polynomial as vectors of $GF(2)^8$.

Given a vectorial representation of an element $a \in GF(2^8)$, we can write the vector as $a = \sum_{i=1}^{8} a_i \cdot e_i$, where e_i is the i'th element of the basis (i.e., the vectors

whose i'th bit is 1, and all the other bits are 0). a_i is the i'th bit of the vector a. Note that $a_i = a_i^2 \in GF(2^8)$, since a_i is either 0 or 1. Note that e_i can be written as $e_i = 2^{i-1}$.

So $a^2 = (\sum_{i=1}^{8} a_i \cdot e_i)^2 = \sum_{i=1}^{8} a_i^2 \cdot e_i^2 = \sum_{i=1}^{8} e_i^2 \cdot a_i = Q \cdot a$, where Q is the matrix whose i'th column ($i \in \{0, \ldots, 7\}$) contain the vectorial representation of e_i^2. The matrix Q of Rijndael is given in Eq. (2.1). It can be seen there that the columns are powers of $4(\equiv 2^2)$, where the first element $1 = 4^0$, the next is 4^1, then 4^2, 4^3, ..., 4^7, i.e., $Q = (4^0, 4^1, 4^2, \ldots, 4^7)$

2.11 Appendix: The Matrix R

We show that the isomorphism transformation denoted by R that replaces the irreducible polynomial $g(x)$ of a representation of $GF(2^8)$ to another representation of $GF(2^8)$ with a different irreducible polynomial $\hat{g}(x)$ is linear: Let x, y be elements in the representation using $g(x)$. They are transformed to $R(x)$ and $R(y)$, respectively, with the representation $\hat{g}(x)$. If we XOR the two elements in the two representation, due to the isomorphism it must hold that $R(x \oplus y) = R(x) \oplus R(y)$, therefore, R is linear with respect to XOR. Using a similar justification, R must be linear with respect to multiplication.

We now show that R is of the structure $R = (1, a, a^2, ..., a^7)$, where the a^i's are computed modulo the irreducible polynomial $\hat{g}(x)$. The value of the first column of R is $R \cdot (1, 0, \ldots, 0)^T$, and the first column is also the value that 01_x is transformed to. From the multiplicative linearity of R it follows that

$$Rx = R(x \cdot 1) = (Rx) \cdot (R \cdot (1, 0, \ldots, 0)^T),$$

i.e., $(R \cdot (1, 0, \ldots, 0) = 1$. Therefore, the first column of R must be 1. The second column determines were 02_x is transformed to, i.e., the second column is $R \cdot (0, 1, 0, 0, \ldots, 0)^T$. We denote the value of the second column of R (i.e., $R \cdot 02_x$) by a. The third column of R is the value that 04_x is transformed to, i.e., $R \cdot 04_x$. From the multiplicative linearity of R, it follows that:

$$R \cdot 04_x = R \cdot (02_x \cdot 02_x) = (R \cdot 2) \cdot (R \cdot 2) = a \cdot a = a^2 \quad (mod\ \hat{g}(x)).$$

We continue this way to show that column $i \in \{1, \ldots, 8\}$ of R is $R \cdot (02_x)^{i-1} = a^{i-1} \quad (mod\ \hat{g}(x))$. Note that the Q matrix computed in Appendix 2.10 is of the same form as the R matrix, for the same reasons, with $\hat{g}(x) = g(x)$, and $a = 04_x$ (i.e., $Q(02_x) = 04_x$).

There are 240 non-trivial dual ciphers of an $EGF(2^8)$ cipher. We now show how to find the exact description of these dual ciphers. For simplicity we focus on showing the 240 non-trivial dual ciphers of Rijndael. All we need to find in order to describe a dual cipher is the R matrix that takes from Rijndael's representation to that dual cipher. After we choose $\hat{g}(x)$ of the dual cipher, we need to know which a can be used. Obviously, there are 8 different a's that can be used, each one creates a different cipher, and all of them are square dual ciphers (or higher degree dual ciphers) of each other. We cannot choose any a, as the resulting R might not be a transformation **from** Rijndael's representation, but from another dual ciphers. A practical solution is to first find R^{-1} and then find its inverse R.

R^{-1} takes an element in a dual cipher to Rijndael's representation. Thus, R^{-1} is of the form $R^{-1} = (1, a, a^2, ..., a^7)$, where the a^i's are computed modulo the irreducible polynomial of Rijndael. There are 240 possible a values, one for each dual cipher. The a can take any value in $GF(2^8)$, which does not belong to $GF(2^4)$ (if a is in $GF(2^4)$ then R^{-1} is singular and does not span $GF(2^8)$). We compute R as the inverse matrix of R^{-1}.

We can find the polynomial $\hat{g}(x)$ out of R as follows: $\hat{g}(x)$ is of the form: $x^8 \oplus \alpha_7 x^7 \oplus \alpha_6 x^6 \cdots \oplus \alpha_1 x \oplus 1$, where $\alpha_i \in \{0, 1\}$. In polynomial representation of elements in $GF(2^8)$ $x^8 = \hat{g}(x) \oplus x^8 \pmod{\hat{g}(x)}$, since 2 is x in polynomial representation and $x^8 = \hat{g}(x) \oplus x^8 \pmod{\hat{g}(x)}$ it follows that $2^8 = \hat{g}(2) \oplus 2^8$. Let $b = 2^8 \pmod{\hat{g}(x)}$ then b in polynomial representation is $\hat{g}(x) \oplus x^8$, and also $R^{-1}b = a^8$. Therefore, to obtain $\hat{g}(x)$ compute $b = Ra^8$, transform b to its polynomial representation and add x^8.

2.12 Appendix: Properties of the $T(x)$ Transformation

Theorem 3 *The following properties hold for the $T(x)$ transformation:*

1. $T(x) - T(-x) = x$
2. $T(2x) = 2T(x)$ *(therefore, $\forall i, T(2^i x) = 2^i T(x)$)*
3. $T(T(x)) = x$
4. *Let $g \triangleq g'^y$, $yT_g(x) = T_{g'}(yx)$*
5. $T_g = T_{g^{2^i}}$

6. $T(x) = -T(-T(-x))$
7. $T(85) = 170$, $T(170) = 85$, *and if* $T(x) = x/2$ *then* $x \in \{85, 170\}$. *Note that* $85/2 \equiv 170 \pmod{255}$
8. $T(0) = -\infty$.
9. $T(-\infty) = 0$.
10. $T(x) = T(x \pm 255)$ - *The cycle size of T is 255.*

Proof Omitted. ■

Also if we define $S(x) = T(-x)$, we find the following properties:

1. $S(x) - S(-x) = -x$
2. $S(2x) = 2S(x)$
3. $S(S(S(x))) = x$
4. $S(-S(x)) = -x$
5. Let $g \triangleq g'^y, yS_g(x) = S_{g'}(yx)$
6. $S_g = S_{g^{2^i}}$
7. $S(170) = 170$, $S(85) = 85$, and there is no other x that satisfies S(X)=X.
8. $S(x) = S(x \pm 255)$. The cycle of S is 255.

The table of $T(x)$ with generator 03_x is described in Table 2.1.

2.13 Appendix: How to Enumerate the Keys of Self-Dual Ciphers

First thing to notice, is that the square operation in $GF(2^8)$ organizes the elements in groups of different sizes, as summarized in Table 2.2. Each element a belongs to the group of its square closure: $\{a, a^2, a^4, a^8, a^{16}, a^{32}, a^{64}, a^{128}\}$. We also refer to the square closure as *cycle*. Note that no matter which element of this group we choose as a, we would still get the same group by repeatedly squaring a. The size of the group is limited by 8, as in $GF(2^8)$ for any element a it holds that $a^{2^8} = a^{256} = a$.

Table 2.1: The Table $T(x)$ with Generator 03_x and Irreducible Polynomial $11B_x$ (Rijndael)

טבלת $T(x)$ עם גנרטור 03_x ופולינום לא פריק $11B_x$ (ריינדל)

$T[x] =$

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	$-\infty$	25	50	223	100	138	191	112	200	120	21	245	127	99	224	33
16	145	68	240	92	42	10	235	196	254	1	198	104	193	181	66	45
32	35	15	136	32	225	179	184	106	84	157	20	121	215	31	137	101
48	253	197	2	238	141	147	208	63	131	83	107	82	132	186	90	55
64	70	162	30	216	17	130	64	109	195	236	103	199	113	228	212	174
80	168	160	59	57	40	170	242	167	175	203	62	209	19	158	202	176
96	251	190	139	13	4	47	221	74	27	248	39	58	161	71	126	246
112	7	76	166	243	214	122	164	153	9	43	117	183	180	194	110	12
128	140	239	69	56	60	250	177	144	34	46	5	98	128	52	218	150
144	135	16	217	53	206	188	143	178	226	119	201	159	169	41	93	155
160	81	108	65	182	118	227	114	87	80	156	85	211	229	232	79	88
176	95	134	151	37	124	29	163	123	38	249	61	204	149	219	97	6
192	247	28	125	72	23	49	26	75	8	154	94	89	187	207	148	205
208	54	91	241	171	78	233	116	44	67	146	142	189	252	102	237	3
224	14	36	152	165	77	172	231	230	173	213	244	22	73	222	51	129
240	18	210	86	115	234	11	111	192	105	185	133	96	220	48	24	—

and

$$T[-\infty] = 0.$$

$$T[i - (-\infty)] = -(-\infty) + i$$

Elements of order 255 will therefore organize in groups of 8. Groups of 8 are also formed by the elements of orders: 17, 51 and 85. So in total there are 30 groups of 8 elements each. Elements of order 15 will organize in groups of 4 elements, as for any element a of order 15 it follows that $a^{16} = a$. A similar argument holds for elements of the order 5, as $a^{16} = (a^5)^3 a = a$. In total there are 3 groups of size 4. The two elements of order 3 fall into one group of size 2, and the remaining two elements are 0 and 1, each in a group of his own.

These cycles induce cycles on the keys. For each key, the length of the cycle is the maximal length of the cycles of its bytes. We want to find the minimal subset of keys covering all the cycles, e.g., a set of exactly one key from each cycle of $\{K, K^2, \ldots, K^{2^7}\}$.

To find them, use the following algorithm:

1. Output a representative from each cycle of keys that has an element of order 8 in at least one of its bytes.

2. Then, output a representative from each cycle of keys that has an element of

Table 2.2: The Element Cycles Under the Squaring Operation
מחזור האיברים תחת פעולת הריבוע

Group Size	Number of Groups	Total
1	2	2
2	1	2
4	3	12
8	30	240
Total	36	256

order 4 in at least one of its bytes, but has no byte that belongs to a cycle of higher order.

3. Then, output a representative from each cycle of keys that has an element of order 2 in at least one of its bytes, but has no byte that belongs to a cycle of higher order.

4. Finally, output all the keys with cycle of size 1, in all their bytes.

It can be easily verified that the algorithm outputs exactly one representative key from each cycle.

The following algorithm outputs all the representative keys that have at least one byte that belongs to a cycle of size $c > 1$, such that there is no byte that belongs to a cycle with size higher than c.

This algorithm holds for $c \in \{2, 4, 8\}$:

For each byte $i = 0 \ldots 15$, For each cycle of size c:

1. fix the value of byte i to be an element of the cycle.

2. Output all the possibilities of keys such that

   (a) for bytes $j \in \{0, \ldots, i-1\}$, choose their values as all the combinations of bytes that belong to cycles of size smaller than c.

   (b) for bytes $j \in \{i+1, \ldots, 15\}$, choose their values as all the possible combinations of bytes that belong to cycles of size smaller or equal to c.

The algorithm for $c = 1$ is: output all the 2^{16} combinations of 16 bytes of $\{0, 1\}$ (whose order is 1).

The number of cycles of keys of each size and the number of keys in each cycle are summarized in Table 2.3.

Table 2.3: The Key Cycles Under the Squaring Operation
מחזור המפתחות תחת פעולת הריבוע

Key Cycle Order	Number of Cycles	Total Keys
8	$\sum_{i=1}^{16} 30 \cdot (16)^{i-1} \cdot (256)^{16-i} = 2^{125} - 2^{61}$	$2^{128} - 2^{64}$
4	$\sum_{i=1}^{16} 3 \cdot (4)^{i-1} \cdot (16)^{16-i} = 2^{62} - 2^{30}$	$2^{64} - 2^{32}$
2	$\sum_{i=1}^{16} 1 \cdot (2)^{i-1} \cdot (4)^{16-i} = 2^{31} - 2^{15}$	$2^{32} - 2^{16}$
1	2^{16}	2^{16}
Total	$(2^{125} - 2^{61}) + (2^{62} - 2^{30}) + (2^{31} - 2^{15}) + 2^{16} = 2^{125} + 2^{61} + 2^{30} + 2^{15}$	

Chapter 3

Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communications

In this chapter we present a very practical ciphertext-only cryptanalysis of GSM encrypted communication, and various active attacks on the GSM protocols. These attacks can even break into GSM networks that use "unbreakable" ciphers. We first describe a ciphertext-only attack on A5/2 that requires a few dozen milliseconds of encrypted off-the-air cellular conversation and finds the correct key in less than a second on a personal computer. We extend this attack to a (more complex) ciphertext-only attack on A5/1. We then describe new (active) attacks on the protocols of networks that use A5/1, A5/3, or even GPRS. These attacks exploit flaws in the GSM protocols, and they work whenever the mobile phone supports a weak cipher such as A5/2. We emphasize that these attacks are on the protocols, and are thus applicable whenever the cellular phone supports a weak cipher, for example, they are also applicable for attacking A5/3 networks using the cryptanalysis of A5/1. Unlike previous attacks on GSM that require unrealistic information, like long known plaintext periods, our attacks are very practical and do not require any knowledge of the content of the conversation. Furthermore, we describe how to fortify the attacks to withstand reception errors. As a result, our attacks allow attackers to tap conversations and decrypt them either in real-time, or at any later time. We present several attack scenarios such as call hijacking, altering of data messages and call theft.

The work in this chapter is a joint work with Prof. Eli Biham and Nathan Keller. A significantly shorter version of this chapter was published in [8, 10].

3.1 Introduction

GSM is the most widely used cellular system in the world, with over a billion customers around the world. The system was developed during the late 1980s, and the first GSM network were deployed in the early 1990s. GSM is based on second generation cellular technology, i.e., it offers digitalized voice (rather than analog, as used in prior systems).

GSM was the first cellular system which seriously considered security threats. One example is a secure cryptographic hardware in the phone (the SIM — Subscriber Identity Module), which was introduced in GSM. Previous cellular systems had practically no security, and they were increasingly the subject of criminal activity such as eavesdropping on cellular calls, phone cloning, and call theft.

The security threat model of GSM was influenced by the political atmosphere around cryptology at the 1980s, which did not allow civilians to use strong cryptography. Therefore, the objective was that the security of GSM would be equivalent to the security of fixed-line telephony. As a result, only the air-interface of GSM was protected, leaving the rest of the system un-protected. The aim of the protection on the air-interface is to provide two kinds of protections: protect the privacy of users (mostly through encryption), and protect the network from unauthorized access to the network (by cryptographic authentication of the SIM).

The privacy of users on the air-interface is protected by encryption. However, encryption can start only after the mobile phone identified itself to the network. GSM also protects the identity of the users by pre-allocating a temporary identification (TMSI — Temporary Mobile Subscriber Identity) to the mobile phone. This temporary identification is used to identify the mobile phone before encryption can commence. The temporary identification for the next call can safely be replaced once the call is encrypted.

Authentication of the SIM by the network occurs at a beginning of a radio conversation between the mobile phone and the network. After the phone identifies itself (e.g., by sending its TMSI), the network can initiate an authentication procedure. The procedure is basically a challenge-response scheme based on a pre-shared secret Ki between the mobile phone and the network. In the scheme, the network challenges the mobile phone with a 128-bit random number $RAND$; the mobile phone transfers $RAND$ to the SIM, which calculates the response $SRES = A3(Ki, RAND)$, where $A3$ is a one-way function; then, the mobile phone transmits $SRES$ to the network, which compares is to the $SRES$ value that it pre-calculated. The encryption key Kc for the conversation is created in parallel to the authentication by $Kc = A8(Ki, RAND)$, where $A8$ is also a one-way function. The remainder of the

call can be encrypted using Kc, and thus, the mobile phone and the network remain mutually "authenticated" due to the fact that they use the same encryption key. However, encryption is controlled by the network, and it is not mandatory. Therefore, an attacker can easily impersonate the network to the mobile phone using a false base station with no encryption. In general, it is not advisable to count on an encryption algorithm for authentication, especially in the kind of encryption that is used in GSM.

The exact design of A3 and A8 can be selected by each operator independently. However, many operators used the example, called *COMP128*, given in the GSM memorandum of understanding (MoU). Although never officially published, the design of COMP128 was reverse engineered by Briceno, Goldberg, and Wagner [25]. They have performed cryptanalysis of COMP128 [26], allowing to find the pre-shared secret Ki of the mobile phone and the network. Given K_i, A3 and A8 it is easy to perform cloning. Their attack requires the $SRES$ for about 2^{17} values of $RAND$. The required data for this kind of attack can obtained within a few hours over-the-air using a fake base station.

The original encryption algorithm for GSM was A5/1. However, A5/1 was export restricted, and as the network grew beyond Europe there was a need for an encryption algorithm without export restrictions. As a result, a new (weakened) encryption algorithm A5/2 was developed. The design of both algorithms was kept secret (it was disclosed only on a need-to-know basis, under an non-disclosure agreement, to GSM manufacturers). In 2002, an additional new version A5/3, was added to the A5 family. Unlike, A5/1 and A5/2, it's internal design was *published*. A5/3 is based on the block-cipher KASUMI, which is used in third generation networks [1]. A5/3 is currently not yet deployed in GSM, but deployment should start soon.

The internal design of both A5/1 and A5/2 was reverse engineered from an actual GSM phone by Briceno [24] in 1999. The internal design was verified against known test-vectors, and it is available on the Internet [24].

After the reverse engineering of A5/1 and A5/2, it was demonstrated that A5/1 and A5/2 do not provide an adequate level of security for GSM. However, most of the attacks are in a known-plaintext attack model, i.e., they require the attacker not only to intercept the required data frames, but also to know their contents before they are encrypted.

A5/1 was initially cryptanalyzed by Golic [46] when only a rough outline of A5/1 was leaked. After A5/1 was reverse engineered, it was analyzed by Biryukov, Shamir, and Wagner [21]; Biham and Dunkelman [15]; Ekdahl and Johansson [35]; Maximov, Johansson and Babbage [58]; and recently by Barkan and Biham [6].

As for A5/2, it was cryptanalyzed by Goldberg, Wagner and Green [45] imme-

diately after the reverse engineering. This attack on A5/2 works in a negligible time complexity and it requires only two known-plaintext data frames which are exactly $26 \cdot 51 = 1326$ data frames apart (about 6 seconds apart). Another attack on A5/2 was proposed by Petrović and Fúster-Sabater [76]. This attack works by constructing a systems of quadratic equations whose variables describe the internal state of A5/2 (i.e., equations of the form $c = \bigoplus_{i,j} a_i \cdot a_j$, where $a_i, a_j, c \in \{0, 1\}$, a_i and a_j are variables and c is a constant). This attack has the advantage that it requires only four known-plaintext data frames (thus the attacker is not forced to wait 6 seconds), but it does not recover the encryption key, rather, it allows to decrypt most of the remaining communications.

3.1.1 Executive Summary of the New Attacks

In this chapter we describe several attacks on the A5 variants and on the GSM protocols. We first show a passive known-keystream attack on A5/2 that requires a few dozen milliseconds of known keystream. In this attack, we construct systems of quadratic equations that model the encryption process. Then, we solve the system to recover the internal state, and thus the key that was used.

We improve this attack on A5/2 to work in real time (finding the key in less than a second on a personal computer) by dividing the attack into two phases, a precomputation phase and a real-time phase. The attacker first performs a one-time precomputation of a few hours, in which he finds how to solve all the equation systems and stores instructions for the solution in memory. In the real-time phase, the attacker uses the instructions quickly solve the equations.

We then transform this known-keystream attack on A5/2 into a ciphertext-only attack. The key idea is to take advantage of the fact that GSM employs error correction before encryption in the transmission path (instead of the well established reverse order). The error correction introduces linear dependencies between the bits. Assume that it is known that the parity (XOR) of some subset of bits is 0. XORing the same subset of bits after encryption reveals the parity of the corresponding keystream bits. We use an attack similar to the known-keystream attack, in which the parity of keystream bits is used instead of the keystream bits themselves. The resulting optimized attack completes in less than a second on a personal computer.

The above attacks assume that there are no reception errors. To overcome this restriction, we improve the attack on A5/2 to withstand a class of reception errors.

Next, we present a ciphertext-only attack on A5/1 whose complexity is considerably higher than the previous two attacks on A5/2. However, it demonstrates that passive A5/1 eavesdropping is feasible even for a medium-sized organization. We uti-

lize the same technique as in the passive attack on A5/2, to reveal the parity of bits of the keystream. We then view the function from the internal state to the known-keystream bits as a random function, and perform a (generic) time/memory/data tradeoff attack, taken from the published literature [20]. Once the internal state is found, a candidate key is found (and can be checked using trial encryptions). It should be noted that the time/memory/data tradeoff requires a lengthy preprocessing phase and huge storage, but still the key can be recovered in a relatively short time. It should also be noted that the recovery process is probabilistic in nature, and that given enough data the success probability becomes close to one.

We then deal with another family of attacks, which are active attacks on the GSM protocol. These attacks can work even if the network supports only A5/1 or A5/3, as long as the mobile supports A5/2. The key flaw that allows the attacks is that the same key is used regardless of whether the phone encrypts using A5/2, A5/1, or A5/3. Therefore, the attacker can mount a man-in-the-middle attack, in which the attacker impersonates the mobile to the network, and the network to the mobile (by using a fake base station). The attacker might use A5/1 for communication with the network and A5/2 for communications with the mobile, and due to the flaw, both algorithms encrypt using the same key. The attacker can gain the key through the passive attack on A5/2. Since the attacker is in the middle, he can eavesdrop, change the conversation, perform call theft, etc. The attack applies to all the traffic including short message service (SMS).

A similar active attack applies to GPRS, which is a 2.5 generation service that allows mobile internet supporting services such as Internet browsing, e-mail on the move, and multimedia messages.

The security of GPRS is based on the same mechanisms as of GSM: the same A3A8 algorithm is used with the same K_i, but the authentication and key agreement of GPRS occurs in different times than in GSM, using a different $RAND$ value. Since the $RAND$ is different, the resulting $SRES$ and K_c are different, and are referred to as $GPRS\text{-}SRES$ and $GPRS\text{-}K_c$, respectively. The GPRS cipher is different from A5/1 and A5/2, and is referred to as GPRS-A5, or GPRS Encryption Algorithm (GEA). Similarly to A5, GEA is implemented in the phone (rather than in the SIM), thus an old SIM card can work in a GPRS-enabled phone. There are currently three versions of the algorithm: GEA1, GEA2, and GEA3 (which is similar to A5/3). Much like A5/1 and A5/2, the internal design of GEA1 and GEA2 was never made public.

Although GPRS uses a different set of encryption algorithms, the key for GPRS is generated using the same A3A8 algorithm using the same K_i but with a different $RAND$ called $GPRS\text{-}RAND$. Therefore, an attacker can use a fake base

station to initiate a (non-GPRS) conversation with the mobile using A5/2, and send the $GPRS\text{-}RAND$ instead of $RAND$. Thus, the resulting key is identical to the key that is used in GPRS, and the attacker can recover it using the attack on A5/2.

3.1.2 Organization of this Chapter

This chapter is organized as follows: In Section 3.2, we give a short description of A5/2 and the way it is used. We present our new known plaintext attack in Section 3.3. This attack is improved in Section 3.4 to a ciphertext-only attack. We enhance our attack to withstand radio reception errors in Section 3.5. We then describe a passive ciphertext-only attack on A5/1 in Section 3.6. Active attacks on GSM are presented in Section 3.7, in which we show how to leverage the ciphertext-only attack on A5/2 to an active attack on any GSM network. We discuss the implications of the attacks under several attack scenarios in Section 3.8. Finally, we describe several ways of identifying and isolating a specific victim in Section 3.9. Section 3.10 summarizes the chapter. In Appendix 3.11, we improve Goldberg, Wagner, and Green's attack to a ciphertext-only attack. We give a technical background on GSM in Appendix 3.12.

3.2 Description of A5/2

The stream cipher A5/2 accepts a 64-bit key K_c, and a 22-bit publicly known initial value (IV) called COUNT (which is derived from the publicly known frame number, as described in Appendix 3.12). We denote the value of COUNT by f. The internal state of A5/2 is composed of four maximal-length Linear Feedback Shift Registers (LFSRs): $R1$, $R2$, $R3$, and $R4$, of lengths 19-bit, 22-bit, 23-bit, and 17-bit, respectively, with linear feedback as shown in Figure 3.1. Before a register is clocked the feedback is calculated (as the XOR of the feedback taps). Then, the register is shifted one bit to the right (discarding the rightmost bit), and the feedback is stored into the leftmost location (location zero).

A5/2 is initialized with K_c and f in four steps, as described in Figure 3.2, where the i'th bit of K_c is denoted by $K_c[i]$, the i'th bit of f is denoted by $f[i]$, and $i = 0$ is the least significant bit. We denote the internal state after the key setup by $(R1, R2, R3, R4) = \text{keysetup}(K_c, f)$. This initialization is referred to as the *key setup*. Note that the key setup is linear in both K_c and f (without bits $R1[15]$, $R2[16]$, $R3[18]$, and $R4[10]$ that are always set to 1).

A5/2 works in cycles, where at the end of each cycle one output bit is produced. During each cycle two or three of registers $R1$, $R2$, and $R3$ are clocked, according to

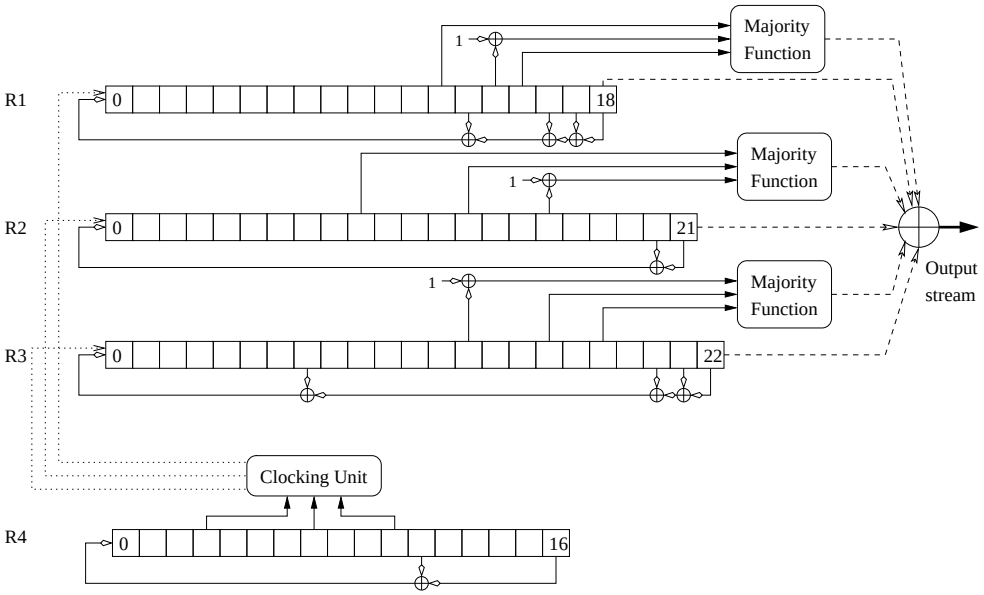


Figure 3.1: The Internal Structure of A5/2
המבנה הפנימי של A5/2

the value of three bits of $R4$. Then, $R4$ is clocked. At the beginning of each cycle, the three bits $R4[3]$, $R4[7]$, and $R4[10]$ enter a clocking unit. The clocking unit performs a majority function on the bits. Then, the registered are clocked as follows: $R1$ is clocked if and only if $R4[10]$ agrees with the majority. $R2$ is clocked if and only if $R4[3]$ agrees with the majority. $R3$ is clocked if and only if $R4[7]$ agrees with the majority. After these clockings, $R4$ is clocked, and an output bit is generated from the values of $R1$, $R2$, and $R3$, by XORing their rightmost bits to three majority values, one of each register. See Figure 3.1 for the exact details. It is important to note that the majority function (used for the output) is quadratic in its input: $maj(a, b, c) = a \cdot b \oplus b \cdot c \oplus c \cdot a$. Thus, an output bit is a quadratic function of bits of $R1$, $R2$, and $R3$.

The first 99 bits of output are discarded,[1] and the following 228 bits of output are used as the output keystream. The keystream generation can be summarized as follows:

[1]Some references state that A5/2 discards 100 bits of output, and that the output is used with a one-bit delay. This is equivalent to stating that it discards 99 bits of output, and that the output is used without delay.

1. Set $R1 = R2 = R3 = R4 = 0$.
2. For $i = 0$ to 63
   - Clock all four registers.
   - $R1[0] \leftarrow R1[0] \oplus K_c[i]$; $R2[0] \leftarrow R2[0] \oplus K_c[i]$; $R3[0] \leftarrow R3[0] \oplus K_c[i]$; $R4[0] \leftarrow R4[0] \oplus K_c[i]$.
3. For $i = 0$ to 21
   - Clock all four registers.
   - $R1[0] \leftarrow R1[0] \oplus f[i]$; $R2[0] \leftarrow R2[0] \oplus f[i]$; $R3[0] \leftarrow R3[0] \oplus f[i]$; $R4[0] \leftarrow R4[0] \oplus f[i]$.
4. Set the bits $R1[15] \leftarrow 1$, $R2[16] \leftarrow 1$, $R3[18] \leftarrow 1$, $R4[10] \leftarrow 1$.

Figure 3.2: The Key Setup of A5/2
אלגוריתם אתחול המפתח של A5/2

1. Run the key setup with K_c and f (Figure 3.2).
2. Run A5/2 for 99 cycles and discard the output.
3. Run A5/2 for 228 cycles and use the output as keystream.

The output of 228 bits (referred to as keystream) is divided into two halves. The first half of 114 bits is used as a keystream to encrypt the link from the network to the phone, and the second half of 114 bits is used to encrypt the link from the phone to the network. Encryption is performed as a bitwise XOR of the message with the keystream.

It is worth noting that A5/2 is built on top of A5/1's architecture. The feedback functions of $R1$, $R2$ and $R3$ are the same as A5/1's feedback functions. The initialization process of A5/2 is also similar to that of A5/1, with the only differences is that A5/2 also initializes $R4$, and that one bit in each register is forced to be 1 after initialization, while A5/1 does not use $R4$, and no bits are forced. Then A5/2 discards 99 bits of output while A5/1 discards 100 bits of output. The clocking mechanism is the same, but the input bits to the clocking mechanism are from $R4$

in the case of A5/2, while in A5/1 they are from $R1$, $R2$, and $R3$. The designers meant to use similar building blocks to save hardware in the mobile phone [74].

3.3 Known Plaintext Attacks on A5/2

In this section we present a new known plaintext attack (known keystream attack) on A5/2. Namely, given a keystream divided into frames, and the respective frame numbers, the attack recovers the session key. For completeness we start by describing in details Goldberg, Wagner, and Green's attack on A5/2.

3.3.1 Goldberg, Wagner, and Green's Known Plaintext Attack on A5/2

The first observation that this attack is based on is that since $R4[10]$ is forced to be "1" after initialization, $R4$ has the same value after initialization regardless of whether the bit $f[10]$ of COUNT is zero or one. Since $R4$ controls the clockings of $R1$, $R2$, and $R3$, the clockings of these registers is independent of the value of $f[10]$. Taking into account the fixed permutation between the TDMA frame number and COUNT (see [41, annex C] or Appendix 3.12), two frames which are exactly $26 \cdot 51 = 1326$ TDMA frames (about 6 seconds) apart are required, where the first frame's $f[10]$ is zero. Note that the first frame's $f[10]$ might be one, in this case the attacker is forced to wait at most another six seconds for $f[10]$ to be zero. The attacker cannot use a frame with $f[10] = 1$ as a first frame, since due to the carry (remember that the TDMA frame number is incremented by one every frame) other bits of the COUNT are changed, and thus register $R4$ is different in the two frames. We conclude that the attacker is forced to wait between 6 to 12 seconds to obtain the required data for the attack.

The attack is as follows: Let f_1 and f_2 be the respective COUNT value for two frame numbers as described above, with respective key-streams k_1, k_2. Denote the values of registers $R1$, $R2$, $R3$, and $R4$ in the first frame, just after the key setup (before the 99 clockings), by $R1_1$, $R2_1$, $R3_1$, and $R4_1$, respectively. We use a similar notation for the initial internal state of the second frame, i.e., we denote the value of the registers in the second frame after the key setup by $R1_2$, $R2_2$, $R3_2$, and $R4_2$. Note that the special choice of f_1 and f_2 ensures that $R4_1 = R4_2$, and we denote its value by $R4$. The other registers are not equal, however, since the initialization process is linear in f_1 and f_2, the difference between $R1_1$, $R2_1$, $R3_1$ and $R1_2$, $R2_2$, $R3_2$, respectively, is also linear in the difference between f_1 and f_2. These differences

are fixed, as $f_1 \oplus f_2 = 00000000000100000000000_b$. Thus, we can write $R1_1 = R1_2 \oplus \delta_1$, $R2_1 = R2_2 \oplus \delta_2$, $R3_1 = R3_2 \oplus \delta_3$, where δ_1, δ_2, and δ_3 are some constants.

We now show that given the value of $R4$, the keystream difference $k_1 \oplus k_2$ is linear in $R1_1$, $R2_1$, and $R3_1$. Given $R4$, the entire clocking of the registered is known (and is equal in the two frames as $R4_1 = R4_2$. Let l_1, l_2, and l_3 be the number of clocks that registers $R1$, $R2$, and $R3$ have been clocked by the end of cycle i. Therefore, the values of the three registers at the end of cycle i of the first frame are $L1^{l_1} \cdot R1_1$, $L2^{l_2} \cdot R2$, and $L3^{l_3} \cdot R3$, where $L1$, $L2$, and $L3$ are matrices that express one clocking of the respective registers. Similarly, the values of the registers at the second frame at the end of cycle i are $L1^{l_1} \cdot (R1_1 \oplus \delta_1)$, $L2^{l_2} \cdot (R2 \oplus \delta_2)$, and $L3^{l_3} \cdot (R3 \oplus \delta_3)$.

Let $g_1(R1) \oplus g_2(R2) \oplus g_3(R3)$ be the output bit of A5/2 given that the internal state of the registers is $R1$, $R2$, and $R3$; $g_1(\cdot)$, $g_2(\cdot)$, and $g_3(\cdot)$ are quadratic (as they involve one application of the majority function). To better understand that the output is quadratic in the internal state, consider the following example. Let $x_0, \ldots, x_{18}$, $y_0, \ldots, y_{21}$, $z_0, \ldots, z_{22}$ be variables representing the bits of R1, R2, and R3, respectively, just after the first bit of the keystream is produced. Then, the first bit of the keystream is

$$k_1[0] = x_{12}x_{14} \oplus x_{12} \oplus x_{12}x_{15} \oplus x_{14}x_{15} \oplus x_{15} \oplus x_{18} \oplus y_9y_{13} \oplus \ldots \oplus z_{16}z_{18} \oplus z_{22}$$

(which is quadratic in the variables representing the internal state).

Goldberg, Wagner, and Green observed that the difference of the output bits can be expressed as a linear function of the internal state of the first frame. The difference in the output bit of cycle i is given by:

$$\begin{aligned} &g_1(L1^{l_1} \cdot R1_1) \oplus g_1(L1^{l_1} \cdot R1_1 \oplus \delta_1) \oplus \\ &g_2(L2^{l_2} \cdot R2_1) \oplus g_2(L2^{l_2} \cdot R1_2 \oplus \delta_2) \oplus \\ &g_3(L3^{l_3} \cdot R3_1) \oplus g_3(L3^{l_3} \cdot R1_3 \oplus \delta_3) = \\ &\qquad\qquad g_{\delta_1}(L1^{l_1} \cdot R1_1) \oplus g_{\delta_2}(L2^{l_2} \cdot R2_1) \oplus g_{\delta_3}(L3^{l_3} \cdot R3_1), \end{aligned}$$

where $g_{\delta_1}(\cdot)$, $g_{\delta_2}(\cdot)$, and $g_{\delta_3}(\cdot)$ are linear function. Thus, the output difference is linear in $R1_1$, $R2_2$, and $R3_3$. It remains to show that given a quadratic function $g(x_1, \ldots, x_n)$ and $\Delta = \Delta_1, \ldots, \Delta_n$, $g_\Delta \triangleq g(x_1, \ldots, x_n) \oplus g(x_1 \oplus \Delta_1, x_2 \oplus \Delta_2, \ldots, x_n \oplus \Delta_n)$ is linear in $x_1, \ldots x_n$, where $x_i, \Delta_i \in \{0, 1\}$.

Since g is quadratic, it can be written as

$$g(x_1, \ldots, x_n) = \sum_{1 \le i,j \le n} a_{i,j}x_ix_j \oplus a_{0,0},$$

where $a_{i,j} \in \{0,1\}$ are fixed for a given g. Thus,

$$\begin{aligned} g_\Delta &= \sum_{1 \le i,j \le n} a_{i,j}(x_i x_j \oplus (x_i \oplus \Delta_i)(x_j \oplus \Delta_j)) \\ &= \sum_{1 \le i,j \le n} a_{i,j}(x_i x_j \oplus x_i x_j \oplus x_i \Delta_j \oplus \Delta_i x_j \oplus \Delta_i \Delta_j) \\ &= \sum_{1 \le i,j \le n} a_{i,j}(x_i \Delta_j \oplus \Delta_i x_j \oplus \Delta_i \Delta_j). \end{aligned}$$

The last expression is linear in $x_1, \ldots, x_n$ given $\Delta_1, \ldots, \Delta_n$.

Therefore, given $R4$ and $k_1 \oplus k_2$, the initial internal state $R1_1$, $R2_1$, and $R3_1$ can be recovered (solving a linear systems of equations). K_c can be recovered from the initial internal state ($R1_1$, $R2_1$, $R3_1$, $R4_1$) and f_1 by reversing the key setup of A5/2. As $R4$ is not known, the attacker needs to guess all possible 2^{16} values of $R4$, and for each value solve the resulting linear equation, until a consistent solution is found.

A faster solution is possible by filtering for the correct $R4$ values. The initial internal state of $R1$, $R2$, and $R3$ is 61 bits (recall that three bits of $R1$, $R2$, and $R3$ are set to 1). Thus, 61 bits of $k_1 \oplus k_2$ are required to reconstruct K_c, while $k_1 \oplus k_2$ is 114 bits long. It is therefore possible to construct an overdetermined linear system whose solution is the internal state. The $114 - 61 = 53$ dependent equations would zero during the Gauss elimination. These equations depend on the value of $R4$, thus, for every value of $R4$, it is possible to write 53 equations $V_{R4} \cdot (k_1 \oplus k_2) = 0$, where V_{R4} is a 53×114 bits matrix, and 0 is a vector of 53 zeros. The redundancy is used to filter wrong $R4$ values by checking that $V_{R4} \cdot (k_1 \oplus k_2) = 0$. On average it takes two dot products (out of the 53 equations) to disqualify a wrong $R4$ value. As there are 2^{16} possible values for $R4$, and as on average the correct $R4$ would be found after trying $2^{16}/2$ values, the average attack time is about 2^{16} dot products, plus a single solution of the equation system. A straightforward implementation on a 32-bit personal computer, where all possible V_{R4} systems are pre-loaded to memory, consumes $2^{16}(16 \cdot 114)/8 = 2^{16} \cdot 228$ bytes (about 15 MBs of volatile memory), and requires a few milliseconds of CPU time (on a 2GHz personal computer) to filter for the correct value of $R4$. Once $R4$ is found, we can solve the linear equations for this specific $R4$ in order to recover $R1_1$, $R2_1$, and $R3_1$. Storing these systems of equations after Gauss elimination takes about $2^{16} \cdot 64 \cdot 114/8 = 2^{16} \cdot 912$ bytes, i.e., about 60 MBs of memory. Note that this memory can be stored on a hard-disk, and can be indexed by $R4$. Given $R4$, the relevant system can be fetched to volatile memory. The complexity can be further reduced by considering fewer bits of $k_1 \oplus k_2$.

The attack as described above requires a relatively short preprocessing consisting

of the computation of the equations. The preprocessing can be completed within a few minutes on a personal computer.

3.3.2 Our Non-Optimized Known-Plaintext Attack on A5/2

We present an attack on A5/2 that requires the keystream of (any) four frames. Our attack recovers the internal state ($R1$, $R2$, $R3$, and $R4$), and by reversing the key setup, it finds the session key.

Our known-plaintext attack can be viewed as an improvement of Goldberg, Wagner, and Green's attack. We guess the initial value of $R4$, and write every output bit as a quadratic term in $R1$, $R2$, and $R3$. We describe a way to write every output bit — even if on different frames — as a quadratic term of $R1$, $R2$, and $R3$ of the first frame. Given the output bits of four frames, we construct a system of quadratic equations, and solve it using linearization. Thus, we recover the initial value of $R1$, $R2$, and $R3$.

Let k_1, k_2, k_3, and k_4 be the keystream of A5/2 for frames f_1, f_2, f_3, and f_4, respectively. Note that each k_j is the output keystream for a whole frame, i.e., each k_j is 114-bit long.[2] We denote the i'th bit of the keystream of f_j by $k_j[i]$. The initial internal state of register Ri of frame j (after the initialization but before the 99 clockings) is denoted by Ri_j.

As we discussed in Section 3.3.1, given $R4$, each output bit can be written as a quadratic function of the initial internal state of $R1$, $R2$, and $R3$. We like to construct a system of quadratic equations that expresses the equality of the quadratic terms for each bit of the output, and the actual value of that bit from the known-keystream. The solution of such a system would reveal the internal state. However, solving a general system of quadratic equations is NP complete. Fortunately, there are shortcuts when the quadratic system is over defined (in our case there are 61 variables and 114 quadratic equations, so the system is overdefined). The complexity drops significantly as the system becomes more and more overdefined. Therefore, we improve this attack by adding equations from other frames, while making sure the equations are over the same variables, i.e., the initial value of $R1$, $R2$, $R3$ at frame f_1. Once we combine the equations of four frames, we solve the system by linearization.

A system of equations is built for each of the 2^{16} possible values for $R4_1$ and solved, until we find a consistent solution. The solution of such a system is the initial internal state at frame f_1.

[2]Note that by keystream for a frame, we refer to the 114-bit keystream half that is used in the encryption process of the frame for a single direction, e.g., the network-to-mobile link.

There are at most 656 variables after linearization: We observe that each majority function operates on bits of a single register. Therefore, the quadratic terms consist of pairs of variables of the same register only. Taking into account that one bit in each register is set to 1, $R1$ contributes 18 linear variables and all their $\frac{17\cdot 18}{2} = 153$ products. In the same way $R2$ contributes $21 + \frac{21\cdot 20}{2} = 21 + 210$ variables and $R3$ contributes $22 + \frac{22\cdot 21}{2} = 22 + 231$ variable, totaling $18 + 153 + 21 + 210 + 22 + 231 = 655$ variables after linearization. We include the constant 1 as a variable to represent the affine part of the equations, thus our set of variables contains 656 variables. We denote the set of these 656 variables for frame f_i by S_i.

It remains to show how given the variables in the set S_1 of frame f_1, we can describe the output bits of frames f_2, f_3, and f_4 as linear combinations of variables from the set S_1. Assume that we know the value of $R4_1$, and recall that the key setup is linear in COUNT (see Section 3.2) (and that COUNT is publicly known for both frames). Therefore, given the COUNT difference of the frames, we know the difference in the values of each register after key setup: $R4_1$ is given, and thus we know $R4_2$. As $R1_1$, $R2_1$, and $R3_1$ are unknown, we only know the XOR-differences between $R1_1, R2_2, R3_3$ and $R1_2, R2_2, R3_2$ respectively.

We translate each variable in S_2 to variables in S_1: Let x_1 be the concatenated value of the linear variables in S_1, and g a quadratic function such that $V_1 = g(x_1)$. We know that the concatenated value of the linear variables of S_2 can be written as $x_2 = x_1 \oplus \delta_{1,2}$, and clearly $S_2 = g(x_2)$. Much like in Section 3.3.1, the difference between S_2 and S_1 is linear in x_1, which implies that S_2 can be expressed in linear terms of the variables in S_1. Thus, we construct a system of quadratic equations using the keystream of four frames with the variables taken only from S_1. In total, we create an equation system of the form: $S_{R4_1} \cdot S_1 = k$, where S is the system's matrix, $k = k_1||k_2||k_3||k_4$, and "||" denotes concatenation. Note that S_{R4_1} depends on the value of $R4_1$, and on the difference between COUNT value of the frames.

Clearly, once we obtain 656 linearly independent equations the system can be easily solved using Gauss elimination. We observe that it is practically very difficult to collect 656 *linearly independent* equations, due to the low order of the output function and the frequent initializations of A5/2 (A5/2 is re-initialized once 228 of output bits are generated). However, we do not actually need to solve all the variables, as it suffices to solve the linear variables of the system. We have tested experimentally and found that about 450 linearly-independent equations are always sufficient to solve the original linear variables in V_1 using linearization and Gauss elimination.[3]

[3]In case the data available for the attacker is scarce, there are additional methods that can be used to reduce the number of required equations. For example, whenever a value of a linear variable

It is interesting to see that we can gain 13 additional linear equations for free, due to the knowledge of $R4_1$, and the frame number. Let $R1234_1 \triangleq R1_1||R2_1||R3_1||R4_1$, where '||' denotes concatenation. We treat $R1234_1$ as a 77-bit vector, throwing away the four bits that are set to 1 during the key setup. $R1234_1$ is linear in the bits of K_c and f_1, i.e., we can write

$$R1234_1 = N_K \cdot K_c \oplus N_f \cdot f_1, \tag{3.1}$$

where N_K is a 77×64 matrix, and N_f is a 77×22 matrix that represents the key setup. The linear space which is spanned by the columns of N_k is of degree 64, but each vector in that space has 77 bits, therefore, 13 linear equations always hold on $N_K \cdot K_c$; let H_K be the matrix 13×77 that expresses these equations, i.e.,

$$H_K \cdot N_K = 0,$$

where 0 is the 13×64 zero matrix. We multiply Equation (3.1) on the left by H_K:

$$H_K \cdot R1234_f = H_K \cdot N_K \cdot K_c \oplus H_k \cdot N_f \cdot f_1 = H_K N_f \cdot f_1.$$

We can divide H_K into two parts H_K^L and H_K^R such that

$$H_K \cdot R1234_f = H_K^L \cdot R123_f \oplus H_K^R \cdot R4_f,$$

where $H_K = H_K^L||H_K^R$, H_K^L is 13×61 (the leftmost 61 columns of H_K), H_K^R is 13×16 (the rightmost 16 columns of H_K), and $R123_f = R1_f||R2_f||R3_f$. It follows that

$$H_K N_f \cdot f_1 = H_K \cdot R1234_f = H_K^L \cdot R123_f \oplus H_K^R \cdot R4_f,$$

which we can reorganize to:

$$H_K^L \cdot R123_f = H_K N_f \cdot f_1 \oplus H_K^R \cdot R4_f.$$

Namely, given $R4_1$ and the relevant COUNT (i.e., f_1), we gain 13 linear equations (H_K^L) over the bits of registers $R1$, $R2$, and $R3$.

We summarize the attack of this section as follows: we try all the 2^{16} possible values for $R4_1$, and for each such value, we solve the linearized system of equations that describe the output bits for four frames. The solution of each system gives us a suggestion for the internal state of R1, R2, and R3, which together with R4 is a

x_i is discovered, any quadratic variable of the form $x_i \cdot x_j$ can be simplified to 0 or x_j depending whether $x_i = 0$ or $x_i = 1$, respectively. The XL algorithm [30] can also be used in cases of scarce data.

suggestion for the full internal state. Most of the $2^{16} - 1$ wrong states can be easily identified due to inconsistencies in the Gauss elimination. If two or more consistent internal states remain, they are verified by trial encryptions.

The time complexity of the attack is as follows: There are 2^{16} guesses of the value of $R4_f$. For each guess, we solve a linear binary system of 656 variables, which is about $656^3 \approx 2^{28}$ XOR operations. Thus, the total complexity is about 2^{44} bit-XOR operations. When performed on a 32-bit machine, the complexity is 2^{39} register-XOR operations.

An implementation of this algorithm on a Linux 800MHz Pentium III personal computer finds the internal state within about 40 minutes, and requires relatively small amount of memory (holding the linearized system in memory requires 656^2 bits $\approx 54KB$).

3.3.3 An Optimized Attack on A5/2

We now describe an optimized implementation of the attack. The optimized version of the attack finds K_c in a few milliseconds of CPU time, and uses precomputed tables stored in memory. However, it requires slightly more data compared to the un-optimized attack.

The key idea of the optimized attack is similar to the one used in 3.3.1 for a faster attack: In a precomputation phase, we compute the dependencies that occur during the gauss elimination of the system of equations for each $R4_1$ value. Then, in the realtime phase, we filter for the correct $R4_1$ value by applying the consistency checks on the known keystream, and keeping only the $R4_1$ values that are consistent with the keystream.

In other words, we perform a precomputation phase, in which we calculate the equation systems for all values of $R4_1$ in advance. We solve each such system in advance, i.e., given a system of equations $S_{R4_1} \cdot S_1 = k$, we compute a "solving matrix" T_{R4_1}, such that $T_{R4_1} \cdot S_{R4_1}$ is the result of Gauss elimination of S_{R4_1}. Since S_{R4_1} not only depends on $R4_1$ but also on the difference between the COUNT values of the frames, we have to perform the precomputation for several COUNT value differences, as we discuss later. In the realtime phase, we calculate $t = T_{R4_1} \cdot k$ for each value of $R4_1$. The first elements of the vector t are the (partially solved) variables in S_1, but as some of the equations are linearly dependent (described in Section 3.3), the remaining elements of t should be zeros (representing the dependent equations). Therefore, we check that the last elements in t are indeed zero, i.e,. that the keystream k is consistent with the tested value for $R4_1$. Once a consistent value for $R4_1$ is found, we can verify it by calculating the key and performing trial

encryptions. In an even faster implementation, we do not need to hold in memory the entire matrices T_{R4_1}. We only hold the last rows $T^0_{R4_1}$ of the matrices T_{R4_1}, i.e., the rows that correspond to the zero elements in t). Then, to verify consistency of a value $R4_1$, we only need to check that $t' = T^0_{R4_1} \cdot k$ is a vector of zeros. We do not need to keep more than 16 rows in $T^0_{R4_1}$, as 16 would ensure that on the average case there would be two values of $R4_1$ that are consistent, one of them is the correct $R4_1$.

We now analyze the time and memory complexity of the attack using a single precomputed table (for a single difference between the COUNT value of the frames). The time that is required for the precomputation is comparable to performing the un-optimized attack, i.e., takes about 40 minutes on our computer. In the realtime phase, we must keep the filtering matrices in volatile memory for fast operation. A single system matrix is about $456 \cdot 16$ bits, thus, about 60 MBs are required to hold the table for the 2^{16} possible values of $R4_1$. Additional $64 \cdot 456 \cdot 2^{16} \approx 240$ MBs are required to hold the matrices that are used to find the full internal state given $R4_1$ and the keystream. However, these matrices can be stored on hard-disk. The attack time is about 250 CPU cycles for multiplying and checking a single matrix, or about 16M cycles in total (a few milliseconds on a personal computer). The limiting factor is the bus speed between the memory and the CPU. After finding an $R4_1$ candidate, loading the relevant solution matrix from disk takes another few tens of milliseconds (and a negligible time to find K_c). In our implementation, the attack takes less than a second on a personal computer.

As we mentioned, S_{R4_1} depends on the value of $R4_1$ and on the difference between the COUNT value of the different frames, i.e., when we perform the precomputation, we must know the XOR difference between the COUNT values of the frames. The difference between the COUNT values is used while translating the sets of variables S_2, S_3, and S_4, to S_1.

We satisfy the requirement of knowing in advance the XOR difference between the COUNT values of the frames as follows: We perform the precomputation several times, for different possible difference, and store the results in different tables. Then, in the real time phase, we use the tables that are appropriate for the COUNT values of our frames. If we are given known keystream for frames with COUNT values that is not covered by our precomputation, then we are forced to abandon this keystream, and wait a for keystream with COUNT difference as we precomputed.

From this point to the end of the section, we give a technical example of a real GSM channel and how we deal with the requirement of knowing in advance the XOR difference between COUNT values. Consider the downlink of the SDCCH/8 channel (see Appendix 3.12 for more details about the channel). This channel is used many times in GSM call initiation, even before the mobile phone rings. In this channel, a

message is transmitted over four consecutive frames out of a cycle of 51 frames. The four frames are always transmitted on the same values of the frame number modulo 51 and starting when the two least significant bits of the frame number modulo 51 are zero. Clearly, the frame number modulo 26 can take any value between zero to 25 (and it is actually decreased by one every cycle as $51 \equiv -1 \pmod{26}$). Let fr denote the first frame number of these four frames, i.e., the four frames are $f_1 = fr$, (and the two lower bits of $fr \pmod{51}$ are zero) $f_2 = fr+1$, $f_3 = fr+2$, and $f_4 = fr+3$. Detailed analysis shows that by repeating the precomputation for specific 13 values of fr mod 26, a success rate of 100% is reached. Alternatively, we can perform the precomputation for only some of the values, and discard some frames until the received frames match the ones meeting the pre-computed conditions.

During the precomputation for a specific fr in the downlink SDCCH/8, the differences $fr \oplus f_2 \pmod{26}$, $fr \oplus f_3 \pmod{26}$, and $fr \oplus f_4 \pmod{26}$ must be fixed. By performing precomputation for the cases where the lower bits of fr mod 26 are $00, 001, 010$, and 011 we cover the XOR-difference for the cases where the first frame number fr modulo 26 is $0, 1, 2, 3, 4, 8, 9, 10, 11, 12, 16, 17, 18, 19, 20$. When the lower bits $fr \pmod{26}$ are 0101, we cover the cases where fr mod 26 is: 5 and 21. When the lower bits fr mod 26 are 0110, we cover fr mod 26 values 6 and 22. We cover each of the following fr mod 26 values by its own: 7, 13, 14, 15, 23, 24, 25. Thus, by repeating the precomputation 13 times we build a full coverage, i.e., given the output of A5/2 for four consecutive frames, we use the relevant precomputed tables to perform the attack. Alternatively, we can perform precomputation only for some of the possible values of fr mod 26, and during the attack, discard frames until we reach a set of four frames whose differences are covered by the precomputation. For example, if we precompute the equation systems for the cases where the lower bits of fr mod 26 are 00, then the following fr mod 26 values are covered by the tables: $0, 4, 8, 12, 16, 20$. The worst case is when fr mod 26 equals 25. In this case, the next quartets of frames begin with fr mod 26 of $24, 23, 22, 21$, i.e., we throw five quartets of frames, and perform the attack using the sixth quartet for which fr mod 26 equals 20 (i.e., we waste about 1.1 second of data).

In the above example of the SDCCH/8, a full optimized implementation requires the keystream of four consecutive frames. After a one-time precomputation of about $40 \cdot 13 = 520$ minutes, and using 780 MBs of RAM, and another 3.1 GBs on disk, the attack works in less than a second. Note that we can refrain from saving the K_c matrices, and thus save 3.1 GBs on the hard-disk, and in return recompute the system of equations for the correct $R4_1$, once found (in this case the total attack time is still less than one second on a personal computer).

3.4 An Instant Ciphertext-Only Attack on A5/2

In this section, we transform the attacks of Section 3.3.2 and Section 3.3.3 to a ciphertext-only attack on A5/2.

GSM must use error correction to withstand reception errors. However, in the GSM protocol a message is first subjected to an error-correction code, which considerably increases the size of the message. Only then, the coded message is encrypted and transmitted (see [40, Annex A]). This transmission path contradicts the common practice of first encrypting a message, and only then subjecting it to error-correction codes. Some readers may wonder how it is even possible to correct errors (on the reception path) after decryption, as decryption often causes single bit errors to propagate through the entire message. However, since GSM decrypts by bitwise XORing the keystream to the ciphertext, an error in a bit before decryption causes an error in the corresponding bit after decryption, without any error-propagation. This trick of reversing the order of encryption and error-correction would not have been possible if a block-cipher was used for encryption. Subjecting a message to error-correction codes before encryption introduces a structured redundancy in the message, which we use to mount a ciphertext-only attack.

There are several kinds of error-correction methods that are used in GSM, and different error-correction schemes are used for different channels (see [36] for exact description of GSM channel coding). For readers unfamiliar with GSM channels, we recommend reading Appendix 3.12. However, most of this section is intelligible without reading the appendix.

We focus on the error-correction codes of the Slow Associated Control Channel (SACCH), which is also used in the SDCCH/8 channel. Both channels are commonly used in the beginning of the call. Other channels are used in other stages of the conversation, and our attack can be adapted to these channels (although it's enough to find the key on the SDCCH/8 at the beginning of the call, as the key does not change during the course of a conversation).

In the SACCH, the message to be coded with error-correction codes has a fixed size of 184 bits. The result after the error-correction codes are employed is a 456-bit long message. The 456 bits of the message are then interleaved, and divided into four frames. These frames are then encrypted and transmitted.

The coding operation and the interleaving operation can be modeled together as a multiplication of the message (represented as a 184-bit binary vector, and denoted by P) by a constant 456×184 matrix over $GF(2)$, which we denote by G, and XORed to a constant vector denoted by g. The result of the coding-interleaving operation is: $M = (G \cdot P) \oplus g$. The vector M is divided into four data frames. In the

encryption process, each data frame is XORed with the output keystream of A5/2 for the respective frame.

Since G is a 456×184 binary matrix, there are $456 - 184 = 272$ equations that describe the kernel of the inverse transformation. The dimension of the kernel is exactly 272 due to the properties of the matrix G. In other words, for any vector $M \oplus g$, such that $M = G \cdot P \oplus g$, there are 272 linearly independent equations on its elements. Let H be a matrix that describes these 272 linear equations, i.e., $H \cdot (M \oplus g) = 0$ for any such M (In coding theory such H is called the parity-check matrix).

We now show how to use the redundancy in M to mount a ciphertext-only attack. The key observation is that given the ciphertext, we can find linear equations on the keystream bits. Recall that the ciphertext C is computed by $C = M \oplus k$, where $k = k_1||k_2||k_3||k_4$ is the keystream of the four frames, and "||" denotes concatenation. We use the same 272 equations on $C \oplus g$, namely:

$$H \cdot (C \oplus g) = H \cdot (M \oplus k \oplus g) = H \cdot (M \oplus g) \oplus H \cdot k = 0 \oplus H \cdot k = H \cdot k.$$

Since the ciphertext C is known (and g is fixed and known), we actually have linear equations over the bits of k. Note that the linear equations are independent of P — they depend only on k. Thus, we now have a linear equation system over the bits of the keystream. For each guess of $R4_1$, we substitute each bit of k in this equation system with its description as linear terms over V_1 (see Section 3.3.2), and thus get a system of equations on the 656 variables of V_1. Each 456-bit coding block provides 272 equations, hence after two blocks, we have more than 450 equations. In a similar way to the attack of Section 3.3.2, we perform Gauss elimination, and about 450 equations are enough to find the value of all the original linear variables in V_1. K_c is then found by inverting the key setup of A5/2.

The rest of the details of the attack and its time complexity are similar to the case in the previous sections. The major difference is that in the known-plaintext attacks we know the keystream bits, and in the ciphertext-only attack, we know only the value of linear combinations of keystream bits (through the ciphertext and error-correction codes). Therefore, the resulting equations in the ciphertext-only attack are the linear combinations of the equations in the known-plaintext attack: Let $S_{R4_1} \cdot V_1 = k$ be a system of equations from Section 3.3.3, where S_{R4_1} is the system's matrix. In the ciphertext-only attack, we multiply this system by H on the left as follows: $(H \cdot S_{R4_1}) \cdot V_1 = (H \cdot k)$. Recall that H is a fixed known matrix that depends only on the coding-interleaving matrix G, and that $H \cdot k$ is computed from the ciphertext as previously explained. Therefore, we can solve this system and continue like in previous sections. In the known-keystream attack, we try all the

2^{16} possible equation systems S. In the ciphertext-only attack, we try all the 2^{16} possible equation systems $H \cdot S_{R4_1}$ instead. In the pre-computation of the optimized ciphertext-only attack, for such system we find linear dependencies of rows by a Gauss elimination. In the real-time phase of the ciphertext-only attack, we filter wrong values of $R4_1$ by checking if the linear dependencies that we found in the pre-computation step hold on the bits of $H \cdot k$.

A technical difference between the ciphertext-only attack and the known plaintext attacks is that while four frames of known plaintext provide enough equations, about eight ciphertext frames are required in the ciphertext-only attack. The reason is that in the ciphertext-only attack from 456 bits of ciphertext, we extract only 272 equations. A consequence of using eight frames instead of four in the optimized version of the attack is that the constraint on the XOR differences of the frame numbers is stronger, as we need to know in advance the XOR differences between eight frames (instead of four in the case of known-keystream). This constraint has a very slight implication, for example, in the case of the SDCCH/8 channel, it increases the number of precomputations that need to be performed to 16 (compared to 13 in the optimized known-plaintext attack). However, depending on the attack configuration, with a small probability we might need extra four frames of data (as T1 might change, see Appendix 3.12).

We summarize that the time complexity of an optimized ciphertext-only attack is identical to the case of the optimized known-plaintext attack. The preprocessing and memory consumption of the optimized attack (in case of downlink SDCCH/8 channel) is $16/13 \approx 1.23$ times the respective complexity of known plaintext attack. We have implemented a simulation of the attack, and verified these results.

Our methods allow to enhance the attack of Goldberg, Wagner, and Green and the attack of Petrović and Fúster-Sabater to ciphertext-only attacks. We give a description of the enhancement of Goldberg, Wagner, and Green's attack in Appendix 3.11.

3.5 Withstanding Errors in the Reception

A possible problem in a real-life implementation of the attacks is the existence of radio reception errors. A single flipped bit might fail an attack (i.e., the attack ends without finding K_c). Once the attack fails, the attacker can abandon the problematic data, and start again from scratch. But in a noisy environment, the chances are high that the new data will also contain errors. An alternative approach that we present in this section is to correct these errors.

Two kinds of reception error can occur: flipped bits, and erasures. A flipped bit is

a bit that was transmitted as "1" and received as "0", or vice versa. Erasures occur when the receiver cannot determine whether a bit is "1" or "0". Many receivers can report erased bits (rather than guessing a random value).

A possible inefficient algorithm to correct reception errors exhaustively tries all the possibilities for errors. For flipped bits, we can first try to employ the attack without any changes (assuming no errors occurr), and if the attack fails we repeat it many times, each we time guess different locations for the flipped bits. We try the possibilities with the least amount of errors first. The time complexity is exponential in the number of errors, i.e., about $\binom{n}{e}A$, where A is the time complexity of the original attack, n is the number of input bits, and e is the number of errors. The case with erasures is somewhat better, as we only need to try all the possible values for the erased bits. The time complexity is thus $2^e A$, where e is the number of erasures. In the un-optimized known-plaintext attack, an erased plaintext bit translates to an erased keystream bit. Each keystream bit contributes one equation, thus, we can simply remove the equations of the erased keystream bits. If not too many erasures occur, we still have sufficiently many equations to perform the attack. However, in the optimized attack, we pre-compute all the equation systems, and thus we cannot remove an equation a posteriori. We could pre-compute the equation systems for every possible erasure pattern, but it would take a huge time to compute, and it would require huge storage. Therefore, another method is needed.

In the rest of this section, we present an (asymptotically) better method to apply the optimized attack with the presence of erasures. For simplicity, we focus on the optimized known-plaintext attack on A5/2, but note that the optimized ciphertext-only attack can be similarly improved.

Assume that e erasures occur with their locations known, but no flips. We view the keystream as the XOR of two vectors, the first vector contains the undoubted bits of the keystream (with the erased bits set to zero), and the second vector has a value for the erased bits (with the undoubted bits set to zero). Let r be the first vector. Let w_i be the i^{th} possibility (out of the 2^e possibilities) for the second vector, where i is the binary value of the concatenated erased bits. Thus, given the correct value for i, the correct keystream is $k = r \oplus w_i$.

We can find the correct value of i without an exhaustive search. Recall the consistency-check matrices T_{R4_1} of Section 3.3.3. The linear space spanned by $T_{R4_1} \cdot w_i$, where $i \in [0, \ldots, 2^e - 1]$, has a maximum dimension of e (if the columns of T_{R4_1} are linearly independent the degree is exactly e, for simplicity we assume that this is indeed the case). We denote this linear space by $\vec{T_{R4_1}}$.

We reduce the problem of finding the correct i to a problem of solving a linear system. For each candidate $R4_1$, we compute $T_{R4_1} \cdot r$. Clearly, for the correct $R4_1$

value and for the correct w_i value, $T_{R4_1} \cdot (w_i \oplus r)$ is a vector of zeros. Therefore, for the correct w_i, $T_{R4_1} \cdot w_i = T_{R4_1} \cdot r$. Thus, the problem of finding the correct i is reduced to finding the w_i that solves this equation.

An efficient way to solve such a system is as follows: First find e vectors that span the space $\vec{T_{R4_1}}$. Such e vectors are given by $b_j = T_{R4_1} \cdot w_{2^j}$, where $j \in \{0, 1, 2, \ldots, e-1\}$. Then, we define a new matrix B whose columns are the vectors b_j: $B = (b_0, \ldots, b_{e-1})$. Finally, we find the correct i by requiring that $B \cdot i = T_{R4_1} \cdot r$, and solving the system (e.g., using Gauss elimination) to find i. If inconsistencies occur during the Gauss elimination, we move on to the next candidate $R4_1$, otherwise we assume we found the value of $R4_1$ and the keystream, and use the attack to recover K_c (which is verified using a trial encryption). Note that if the degree of $\vec{T_{R4_1}}$ is smaller than e, then Gauss elimination might result in more than one option for i. In such case, the number of options for i is always less or equal to 2^e.

The number of needed rows in T_{R4_1} in order to correct e erasures is about $16 + e$: For each of the 2^{16} candidate values of $R4_1$ the e erasures span a space of at most 2^e vectors, thus, there are about 2^{16+e} candidate solutions. Therefore, the number of rows in T_{R4_1} needs to be about $16+e$ in order to ensure that only about two consistent solution remain.

The time complexity of correcting the erasures for a single candidate of $R4_1$ is composed of first calculating the matrix B and $T_{R4_1} \cdot r$, and then solving the equation system $B \cdot i = T_{R4_1} \cdot r$. Calculating B and $T_{R4_1} \cdot r$ is comparable to one full vector by matrix multiplication, i.e., about $456(16 + e)$ bit-XORs. The Gauss elimination takes about $O((16 + e)^3)$ bit-XOR operations. The processes is repeated for every possible value of $R4_1$. Thus, the time complexity is about $2^{16}(456(16+e)+(16+e)^3)$ bit-XOR operations. Assuming that ten erasures need to be corrected, the total time complexity is about 2^{31} bit-XOR operations, i.e., about three and a half times the complexity of the optimized known-plaintext attack without reception errors. A naive implementation for correcting ten erasures would take about $2^{10} \approx 1000$ times longer to execute than the optimized known-plaintext attack. It can be seen that the benefit of the method grows as the number of erasures increases because the method's time complexity is polynomial in the number of erasures, compared to an exponential time complexity in the case of the naive method.

For the ciphertext-only attack, the time and memory complexity is doubled, as the length of the required bits is doubled. Therefore, instead of working with $T^0_{R4_1}$ in memory, we would have to store $T^0_{R4_1}H$ (which is about twice as large). Using another approach, we can leave the required memory as in the optimized attack, and pay with higher time-complexity. We can store $T^0_{R4_f}$ in memory, and calculate the multiplication by H on the fly. This method increases the time complexity by a

factor of about $e + 1$ compared to the optimized ciphertext-only attack.

3.6 A Passive Ciphertext-Only Cryptanalysis of A5/1 Encrypted Communication

In this section, we generalize the attack of Section 3.4. We show how to construct passive ciphertext-only attacks on networks that use A5/1, i.e., attacks that require the attacker to receive transmissions, but do not require the attacker to transmit. This attack can be adapted to other ciphers, as long as the network performs error-correction before encryption.

The classic approach of implementing a ciphertext-only attack is guessing the GSM traffic (or control messages), thus, known plaintext is gained. In such a case, we can use one of the known-plaintext attacks on A5/1, as published in the literature. In this section, we discuss a different approach of implementing a ciphertext-only attack — using the fact that error-correction codes are employed before encryption. An advantage of this approach over the classic approach is that the attacker is not required to guess the contents of the traffic. The disadvantage is that the complexity of the attack is higher in the new approach.

We overview the process of the attack on A5/2 of Section 3.4, and generalize it. In Section 3.4, we constructed a function $H \cdot k$ of the keystream k. This function can be seen as a function $h(x)$ from the internal state x of the cipher at the first frame, where the internal state x determines the keystream k. The special property of this function is that it can also be efficiently computed from the ciphertext of any message that was encrypted using k, as $H \cdot k = H \cdot (C \oplus g)$, where g is a known constant. Therefore, we have a function $h(x)$ from the internal state x of the cipher, such that $h(x)$ can be also computed from the ciphertext. $h(x)$ was then reversed to reveal the internal state x (by guessing all possible $R4_1$ values, and solving a system of equations). We can find the key K_c from the internal state x by reversing the (linear) key setup.

We now follow the same lines to mount an attack in case A5/1 is used instead of A5/2. We begin by constructing the same function $h(x) : \{0,1\}^{64} \rightarrow \{0,1\}^{64}$ from the internal state of A5/1 just after the key setup (i.e., $H \cdot k$, where k is the keystream resulting from initial internal state x at the first frame). We would like to reverse $h(x) = H \cdot k$ to reveal the internal state x, knowing that the inversion of $h(x)$ is expected to be computationally intensive, as it includes inversion of A5/1. Given D data points (i.e., images under $h(x)$), it suffices to invert $h(x)$ for only one of them, as it would reveal K_c. Therefore, we treat $h(x)$ as if it is a random function,

and we can use a time/memory/data tradeoff from the literature to invert it. In this discussion, we use the time/memory/data tradeoff presented by Biryukov and Shamir in [20].

Time memory tradeoffs are composed of two phases: a one-time precomputation phase and a real-time phase. The time/memory/data tradeoff in [20] has a preprocessing time complexity of N/D applications of $h(x)$, where N is the search space (2^{64} in our case), and D is the number of data points $h(x)$ that are available. The real-time phase is composed of T application of $h(x)$ and $\sqrt{T}$ disk accesses. The attack has a good success rate (greater than 60%) when the parameters are on the tradeoff curve $TM^2D^2 = N^2$ and $D^2 \leq T \leq N$, where M is the disk space of the attacker divided by $2 \log_2 N$, e.g., $M = 2^{40}$ is a $2^{40} \times 128$-bit of disk space — about 17.6 terabytes (using efficient representation, the memory complexity can drop by a factor of about 3). From the tradeoff curve, it is clear that increasing the number of available data points D by a factor of 2 reduces the time complexity of the precomputation by a factor of 2, and reduces the time complexity of the real-time phase by a factor of 4. Thus, the number of available data points is an important parameter of the attack, and the attacker benefits from having many data points.

There are a few technical issues that reduce the number of available data points of our desired form. The problem is very similar to the problem of knowing the differences between COUNT value that we encounter in Section 3.3.3. At the time of the preprocessing, we must be able to derive the initial internal state of A5/1 over four frames (in case of SDCCH/8) from the initial internal state x in the first frame. In Section 3.3.3, this problem was solved by repeating the precomputation 13 times. In this section, we would not perform the precomputation several times, rather, we would wait for a data point that is covered by the precomputation, and use some other tricks.

In the rest of this section, we discuss implementations of the ciphertext-only passive attack on A5/1 under various GSM channels, and various parameters of the time/memory/data tradeoff. We compare the attacks in Table 3.1. Readers that are not interested in the technicalities of GSM can skip the rest of this section.

For comparison with our attacks, we analyze the time/memory/data tradeoff attack of [20] given a single known message (four frames).[4] The random function that is analyzed $h(x)$ is the function from internal state x to the 64 bits of output that are generated from x, i.e., the first bit of output is generated when the internal state is x. Thus, in a 114-bit frame, there are $114 - 64 + 1 = 51$ (overlapping) strings of 64 consecutive bits (the first 64 are at the beginning of the frame; the next 64 bits begin in the second bit of the frame, etc), with 51 internal states that are associated

[4] In Section 3.7 we show that it is possible to gain a known message in certain conditions.

Table 3.1: Four Points on the Time/Memory/Data Tradeoff Curve for a Ciphertext-Only attack on A5/1

ארבע נקודות על עקומת החלפת התמורות זמן/זכרון/מידע

Attacked Channel	Available Data in Coded Messages (Four Frames)	Number of 250GBs Disks	Number of PCs to Complete Preprocessing in One Year	Duration of Online Phase on a Single PC in Minutes
KP* [21]	A Single Message	≈ 200	680	3.33
SACCH**	204 (≈ 3.5 min)	≈ 200	2800	13.33
SACCH**	600 (≈ 10 min)	≈ 200	930	1.53
SACCH**	600 (≈ 10 min)	≈ 67	930	13.83
SDCCH/8	204 (≈ 64 sec)	≈ 200	2800	13.33

* Known plaintext.

** The SACCH of the TCH/FS.

with them. It is enough to recover one of these internal states, as A5/1's internal state can be rolled back efficiently. As a message is transmitted over four frames, it is enough to invert $h(x)$ on one out of the $51 \cdot 4 = 204$ available 64-bit outputs of A5/1 (i.e., $D = 204$).

The preprocessing phase invokes A5/1 $2^{64}/204$ times (therefore, it takes about 684 computer years, assuming 2^{22} applications of A5/1 per second can be performed on a personal computer). On a network of 1000 personal computers, the preprocessing can be completed in about eight months. Using about 50 terabytes of disk storage (200 disks of 250GBs, with $M \approx 2^{41.5}$), finding a key takes about 200 seconds of CPU time ($T \approx 2^{29.65}$), and about 30000 disk accesses (which takes less than a second when averaged on the 200 disks). Note that it is possible to reduce the number of disk accesses using A5/1's low sampling resistance (see [20, 21] for details).

We now analyze the ciphertext-only attack when employed on the SACCH of a TCH/FS and on an SDCCH/8 channel (see Appendix 3.12 for more details on these channels). We assume that $h(x)$ can be applied 2^{20} times every second on a personal computer, and that a random access to disk takes about 5 milliseconds.

Focus on the SACCH of a TCH/FS. In this channel, a frame is transmitted every 26 frames, therefore, the counter $T2$ (frame number modulo 26) remains fixed. The counter $T3$ (frame number modulo 51) is increased by 26 modulo 51 with each

frame of the SACCH. Note that every two frames of SACCH $T3$ is increased by one modulo 51 (as $26 \cdot 2 \equiv 1$ modulo 51).

We have to make an assumption on the frame number, such that given the internal state x of A5/1 after initialization at the first frame, we know the internal state after initialization in the other three frames of the message. We show a method that slightly loosens the assumption on the frame numbers. In the method, we use only two of the four encrypted frames. Furthermore, 20 bits of each SACCH message are fixed (the protocol requires that these bits always have the same value), therefore, we construct H with additional 20 rows, i.e., H is 292×456. While creating H, we change the order of bits in k such that $k = k_1||k_3||k_0||k_2$, where k_i are the keystream of the individual frames (we make the corresponding changes in H's columns). Since the number of rows is 292, and due to the structure of H, we can eliminate the variables of k_1 and k_3 (i.e., $114 \cdot 2 = 228$ variables) from all the rows except for the first 228 rows by using Gauss's elimination. We define the matrix H' as the rows 229–292 and columns 229–456, i.e., H' is 64×228. Using H', we define h' in a similar way to the way H defines h. Our assumption on the frame numbers is that $T1$ (the frame number divided by $26 \cdot 51 = 1326$) is the same in both the generation of k_0 and k_2, in addition we know that $T2$ remains fixed. We further assume that the value of $T3$ is even when k_0 is generated, therefore, $T3$ is larger by one in the generation of k_2 (and the two $T3$ values differ only in their LSB). These conditions are met on average about once a second. To achieve a similar tradeoff to the one given above in the BSW example, we need $D = 204$, i.e,. about three and a half minutes of conversation (since this time a single data point is four frames, compared to 51 data points in one frame in the case of known plaintext). Furthermore, the attack time, and preprocessing time is expected to take about four times longer, as the application of h' takes more CPU time than finding the output of A5/1 given an internal state. Other possible choice of parameters are given in Table 3.1.

Another example is the downlink SDCCH/8 channel with SACCH. In every cycle of 102 frames, three messages are transmitted for a specific phone (two SDCCH messages and one SACCH with the same error-correction code), i.e., about 6.37 messages a second. We would like to be able to calculate the XOR difference between of the COUNT values in the four frames that constitute the message. Therefore, our assumption on the frame numbers is that lower two bits of the counter $T3$ are zero (this part of the assumption always holds), and that the lower two bits of the counter $T2$ are zero (and the rest of the bits of $T2$ are the same in all four frames, i.e., the counter's values (not modulo 26) in the three other frames are $T2 + 1$, $T2 + 2$, and $T2 + 3$). The assumption on $T2$ holds in six out of the 26 cases, therefore, on average the assumption holds for 1.47 messages in a second. To follow

the previous tradeoff with $D = 204$, two minutes and 19 seconds are needed, which is unreasonably long data requirements for a SDCCH/8 channel on a single session. We increase D by employing a similar trick to the one we employ in the SACCH of a TCH/FS: each GSM message can contain 184 bits, but if the message is shorter the message is padded with fill bits at its end. Assume that at least 20 such bits are fill bits. It's a reasonable assumption, although not always true. We perform a similar trick to one we made for the SACCH of the TCH/FS, to construct h' from the keystream of the first two frames of the message. We modify our assumption on the frame numbers, and assume that the LSB of $T2$ is zero in the first frame, therefore, $T2$ in the second frame equals to $T2$ of the first frame with the LSB changed to 1. This assumption holds for exactly half of the possible values of $T2$, i.e., for about $6.37/2 \approx 3.18$ messages a second. To achieve the previous tradeoff of $D = 204$, we need to collect encrypted data for a duration of about $204/(3.18) \approx 64$ seconds. The data complexity can be lowered using the tradeoff curve with a price of increased preprocessing complexity, and higher time/memory complexity. Note that the available data can be taken from several conversations, as long as they are encrypted with the same key.

3.7 Leveraging the Attacks to Any GSM Network by Active Attacks

In this section, we present several attacks which are based on flaws in the GSM call-establishment protocol (which is shortly described in Appendix 3.12.1). Through these flaws, an attacker can compromise any GSM encrypted communication based on his ability to break one weak cipher of the GSM family that is supported by the victim handset. The time complexity of the new attacks are the same time complexity of breaking the weak cipher. For the sake of simplicity, we assume that the attacker wishes to compromise conversations in networks that use A5/1 through the cryptanalysis of the weaker A5/2.

Unlike the attacks of Section 3.4 and Section 3.6 which requires only tapping the communications, the attacks in this section also require the attacker to transmit, and thus, the attacker takes a greater risk of being detected. However, active attacks brings many advantages to the attacks.

The major advantage that comes with the active attacks of this section is tapping into A5/1 networks with the time complexity of breaking A5/2, but there are also other advantages. In most of the active attacks that we present, the attacker impersonates the network towards the victim handset by using a fake base station.

As the handset views the attacker as the network, the attacker controls the transmission power of the mobile phone, and command it to first use high power to reduce reception errors that can cause problems during the cryptanalysis, but then use a lower power to reduce the chances of detection. Another advantage is the freedom of choosing the channel that is used, including the time slot in the TDMA frame that is allocated to the mobile. The attacker can use this freedom to reduce the complexity of the attack. For example in SDCCH/8, the uplink subchannel allocation is not as uniform as the downlink subchannel allocation. It is easier for an attacker employing a ciphertext-only attack to allocate the victim to an SDCCH/8 subchannel that he prepared for in advance (by pre-computing tables for it). The attacker can also wait a little before he commands the mobile to start encryption, such that the mobile starts encryption in a TDMA frame number that the attacker prepared for in advance (for example the attacker can precompute tables only for some values of the TDMA frame number modulo 26). For similar reasons, the attacker can also allocate a TDMA slot that is convenient to him, and he can choose the frequencies that he favors (for example, frequencies that minimize the risk of detection).

The protocol flaws that are used by the attacks are as follows:

1. The authentication and key agreement protocol can be executed between the mobile and the network at the beginning of a call, at the sole discretion of the network. The phone cannot ask for authentication. If no authentication is performed, K_c stays the same as in the previous conversation. In this case, the network can "authenticate" the phone through the fact that the phone encrypts using K_c, and thus the phone "proves" that it knows K_c.

2. The network chooses the encryption algorithm (or either not to encrypt at all).[5] The phone only reports the list of ciphers that it supports (in a message called *class-mark*.

3. The class-mark message is not protected, and can be modified by an attacker.

4. During authentication, only the phone is authenticated to the network, while there is no mechanism that authenticates the network to the phone. This fact allows for fake base-stations.[6]

[5]Note that if the conversation is not encrypted, a *ciphering indicator* in the phone might indicate the situation to the user.

[6]It should be noted that the network "authenticates" itself to the phone through the fact that it knows how to encrypt, and thus proves knowledge of K_c. This "authentication" cannot be considered a real authentication, especially since the network can choose not to encrypt. As a result, a fake base station does not need to know the encryption key.

5. There is no key separation: the key-agreement protocol is independent of the encryption algorithm that is used, and it is even independent of method of communication, i.e., K_c depends only on $RAND$ (which is chosen by the network), regardless of whether A5/1, A5/2, A5/3, or even GPRS encryption algorithms is used.

6. RAND reuse is allowed: the same $RAND$ can be used as many times as the network pleases, and for different types of communications (i.e., GSM or GPRS).

3.7.1 Class-Mark Attack

In the simplest attack on the protocol, the attacker changes the class-mark information that the phone sends to the network at the beginning of the conversation, such that the network thinks that the phone supports only A5/2. Although the network prefers to use A5/1, it must use either A5/2 (or A5/0 — no encryption), as it believes that the phone does not support A5/1. The attacker can then listen in to the conversation through the cryptanalysis of the weaker A5/2 cipher.

The attacker can change the class-mark message in several ways. He can transmit his alternative class-mark message at the same time that the victim's handset transmits the class-mark message, but using a much stronger radio signal. Thus, at the cellular tower, the attacker's signal overrides the handsets original message. As an alternative, the attacker can perform a man-in-the-middle attack (enter between the handset and the cellular tower by using a fake handset and a fake base station), such that all messages pass through the attacker. Then, he can simply replace the class-mark message with another message.

Note that some networks may decide not to select A5/2, but drop the conversation. As all phones should support A5/1, this kind of attack can be easily spotted by the network, and can be prevented by insisting that the phone uses A5/1 or dropping the conversation.

3.7.2 Recovering K_c of Past or Future Conversations

The remaining attacks are mostly based on the fact that the protocol does not provide any key separation, i.e., the key is fixed regardless of the encryption algorithm that is used. The idea behind the attacks is to use a fake base-station[7] that instructs the phone to use A5/2, and through the attack of Section 3.4 on A5/2 the value of K_c

[7]It is easy (and cheap) to build and operate a fake base station in GSM, using off-the-shelf equipment. The fact that the phone does not authenticate the network also helps.

is retrieved. As there is no key separation, this key is the same one used for the stronger cipher. Thus, the phone with A5/2 acts as an oracle for retrieving K_c.

In this section we present an attack in which we recover the encryption key of an encrypted conversation that was recorded in the past. As the encryption key might not change during next few conversation (the network might choose not to perform the key-agreement protocol), the encryption key that we obtain might be valid for future conversations.

The simplest way of decrypting recorded conversations is when the attacker has access to the SIM card of the victim. Then, the attacker can feed the SIM card with the $RAND$ that was used in the conversation. The SIM card then calculates and returns to the attacker the respective value of K_c (this attack is possible as GSM allows re-use of $RAND$s).

Clearly, it might not be easy for the attacker to gain physical access to the victim's SIM card. Instead, the following attack simulates such an access through the use of a fake base station. As a preparation for the attack, the attacker records encrypted conversations (that may be encrypted using different K_c's). At the time of the attack, the attacker initiates a radio-session with the victim phone through the fake base station. Then, the attacker initiates an authentication procedure, using the same $RAND$ value that was used during the encrypted conversation. The phone returns $SRES$, which is equal to the $SRES$ of the recorded conversation. Next, the attacker commands the phone to start encryption using A5/2. The phone sends an acknowledgement which is already encrypted using A5/2 and the same K_c that was used in the recorded conversation (as K_c is a function of $RAND$, and the $RAND$ is identical to the one in the recorded conversation). Finally, the attack employs the attack on A5/2 of Section 3.4 to obtain K_c from the encrypted response. The attack can be repeated several times for all the $RAND$s that appear in the recording.

The above attack leaves some traces, as the phone remembers the last K_c for use in the next conversation. The attacker can return the phone to its state before the attack by performing another authentication procedure using the last (legitimate) $RAND$ that was issued to the phone.

In a variation of this attack, the attacker can recover the current K_c that is stored in the phone by performing the attack, but skipping the authentication procedure. In this case, the attack does not change the state of the phone with respect to K_c. The attacker can use this K_c to tap into future conversations until the network initiates a new authentication procedure.

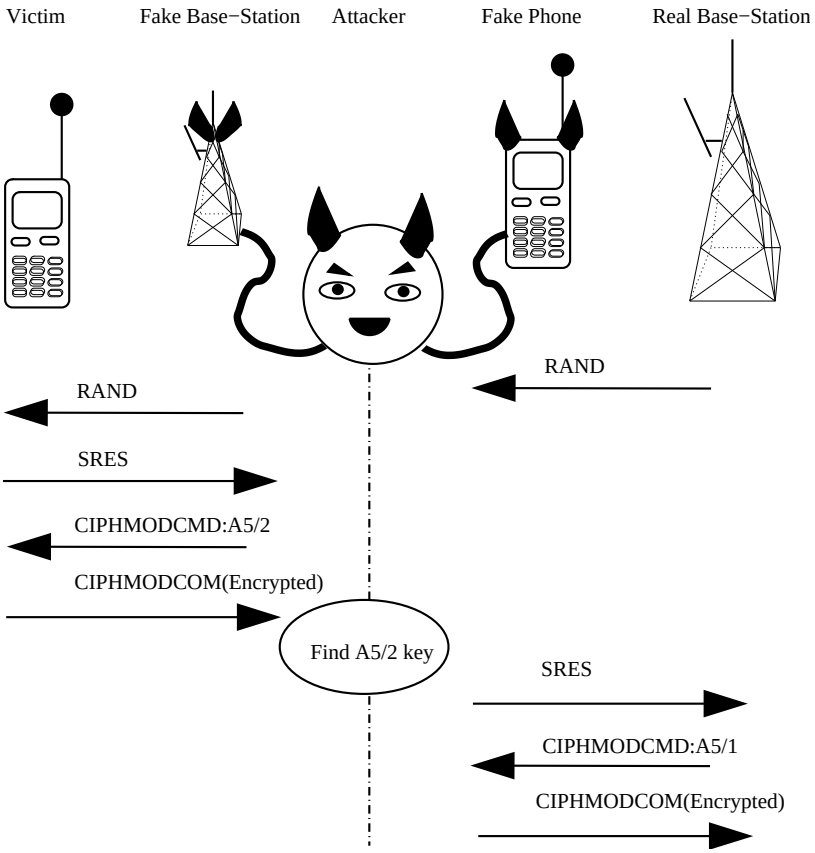


Figure 3.3: The Man-in-the-Middle Attack
התקפת האיש שבאמצע

3.7.3 Man in the Middle Attack

The attacker can tap conversations in real time by performing a man-in-the-middle attack, as depicted in Figure 3.7.3. The attacker uses a fake base-station in its communications with the mobile phone, and impersonates the mobile phone to the network. When authentication is initiated by the network, the network sends an authentication request to the attacker, and the attacker forwards it to the victim. The victim computes $SRES$, and returns it to the attacker, which holds it and does not send it back to the network, yet. Next, the attacker asks the phone to start encryption using A5/2. This request seems legitimate to the phone, as the attacker impersonates the network. The phone starts encryption using A5/2, and sends an encrypted acknowledgment. The attacker employs the ciphertext-only attack of Section 3.4

to find K_c in less than a second. Only then, the attacker returns $SRES$ to the network. Now, when the attacker is "authenticated" to the network, the network asks the attacker to start encryption using A5/1. The attacker already knows K_c, and can send the response encrypted using A5/1 under the correct K_c. From this point on, the network views the attacker as the mobile phone, and the attacker can continue the conversation, relay the conversation to the mobile, etc. It should be clear that the same attack applies when using A5/3 instead of A5/1, and we note that although A5/3 can be used with key lengths of 64–128 bits, the current GSM standard only allows the use of 64-bit A5/3.

Some readers may suspect that the network may identify this attack, by identifying a small delay in the authentication procedure. However, the GSM standard allows 12 seconds for the mobile phone to complete his authentication calculations and to return an answer, while the delay incurred by this attack is less than a second.

Another issue that might concern some readers is whether the amount of information available from the mobile is suffices to mount the ciphertext only attack of Section 3.4. After the attacker asks the mobile to start encryption using A5/2, the mobile must reply with (an encrypted) *Cipher mode complete* (CIPHMODCOM) message, which acts as an acknowledgment that encryption has started. This message is 456 bits long (after the error-correction coding takes place). It is enough for a known-plaintext attack, but the ciphertext-only attack of Section 3.4 requires two such messages. Note that the attacker cannot acknowledge the CIPHMODCOM message, as he needs K_c for that. Therefore, he can wait for the retransmission mechanism of the mobile phone to transmit the encrypted CIPHMODCOM message again. Thus, the attacker obtains two differently encrypted messages, enough for the ciphertext-only attack.

It should be noted that the retransmission mechanism of GSM ensures that the CIPHMODCOM is retransmitted immediately (in the first opportunity) after the first CIPHMODCOM not acknowledged by the network, as the size of the transmission window is one. Therefore, the same message (CIPHMODCOM) is retransmitted by the mobile (but under a different frame number), and only one message bit is changed from zero to one to indicate that the message is a retransmission. As a result, not only do we gain another encrypted message, but we also gain 184 extra bits of information, which we can express as 184 extra equations for the attack of Section 3.4 (but we can apply the attack even without these extra equations). For full details on the data-link layer of GSM, we refer the reader to [38].

It appears that with a small preparation, we can infer the plaintext of the CIPHMODCOM and use the known-plaintext attack of Section 3.3.3. The contents of the CIPHMODCOM message that the mobile returns is known or can be easily

derived, except for an optional field called IMEISV. When the network asks the mobile to start encryption, it can ask that the phone's 64-bit IMEISV — International Mobile Equipment Identity (the hardware number of the phone) plus the Software Version — would be included in the CIPHMODCOM that the phone returns. If the network does not ask the phone to include the IMEISV, then the entire contents of CIPHMODCOM can be inferred from the previous un-encrypted messages.

For the case that the network asks for the IMEISV, the attacker can find the IMEISV of a victim phone by some preparation. The IMEISV does not change unless the phone is replaced, or its software is upgraded. In the preparation work, the attacker can ask the mobile (through a fake base station) not to encrypt, but to include its IMEISV. Thus he gains the IMEISV, and in future attacks he can employ the known-plaintext attack of Section 3.3.3. Alternatively, the attacker can ask the mobile to encrypt, but not to include the IMEISV, and employ the known-plaintext attack to find K_c. Then, the attacker releases the connection, and initiates a new connection skipping the authentication, this time the attacker asks the mobile to encrypt using A5/2 and to include the IMEISV. Since K_c is known from the previous section, the attacker gains the IMEISV for future attacks. It should be noted that the known plaintext that is achieved through guessing the CIPHMODCOM can be used for attacks on other GSM ciphers, such as A5/1. For a full description of the CIPHMODCOM message, see [37].

A possible pitfall of the attack is that some networks employ protective measures that spot the event that two radio sessions are maintained from a single identity. This event implies that the phone has been cloned, and the network freezes the subscriber's account. This kind of event might occur during the establishment of a man-in-the-middle attack, when the attacker impersonates the phone to the network, but lost the acquisition on the mobile victim, which holds another radio-session. It is very easy to avoid this event if the attacker identifies (as the victim) to the network, only after he has an active radio-session with the victim. The GSM protocol also allows the attacker to prevent the mobile from accessing (non-faked) base station, by noting to the mobile that there are no other base stations except the faked one.

3.7.4 Attack on GPRS

GPRS can be attacked by an active attack, due to the fact that there is no key separation between voice conversation and GPRS data, even if the ciphers used in GPRS are secure. For example, the attacker can listen in to the *GPRS-RAND* sent by the network to the handset, while impersonating the voice network towards

the handset.[8] Then, the attacker initiates a radio session on the voice network with the handset and performs the attack that retrieves the K_c using $RAND = GPRS\text{-}RAND$. As GPRS uses the same SIM (with the same algorithms and without any key separation from regular GSM), K_c equals $GPRS\text{-}K_c$. The attacker can now decrypt/encrypt the customer's GPRS traffic using the recovered K_c. Alternatively, the attacker can record the customer's traffic, and perform the impersonation at any later time to retrieve the $GPRS\text{-}K_c$. Then, the recorded data can be decrypted. It is rumored that the first two GPRS encryption algorithms (which are kept in secret) are weaker than the newer ones. If indeed they are weak, it is also possible to mount the attack the other way round, finding $GPRS\text{-}K_c$, and using it to decrypt voice communication.

3.8 Possible Attack Scenarios

The attacks presented in this chapter can be used in several scenarios. In this section, we present four of the scenarios: call wire-tapping, call hijacking, altering of data messages (SMS), and call theft — dynamic cloning.

3.8.1 Call Wire-Tapping

The most naive scenario that one might anticipate is eavesdropping conversations in real-time. Communications encrypted using GSM can be decrypted and eavesdropped by an attacker, once the attacker has the encryption key. The attacker can tap voice conversation, but he can also tap data conversations and SMS messages. The attacker can tap video and picture messages that are sent over GPRS, etc. Real-time eavesdropping on A5/2 networks can be performed using a passive attack on A5/2 as shown in Section 3.4. On networks using encryption other than A5/2, the man-in-the-middle attack of Section 3.7 is required, or the passive attack of Section 3.6 can be used (but with a very long precomputation, and a very large storage).

In another possible wire-tapping attack against ciphers such as A5/1, the attacker records the encrypted conversation (making sure that he knows the $RAND$ value that that is sent unencrypted). Then, he uses a fake base station to attack the victim phone and retrieve the respective K_c. Once the attacker has the key, he simply decrypts the conversation. Note that an attacker can record many conversations, and

[8] The handset can work with one cellular tower for regular GSM, and another cellular tower for GPRS.

with subsequent later attacks recover all the keys. This attack has the advantage of transmitting only in the time that is convenient for the attacker. Possibly even years after the recording of the conversation, or when the victim is in another country, or in a convenient place for the attacker.

3.8.2 Call Hijacking

While a GSM network can perform authentication at the initiation of the call, encryption is the means of GSM for preventing impersonation at later stages of the conversation. The underlying assumption is that an imposter does not have K_c, and thus cannot conduct encrypted communications. Using our passive attacks, the attacker can obtain the encryption key. Once an attacker has the encryption keys, he can cut the victim off the conversation (by transmitting a stronger signal, for example), and impersonate the victim to the other party using the retrieved key. Therefore, hijacking the conversation after authentication is possible. Hijacking can occur during early call-setup, even before the victim's phone begins to ring. The operator can hardly suspect that an attack is performed. The only clue of an attack is a moment of some increased electro-magnetic interference.

In another way of call hijacking, the attacker mounts the man-in-the-middle attack. Then, at any point in time (even before the phone rings), the attacker can disconnect the victim handset and take over the conversation (including forwarding the conversation to another location).

3.8.3 Altering of Data Messages (SMS)

Once a call has been hijacked, the attacker decides on the content, including on the content of SMS messages (which are encrypted by the same K_c as the speech). The attacker can eavesdrop on the contents of a data message being sent by the victim (or being received), and send his own version instead. The attacker can also stop the message from being received, or even send his own SMS message, thus compromising the integrity of GSM traffic.

3.8.4 Call Theft — Dynamic Cloning

GSM was believed to be secure against call theft due to the authentication procedures of A3A8 (at least for operators that use a strong primitive for A3A8 rather then COMP128).

However, due to the weaknesses discussed in this chapter, an attacker can make outgoing calls on the expense of a victim. When the network asks for authentication, the attacker performs the attack that uses the victim's phone as an oracle for obtaining the $SRES$ and K_c for the given $RAND$ (as described in Section 3.7): the attacker initiates an outgoing call to the cellular network in parallel to a radio session to a victim. When the network asks the attacker for authentication, the attacker asks the victim for authentication, and relays the resulting authentication back to the network. The attacker then recovers K_c as described in Section 3.7. Now the attacker can close the session with the victim, and continue the outgoing call to the network. This attack is hardly detectable by the network, as the network views it as normal access. The victim's phone does not ring, and the victim has no indication that he is a victim (until his monthly bill arrives).

3.9 How to Acquire a Specific Victim

We distinguish between attacks that are targeted against a specific victim (e.g., eavesdropping), and attacks that are not targeted against a specific victim (e.g., call-theft). When performing eavesdropping, the attacker is usually interested in a specific victim which he targets. However, in call theft, the attacker's aim is to steal calls, and he does not care whether victim A pays the bill, or victim B pays the bill, as long as the attacker does not pay. This section focuses on targeting a specific victim.

GSM includes a mechanism that is intended to provide protection on the identity of the mobile phone. Each subscriber is allocated a TMSI (Temporary Mobile Subscriber Identity) over an encrypted link. The TMSI can be reallocated every once in a while, in particular when the subscriber changes his location. The TMSI is used to page the subscriber on incoming calls and for identification during the un-encrypted part of a session. On first sight, it seems that an attacker that performs eavesdropping with cryptanalysis using one of the methods of the previous sections can follow the decrypted data, and obtain the TMSI of his targeted victim. However, the fixed identification of a mobile is its International Mobile Subscriber Identity (IMSI), which might be unknown to the attacker. If both the IMSI and TMSI are unknown to the attacker, he may be forced to listen in to all the conversations in the area until he recognizes the victim's voice.

The attacker might only have the victim's phone number, and wish to associate the phone number with the subscriber's IMSI or TMSI. There are several possible solutions to this problem: In one solution the attacker calls the victim's phone, and pretend it to be a mistake in dialing. By monitoring all communications in the area

the attacker can distinguish the victim's phone, by recognizing his own caller ID, for example. Another more covert solution is to send a malformed SMS message to the target phone. For example, the attacker can send an SMS message as if it is part of a multi-part SMS message, but actually send only one part of the SMS. This part is received in the victim's phone, but since the entire SMS message is never fully received, the phone does not indicate to the user of the received SMS. However, the SMS passes through the radio-interface, and thus the victim can be identified. This solution can also be used as a source of known-plaintext, even during a call (when an SMS is transmitted during a call on a voice channel, an un-encrypted flag signals that data is transmitted instead of voice. If the SMS is transmitted on the SACCH, the attacker would have to guess on which bursts the SMS is carried). The attacker might be successful in identifying the victim's TMSI by correlating the paging information on the serving base station with, for example, the SMS that the attacker sends.

When performing an active attack, the attacker needs to lure the mobile into his own (fake) base station. The luring is accomplished by a suitable choice of the parameters of the fake base station, causing the victim mobile to prefer the attacker's base station. However, the fake base station might lure "innocent" handsets in addition to the victim handset. Therefore, the acquisition is composed of four phases:

1. luring many mobiles including the victim,
2. sensing the victim,
3. isolating the victim, and
4. returning the "innocent" mobiles back to the original network.

The sensing of the victim can be performed in a few ways. One way to sense the victim is to set a parameter called the *location area* of the fake base station to be different than the surrounding legitimate base stations. Once lured, the mobile has to perform a procedure called *location area update*, which includes contacting the fake base station and identifying (a mobile must perform location area update when switching between base stations with different values of the location area parameter). Another way (assuming the TMSI or the IMSI is known) is to use the same location area, and to page the victim in the fake base station using its TMSI/IMSI until the victim responds (once the victim handset is parked on the fake base station, it must respond). If the TMSI/IMSI is not known, the attacker can use the radio-session of the location area update to interrogate the mobile for its IMSI (if only the TMSI is known), or to perform an acquisition as previously described. The attacker can relay

the paging messages of the real network to the lured mobiles, so they do not miss incoming calls.

The next steps for the attacker are to isolate the victim and return the "innocent" handsets to the real network. The isolation can be performed by changing the fake base station parameters, such that it transmits on its beacon frequency that the fake base station is the only cell in the area. This change prevents the lured mobiles from switching to another base stations. The attacker can now page the victim to make sure that the victim is still parked on the fake base station.

Next, the attacker returns the "innocent" handsets back to the real network by initiating a radio-session with each one of them, and return them to the real network: During the radio session, the handsets are made to believe that they are handed-over to a neighbor base station, while actually the attacker uses another transceiver (fake base station without the beacon frequency) to impersonate that neighbor base station. After the "handover" is complete, the radio-session is released, and the "innocent" mobile returns to the real neighbor base station. In another option for returning innocent mobiles to the real network, the attacker establishes a radio-session with the victim, and "scares away" all the other mobiles, for example by stopping transmission on the beacon frequency. After a short time, the beacon can be restored with parameters that are unlikely to attract mobiles, but claiming to be the only base station in the area. Before releasing the radio-session with the victim, the victim is handed over to the fake base station with the new parameters. Accidental entrance of other mobiles to the base station can be identified using a different location area for the fake base station, and a radio session can then be established with these mobiles, during which they are returned to the real network. It is stressed that a correct choice of parameters for the fake-base station should almost entirely eliminate accidental entries to the base station.

3.10 Summary

In this chapter, we present new methods for attacking the encryption and the security protocols used by GSM and GPRS. The described attacks are easy to apply, and do not require knowledge of the conversation. We stress that GSM operators should replace the cryptographic algorithms and protocols as soon as possible, or switch to the more secure third generation cellular system (although it still possess some of the weaknesses described in this chapter).

Even GSM networks that use the new A5/3 succumb to our attacks. We suggest to change the way A5/3 is integrated into GSM, in order to protect the networks from such attacks. A possible correction is to make the keys used in A5/1 and A5/2

unrelated to the keys that are used in A5/3. The integration of GPRS suffers from similar flaws that should be taken into consideration.

We would like to emphasize that our ciphertext-only attack is made possible by the fact that the error-correction codes are employed before the encryption. In the case of GSM, the addition of such a structured redundancy before encryption is performed crucially reduces the security of the system.

As a result of the initial publication of these attacks, the GSM association security group together with the GSM security working group are working to remove the A5/2 algorithm from handsets (which should be completed during 2006).

Acknowledgements

We are grateful to Orr Dunkelman for his great help and various comments on early versions of this work, and to Adi Shamir for his advice and useful remarks. We would like to thank David Wagner for providing us with information on his group's attack on A5/2. We also acknowledge the anonymous referees for their important comments. Finally, we would like to thank the many people that expressed their interest in this work.

3.11 Appendix: Enhancing The Attack of Goldberg, Wagner, and Green on GSM's A5/2 to a Ciphertext-Only Attack

We now describe a ciphertext-only attack on A5/2 based on Goldberg, Wagner, and Green's Attack [45]. We use the same matrix H as in Section 3.4. Recall that the attack of [45] requires the XOR difference of the keystream of two frames. The enhanced ciphertext-only attack uses eight encrypted frames. We denote the eight encrypted frames by $C_1, \ldots, C_8$, where the first four frames have consecutive frame numbers f_1, f_2, f_3, f_4, and the second four frames have consecutive frame numbers f_5, f_6, f_7, f_8. We require that f_{i+4} is exactly $51 \cdot 26 = 1326$ frames after f_i, for $i \in \{1, 2, 3, 4\}$. We also require that $f_1/1326$ is even (required by the original attack), and that $C_i, C_{i+1}, C_{i+2}, C_{i+3}$, where $i \in \{1, 5\}$, constitute an encrypted message. The latter requirement does not hold for the SACCH of the TCH/FS, due to the locations of TDMA frame numbers that can be used to transmit a SACCH message, however, it holds for the SDCCH/8 channel (an adjusted requirement can be constructed for other channels, including the TCH/FS).

Due to the reasons shown in Section 3.4, it holds that

$$H \cdot (C_1 \oplus g||C_2 \oplus g||C_3 \oplus g||C_4 \oplus g) = H \cdot (k_1||k_2||k_3||k_4),$$

where k_i is the keystream used in frame f_i. Similarly it holds that

$$H \cdot (C_5 \oplus g||C_6 \oplus g||C_7 \oplus g||C_8 \oplus g) = H \cdot (k_5||k_6||k_7||k_8).$$

Due to linearity, it holds that:

$$H \cdot ((C_1||C_2||C_3||C_4) \oplus (C_5||C_6||C_7||C_8)) = H \cdot ((k_1||k_2||k_3||k_4) \oplus (k_5||k_6||k_7||k_8)).$$

Let

$$C' = (C_1||C_2||C_3||C_4) \oplus (C_5||C_6||C_7||C_8),$$

and let

$$k' = (k_1||k_2||k_3||k_4) \oplus (k_5||k_6||k_7||k_8).$$

Therefore, $HC' = Hk'$.

The rest of the attack is similar to the attack of [45], using $Hk' = HC'$ instead of the keystream difference. Using a similar argument to the one in Section 3.3.1 and given the initial value of $R4_1$, we express the bits of the 272-bit $H \cdot C'$ as linear expressions of the bits of the initial value of $R1_1$, $R2_1$, and $R3_1$ at the first frame. The flaw observed in [45] causes $R4$ to have the same value in f_i and f_{i+4}, where $i \in \{1, 5\}$. Thus, the clockings are the same in these frames, and each bit of k_i and k_{i+4} can be expressed using exactly the same quadratic terms over the bits of $R1$, $R2$, and $R3$. The XOR difference of these terms is linear in the bits of $R1$, $R2$, and $R3$. To further simplify the analysis, we assume that the XOR difference among the frame numbers is known in advance. Since the difference between the frame numbers is known, a guess for a value for $R4$ of the first frame causes a known value for $R4$ of the other frames. In addition, the respective differences between the values of registers $R1$, $R2$, and $R3$ in the four frames are also known in advance. In this way, we can express Hk' as linear terms. It should be noted that we do not have to use the whole 272 bits of $H \cdot C'$, and actually less than a hundred bits suffices.

The attack follows a similar path as the original attack, using the redundancy to filter wrong $R4$ values. The time complexity of this attack is similar to the one of the original attack (i.e., a few milliseconds on a personal computer), and the memory requirement is also similar, i.e., about 15 MBs of volatile memory and another 60 MBs of memory that can be stored on disk. The pre-computation takes similar time. The time complexity of this enhanced attack is better than the ciphertext-only attack of Section 3.4, however, the fact that f_5 should be exactly 1326 frames after f_1 (about six seconds) limits the usability of this attack compared to the one in Section 3.4, which can complete in less than a second given eight encrypted frames.

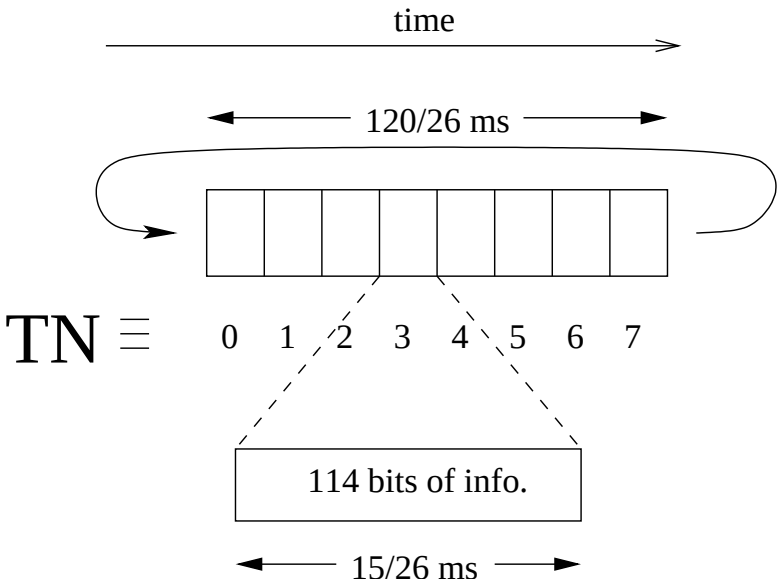


Figure 3.4: A TDMA frame
TDMA מסגרת

3.12 Appendix: Technical Background on GSM

In this appendix we describe some technical aspects of the GSM system, which are relevant to attacks presented in this chapter.

We first elaborate on the concept of a TDMA frame. In GSM the same physical channel can serve up to eight different phones, by allocating the physical channel to different phones through round-robin, where each phone transmits in a *time slot* that lasts 15/26 ms. This method is known as *Time Division Multiple Access* (TDMA). Each frame is composed of eight time slots, which are referred to by their *Time slot Number* (TN). In Figure 3.4 we depict a typical TDMA frame. Each TDMA frame has a TDMA frame number associated with it. The TDMA frame number is fixed for all the time slots in the TDMA frame, and is incremented by one before the next TDMA frame begins. In each time slots, 114 bits of information can be transmitted. Therefore, the physical channel between the network and a phone has a maximum throughput of 114 bits per TDMA frame, or 24.7 Kbits/second.[9] In this chapter,

[9]Note that the actual throughput is lower due to error-correction codes that must be employed, protocols overhead, and the fact that several logical channels between the phone and the network share the same physical channel. In GPRS, a higher data rate is accomplished by allocating several time slots to the same phone.

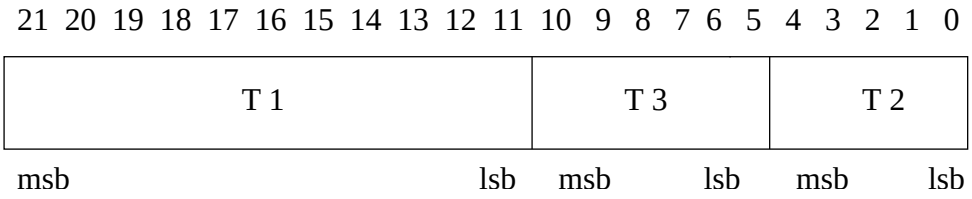


Figure 3.5: The coding of COUNT
COUNT קידוד

we always focus on the link between a single phone and the network, and therefore, when referring to a frame we refer to the data in the relevant slot for the phone in the TDMA frame.

The keystream generation (using A5) for a specific frame depends on the TDMA frame number. In Section 3.2, we describe the way that COUNT affects the A5 key setup. COUNT is derived from the TDMA frame number as shown in Figure 3.5, where $T1$ is the quotient of the frame number divided by $51 \cdot 26 = 1326$, $T2$ is the remainder of the frame number divided by 26, and $T3$ is the remainder of the frame number divided by 51. It should be noted that many times in our attacks, we know in advance the additive difference between two frame numbers, but we do not know in advance (with 100% certainty) the XOR-difference between the COUNT values of the two frames. This fact complicates our attack at certain points. Note that the above description is true only when the mobile is allocated a single time slot. When the mobile is allocated several time slots (or in GPRS), a different method is used.

There are many kinds of messages in GSM, but most of them consume 456 bits after error correction. The allocation of the 456-bit message into frames depends on the channels. Here are two extreme examples: the 456-bit message is transmitted on four consecutive frames in some channels, but there is also a channel in which the 456-bit message is transmitted over 22 frames (interleaved with other messages). In the following paragraphs, we give two examples of two specific channels. For exact description of GSM channels see [40].

The slowest dedicated channel in GSM is a Stand alone Dedicated Control CHannel (SDCCH/8), which is used mostly for signaling in the beginning of a call, or for SMS transfer (while not in a voice conversation). In this channel, the same TN is used by up to eight different mobiles, i.e., the SDCCH contains eight *subchannels* $0, \ldots, 7$. The subchannel is determined by the value of $T3$ and the LSB of $T2$. Each mobile is also allocated a Slow Associated Control CHannel (SACCH). The downlink (from the network to the mobile) frame arrangement is shown in Figure 3.6, where a

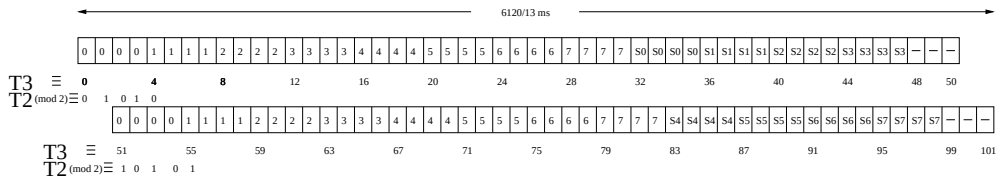


Figure 3.6: The SDCCH/8 channel — downlink.
SDCCH/8 — ערוץ יורד

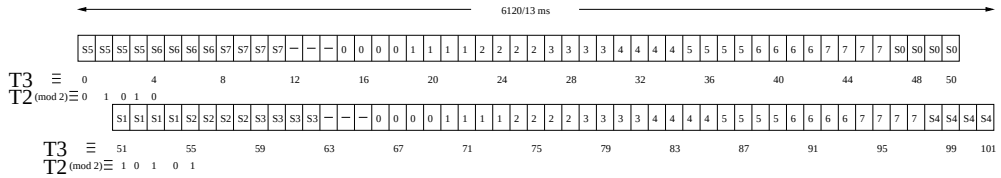


Figure 3.7: The SDCCH/8 channel — uplink.
SDCCH/8 — ערוץ עולה

number "x" denotes messages belonging to a SDCCH subchannel x, Sx denotes the SACCH of subchannel x, and an empty frame is denoted by "–". Each 456-bit message is transmitted in four consecutive frames. When $T3 \equiv 48, 49$, or 50 no frames are transmitted. The uplink frame arrangement of SDCCH/8 is shown in Figure 3.7.

Another highly-used channel in GSM is the full rate traffic channel for speech (TCH/FS), which is used to carry speech. In this channel, the 456-bit speech messages are transmitted on eight frames, using the even-numbered bits of the first four frames, and the odd-numbered bits of the second four frames (the remaining bits carry parts of the previous and next speech messages). Each mobile in TCH/FS is also allocated a SACCH channel, as shown in Figure 3.8, where a SACCH frame is

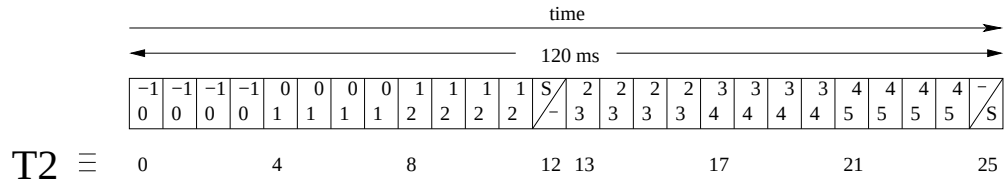


Figure 3.8: The TCH/FS.
TCH/FS-ערוץ ה

denoted by "S", a number inside a frame denotes a speech message (the value at the top of an entry denotes a speech message carried on odd-numbered bits, and the value at the bottom of an entry denotes a speech message carried on even-numbered bits), and an empty frame is denoted by "–". In each period of $T2$ one SACCH frame is transmitted, either when $T2$ is 12 or when $T2$ is 25 (using both the even-numbered bits and the odd-numbered bits), the the other frame (when $T2$ is 25 or 12, respective) is left empty. The choice if the frame in which the SACCH is transmitted depends on the LSB of the TN that is allocated to the mobile (when the LSB is zero the SACCH is transmitted when $T2$ is 12). A 456-bit SACCH message starts whenever the TDMA frame number modulo 104 equals $12 + 13 \cdot TN$. For further details on the TDMA frame number in which a message can begin, see [39].

There are many types of channels, the above are only a few examples.

3.12.1 GSM Call Establishment

Calls in GSM are established as follows:

1. (In case the call is initiated by the network:) The network pages the phone with PAGING REQUEST by its IMSI or TMSI on the cell's paging channel (PAGCH). The configuration of the PAGCH is a part of a cell's broadcast information. If the call is initiated by the mobile it starts directly from stage 2.

2. Immediate assignment procedure[10]:

   (a) The phone sends a CHANNEL REQUEST message on the random access channel (RACH). The CHANNEL REQUEST message includes a very small amount of information — only 8 bits. It does not contain an identification of the mobile, rather it includes a random discriminator (5 bits). The remaining three bits contain the establishment cause.

   (b) The network broadcasts an IMMEDIATE ASSIGNMENT message on the PAGCH. This message contains the random discriminator (and also the TDMA frame number in which the CHANNEL REQUEST was received), and the details of the channel that is allocated to the mobile (including frequency hopping information, if needed). The messages also includes other

[10]The procedure is initiated by the mobile phone. It can be triggered by a PAGING REQUEST, or by a service request originated by the mobile.

technical information such as timing advance. The mobile immediately tunes to the the assigned traffic channel.[11]

3. Service Request and Contention Resolution:

   (a) The mobile sends a service request message (e.g., paging response, service request, etc.), this message includes the TMSI of the mobile. The message also includes the mobile class-mark (including the A5 versions that are supported), and a ciphering key sequence number $(0, \ldots, 6)$.

   (b) The network acknowledges the service request message, and repeats the TMSI. The reason for repeating the TMSI is contention resolution: It is possible that two mobiles used the same random discriminator on the same TDMA frame, and therefore, both "think" that they are assigned to the same channel. The mobile that his TMSI is acknowledged by the network, stays on the channel, and the other mobile quits.

4. Authentication:[12]

   (a) The network sends authentication request (AUTHREQ). The authentication request includes a random 128-bit value RAND, and a ciphering key sequence number, in which the resulting Kc should be stored.

   (b) The mobile answers the authentication with the computed signed response (SRES), in an authentication response message (AUTHRES).

   (c) The network asks the mobile to start encryption using a cipher mode command (CIPHMODCMD). The network can specify the encryption algorithm to be used, and it specifies the encryption key by a ciphering key sequence number $(0, \ldots, 6)$. The network starts to decipher incoming communication. This message can also be used to ask the mobile to send its international mobile equipment identity, and software version (IMEISV).

   (d) The mobile starts to encrypt and decrypt, and responds with (encrypted) cipher mod complete message (CIPHMODCOM). If requested, the mobile sends its IMEISV.

[11] Unlike the PAGCH and the RACH which are uni-directional, a traffic channel is a bi-directional channel

[12] The network can choose to perform authentication every call, but may also choose to skip this procedure (and use an already existing Kc for encryption, or choose not to encrypt).

5. The network and the mobile "talk" on the channel. It might well be that the network changes the channel. For example, if it is a voice conversation the channel might need to be changed to suit a voice conversation, etc. In case a channel is changed or a handover is needed, the new channel information is sent by the network (including the frequency hopping information). Note that if the conversation is encrypted, then the new channel information is encrypted as well.

It is important to understand the concept of traffic channels in GSM. A traffic channel in GSM is composed of a list of frequencies, and frequency hopping parameters: Mobile Allocation Index Offset (MAIO), which takes a value from zero to the number of frequencies in the list minus one, and the Hopping Sequence Number (HSN), which takes a value from zero to 63. Therefore, given n frequencies there are $64n$ different hopping sequences. Usually, traffic channels in the same cell bear the same HSN and different MAIOs. After a traffic channel is assigned, the mobile and the network compute the frequency for each burst according to the above information given at the time of assignment, and according to the TDMA frame number (which is publicly known). The channel remains the same one even when encryption is turned on. The channel may be changed during the course of the conversation. In this case, the new channel parameters are passed on the current channel.

Chapter 4

Conditional Estimators: an Effective Attack on A5/1

Irregularly-clocked linear feedback shift registers (LFSRs) are commonly used in stream ciphers. We propose to harness the power of conditional estimators for correlation attacks on these ciphers. Conditional estimators compensate for some of the obfuscating effects of the irregular clocking, resulting in a correlation with a considerably higher bias. On GSM's cipher A5/1, a factor two is gained in the correlation bias compared to previous correlation attacks. We mount an attack on A5/1 using conditional estimators and using three weaknesses that we observe in one of A5/1's LFSRs (known as $R2$). The weaknesses imply a new criterion that should be taken into account by cipher designers. Given 1500–2000 known-frames (about 6.9–9.2 conversation seconds of known keystream), our attack completes within a few tens of seconds to a few minutes on a PC, with a success rate of about 91%. To complete our attack, we present a source of known-keystream in GSM that can provide the keystream for our attack given 3–4 minutes of GSM ciphertext, transforming our attack to a ciphertext-only attack.

The work described in this chapter is a joint work with Prof. Eli Biham. It was originally published in [6].

4.1 Introduction

Correlation attacks are one of the prominent generic attacks on stream ciphers. There were many improvements to correlation attacks after they were introduced by Siegenthaler [75] in 1985. Many of them focus on stream ciphers composed of one or more regularly clocked linear feedback shift registers (LFSRs) whose output is filtered

through a non-linear function. In this chapter, we discuss stream ciphers composed of irregularly-clocked linear feedback shift registers (LFSRs), and in particular, on stream ciphers whose LFSRs' clocking is controlled by the mutual value of the LFSRs. The irregular clocking of the LFSRs is intended to strengthen the encryption algorithm by hiding from the attacker whether a specific register advances or stands still. Thus, it should be difficult for an attacker to correlate the state of an LFSR at two different times (as he does not know how many times the LFSR has been clocked in between).

Assume the attacker knows the numbers of clocks that each LFSR has been clocked until a specific output bit has been produced. Then with some success probability $p < 1$, the attacker can guess the numbers of clocks that each LFSR is clocked during the generation of the next output bit. A better analysis that increases the success probability of guessing the number of clocks for the next output bit could prove devastating to the security of the stream cipher. Our proposed conditional estimators are aimed at increasing this success probability.

In this chapter, we introduce conditional estimators, aimed to increase the probability of guessing the clockings of the LFSRs correctly. We apply conditional estimators to one of the most fielded irregularly clocked stream ciphers — A5/1, which is used in the GSM cellular network. GSM is the most heavily deployed cellular phone technology in the world. Over a billion customers world-wide own a GSM mobile phone. The over-the-air privacy is currently protected by one of two ciphers: A5/1 — GSM's original cipher (which was export-restricted), or A5/2 which is a weakened cipher designated for non-OECD (Organization for Economic Co-operation and Development) countries. As A5/2 was discovered to be completely insecure [10] (see Chapter 3), the non-OECD countries are now switching to A5/1.

4.1.1 Previous Correlation Attacks on A5/1

The first correlation attack on A5/1 was publish in 2001 by Ekdahl and Johansson [35]. Their attack requires a few minutes of known-keystream, and finds the key within minutes on a personal computer. In 2004, Maximov, Johansson, and Babbage [58] discovered a new correlation between the internal state and the output bits and used it to improved the attack. Given about 2000–5000 frames (about 9.2–23 seconds of known-plaintext), their attack recovers the key within 0.5–10 minutes on a personal computer. These are not the fastest attacks, and in some scenarios other (non-correlation) attacks can perform better (for a comprehensive list of attacks on A5/1 see Chapter 3).

The attacks on GSM demonstrate that fielded GSM systems do not provide an

adequate level of privacy for their customers. However, breaking into fielded A5/1 GSM systems using previous attacks requires either active attacks (e.g., man-in-the-middle attacks), a lengthy (although doable) precomputation step, a high time complexity, or a large amount of known keystream.

One advantage of correlation attacks on A5/1 over previous attacks is that they do not require long-term storage or a preprocessing phase, yet given a few seconds of known-keystream, they can find the key within minutes on a personal computer. Another advantage of correlation attacks over some of the previous attacks is the immunity to transmission errors. Some of the previous attacks are susceptible to transmission errors, e.g., a single flipped bit defeats Golic's first attack. Correlation attacks can naturally withstand transmission errors, and even a high bit-error-rate can be accommodated for.

4.1.2 Our Contribution

In this chapter, we introduce conditional estimators, which can compensate for some of the obfuscating effects caused by the irregular clocking. Using conditional estimators, we improve the bias of the correlation equation that was observed in [58] by a factor of two. In addition, we discover three weaknesses in one of A5/1's registers. We mount a new attack on A5/1 based on the conditional estimators and the three weaknesses. Finally, we describe a source for known keystream transforming our attack to a ciphertext-only attack.

One of the weaknesses relates to the fact that register $R2$ of A5/1 has only two feedback taps, which are adjacent. This weakness enables us to make an optimal use of the estimators by translating the problem of recovery of the internal state of the register to a problem in graph theory. Thus, unlike previous attacks [35, 58], which were forced to use heuristics, we can exactly calculate the list of the most probable internal states. We note that in 1988, Meier and Staffelbach [59] warned against the use of LFSRs with few feedback taps. However, it seems that their methods are difficult to apply to A5/1.

An alternative version of our attack can take some advantage of the fact that many operators set the first bits of the key to zero (as reported in [24]); this alternative version slightly simplifies the last step of our attack, and results with a somewhat higher success rate. We are not aware of any other attack on A5/1 (except for exhaustive search) that could benefit from these ten zero bits.

Our last contribution is a new source for known-plaintext in GSM. We point at the Slow Associated Control CHannel (SACCH), which is a control channel that accompanies any voice channel, and show that its content can be derived from in-

formation which is available to the attacker. We also discuss the frequency hopping in GSM and how to overcome it. Using this new source for known-plaintext, our attacks can be converted to ciphertext-only attacks. However, this is a slow channel, that provides only about eight known frames each second.

We have performed simulations of our attacks. Given 2000 frames, our simulations take between a few tens of seconds and a few minutes on a PC to find the key with a success rate about 91%. For comparison, the simulations of [58] with a similar number of frames take about four times longer to run and achieve a lower success rate of about only 5%. A comparison of some of the results of previous works and our results is given in Table 4.1. With our new source for known keystream, the required 1500–2000 *known frames* can be obtained from the *ciphertext* of about 3–4 minutes of conversation.

4.1.3 Organization of the Chapter

This chapter is organized as follows: We give a short description of A5/1 in Section 4.2. Then, we set our notations and review some of the main ideas of previous works in Section 4.3. In Section 4.4 we describe the conditional estimators and three weaknesses, and then use them in our new attack in Section 4.5. The results of our simulations are presented in Section 4.6. We describe the new source of known-plaintext in Section 4.7. Finally, the chapter is summarized in Section 4.8.

4.2 A Description of A5/1

The stream cipher A5/1 accepts a 64-bit session key K_c and a 22-bit publicly-known frame number f. GSM communication is performed in frames, where a frame is transmitted every 4.6 millisecond. In every frame, A5/1 is initialized with the session key and the frame number. The resulting 228 bit output (keystream) is divided into two halves: the first half is used to encrypt the data from the network to the mobile phone, while the second half is used to encrypt the data from the mobile phone to the network. The encryption is performed by XORing the data with the appropriate half of the keystream.

A5/1 has a 64-bit internal state, composed of three maximal-length Linear Feedback Shift Registers (LFSRs): $R1$, $R2$, and $R3$, with linear feedbacks as shown in Figure 4.1. The basic operation of each register is called *clocking*, in which the feedback of the register is calculated (as the XOR of the feedback taps), then, the register is shifted one bit to the right (discarding the rightmost bit), and the feedback is stored into the leftmost location (location zero). The registered are clocked

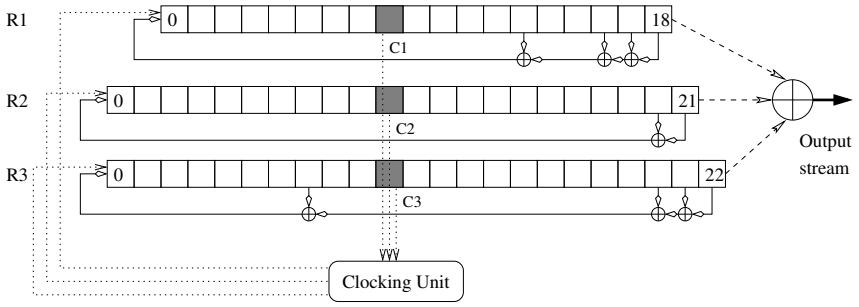


Figure 4.1: The Internal Structure of A5/1
המבנה הפנימי של A5/1

1. Set $R1 = R2 = R3 = 0$.
2. For $i = 0$ to 63
   - Clock all three registers.
   - $R1[0] \leftarrow R1[0] \oplus K_c[i]$; $R2[0] \leftarrow R2[0] \oplus K_c[i]$; $R3[0] \leftarrow R3[0] \oplus K_c[i]$.
3. For $i = 0$ to 21
   - Clock all three registers.
   - $R1[0] \leftarrow R1[0] \oplus f[i]$; $R2[0] \leftarrow R2[0] \oplus f[i]$; $R3[0] \leftarrow R3[0] \oplus f[i]$.

Figure 4.2: The Key Setup of A5/1.
אלגוריתם אתחול המפתח של A5/1

regularly during the initialization of the state with K_c and f (the *key setup*), and *irregularly* during the keystream generation, as described in detail later on.

A5/1 is initialized with K_c and f in three steps, as described in Figure 4.2, where the i'th bit of K_c is denoted by $K_c[i]$, the i'th bit of f is denoted by $f[i]$, and $i = 0$ is the least significant bit. We denote the internal state after the key setup by $(R1, R2, R3) = \text{keysetup}(K_c, f)$.

Observe that the key setup is linear in the bits of both K_c and f, i.e., once the key setup is completed, every bit of the internal state is an XOR of bits in fixed

locations of K_c and f. This observation is very helpful in correlation attacks.

The keystream generation is performed in cycles, where in each cycle one output bit is produced. A cycle is composed of irregularly clocking $R1$, $R2$, and $R3$ according to a clocking mechanism (described later), and then outputting the XOR of the rightmost bits of the three registers (as shown in Figure 4.1). The first 100 bits of output are discarded (bits $0, \ldots, 99$), i.e., the 228 bits that are used in GSM are output bits $100, \ldots, 327$. The keystream generation can be summarized as follows:

1. Run the key setup with K_c and f (Figure 4.2).
2. Run A5/1 for 100 cycles and discard the output.
3. Run A5/1 for 228 cycles and use the output as keystream.

It remains to describe the clock control mechanism, which is responsible for the irregular clocking. Each register has a special *clocking tap* near its middle (in locations $R1[8]$, $R2[10]$, and $R3[10]$). The clocking mechanism algorithm:

1. Calculate the majority of the values in the three clocking taps.
2. Then, clock a register if and only if its clocking tap agrees with the majority.

For example, assume that $R1[8] = R2[10] = c$ and $R3 = 1 - c$ for some $c \in \{0, 1\}$. Clearly, the value of the majority is c. Therefore, $R1$ and $R2$ are clocked, and $R3$ stands still.

Note that in each cycle of A5/1, either two or three registers are clocked (since at least two bits agree with the majority). Assuming that the clocking taps are uniformly distributed, each register has a probability of 1/4 for standing still and a probability of 3/4 for being clocked.

4.3 Notations and Previous Works

In this section, we set our notations, and describe some of the main ideas of the previous works. Let S_1, S_2, and S_3 be the initial internal state of registers $R1$, $R2$, and $R3$ after the key-setup using the correct K_c, where the frame number is chosen to be zero, i.e., $(S_1, S_2, S_3) = \text{keysetup}(K_c, 0)$. For $i = 1, 2, 3$, denote the output bit of Ri after it is clocked l_i times from its initial state S_i by $\tilde{S}_i[l_i]$.[1] Similarly, let F_1^j, F_2^j, and F_3^j be the initial internal state of registers $R1$, $R2$, and $R3$ after a key setup using

[1]Note that as a register has a probability of 3/4 of being clocked in each cycle, it takes about $l_i + l_i/3$ cycles to clock the register l_i times.

all zeros as the key, but with frame number j, i.e., $(F_1^j, F_2^j, F_3^j) = \text{keysetup}(0, j)$. For $i = 1, 2, 3$, denote by $\tilde{F}_i^j[l_i]$ the output of Ri after it is clocked l_i times from its initial state F_i^j. Ekdahl and Johansson [35] observed that due to the linearity of the key setup, the initial internal value of Ri at frame j is given by $S_i \oplus F_i^j$, i.e., $\text{keysetup}(K_c, j) = \text{keysetup}(K_c, 0) \oplus \text{keysetup}(0, j) = (S_1 \oplus F_1^j, S_2 \oplus F_2^j, S_3 \oplus F_3^j)$. Furthermore, due to the linear feedback of the shift register, the output of LFSR i at frame j after being clocked l_i times from its initial state is given by $\tilde{S}_i[l_i] \oplus \tilde{F}_i^j[l_i]$.

Maximov, Johansson, and Babbage [58] made the following assumptions:

1. *clocking assumption* (j, l_1, l_2, t): Given the keystream of frame j, registers $R1$ and $R2$ were clocked exactly l_1 and l_2 times, respectively, until the end of cycle t. The probability that this assumption holds is denoted by $Pr((l_1, l_2) \text{ at time } t)$ (this probability can be easily computed, see [58]).

2. *step assumption* (j, t): Given the keystream of frame j, both $R1$ and $R2$ are clocked in cycle $t + 1$, but $R3$ stands still. Assuming the values in the clocking taps are uniformly distributed, this assumption holds with probability 1/4 (the clocking mechanism ensures that if the values of the clocking taps are uniformly distributed, each register stands still with probability 1/4).

They observed that under these two assumptions, $R3$ contributes the same bit to output bits t and $t + 1$. Thus, $R3$'s contribution is eliminated from the difference of these two output bits, and the following equation holds:

$$(\tilde{S}_1[l_1] \oplus \tilde{S}_2[l_2]) \oplus (\tilde{S}_1[l_1 + 1] \oplus \tilde{S}_2[l_2 + 1]) = \\ \tilde{Z}^j[t] \oplus \tilde{Z}^j[t + 1] \oplus (\tilde{F}_1^j[l_1] \oplus \tilde{F}_2^j[l_2]) \oplus (\tilde{F}_1^j[l_1 + 1] \oplus \tilde{F}_2^j[l_2 + 1]), \quad (4.1)$$

where $\tilde{Z}^j[t]$ is the output bit of the cipher at time t of frame j (and under the two assumptions above). Thus, the value of $(\tilde{S}_1[l_1] \oplus \tilde{S}_2[l_2]) \oplus (\tilde{S}_1[l_1 + 1] \oplus \tilde{S}_2[l_2 + 1])$ can be estimated from the known keystream and the publicly available frame numbers.

Equation (4.1) holds with probability 1 if both the clocking assumption and the step assumption hold. If either or both assumptions do not hold, then Equation (4.1) is assumed to hold with probability 1/2 (i.e., it holds by pure chance). Therefore, Equation (4.1) holds with probability $(1 - Pr((l_1, l_2) \text{ at time } t))/2 + Pr((l_1, l_2) \text{ at time } t)((3/4)/2 + 1/4) = 1/2 + Pr((l_1, l_2) \text{ at time } t)/8$. The value of the bias $Pr((l_1, l_2) \text{ at time } t)/8$ is typically two to three times higher compared to the bias shown in [35]. Such a difference in the bias is expected to result in an improvement of the number of frames needed by a factor between four and ten, which is indeed the case in [58].

We simplify Equation (4.1) by introducing the notation $\tilde{S}'_i[l_i]$ defined as $\tilde{S}_i[l_i] \oplus \tilde{S}_i[l_i+1]$. Similarly denote $\tilde{F}_i^j[l_i] \oplus \tilde{F}_i^j[l_i+1]$ by $\tilde{F}_i'^j[l_i]$, and denote $\tilde{Z}^j[t] \oplus \tilde{Z}^j[t+1]$ by $\tilde{Z}'^j[t]$. Thus, Equation (4.1) can be written as:

$$(\tilde{S}'_1[l_1] \oplus \tilde{S}'_2[l_2]) = \tilde{Z}'^j[t] \oplus (\tilde{F}_1'^j[l_1] \oplus \tilde{F}_2'^j[l_2]) \tag{4.2}$$

Observe that due to the linearity of the LFSR, $\tilde{S}'_i[l_i]$ can be viewed as the output of Ri after it has been clocked l_i times from the initial state $S'_i \triangleq S_i \oplus S_i^+$, where S_i^+ denotes the internal state of Ri after it has been clocked once from the internal state S_i. Note that due to the irreducible polynomial there is a one-to-one correspondence between S_i and S'_i (S'_i can be seen as a multiplication of S_i in its polynomial representation by $(x+1)$ modulo the irreducible polynomial, and $x+1$ is always invertible modulo an irreducible polynomial of degree 2 or more). Therefore, once we recover S'_i, we can easily find S_i.

In [58] it was observed that better results are obtained by working simultaneously with d consecutive bits of the output of S'_i, where d is a small integer. A *symbol* is defined to be the binary string of d consecutive bits $S'_i[l_i] \triangleq \tilde{S}'_i[l_i]||\tilde{S}'_i[l_i+1]||\cdots||\tilde{S}'_i[l_i+d-1]$, where "||" denotes concatenation. For example, $S'_2[81] = \tilde{S}'_2[81]$ is a 1-bit symbol, and $S'_1[90] = \tilde{S}'_1[90]||\tilde{S}'_1[91]$ is a 2-bit symbol.

In the first step of [58], estimators are calculated based on the above correlation and on the available keystream. For every pair of indices l_1 and l_2 for which estimators are computed, and for every possible symbol difference $\delta = S'_1[l_1] \oplus S'_2[l_2]$, the estimator $E_{l_1,l_2}[\delta]$ is defined as the logarithm of the a-posteriori probability that $S'_1[l_1] \oplus S'_2[l_2] = \delta$. For example, the symbol is a single bit for $d = 1$, thus, the symbol difference can be either zero or one. Then, for $l_1 = 80$ and $l_2 = 83$, the estimator $E_{80,83}[0]$ is the logarithm of the probability that $S'_1[80] \oplus S'_2[83] = 0$, and $E_{80,83}[1]$ is the logarithm of the probability that $S'_1[80] \oplus S'_2[83] = 1$. For $d = 2$, there are four estimator for every pair of indices, e.g., $E_{80,83}[00_2]$, $E_{80,83}[01_2]$, $E_{80,83}[10_2]$, and $E_{80,83}[11_2]$ (where "$_2$" denotes the fact that the number is written in its binary representation, e.g., 11_2 is the binary representation of the number 3). The value of $E_{80,83}[10_2]$ is the logarithm of the probability that $S'_1[80] \oplus S'_2[83] = 10_2$, and so on. Note that the higher d is — the better the estimators are expected to be (but the marginal benefit drops exponentially as d grows).

We do not describe here how to calculate the estimators given d and the known keystream, as this calculation is a special case of the calculation of our conditional estimators (see Appendix 4.11). We note that the time complexity of this step is roughly proportional to 2^d. With 2000 frames, the simulation in [58] takes about eleven seconds to complete this step with $d = 1$, and about 40 seconds with $d = 4$.

The rest of the details of the previous attacks deal with how to decode the esti-

mators and recover candidate values for S_1, S_2, and S_3 (and thus recovering the key). Further details are given in Appendix 4.9, but for the complete details, we refer the reader to [58].

4.4 The New Observations

In this section, we describe tools and observations that we later combine to form the new attack.

4.4.1 The New Correlation — Conditional Estimators

In Section 4.3, we reviewed the correlation equation used by Maximov, Johansson, and Babbage. This correlation equation is based on two assumptions, the clocking assumption and the step assumption. Recall that the step assumption (i.e., that the third register stands still) holds in a quarter of the cases (assuming that the values in the clocking taps are independent and uniformly distributed).

Consider registers $R1$ and $R2$, and assume that for a given frame j and for the t'th output bit, the clocking assumption holds, i.e., at the t'th output bit of frame j, $R1$ and $R2$ were clocked a total of l_1 and l_2 times, respectively, from their initial state. Also assume that we know the value of $\tilde{S}_1[l_1 + 10]$ and $\tilde{S}_2[l_2 + 11]$. We use the publicly known frame number j to find the value of the clocking taps $C1 = \tilde{S}_1[l_1 + 10] \oplus \tilde{F}_1^j[l_1 + 10]$ of $R1$ and $C2 = \tilde{S}_2[l_2 + 11] \oplus \tilde{F}_2^j[l_2 + 11]$ of $R2$ at output bit t.

We observe that the bias of the correlation can be improved by dividing the step assumption into two distinct cases. The first of the two cases is when $C1 \neq C2$. Due to the clocking mechanism, $R3$ is always clocked in this case along with either $R1$ or $R2$. The step assumption does not hold, and therefore, Equation (4.2) is assumed to hold in half of the cases. In other words, the case where $C1 \neq C2$ provides us no information.

However, in the second case, when $C1 = C2$, we gain a factor two increase in the bias. In this case, both $R1$ and $R2$ are clocked (as $c = C1 = C2$ is the majority), and $R3$ is clocked with probability $1/2$ under the assumption that the values of the clocking taps are uniformly distributed ($R3$ is clocked when its clocking tap agrees with the majority, i.e., when $C3 = c$). Therefore, when $C1 = C2$, the step assumption holds with probability $1/2$ compared to probability $1/4$ in [58].

We analyze the probability that Equation (4.2) holds when $C1 = C2$. If either the step assumption or the clocking assumption do not hold, then we expect that Equation (4.2) holds with probability $1/2$ (i.e., by pure chance). Together with

the probability that the assumptions hold, Equation (4.2) is expected to hold with probability $Pr((l_1, l_2) \text{ at time } t)(1/2 + 1/2 \cdot 1/2) + 1/2(1 - Pr((l_1, l_2) \text{ at time } t)) = 1/2 + Pr((l_1, l_2) \text{ at time } t)/4$ compared to $1/2 + Pr((l_1, l_2) \text{ at time } t)/8$ in [58]. Therefore, when $C1 = C2$, we gain a factor two increase in the bias compared to [58].[2]

We use the above observation to construct *conditional estimators* (which are similar to conditional probabilities). We define a d-bit *clock symbol* $S_i[l_i]$ in index l_i as the d-bit string: $S_i[l_i] = \tilde{S}_i[l_i]||\tilde{S}_i[l_i + 1]|| \cdots ||\tilde{S}_i[l_i + d - 1]$, where "||" denotes concatenation. The conditional estimator $E_{l_1,l_2}[x|Sc]$ for indices l_1, l_2 is computed for every possible combination of a clock symbol difference $Sc = S_1[l_1 + 10] \oplus S_2[l_2 + 11]$ and a symbol difference $x = S'_1[l_1] \oplus S'_2[l_2]$. The estimator $E_{l_1,l_2}[x|Sc]$ is the logarithm of the a-posteriori probability that the value of the symbol difference is x, given that the value of the clock symbol difference is Sc. The computation of conditional estimators is similar to the computation of the estimators as described in [58], taking the dependence on the clock symbol difference into account. The complete description of the calculation of conditional estimators is given in Appendix 4.11.

One way of using conditional estimators is to remove the conditional part of the estimators, and use them as regular estimators, i.e., we can compute $E_{l_1,l_2}[x] = \log\left(\frac{1}{2^d}\sum_y e^{E_{l_1,l_2}[x|y]}\right)$. Nevertheless, the benefit would not be large. A better use of the conditional estimators is to use them directly in the attack as is shown in Section 4.5.1, but before we present this attack, we need to present a few additional observations.

4.4.2 First Weakness of $R2$ — the Alignment Property

The first weakness of $R2$ uses the fact that the feedback taps of $R2$ coincide with the bits that are estimated by the correlation equation. Assume that the value of S_1 is known. Then, for every index i, the correlation equation estimates the value of $S_2[i] \oplus S_2[i+1]$. On the other hand the linear feedback of $R2$ forces $S_2[i] \oplus S_2[i+1] = S_2[i + 22]$. Thus, the correlation equation actually estimates bits which are 22 bits away. Using our notations, this property can be written as

$$S'_2[i] = S_2[i + 22].$$

[2] As a refinement of these observations, note that it suffices to know the value of $\tilde{S}_1[l_1 + 10] \oplus \tilde{S}_2[l_2 + 11]$, since we only consider $C1 \oplus C2$ rather than the individual value of $C1$ and $C2$.

4.4.3 Second Weakness of $R2$ — the Folding Property

The second weakness of $R2$ is that it has only two feedback taps, and these taps are adjacent. Let $X[*]$ be a bit-string which is an output of $R2$, and let $cost(i, x)$ be a cost function that sets a cost for every possible d-bit string x in index i of the string $X[*]$ (the cost function is independent of the specific stream $X[*]$). We calculate the total cost of a given string $X[*]$ (i.e., calculate its "score") by

$$\sum_{i=i_s}^{length(X)-d+1} cost(i, X[i]||X[i+1]||\cdots||X[i+d-1]) \tag{4.3}$$

, where i_s is the first bit that is scored, and $length(X)$ is the last bit being scored. Given the cost function, we can also ask what is the string X_{max} that maximizes the above sum, i.e., the string with the highest score.

The folding property allows to create a new cost function $cost'(i, x)$, where i is one of the first 22 indices. The special property of $cost'$ is that the score calculated on the first 22 indices using $cost'$ is equal to the score using Equation (4.3) over all the indices (using $cost$). $cost'$ is very helpful in finding the highest scored string X_{max} for a given cost function $cost$. However, the transition from $cost$ to $cost'$ has the penalty that $cost'(i, x)$ operates on d'-bit strings x that are slightly longer than d. In general, every 22 additional indices (beyond the first 22 indices) in $X[*]$ add one bit of length to x, so $d' = d + \lceil (length(X) - i_s - d + 1 + 1 - 22)/22 \rceil$ (in our simulation we work with strings of 66 indices and $d = 1$, therefore, our $cost'$ operates on strings of length $d' = d + 2 = 3$).

How should $cost'$ be calculated? For every index i of the first 22 indices, the equality $X[i] \oplus X[i+1] = X[i+22]$ holds due to the linear feedback taps of $R2$. In other words, the d'-bit string at index i determines a $(d'-1)$-bit string at index $i+22$ (which is the XOR difference between every two adjacent bits of the d'-bit string at index i). This string also determines a $(d'-2)$-bit string at index $i+2 \cdot 22$, a $(d'-3)$-bit string at index $i + 3 \cdot 22$, etc. The score is calculated as the sum of the $cost$ of all the indices by Equation (4.3). We can reach the same value of the score if for all $i \in \{i_s, \ldots, i_s + 21\}$, we sum up all the $cost$ of indices equal to i modulo 22, store the result in $cost'$ of index i, and then sum up only the $cost'$ of the first 22 indices $(i_s, \ldots, i_s + 21)$. Thus we "fold" the cost function over all the indices to the $cost'$ function defined for the first 22 indices.

We formally describe an algorithm that calculates $cost'$ from $cost$. For the sake of simplicity, assume that the number of indices is $22k$ (divisible by 22), i.e., $22k+d-1$ bits of $X[*]$ are included in the score computation (though the attack also work when the number of indices is not divisible by 22). The calculation of $cost'$ from $cost$ is given

For each $i \in \{i_s, \ldots, i_s + 21\}$
  For each $e \in \{0,1\}^{d+k-1}$
    $cost'(i,e) \triangleq \sum_{j=0}^{k-1} cost(i + 22k, \mathrm{lsb}_d(\underbrace{D(D(D \cdots D(}_{j \text{ times}} e) \cdots))$

Figure 4.3: The Folding Property: Calculating $cost'$ From $cost$
תכונת ההתקפלות: חישוב $cost'$ מ-$cost$

in Figure 4.3, where the first index of $X[*]$ is denoted by i_s, $D(\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_{d'}) = (\alpha_1 \oplus \alpha_2, \alpha_2 \oplus \alpha_3, \ldots, \alpha_{d'-1} \oplus \alpha_{d'})$ is the operator that calculates the XOR-difference between each pair of adjacent bits (note that the linear feedback of $R2$ actually calculates $D(\cdot)$, as discussed in Section 4.4.2), $\mathrm{lsb}_d(x)$ returns the first d bits of x, and $\mathrm{lsb}_d(\underbrace{D(D(D \cdots D(}_{j \text{ times}} e) \cdots))$ is the d-bit string in index $i + 22k$ that is determined by the $(d + k - 1)$-bit string e in index i. We call the d'-bit strings *representative symbols*.

Note that not every choice of the 22 representative symbols is a consistent output of $R2$, as the 22 representative symbols span $22+d'-1$ bits (and thus there are $2^{22+d'-1}$ possibilities for these bits), while $R2$'s internal state has 22 bits. In particular, the last $d' - 1$ bits of the last (22nd) representative symbol (i.e., the bits in indices $i_s + 22, \ldots, i_s + 20 + d'$) are determined by the first d' bits of the first representative symbol (i.e., bits $i_s, \ldots, i_s + d' - 1$) through the linear feedback. Denote these last $d' - 1$ bits by w. For the first bits to be consistent with the last bits w, we require that the first bits are equal to $D_0^{-1}(w)$ or $D_1^{-1}(w)$, where $D_0^{-1}(w)$ is the value such that $D(D_0^{-1}(w)) = w$, with the first bit of $D_0^{-1}(w)$ being zero (i.e., D_0^{-1} is one of two inverses of D), and where $D_1^{-1}(w)$ is the 1-complement of $D_0^{-1}(w)$ (it also satisfies $D(D_1^{-1}(w)) = w$, i.e., D_1^{-1} is the other inverse of D with the first bit being one).

4.4.4 Third Weakness of $R2$ — the Symmetry Property

The third weakness of $R2$ is that its clock tap is exactly in its center. Combined with the folding property, a symmetry between the clocking tap and the output tap of $R2$ is formed. The symmetry property allows for an efficient attack using conditional estimators. Assume that S_1 is known. $S_2[i]$ is at the output tap of $R2$ when $S_2[i + 11]$ is at the clock tap of $R2$. When $S_2[i + 11]$ reaches the output tap, $S_2[i + 11 + 11] = S_2[i + 22]$ is at the clock tap. However, the representative symbol

at i determines both the bits of $S_2[i]$ and $S_2[i+22]$. Therefore, the representative symbols can be divided into pairs, where each pair contains a representative symbol of some index i and a representative symbol of index $i+11$. When the representative symbol of index i serves for clocking, the other representative symbol is used for the output, and vice versa. As a result, the representative symbols in the pair control the clocking of each other. If the clocking taps were not in the middle, we could not divide the representative symbols into groups of two.

4.5 The New Attack

The attack is composed of three steps:

1. Compute the conditional estimators.

2. Decode the estimators to find a list of the best candidate pairs for (S_1, S_2) by translating the problem of finding the best candidates to a problem in graph-theory.

3. For each candidate in the list for (S_1, S_2), recover candidates for S_3. For each such candidate, the key is recovered and then verified through trial encryptions.

The computation of conditional estimators in Step 1 is based on Section 4.4.1, and similar to the computation of estimators in [58]. A detailed description of this computation is given in Appendix 4.11. Step 2 is described in Section 4.5.1.

In Step 3, given a candidate pair for (S_1, S_2), we find candidates for S_3 based on (S_1, S_2) and the keystream of a particular frame. The method is similar to the one briefly described by Ross Anderson in [2]. However, some adjustments are needed as the method of [2] requires the internal state right at the beginning of the keystream (after discarding 100 bits of output), whereas Step 2 provides candidates for the internal state after the key setup but before discarding 100 bits of output (the candidates for $(S_1, S_2) \oplus (F_1^j, F_2^j)$ are the internal state right after the key-setup and before discarding 100 bits of output).

An alternative Step 3 exhaustively tries all 2^{23} candidate values for S_3. Taking into account that many operators set ten bits of the key to zero (as reported in [24]), we need to try only the 2^{13} candidate values for S_3 which are consistent with the ten zero bits of the key. A more detailed description of Step 3 can be found in Appendix 4.12

4.5.1 Step 2 — Decoding of Estimators

The aim of Step 2 is to find the list of best scored candidates for (S_1, S_2), based on the conditional estimators. The score of (s_1, s_2) (denoting candidate values for S_1 and S_2) is simply the sum of the simultaneous estimators of s_1 and s_2 (which is the logarithm of the product of the a-posteriori probabilities), i.e.,

$$score(s_1, s_2) = \sum_{l_1,l_2} E_{l_1,l_2}[s'_1[l_1] \oplus s'_2[l_2] \;\; | \;\; s_1[l_1 + 10] \oplus s_2[l_2 + 11]].$$

The list of best candidates is the list of candidates $\{(s_1, s_2)\}$ that receive the highest values in this score. For the case of non-conditional estimators, the *score* is defined in a similar manner but using non-conditional estimators (instead of conditional estimators).

Surprisingly, the list of best candidate pairs can be efficiently computed using the three weaknesses of $R2$. We translate the problem of calculating the list of best scored candidates into a problem in graph theory. The problem is modeled as a huge graph with a source node s and target node t, where each path in the graph from s to t corresponds to a candidate value for (S_1, S_2), with the score of the pair being the sum of the costs of the edges along the path (there is a one-to-one correspondence between candidate pairs (s_1, s_2) and path from s to t in the graph). Thus, the path with the heaviest score ("longest" path) corresponds to the highest scored pair. A Dijkstra-like algorithm for finding shortest path [34] can find the longest path in our graph, since the weights on the edges in our graph are negative (logarithm of probability). The list of best candidates corresponds to the list of paths whose scores are closest to the heaviest path. The literature for graph algorithms deals with finding N-shortest paths in a graph (e.g., [49]). These algorithms can be adapted to our graph, and allow to find the heaviest paths.

Our graph contains 2^{19} subgraphs, one for each candidate value for S_1. All the subgraphs have the same structure, but the weights on the edges are different. Each such subgraph has one incoming edge entering the subgraph from the source node s, and one outgoing edge from the subgraph to the target node t. Both edges have a cost of zero.

The Structure of the Sub-Graph Using non-Conditional Estimators

Our method for decoding the estimators can be used with non-conditional estimators, and in fact the structure of the subgraph is best understood by first describing the structure of the subgraph for the case of non-conditional estimators. In this case, the subgraph for the j^{th} candidate of S_1 has a source node s_j and a target node t_j. The

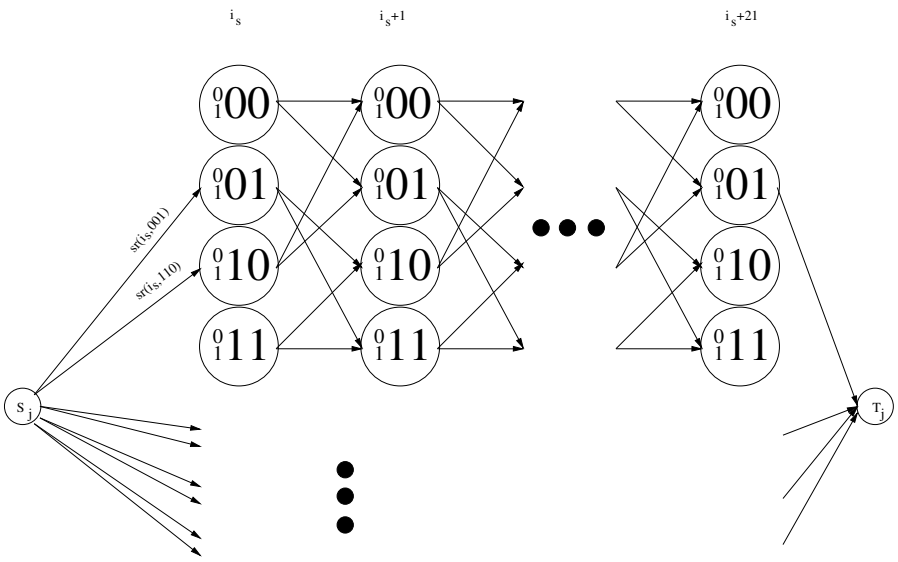


Figure 4.4: The Subgraph for the j^{th} Candidate Value of S_1.
תת-הגרף עבור המועמד ה-j-י לערך של S_1

subgraph is composed of $2^{d'-1}$ mini-subgraphs. Each mini-subgraph corresponds to one combination w of the last $d'-1$ bits of the representative symbol in index i_s+21 (last representative symbol). Figure 4.4 shows an example of a subgraph for $d'=3$, in which only the mini-subgraph for $w=01$ is shown. The full subgraph contains a total of four mini-subgraphs, which differ only in the locations of the two incoming edges (and their weight) and the outgoing edge. For each index $i \in \{i_s, \ldots, i_s+21\}$, the mini-subgraph includes $2^{d'-1}$ nodes: one node for each combination of last $d'-1$ bits of the representative symbols in index i. A single outgoing edge connects the mini-subgraph relevant node ${}^{0}_{1}01$ in index i_s+21 to t_j (the other nodes in index i_s+21 can be erased from the mini-subgraph). Two incoming edges (which correspond to $D_0^{-1}(w)$ and $D_1^{-1}(w)$) connect s_j to relevant nodes in index i_s, which in our example are $D_0^{-1}(01)=001$ and $D_1^{-1}(01)=110$ (the nodes ${}^{0}_{1}00$ and ${}^{0}_{1}11$ in index i_s can thus be erased from the mini-subgraph). Thus, any path that goes through the mini-subgraph must include one of these incoming edges as well as the outgoing edge. This fact ensures that each path corresponds to a consistent choice of representative symbols (as discussed at the end of Section 4.4.3).

Consistent transitions between representative symbols in adjacent indices are modeled by edges that connect nodes of adjacent indices (in a way that reminds a de-Bruijn graph). There is an edge from a node to another node if and only if the

last $d'-1$ bits of the former node are the same as the first $d'-1$ bits of the latter node, which is the requirement for consistent choice of representative symbols. For example, a transition between a representative symbol $a_0a_1\ldots a_{d'-1}$ in index i and a representative symbol $a_1a_2\ldots a_{d'}$ in index $i+1$ is modeled by an edge from node ${}^0_1a_1\ldots a_{d'-1}$ to node ${}^0_1a_2\ldots a_{d'}$. The cost of the edge is $sr(i+1, a_1a_2\ldots a_{d'}) \triangleq cost'(i+1, a_1a_2\ldots a_{d'})$, where $cost'$ is folded using the folding property from $cost(i,x) \triangleq func_{i,s_1}[x] \triangleq \sum_{l_1} E_{l_1,i}[s'_1[l_1]\oplus x]$, as described in Section 4.4.3, and s'_1 is fixed for the given subgraph.

The total cost of edges along a path is $\sum_i func_{i,s_1}[s'_2[i]] = \sum_{l_1,i} E_{l_1,i}[s'_1[l_1]\oplus s'_2[i]] = score(s_1,s_2)$, where s'_2 is the candidate for S'_2 that is implied by the path, and s'_1 is the appropriate value for the j^{th} candidate for S_1. After a quick precomputation, the value of $func_{i,s_1}[x]$ can be calculated using a few table lookups regardless of the value of s_1, as described in Appendix 4.10.

The Structure of the Sub-Graph Using Conditional Estimators

Similarly to the case of non-conditional estimators, in case conditional estimators are used, the subgraph for candidate j has a source node s_j, a target node t_j, and the subgraph is composed of several mini-subgraphs, which differ only in the location of the incoming edges (and their cost) and the location of the outgoing edge. However, with conditional estimators, the structure of the mini-subgraphs is different: each pair of indices $i, i+11$ are unified to a single index, denoted by $i|i+11$.

We would like to combine the nodes in index i with nodes in index $i+11$ by computing their cartesian product: for each node a in index i and for each node b in index $i+11$, we form the unified node $a|b$ in unified index $i|i+11$. However, there is a technical difficulty: while (given S_1) a non-conditional estimator depends on a symbol candidate $s'_2[i]$, a conditional estimator depends on both a symbol candidate $s'_2[i]$ and a clock symbol candidate $s_2[i+11]$. As a result, we must apply the D^{-1} operator on nodes in index $i+11$ (to transform them from symbols to clock symbols). This operation divides node $b = {}^0_1 b_1b_2\ldots b_{d'-1}$ in index $i+11$ into two nodes ${}^0_1D^{-1}_0(b_1b_2\ldots b_{d'-1})$ and ${}^0_1D^{-1}_1(b_1b_2\ldots b_{d'-1})$. Only then, we can perform the cartesian product between the nodes in index i and the nodes that results from applying D^{-1}. Thus, from a pair of a and b of the above form, we have two nodes in the product (in index $i|i+11$): $a|{}^0_1D^{-1}_0(b_1b_2\ldots b_{d'-1})$ and $a|{}^0_1D^{-1}_1(b_1b_2\ldots b_{d'-1})$. We refer to the bits on the left of the "|" in the node as symbol bits, and the bits on the right of the "|" as clock bits. In total, there are $2^{d'-1}(2\cdot 2^{d'-1}) = 2^{2d'-1}$ nodes in each index $i|i+11$.

There is an edge from node $x_1|y_1$ in index $i|i+11$ to node $x_2|y_2$ in index $i+1|i+12$

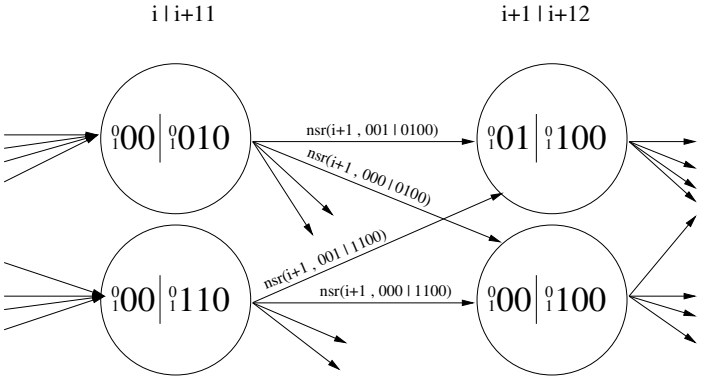


Figure 4.5: Four Nodes of the Mini-Subgraph Using Conditional Estimators for $d' = 3$
ארבעה צמתים במיני תת-הגרף המשתמש במשערכים מותנים עם $d' = 3$

if and only if the last $d'-1$ bits of x_1 are equal to the first $d'-1$ bits of x_2 and the last d' bits of y_1 are equal to the first d' bits of y_2. Figure 4.5 depicts four nodes of a mini-subgraph using conditional estimators.

What should be the cost of an edge? The basic cost function is $cost(i, x|y) \triangleq func_{i,s_1}[x|y] \triangleq \sum_{l_1} E_{l_1,i}[s'_1[l_1] \oplus x | s_1[l_1+10] \oplus y]$, which is folded to the cost function $cost'(i, x|y)$. Since each index $i|i+11$ unifies two indices, the edge that enters $i|i+11$ should contain the sum of contribution of indices i and $i+11$, i.e., the cost of the edge is $nsr(i, s'_2[i]|s_2[i+11]) \triangleq cost'(i, s'_2[i]|\text{lsb}_{d'}(s_2[i+11])) + cost'(i+11, s'_2[i+11]|s_2[i+22])$, where $\text{lsb}_{d'}(x)$ returns the d' first bits of x. Note that $s'_2[i+11] = D(s_2[i+11])$, and (due to the alignment property) $s_2[i+22] = s'_2[i]$. Therefore, $nsr(i, s'_2[i]|s_2[i+11]) = cost'(i, s'_2[i]||\text{lsb}_{d'}(s_2[i+11])) + cost'(i+11, D(s_2[i+11])|s'_2[i])$.

Like the case of non-conditional estimators, we create several mini-subgraphs to ensure that the paths in the subgraph represent consistent choices for S_1 and S_2. We include in the subgraph a mini-subgraph for each combination v of the last $d'-1$ symbol bits and each combination w of the last d' clock bits of the last node (the node near t_j). A single edge (with cost zero) connects the mini-subgraph to t_j from node ${}_1^0 v|_1^0 w$. For consistency with the linear feedback, the bits w must be identical to the symbol bits of the first node (both w and the first symbol bits are d'-bit long). The bits v must be identical to the difference of the first d' bits of the first clock symbol. As v is $(d'-1)$-bit long, and as the clock bits of the first symbol are $(d'+1)$-bit long, there are four possibilities for the clock bits: $D_0^{-1}(v)||0$, $D_1^{-1}(v)||0$, $D_0^{-1}(v)||1$, and $D_1^{-1}(v)||1$. Therefore, four edges $w|D_0^{-1}(v)0$, $w|D_1^{-1}(v)0$, $w|D_0^{-1}(v)1$, and $w|D_1^{-1}(v)1$ connect s_j to the mini-subgraph (the concatenation mark

"||" was removed for clarity). Their costs are $nsr(i_s, w|D_0^{-1}(v)0)$, $nsr(i_s, w|D_1^{-1}(v)0)$, $nsr(i_s, w|D_0^{-1}(v)1)$, and $nsr(i_s, w|D_1^{-1}(v)1)$, respectively.

To reconstruct s'_2 from a path in the mini-subgraph, we first concatenate the symbol bits to form the first half of the path, and separately concatenate the clock bits to form the second half of the path. Then, we compute the difference between the clock bits, and combine the result with the symbol bits to obtain a path of s'_2 (similar to the path in the case of the mini-subgraph using un-conditional estimators).

Note that in an efficient implementation there is no need to keep the entire graph in memory, since the needed parts of the graph can be reconstructed on-the-fly.

4.6 Simulations of our Attacks

We have implemented our attack, and simulated it under various parameters. Our simulations focus on 2000 frames of data, which is the lowest amount of data that gives a non-negligible success rate in the simulations of Maximov, Johansson, and Babbage [58]. We also simulated the attack with 1500 frames. A comparison of simulations of previous attacks and simulations of our new attacks is given in Table 4.1.

In the simulations we use $d = 1$, $l_1 \in \{61, \ldots, 144\}$, $l_2 \in \{70, \ldots, 135\}$, and calculate estimators for $\mid l_1 - l_2 \mid < 10$. We use the first version of Step 3 with 64-bit keys.

We ran the simulations on a 1.8GHz Pentium-4 Mobile CPU with 512MB of RAM. The operating system was Cygwin under Windows XP. In comparison, the simulations of [58] were performed on a 2.4GHz Pentium-4 CPU with 256MB of RAM under Windows XP, and the simulations of [35] were performed on a 1.8GHz Pentium-4 CPU with 512MB of RAM under Linux.

In one simulation, we limited the size of the list of top (s_1, s_2) pairs to 5200. The key was found in about 64 percent of the cases, compared to about 5 percent in previous attacks with 2000 frames. Our attack takes about 7 seconds to complete Step 1. Step 2 takes about 340 seconds for the first pair, after which it can generate about 1500 pairs of candidates per second. Step 3 scans about 20.4 candidate pairs per second. Therefore, the total time complexity varies depending on the location of the correct pair in the list. It takes about 350 seconds (six minutes) in the best case, and up to ten minutes in the worst case.

For better results, we employ two methods: *early filtering* and *improved estimators*.

Table 4.1: Comparison Between Our Attacks and Passive Attacks of Previous Works
השוואה בין ההתקפות שלנו להתקפות פאסיביות קודמות

Attack: (Configuration explained in Section 4.6)	Required Frames: Known Keystream	Required Frames: Ciphertext Only (Section 4.7)	Average Time on a single PC (range)	Success Rate
Ekdahl & Johansson [35]	70000 (322 s)	140 min	5 min	76%
[35]	50000 (230 s)	99 min	4 min	33%
[35]	30000 (138 s)	60 min	3 min	3%
Biham & Dunkelman [15]	20500 (95 s)	40.8 min	$\approx$ 1.5 days	63%
Maximov et al. [58]	10000 (46 s)	20 min	10 min	99.99%
[58]	10000 (46 s)	20 min	76 s	93%
[58]	5000 (23 s)	10 min	10 min	85%
[58]	5000 (23 s)	10 min	44 s	15%
[58]	2000 (9.2 s)	4 min	10 min	5%
[58]	2000 (9.2 s)	4 min	29 s	1%
Biryukov et al. [21]	2000 (9.2 s)	4 min	♯ > 5 years	
Ciphertext only of [10]	—	4 min*	♯ > 2300 years	
This Chapter	2000 (9.2 s)	4 min	(6–10 min)	64%
early filtering $(220000, 40000, 2000, 5200)$	2000 (9.2 s)	4 min	(55–300 s)	64%
early filtering $(100000, 15000, 200, 300)$	2000 (9.2 s)	4 min	(32–45 s)	48%
improved estimators, $(200000, 17000, 900, 2000)$	2000 (9.2 s)	4 min	74 s (50–145 s)	86%
improved estimators, $(\mathbf{200000}, \mathbf{36000}, \mathbf{1400}, \mathbf{11000})$	**2000 (9.2 s)**	**4 min**	**133 s (55–626s)**	**91%**
early filtering $(120000, 35000, 1000, 800)$	1500 (6.9 s)	3 min	(39–78 s)	23%
improved estimators, $(88000, 52000, 700, 1200)$	1500 (6.9 s)	3 min	82 s (44–105 s)	48%
improved estimators, $(\mathbf{88000}, \mathbf{52000}, \mathbf{3200}, \mathbf{15000})$	**1500 (6.9 s)**	**3 min**	**7.2 min (44–780 s)**	**54%**

Only passive attacks are included, i.e., the active attack of [10] is not shown. The attack time for [10, 15, 21] is our estimate. As [10, 21] are time/memory/data tradeoff attacks, we give the tradeoff point that uses data that is equivalent to four minutes of ciphertext.
* based on error-correction codes as described in [10] (not on Section 4.7).
♯ preprocessing time.

4.6.1 Early Filtering

In early filtering, we perform Step 2 several times, using less accurate (and faster) methods. Thus, we discard many candidate values of S_1 that are highly unlikely, and we do not need to build a subgraph for these values. For example, we score all the candidates of S_1 (a score of a candidate s_1 of S_1 is $\max_{s_2} score(s_1, s_2)$) using non-conditional estimators and a less accurate but faster method, in which we construct a subgraph containing only one mini-subgraph for each candidate of s_1, and do not use the folding property. Then, we recalculate the score for the 220000 top candidates, using a similar method, but with conditional estimators. The 40000 top scored candidates are re-scored using conditional estimators with a variation using only one mini-subgraph. Finally, we perform Step 2 of Section 4.5.1 with subgraphs only for the 2000 scored candidates of S_1. The list of the 5200 top candidates of (S_1, S_2) is generated and passed to Step 3. We denote this kind of configuration in a tuple $(220000, 40000, 2000, 5200)$. Simulation results using other configurations for both 2000 and 1500 frames are given in Table 4.1.

4.6.2 Improved Estimators

A disadvantage of the described attack is that only information from the estimators $E_{l_1,l_2}[\cdot|\cdot]$ is taken into consideration, while estimators involving $R3$, i.e., $E_{l_1,l_3}[\cdot|\cdot]$ and $E_{l_2,l_3}[\cdot|\cdot]$, are disregarded. In *improved estimators*, we improve our results by adding to each estimator $E_{l_1,l_2}[x|y]$ the contributions of the estimators of the other registers, i.e., we add to it

$$\sum_{l_3} \log \left(\sum_{\alpha,\beta \in \{0,1\}^d} e^{E_{l_1,l_3}[\alpha|\beta] + E_{l_2,l_3}[x \oplus \alpha | y \oplus \beta]} \right).$$

The resulting estimators include more information, and thus, are more accurate. They significantly improve the success rate with a modest increase in the time complexity of Step 1 (mostly, since we need to calculate three times the number of estimators). This increase in time complexity is compensated by a large decrease in the time complexity of Step 3 (as the correct (S_1, S_2) is found earlier). The results are summarized in Table 4.1.

4.7 A New Source for Known-Keystream

Every traffic channel between the handset and the network is accompanied by a slower control channel, which is referred to as the Slow Associated Control CHan-

nel (SACCH). The mobile uses the SACCH channel (on the uplink) to report its reception of adjacent cells. The network uses this channel (on the downlink) to send (general) system messages to the mobile, as well as to control the power and timing of the current conversation.

The contents of the downlink SACCH can be inferred by passive eavesdropping: The network sends power-control commands to the mobile. These commands can be inferred from the transmission power of the mobile. The timing information that the network commands the mobile can be inferred from the transmission timing of the mobile. The other contents of the SACCH is a cyclical transmission of 2–4 "system messages"(see [37, Section 3.4.1]). These messages can be obtained from several sources, for example by passively eavesdropping the downlink at the beginning of a call (as the messages are not encrypted at the beginning of a call), or by actively initiating a conversation with the network using another mobile and recover these messages (these messages are identical for all mobiles). There is no retransmission of messages on the SACCH, which makes the task of the attacker easier, however, it should be noted that an SMS received during an on-going conversation could disrupt the eavesdropper, as the SMS can be transferred on the SACCH, when system messages are expected.

An attacker would still need to cope with the Frequency Hoping (FH) used by GSM. Using a frequency analyzer the attacker can find the list of n frequencies that the conversation hops on. Given n, GSM defines only $64n$ hopping sequences (n cannot be large since the total number of frequencies in GSM is only about 1000, of which only 124 belong to GSM 900). Thus, the hopping sequence can be determined through a quick exhaustive search.

As the name of SACCH implies, it is a slow channel. Only about eight frames are transmitted every second in each direction of the channel. Therefore, to collect 1500–2000 SACCH frames transmitted from the network to mobile, about 3–4 minutes of conversation are needed.

4.8 Summary

Our contribution in this chapter is multi-faced. We begin by introducing conditional estimators that increase the bias of the correlation equation. Then, we present three weaknesses in $R2$, which were not reported previously. The first weakness — the alignment property — utilizes the fact that the correlation equation coincides with the feedback taps of $R2$. The second weakness — the folding property — uses the fact that $R2$ has only two feedback taps, and they are adjacent. We use the folding property to decode the estimators in an optimal way. In contrast, previous attacks

were forced to use heuristics to decode the estimators. Using this weakness, we present a novel method to efficiently calculate the list of best candidate pairs for S_1 and S_2. Given S_1 and S_2, the value S_3 can be worked back from the keystream.

The last weakness that we report — the symmetry property — is based on the fact that $R2$'s clocking tap is exactly in its middle, which together with the folding property causes a symmetry between the clocking tap and the output of $R2$. This property enables us to efficiently decode the *conditional* estimators.

Finally, we describe a new source for known-plaintext in GSM. This source of known-plaintext transforms our attack to a practical ciphertext-only attack. With 3–4 minutes of raw ciphertext, we can extract the required amount of about 1500–2000 frames of known-plaintext from the SACCH.

We compare some of the previous results and our current simulation results in Table 4.1. Compared to previous attacks on 1500–2000 frames, it can be seen that our new attack has a significantly higher success rate (91% compared to 5%), it is faster, and it does not require any precomputation.

Acknowledgements

We are pleased to thank Alexander Maximov for providing early versions of [58].

4.9 Appendix: Overview of Step 2 and Step 3 of Maximov, Johansson, and Babbage's Attack

In Step 2 of [58], the set of estimators are decoded within short intervals, i.e., each possible value for the interval contents is scored using the estimators, and the list of the r highest-scored candidates is stored in tables. We describe Step 2 using our notations (which results in a factor four decrease in the time complexity compared to the original work).

In [58] the estimators are decoded in intervals of eleven symbols in length, e.g., $S'_1[69,\ldots,79]$, and $S'_2[69,\ldots,79]$. For each such interval and for each possible value of the content $s'_1[69,\ldots,79], s'_2[69,\ldots,79]$ of the interval, a score is calculated. Let $I = [69,\ldots,79]$. Then, $s'_1[I], s'_2[I]$ can take $2^{2(11+d-1)}$ values. A candidate value is scored by calculating

$$score(s'_1[I], s'_2[I]) = \sum_{l_1,l_2 \in I} E_{l_1,l_2}[s'_1[l_1] \oplus s'_2[l_2]]. \tag{4.4}$$

The highest-scored r possibilities for interval values are stored in a table. This process is performed for all the intervals.

We analyze the time complexity of Step 2 as follows. The score calculation for a particular pair of interval values involves summing $|I|^2$ elements, where $|I|$ is the number of symbols covered by the interval (in the above example $|I| = 11$). Therefore, the time complexity of Step 2 for a given interval is $|I|^2 \cdot 2^{2(|I|+d-1)}$. This figure should be multiplied by the number of intervals to obtain the total time complexity of Step 2.

The time complexity of Step 2 presented here is lower by a factor of four compared to Step 2 of the attack described in [58] (there the time complexity for a given interval is $|I|^2 \cdot 2^{2(|I|+d)}$). The factor four savings in our description is the result of the observation that there is a one-to-one correspondence between S' and S (the observation is incorporated into the S' notation).

In Step 3 of [58], the candidate tables are intersected according to various heuristics to recover candidates for the values of S'_1, S'_2, and S'_3. These values are combined to create candidate keys, which are checked against the known-keystream. See [58] for the complete description.

4.10 Appendix: Fast Calculation of $func_{l_2,s_1}[x]$

We show a precomputation that speeds up the calculation of $func_{i,s_1}[\cdot]$. Recall that given s_1, $func_{i,s_1}[x] = \sum_{l_1} E_{l_1,i}[s'_1[l_1] \oplus x]$. The idea behind this precomputation is as follows. Since the contribution $E_{l_1,i}[s'_1[l_1] \oplus x]$ of all the location l_1 is summed to form the value $func_{i,s_1}[x]$, we can precompute the contribution of intervals of l_1, with all the different values for the content of the interval. In this way, given $s'_1[*]$ we can calculate the value of $func_{i,s_1}[x]$ using a few table accesses.

We divide the range of possible l_1 values into intervals $I_1, \ldots, I_k$ of fixed length. Therefore, we can separate the summation of $func_{i,s_1}[x]$ according to the intervals, i.e., $func_{i,s_1}[x] = \sum_{I_j \in \{I_1,\ldots,I_k\}} \sum_{l_1 \in I_j} E_{l_1,i}[s'_1[l_1] \oplus x]$. Each interval covers a sequence of symbol locations, for example, ten locations. Note that every two adjacent intervals intersect on $d-1$ bits. We define $func_{I_j,v[I_j],l_2}[x] = \sum_{l_1 \in I_j} E_{l_1,l_2}[v[l_1] \oplus x]$, where $v[I_j]$ is some interval value (2^{10+d-1} possible interval values for $|I_j| = 10$, where $|I|$ denotes the number of symbols in the interval) and $v[l_1]$ is the d-bit symbol in location l_1 in $v[I_j]$. For each such interval I_j, for each each possible interval value $v[I_j]$, for each l_2 value, and for each possible symbol value x, we precompute $func_{I_j,v[I_j],l_2}[x]$ according to its formula. Assuming the interval size is ten symbols, this precomputation takes $n^2 2^{10+2d-1}$ table accesses (to the estimators), where l_1

runs over n locations. The memory complexity for storing the $func_{I_i,v[I_i],l_2}[x]$'s is $(n/10)2^{10+d-1}n2^d = \frac{n^2}{10}2^{9+2d}$ memory cells. This memory complexity is not negligible, but is not too large either: assuming $n = 70$ and $d = 1$, the required memory is about a few megabytes of memory.

Using the precomputation, we compute a very close approximation to $func_{l_2,s_1}[x]$ in a time complexity of only about three table accesses. Given $s'_1[*]$, we calculate $func_{l_2,s_1}[x] = \sum_{I_i} func_{I_i,s'_1[I_i],l_2}[x]$, where $I_i \in \{I_1, \ldots, I_k\}$, and $s'_1[I_i]$ is the string of symbols of $s'_1[*]$ beginning in the interval I_i. Thus, the time complexity of calculating $func_{l_2 s_1}[x]$ is reduced to k table accesses. Note that when $|l_1 - l_2|$ increases the bias of the estimator decreases. Therefore, when calculating $func_{l_2,s_1}[x]$ we can ignore the contribution of the locations l_1 which are far from l_2. If we limit the contribution to $|l_1 - l_2| < 10$ (for example), we can calculate $func_{l_2,s_1}[e]$ using a fixed number of table accesses. For example if $l_2 \in I_j$ than only I_{j-1}, I_j, and I_{j+1} are considered when calculating $func_{l_2,s_1}[x]$.

4.11 Appendix: Calculating Conditional Estimators

In this appendix, we show how to calculate conditional estimators. The method is similar to the calculation of estimators as described in [58] combined with the conditional step assumption as described in Section 4.4.1 and some corrections.

Denote by $\tilde{R}_i^j[l_i]$ the output of register Ri at frame j after being clocked l_i times from its initial state, i.e., $\tilde{R}_i^j[l_i] = \tilde{S}_i[l_i] \oplus \tilde{F}_i^j[l_i]$. Denote by $C^j_{l_1,l_2}$ the difference between the d-bit clock symbols $S_1[l_1]$ and $S_2[l_2]$ together with the affect of a specific frame number j on it, i.e.,

$$C^j_{l_1,l_2} = \tilde{R}_1^j[l_1] \oplus \tilde{R}_2^j[l_2]||\ldots||\tilde{R}_1^j[l_1 + d - 1] \oplus \tilde{R}_2^j[l_2 + d - 1].$$

We call $C^j_{l_1,l_2}$ the *frame clock symbol*. Clearly, given d, there are 2^d possible values for $C^j_{l_1,l_2}$.

For every possible value of $C^j_{l_1,l_2}$, we perform a one-time precomputation. The precomputation results in two tables: a *pattern table* and a *distribution table*, where the pattern table is only used for the computation of the distribution table, and it is discarded right after. Denote $Z'^j[t] = \tilde{Z}^j[t]||\tilde{Z}^j[t+1]||\ldots||\tilde{Z}^j[t+d-1]$. Then, the distribution table states for every possible difference $\varepsilon \in \{0,1\}^d$ the a priori probability that

$$Z'^j[t] \oplus (\ (S'_1[l_1] \oplus F'^j_1[l_1]) \oplus (S'_2[l_2] \oplus F'^j_2[l_2])\ ) = \varepsilon$$

Table 4.2: A Comparison of Distribution Tables for $d = 4$
$d = 4$ השוואה של טבלת ההתפלגות עם

ε	$Pr\{\varepsilon\|$ $0000_2\}$	$Pr\{\varepsilon\|$ $0001_2\}$	$Pr\{\varepsilon\|$ $0011_2\}$	$Pr\{\varepsilon\|$ $0111_2\}$	$Pr\{\varepsilon\|$ $1111_2\}$	$Pr\{\varepsilon\}$	$Pr\{\varepsilon\}$
0000_2	$81/2^8$	$54/2^8$	$36/2^8$	$24/2^8$	$16/2^8$	$26.9/2^8$	$431/2^{12}$
1000_2	$27/2^8$	$18/2^8$	$12/2^8$	$8/2^8$	$16/2^8$	$14.3/2^8$	$229/2^{12}$
0100_2	$27/2^8$	$18/2^8$	$12/2^8$	$24/2^8$	$16/2^8$	$18.3/2^8$	$293/2^{12}$
1100_2	$9/2^8$	$6/2^8$	$4/2^8$	$8/2^8$	$16/2^8$	$11.4/2^8$	$183/2^{12}$
0010_2	$27/2^8$	$18/2^8$	$36/2^8$	$24/2^8$	$16/2^8$	$21.3/2^8$	$341/2^{12}$
1010_2	$9/2^8$	$6/2^8$	$12/2^8$	$8/2^8$	$16/2^8$	$12.4/2^8$	$199/2^{12}$
0110_2	$9/2^8$	$6/2^8$	$12/2^8$	$24/2^8$	$16/2^8$	$16.4/2^8$	$263/2^{12}$
1110_2	$3/2^8$	$2/2^8$	$4/2^8$	$8/2^8$	$16/2^8$	$10.8/2^8$	$173/2^{12}$
0001_2	$27/2^8$	$54/2^8$	$36/2^8$	$24/2^8$	$16/2^8$	$23.5/2^8$	$377/2^{12}$
1001_2	$9/2^8$	$18/2^8$	$12/2^8$	$8/2^8$	$16/2^8$	$13.1/2^8$	$211/2^{12}$
0101_2	$9/2^8$	$18/2^8$	$12/2^8$	$24/2^8$	$16/2^8$	$17.1/2^8$	$275/2^{12}$
1101_2	$3/2^8$	$6/2^8$	$4/2^8$	$8/2^8$	$16/2^8$	$11.0/2^8$	$177/2^{12}$
0011_2	$9/2^8$	$18/2^8$	$36/2^8$	$24/2^8$	$16/2^8$	$20.1/2^8$	$323/2^{12}$
1011_2	$3/2^8$	$6/2^8$	$12/2^8$	$8/2^8$	$16/2^8$	$12.0/2^8$	$193/2^{12}$
0111_2	$3/2^8$	$6/2^8$	$12/2^8$	$24/2^8$	$16/2^8$	$16.0/2^8$	$257/2^{12}$
1111_2	$1/2^8$	$2/2^8$	$4/2^8$	$8/2^8$	$16/2^8$	$10.6/2^8$	$171/2^{12}$

The rightmost column gives the distribution table from [58]. The column on its left contains the same values after dividing the numerator and the denominator by 2^4. Further on the left are the distribution tables using the conditional estimators given all the possible frame clock symbols. An example of reading the table: the correlation equation holds for all the bits except the last (i.e., $\varepsilon = 0001_2$) with probability $36/2^8$ given that the frame clock symbol is 0011_2, with probability $27/2^8$ given that the frame clock symbol is 0000_2, and with probability $23.5/2^8$ when not given the frame clock symbol (as is calculated in [58]).

given that $R1$ and $R2$ have been clocked l_1 and l_2, respectively, at time t, and given the value of $C^j_{l_1+10,l_2+11}$. As it turns out, this probability depend only on the symbol size d, the value of $C^j_{l_1+10,l_2+11}$, and the value of ε. Furthermore, the values in the distribution tables do not depend on the exact value of $C^j_{l_1,l_2}$, rather, they depend on the first occurrence of "1" in $C^j_{l_1,l_2}$. Therefore, it is enough to calculate the tables for the $(d+1)$ possibilities of a first occurrences of "1".

In Table 4.2, we give a comparison of the distribution tables that are computed using conditional estimators with the one of [58]. Note that the most probable

event is that the correlation equation holds for all the bits of the symbol (e.g., $\varepsilon = 0000_2$ with probability of $81/2^8$ given a frame clock symbol of 0000_2). The least probable event is that the correlation equation fails for all the bits of the symbol (e.g., $\varepsilon = 1111_2$ with probability $1/2^8$ given a frame clock symbol of 0000_2). Note that the differences between the values is typical higher in distribution tables based on conditional estimators compared to the more uniform distribution table of [58] (which uses non-conditional estimators). Note that the bias between the values in the distribution table is at is peak when the frame clock symbol is all zeros (but this value of the frame clock symbol occurs only in 1/16 of the cases); the most uniform distribution table occurs when the frame clock symbol begins with a "1", due to the reasons explained later. The distribution tables using conditional estimators are always better than the distribution table that are unaware of the frame clock symbol. In fact, the table of [58] can be seen as weighted average over the tables using conditional estimators (as the clock had to be guessed), for example, $Pr\{\varepsilon\} = (Pr\{\varepsilon|0000_2\} + Pr\{\varepsilon|0001_2\} + 2Pr\{\varepsilon|0011_2\} + 4Pr\{\varepsilon|0111_2\} + 8Pr\{\varepsilon|1111_2\})/16$ (as we elaborate later). Thus, when using only the non-conditional distribution table $Pr\{\varepsilon\}$, "noise" is induced into the analysis.

We first explain how to calculate the conditional estimators based on the distribution tables, and only then explain how to calculate the pattern tables and the distribution tables. The conditional estimators are first computed for each frame j. For each possible value of l_1, l_2 and each value $S', C \in \{0,1\}^d$, the conditional estimators state the a posteriori probability that $S'_1[l_1] \oplus S'_2[l_2] = S'$ given that the frame clock symbol is C and given the keystream in frame j. The estimators for the frame are calculated by

$$E^j_{l_1,l_2}[S'|C] = \frac{1}{2^d}\left(1 - \sum_t Pr((l_1,l_2) \text{ at time } t)\right) + \sum_t Pr((l_1,l_2) \text{ at time } t) \cdot Pr\{\varepsilon = R' \oplus Z'^j[t] | C^j_{l_1+10,l_2+11}\},$$

where $R' = S' \oplus F'^j_1[l_1] \oplus F'^j_2[l_2]$, $C^j_{l_1+10,l_2+11} = C \oplus F^j_1[l_1+10] \oplus F^j_2[l_2+11]$, $Z'^j[t]$ is calculated from the keystream, $Pr((l_1,l_2)$ at time $t)$ is the probability that register $R1$ at clock t has been clocked (using the irregular clocking) l_1 times from its initial state (respectively, $R2$ has been clocked l_2 times from its initial state), and the sums are taken over the values of t that have a non-negligible $Pr((l_1,l_2)$ at time $t)$ value. If we ignore for a moment the "$|C^j_{l_1+10,l_2+11}$" in the above formula, the formula is nothing more than $Pr(A) = \sum_i Pr(A|B_i)Pr(B_i)$, where $Pr(A|B_i)$ is taken from the distribution table, and $Pr(B_i)$ is $Pr((l_1,l_2)$ at time $t)$. For t's with negligible

probability, we assume the distribution table is uniform, i.e., all the probabilities are $1/2^d$.

The conditional estimators of the frames are then combined over all the frames:

$$E'_{l_1,l_2}[S'|C] = \prod_j E^j_{l_1,l_2}[S'|C].$$

Finally, the estimators are normalized, and their logarithm is taken:

$$E_{l_1,l_2}[S'|C] = \log \frac{E'_{l_1,l_2}[S'|C]}{\sum_i E'_{l_1,l_2}[i|C]}.$$

This completes the calculation of the conditional estimators.

In [58] a closed formula is given to $Pr((l_1, l_2)$ at time $t)$:

$$Pr((l_1, l_2) \text{ at time } t) = \frac{\left(\begin{array}{c} t \\ t-l_1 \end{array} \right) \left(\begin{array}{c} t-(t-l_1) \\ t-l_2 \end{array} \right)}{2^{3t-(l_1+l_2)}}.$$

The formula can be easy derived as follows: There are four equiprobable clocking possibilities for each output bit (either a single register (of $R1$, $R2$, or $R3$) is not clocked and the rest are clocked, or that all three registers are clocked). Therefore, we need to determine how many of the 4^t clocking possibilities result in $R1$ being clocked l_1 times and $R2$ being clocked l_2 times. Out of the t output bits, we choose the $t-l_1$ output bits for which $R1$ is not clocked, and therefore, the rest of the registers are clocked for these output bits. In the remaining $t-(t-l_1)$ output bits, we choose the $t-l_2$ output bits for which $R2$ is not clocked, and thus the rest of the registers are clocked for these $t-l_2$ output bits. For the remaining $(t-(t-l_1))-(t-l_2)$ output bits, either $R3$ is not clocked or all the registers are clocked. Using simple combinatorics, there are

$$\left(\begin{array}{c} t \\ t-l_1 \end{array} \right) \left(\begin{array}{c} t-(t-l_1) \\ t-l_2 \end{array} \right) 2^{(t-(t-l_1))-(t-l_2)}$$

such possibilities out of the 4^t possible clockings.

It now remains to describe how the pattern tables are calculated. Assume that at time t and frame j, $R1$ and $R2$ have been clocked l_1 and l_2 times, respectively, from their initial states. Given the parity of the clock-control bits of $R1$ and $R2$, there are only two ways to clock the registers. Therefore, given a frame clock symbol, there are 2^d possible clock symbols for $R3$, and together with the frame clock symbol, the clockings is completely defined. A row in the pattern table states for each bit

in the output symbol $Z'^j[t]$ which assumptions hold for that bit (i.e., the clocking assumption and/or the step assumption), and the number of cases out of the 2^d possible clockings for which it happens (i.e., the probability is taken over the choices of the clock symbol in $R3$, which is assumed to be random). The clocking assumption for bit $i \in \{0, \ldots, d-1\}$ in the output symbol holds if and only if $R1$ and $R2$ have been clocked $l_1 + i + 1$ and $l_2 + i + 1$ times, respectively. In other words, recall that $Z'^j[t] = (\tilde{Z}^j[t] \oplus \tilde{Z}^j[t+1], \tilde{Z}^j[t+1] \oplus \tilde{Z}^j[t+2], \ldots, \tilde{Z}^j[t+d-1] \oplus \tilde{Z}^j[t+d])$; then the clocking assumption holds for bit i of $Z'^j[t]$ if and only if $R1$ and $R2$ have been clocked $l_1 + i + 1$ and $l_2 + i + 1$, respectively, at time $t + i + 1$. Since we assume that the clocking assumption holds at $Z^j[t]$, it continues to hold as long as $R1$ and $R2$ are clocked at every output bit. Once one of the two registers is not clocked, the clocking assumption cannot hold at least until the end of the symbol, since a register cannot be clocked more than once for each output bit. The frame clock symbol tells us exactly when the clocking assumption holds: If the first bit of the frame clock symbol is "0", then the clocking assumption holds for the first bit. If the clocking assumption holds for the first bit, then it holds for the second bit provided that the second bit of the frame clock symbol is "0", and it continues to hold for the other bits as long as the corresponding bit in the frame clock symbol is "0". Once the frame clock symbol contains a "1", either $R1$ or $R2$ is not clocked. Therefore, the clocking assumption stops holding until the end of the output symbol. As a result, there are $(d+1)$ different distribution tables when using conditional estimators, as the first "1" can appear in any of the d locations, or it might not appear at all. In other words, once a "1" occurs in the frame clock symbol, we lose our ability to gain any information out of the remaining bits of the keystream symbol (and this is the reason that the distribution table for a clock symbol that begins with "1" is uniform).

The step assumption for bit i is meaningful only when the clocking assumption holds for the bit. When the clocking assumption holds, the step assumption holds with probability 1/2 (and then, the clocking assumption holds for bit $i+1$), or it holds with probability 0 (and then, the clocking assumption fails for bit $i+1$). The pattern table for $d = 4$ and the frame clock symbol 0011_2 is given in Table 4.3 (rows with probability zero are not shown). Note the correspondence between the 1's in frame clock symbol 0011_2 and the *cfail* values in the pattern table.

For the purpose of computing the distribution table, all we care is if the assumptions hold or not, regardless of their type (*cfail* or *sfail*). Therefore, we unite *sfail* and *cfail* under *Random*, and *hold* is now denoted by *Correct*. In Table 4.4 we rewrite Table 4.4 under the new notations, and call the resulting table the *united pattern table*.

Table 4.3: The Pattern Table for $d = 4$ and the Frame Clock Symbol 0011_2
טבלת התבנית עבור $d = 4$ וסימבול שעון של מסגרת 0011_2

1^{st} output bit	2^{nd} output bit	3^{rd} output bit	4^{th} output bit	Probability
holds	holds	cfail	cfail	$2^2/2^4$
sfail	holds	cfail	cfail	$2^2/2^4$
holds	sfail	cfail	cfail	$2^2/2^4$
sfail	sfail	cfail	cfail	$2^2/2^4$

holds means both the clocking and the step assumption hold; *sfail* means the clocking assumption holds, but the step assumption fails; *cfail* means the clocking assumption fails (and therefore, the step assumption is meaningless).

We now construct a distribution table given a united pattern table. Every row in the distribution table with bits $\varepsilon \in \{0,1\}^d$ lists the conditional probability that

$$S'_1[l_1] \oplus F'^j_1[l_1] \oplus S'_2[l_2] \oplus F'^j_2[l_2] \oplus Z'^j[t] = \varepsilon$$

given that the frame clock symbol is $FCS \triangleq C^j_{l_1+10,l_2+11}$. The distribution table is built in a similar way to [58]. The probability for bits combination ε is computed as follows: $Pr(\varepsilon|FCS) = \sum_i Pr(\varepsilon|Ev_i, FCS) \cdot Pr(Ev_i|FCS)$; the probability is calculated given the value of FCS and given that the clocking assumption holds before the first bit of $Z^j[t]$. The value of $Pr(Ev_i|FCS)$ is taken from the i^{th} row of the united pattern table for FCS. Let $\varepsilon = (\epsilon_1, \ldots, \epsilon_d)$, then $Pr(\varepsilon|Ev_i) = \prod_j Pr(\varepsilon|Ev_i)_j$, where

$$Pr(\varepsilon|Ev_i)_j = \begin{cases} 0 & \epsilon_j = 1 \text{ and pattern bit } j \text{ at row } i \text{ is correct} \\ 1 & \epsilon_j = 0 \text{ and pattern bit } j \text{ at row } i \text{ is correct} \\ 0.5 & \epsilon_j = 1 \text{ and pattern bit } j \text{ at row } i \text{ is random} \\ 0.5 & \epsilon_j = 0 \text{ and pattern bit } j \text{ at row } i \text{ is random} \end{cases}$$

For example, to compute the probability that $\varepsilon = (1,0,1,0)$, we combine all the events that could cause it (i.e., $Pr(\epsilon|Ev_i) > 0$). The events that could cause it are Ev_5 (where the two random choices are both 1), Ev_7, Ev_{13}, and Ev_{15}. Therefore, $Pr(\varepsilon = 1010_2|0011_2) = Pr(Ev_5)/2^2 + Pr(Ev_7)/2^3 + Pr(Ev_{13})/2^3 + Pr(Ev_{15})/2^4 = (0 \cdot 2^2 + 0 \cdot 2 + 4 \cdot 2 + 4)/2^8 = 12/2^8$. The full distribution table which corresponds to Table 4.4 is given in Table 4.2. We can convert the conditional distribution tables to non-conditional by $Pr(\varepsilon) = \sum_{i=1}^{d} 2^{-i} \cdot Pr(\varepsilon|\text{first } \text{“}1\text{”} \text{ appears in location } i) + 2^{-d} Pr(\varepsilon|0 \ldots 0)$, as 2^{-i} is the probability that a binary string begins with $i-1$ zeros

Table 4.4: The United Pattern Table for $d = 4$ and the Frame Clock Symbol 0011_2
טבלת התבנית המאוחדת עבור $d = 4$ וסימבול שעון של מסגרת 0011_2

1^{st} output bit	2^{nd} output bit	3^{rd} output bit	4^{th} output bit	$Pr(Event\|$ $0011_2)$	Event
Correct	Correct	Correct	Correct	$0/2^4$	Ev_0
Random	Correct	Correct	Correct	$0/2^4$	Ev_1
Correct	Random	Correct	Correct	$0/2^4$	Ev_2
Random	Random	Correct	Correct	$0/2^4$	Ev_3
Correct	Correct	Random	Correct	$0/2^4$	Ev_4
Random	Correct	Random	Correct	$0/2^4$	Ev_5
Correct	Random	Random	Correct	$0/2^4$	Ev_6
Random	Random	Random	Correct	$0/2^4$	Ev_7
Correct	Correct	Correct	Random	$0/2^4$	Ev_8
Random	Correct	Correct	Random	$0/2^4$	Ev_9
Correct	Random	Correct	Random	$0/2^4$	Ev_{10}
Random	Random	Correct	Random	$0/2^4$	Ev_{11}
Correct	Correct	Random	Random	$2^2/2^4$	Ev_{12}
Random	Correct	Random	Random	$2^2/2^4$	Ev_{13}
Correct	Random	Random	Random	$2^2/2^4$	Ev_{14}
Random	Random	Random	Random	$2^2/2^4$	Ev_{15}

Correct means both the clocking and the step assumption hold, *Random* means that at least one of the assumptions do not hold.

and then a one and 2^{-d} is the probability that a d-bit binary string is all zeros. See the example given in Table 4.2.

It is interesting to note again what happens if the frame clock symbol $C^j_{l_1,l_2}$ begins with a "1" (which happens in about half the cases). In such a case, the resulting united pattern table contains only the last row, which is full of *Random*. Therefore, the resulting distribution table has a uniform distribution, and the estimator $E^j_{l_1,l_2}[S'|C]$ has a uniform value for the different values of S', i.e., we gain no information from this frame regarding the value of $S'_1[l_1] \oplus S'_2[l_2]$.

4.12 Appendix: Step 3 — Recovering the Third Register

In this step we receive a list of candidate pair values (s_1, s_2) for the value of (S_1, S_2). For each candidate pair (s_1, s_2), we recover candidate values for S_3 that are consistent with the keystream of one of the frame, as explained later in this section. For each triplet of candidate values for (S_1, S_2, S_3), we construct a candidate key. We discard wrong keys by performing trial encryptions and comparing the results to the known keystream.

We recover a value for S_3 through a method similar to the one briefly described by Ross Anderson at [2]. The idea is as follows. Given the state of $R1$ and the state of $R2$ at the beginning of the keystream (i.e., after 101 irregular clockings have occurred), and given the known-keystream of a particular frame, it is easy to recover the state of $R3$ at the beginning of the keystream: The rightmost bit of $R3$ is simply the XOR of the first bit of the keystream and the rightmost bits of $R1$ and $R2$. We then need to guess the clocking tap of $R3$ and accordingly clock the register. If $R3$ stands still (which happens in a quarter of the cases) then in half of these cases the keystream is inconsistent with the rightmost values of the registers, and we backtrack. If $R3$ stands still and the keystream is consistent, we do not need to re-guess the clock tap of $R3$ for the next clock. Thus, an inconsistency is expected to occur after an average of eight clocks, after guessing an average of about $8 \cdot 3/4 = 6$ bits. Using the linear feedback function of $R3$, we reconstruct the values on the left of the clocking tap from the bits that were already processed. After $R3$ is clocked eleven times, the feedback of the first processed output bits reaches the clocking tap, thus, no more bits needs to be guessed, and we can continue to check for consistency without further guesses. After $R3$ is clocked twelve times, the guesses made for the clocking tap reach the rightmost bit of $R3$, and the entire state of $R3$ is determined. As a result, for half the clocks, the output is inconsistent.

The total time complexity of this algorithm is quite low. Our non-optimized straightforward implementation on a 1.8GHz Intel mobile CPU has a throughput of about 12000 applications of this $R3$ recovery algorithm in a second (on a keystream inconsistent with the internal state of $R1$, $R2$ — which is the case in the majority of the cases in our attack).

We cannot directly use the above $R3$ recovery algorithm, as (S_1, S_2) is not the state of $(R1, R2)$ at the beginning of a keystream of a particular frame. We must first choose the frame whose known keystream is used, say frame j. We then add the contribution of the frame number j to the initial state of the registers, i.e., we compute $s_1 + F_1^j$ and $s_2 + F_2^j$. We now have the initial state of the registers

$R1$ and $R2$, but for the above $R3$ recovery algorithm, we require the internal state after 101 irregular clocks have occurred. Therefore, we guess the number of clocks that $R1$ is clocked during the 101 irregular clocks, and the number of clocks that $R2$ is clocked during this time. For each such guess, we apply the above $R3$ recovery algorithm to find the internal state of $R3$ at the beginning of the keystream. In case a consistent state for $R3$ is found, we need to rewind the state through the 101 irregular clockings to the initial state. We guess the number of clocks that $R3$ performs in the 101 irregular clockings, rewind its state accordingly, and run A5/1 forward to verify our guess. Once a consistent initial internal state is found for $R3$, we eliminate the effect of the frame number on the state of $R3$ by XORing it with F_3^j and obtain a candidate s_3 for S_3. From (s_1, s_2, s_3), we construct the candidate key by reversing the linear key-setup, and we verify the candidate key using a known-keystream of another frame. Our non-optimized implementation uses this method to test about 30 pairs of (s_1, s_2) per second.

4.12.1 Alternative Step 3 Using the Ten Zero Bits of K_c

An alternative Step 3 tries all the 2^{23} possible s_3 values for each pair of (s_1, s_2) and filters for the right key using trial encryptions. In [24] it was observed that in all the implementations they checked, ten bits of the key are fixed to zero, effectively reducing the keyspace of A5/1 from 64 bit to 54 bits. We do not know if operators world-wide continue this practice. However, assuming this practice continues, we can benefit from it: Since the key-setup is linear in the bits of the key, given s_1 and s_2, we can efficiently enumerate all the 2^{13} keys with the fixed zero bits. Thus, the time-complexity drops to 2^{13} trial encryptions for each (s_1, s_2) combination. We are not aware of any other attack on A5/1 (except for a brute-force exhaustive search) that can use the existence of the ten zero bits in the key.

There are two advantages to this alternative method: One advantage is the success rate. Without using this alternative, the number of clocks that $R3$ performs has to be guessed. The attack fails if the number of times that $R3$ is clocked is not covered by the guess. However, in the alternative Step 3, the correct S_3 is *always* found given the correct (S_1, S_2). The second advantage is significant in a situation of a high bit error rate. In such a case, it is difficult to find even a single frame without errors. Therefore, the method in the previous section fails (as we cannot reverse the keystream to find $S3$), while in the alternative Step 3, the correct S_3 can be found (as the keystream based on it would be in almost a complete match to the known keystream of the particular frame).

Chapter 5

Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs

In this chapter we formalize a general model of cryptanalytic time/memory tradeoffs for the inversion of a random function $f : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$. The model contains all the known tradeoff techniques as special cases. It is based on a new notion of *stateful random graphs*. The evolution of a path in the stateful random graph depends on a *hidden state* such as the color in the Rainbow scheme or the table number in the classical Hellman scheme. We prove an upper bound on the number of images $y = f(x)$ for which f can be inverted using a tradeoff scheme, and derive from it a lower bound on the number of hidden states. These bounds hold with an overwhelming probability over the random choice of the function f, and their proofs are based on a rigorous combinatorial analysis. With some additional natural assumptions on the behavior of the *online phase* of the algorithm, we prove a lower bound on its worst-case time complexity $T = \Omega(\frac{N^2}{M^2 \ln N})$, where M is the memory complexity. We describe several new variants of existing schemes, including a method that can improve the time complexity of the online phase (by a small factor) by performing a deeper analysis during the preprocessing phase.

The work described in this chapter is a joint work with Prof. Adi Shamir of the Weizmann Institute of Science, and Prof. Eli Biham. It was submitted to Crypto 2006.

5.1 Introduction

In this chapter we are interested in generic ("black-box") schemes for the inversion of one-way functions such as $f(x) = E_x(0)$, where E is any encryption algorithm, x

is the key, and 0 is the fixed plaintext zero. For the sake of simplicity, we assume that both x and $f(x)$ are chosen from the set $\{0, 1, \ldots, N-1\}$ of N possible values.

The simplest example of a generic scheme is exhaustive search, in which a pre-image of $f(x)$ is found by trying all the possible pre-images x', and checking whether $f(x') = f(x)$. The worst-case time complexity T of exhaustive search is N, and the space complexity M is negligible. Another extreme scheme is holding a huge table with all the images, and for each image storing one of its pre-images. This method requires a *preprocessing phase* whose time and space complexities T and M are about N, followed by an *online inversion phase* whose running time T is negligible and space complexity M is about N (we always measure the running time by the number of applications of f). Cryptanalytic time/memory tradeoffs deal with finding a compromise between these extreme schemes, in the form of a tradeoff between the time and memory complexities of the online phase (assuming that the preprocessing phase comes for free). Cryptanalytic time/memory/data tradeoffs are a variant which accepts D inversion problems and has to be successful in at least one of them. This scenario typically arises in stream ciphers, when it suffices to invert the function that maps an internal state to the output at one point to break the cipher. However, the scenario also arises in block ciphers when the attacker needs to recover one key out of D different encryptions with different keys of the same message [13, 18]. Note that for $D = 1$ the problem degenerates to a the time/memory tradeoff discussed above.

5.1.1 Previous Work

The first and most famous cryptanalytic time/memory tradeoff was suggested by Hellman in 1980 [48]. His tradeoff requires a preprocessing phase with a time complexity of about N and allows a tradeoff curve of $M\sqrt{T} = N$. An interesting point on this curve is $M = T = N^{2/3}$. Since only values of $T \leq N$ are interesting, this curve is restricted to $M \geq \sqrt{N}$. Hellman's scheme consists of several tables, where each table covers only a small fraction of the possible values of $f(x)$ using chains of repeated applications of f. Hellman rigorously calculated a lower bound on the expected coverage of images by a single table in his scheme. However, Hellman's analysis of the coverage of images by the full scheme was highly heuristic, and in particular it made the unjustifiable assumption that many simple variants of f are independent of each other. Under this analysis, the success rate of Hellman's tradeoff for a random f is about 55%, which was verified using computer simulations. Shamir and Spencer proved in a rigorous way (in an unpublished manuscript from 1981) that with overwhelming probability over the choice of the random function f, even the best Hellman table (with unbounded chains created from the best collection of start

points, which are chosen using an unlimited preprocessing phase) has essentially the same coverage of images as a random Hellman table (up to a multiplicative logarithmic factor). However, they could not rigorously deal with the full (multi-table) Hellman scheme.

In 1982, Rivest noted that in practice, the time complexity is dominated by the number of disk access operations (random access to disk can be many orders of magnitude slower than the evaluation of f). He suggested to use distinguished points to reduce the number of disk accesses to about $\sqrt{T}$. The idea of distinguished points was described in detail and analyzed in 1998 by Borst, Preneel, and Vandewalle [22], and later by Standaert, Rouvroy, Quisquater, and Legat in 2002 [68].

In 1996, Kusuda and Matsumoto [55] described how to find an optimal choice of the tradeoff parameters in order to find the optimal cost of an inversion machine. Kim and Matsumoto [51] showed in 1999 how to increase the precomputation time to allow a higher success probability. In 2000, Biryukov and Shamir [20] generalized time/memory tradeoffs to time/memory/data tradeoffs, and discussed specific applications of these tradeoffs to stream ciphers.

A new time/memory tradeoff scheme was suggested by Oechslin [67] in 2003. It saves a factor 2 in the worst-case time complexity compared to Hellman's original scheme. Another interesting work on time/memory tradeoffs was performed by Fiat and Naor [43, 44] in 1991. They introduce a rigorous time/memory tradeoff for inverting *any* function. Their tradeoff curve is less favorable compared to Hellman's tradeoff, but it can be used to invert any function rather than a random function.

A question which naturally arises is what is the best tradeoff curve possible for cryptanalytic time/memory tradeoffs? Yao [79] showed that $T = \Omega(\frac{N \log N}{M})$ is a lower bound on the time complexity, regardless of the structure of the algorithm, and where M is measured in bits. This bound is tight up to a logarithmic factor, in case f is a single-cycle permutation, for which a tradeoff of $TM = N$ is possible [48] (here M is measured in start points which take about $\log N$ bits to represent), but the question remains open for functions which are not single-cycle permutations. Can there be a better cryptanalytic time/memory tradeoff than what is known today?

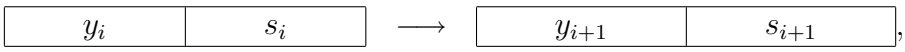
5.1.2 The Contribution of This Chapter

In this chapter we formalize a general model of cryptanalytic time/memory tradeoffs, which includes all the known schemes (and many new schemes). In this model, the preprocessing phase is used to create a matrix whose rows are long chains (where each link of a chain includes one oracle access to f), but only the start points and end points of the chains are stored in a table, which is passed to the online phase

(the chains in the matrix need not be of the same length).

The main new concept in our model is that of a *hidden state*, which can affect the evolution of a chain. Typical examples of hidden states are the table number in Hellman's scheme, and the color in a Rainbow scheme (we give more details on these schemes in Appendix A.3). The hidden state is an important ingredient of time/memory tradeoffs. Without the hidden state, the chains are paths in a single random graph, and the number of images that these chains can cover is extremely small (as shown heuristically in [48] and rigorously by Shamir and Spencer). We observe that in existing schemes, almost all of the online running time is spent on discovering the value of the hidden state (and hence the name *hidden* state). Once the correct hidden state is found, the online phase needs to spend only about a square root of the running time to complete the inversion.

The main effect of the hidden state is that it increases the number of possible states during the evolution of the chains in the preprocessing phase from N to NS, where S is the number of values that the hidden state can assume. The chains can be viewed as paths in a new directed graph, which we call the *stateful random graph*. Two nodes in the stateful random graph are connected by an edge:



if (y_{i+1}, s_{i+1}) is the (unique) successor of (y_i, s_i) defined by a deterministic transition function, where y_i and y_{i+1} are the output of the f function, and s_i, s_{i+1} are the respective values of the hidden state during the creation of y_i and y_{i+1}. The evolution of the y values along a path in the stateful random graph is "somewhat random" since it is controlled by the random function f. However, the evolution of the hidden state (s_i and s_{i+1}) can be totally controlled by the designer of the scheme.

The larger number of states is what allows chains to cover a larger number of images y. We rigorously prove that with an overwhelming probability over the choice of f, the number of images that can be covered by any collection of M chains is bounded from above by $2\sqrt{SNM\ln(SN)}$, where $M = N^\alpha$ for any $0 < \alpha < 1$. Intuitively it might seem that making S larger at the expense of N should cause the coverage to be larger (as S can behave more like a permutation). Surprisingly, S and N play the same role in the bound. The product SN remains unchanged if we enlarge S at the expense of N or vice versa. Note that $\sqrt{SNM}$ is about the coverage that is expected with the Hellman or Rainbow schemes, and thus even for the best choice of start points and path lengths (found with unlimited preprocessing time), there is only a small factor of at most $2\sqrt{\ln SN}$ that can be gained in the coverage. We use the above upper bound to derive a lower bound on the number S of hidden states.

Under some additional natural assumptions on the behavior of the online phase, we give a lower bound on the worst-case time complexity:

$$T \geq \frac{1}{1024 \ln N} \frac{N^2}{M^2},$$

where the success probability is at least 1/2 (the constant 1024 can be greatly improved by using a tighter analysis). Therefore, either there are no fundamentally better schemes, or their structure will have to violate our assumptions. Finally we show a similar lower bound of the form:

$$T \geq \frac{1}{1024 \ln N} \frac{N^2}{D^2 M^2}$$

on time/memory/data tradeoffs.

5.1.3 Structure of the Chapter

The model is formally defined in Section 5.2, and in Section 5.3 we prove the rigorous upper bound on the best achievable coverage of M chains in a stateful random graph. Section 5.4 uses the upper bound to derive a lower bound on the number of hidden states. The lower bound on the time complexity (under additional assumptions) is given in Section 5.5. Additional observations and notes appear in Section 5.6, and the chapter is summarized in Section 5.7.

A description of the main details of the time/memory tradeoffs of [48, 67] is given in Appendix A.3. A new time/memory tradeoff is described in Appendix 5.8. In Appendix 5.9, we describe a time/memory tradeoff scheme that violates our assumptions on the behavior of the online phase, and in Appendix 5.10 we compare the time complexity of the Hellman and Rainbow scheme. Finally, Appendix 5.11 contains the analysis of some new time/memory/data tradeoffs.

5.2 The Stateful Random Graph Model

The class of time/memory tradeoffs that we consider in this chapter can be seen as the following game: An adversary commits to a generic scheme with oracle accesses to a function f, which is supposed to invert f on a given image y. Then, the actual choice of f is revealed to the adversary, who is allowed to perform an unbounded *precomputation* phase to construct the best collection of M chains. Then, during the *online phase*, a value y is given to the adversary, who should find x such that

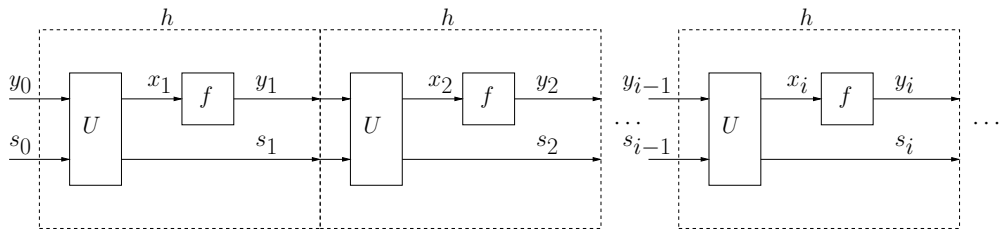


Figure 5.1: A Typical Chain — A Path in a Stateful Random Graph
שרשרת טיפוסית — מסלול בגרף אקראי בעל מצב

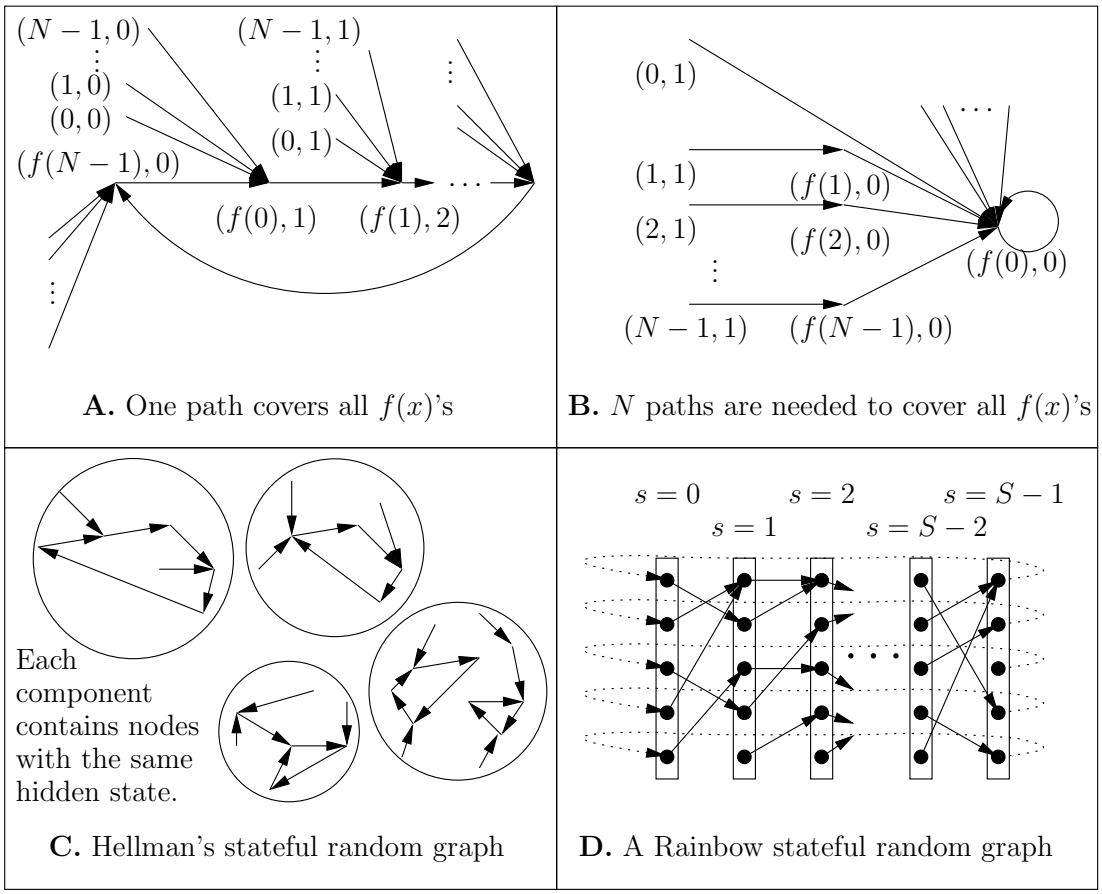
$f(x) = y$ using the scheme it committed to. The chains are not necessarily of the same length, and the collection of the M chains is called the *matrix*. We are interested in the time/memory complexities of schemes for which the algorithm succeeds with probability of at least $1/2$ for an overwhelming majority of random functions f.

In the model that we consider, we are generous to the adversary by not counting the size of the memory that is needed to represent the scheme that it has committed to. Having been generous, we cannot allow the adversary to choose the scheme after f is revealed, as the adversary can use his knowledge to avoid collisions during the chain creation processes, and thus cover almost all the images using a single Hellman table.[1]

We do not impose any restrictions on the behavior of the preprocessing algorithm, but we require that it performs all oracle accesses to f through a *sub-algorithm*. When the preprocessing algorithm performs a series of oracle accesses to f, in which each oracle access can depend on the result of previous oracle accesses in the series, it is required to use the sub-algorithm. We call such a series of oracle accesses a *chain*. The *hidden state* is the internal state of the sub-algorithm (without the input/output of f).

A typical chain of the sub-algorithm is depicted in Figure 5.1, where by U we denote the function that updates the internal state of the sub-algorithm and prepares the next input for f, and by h we denote the entire complex of U together with the oracle access to f. We denote by s_i the hidden state which accompanies the output

[1] A variant of the model is the auxiliary-memory model, in which we allow the scheme to depend on an additional collection of $M \ln N$ bits, which the adversary chooses during the preprocessing. Thus, we allow the adversary some customization of his scheme to the specific function f (within the limits of M memory rows). Analysis shows that the auxiliary-memory model is only marginally stronger (by small constant factor) than this model. Therefore, without loss of generality, we can discuss the model without auxiliary memory.



A. One path covers all $f(x)$'s

B. N paths are needed to cover all $f(x)$'s

C. Hellman's stateful random graph

D. A Rainbow stateful random graph

Figure 5.2: Four Examples of Stateful Random Graphs
ארבע דוגמאות לגרף אקראי בעל מצב

y_i of f in the sub-algorithm. The choice of U by the adversary together with f defines the stateful random graph, and h can be seen as the function that takes us from one node in the stateful random graph to the next node. U is assumed to be deterministic (if a non-deterministic U is desired, then the randomness can be given as part of the first hidden state s_0), and thus each node in the stateful random graph has an out-degree of 1.

Choosing U such that $s_i = s_{i-1} + 1 \pmod N$ and $x_i = s_{i-1}$ creates a stateful random graph that goes over all the possible images of f in a single-cycle (depicted in Figure 5.2.A), and thus represents exhaustive search (note that the y_{i-1} is ignored by U and thus all its N values with the same hidden state s_{i-1} converge to the

same node $(f(s_{i-1}), s_{i-1}+1))$. Such a cycle is very easy to cover even with a single path, but at the heavy price of using N hidden states. At the other extreme, we can construct a stateful random graph (see Figure 5.2.B) that requires a full lookup table to cover all images of f by choosing U as: *if* $s_{i-1}=1$ *then* $x_i = y_{i-1}$ and $s_i = 0$, *else* $x_i = s_i = 0$. In this function, each $(y_{i-1}, 1)$ is mapped by h to $(f(y_{i-1}), 0)$, and all these values are mapped to the same node $(f(0), 0)$.

As another example consider the mapping $x_i = y_{i-1}$ and $s_i = g(s_{i-1})$, where g is some function. This mapping creates a stateful random graph which is the direct product of the random graph induced by f, and the graph induced by g (this graph is not shown in the figure). We can implement Hellman's scheme by setting $x_i = y_{i-1} + s_i \pmod N$ and $s_i = s_{i-1}$, where s_i represents the table number to which the chain belongs. This stateful random graph (see Figure 5.2.C) consists of S disconnected components, where each component is defined by h and a single hidden state. Finally, we can implement a Rainbow scheme by setting $x_i = y_{i-1} + s_{i-1} \pmod N$ and $s_i = s_{i-1} + 1 \pmod S$, where S is the number of colors in the scheme. This stateful random graph (see Figure 5.2.D) looks like a layered graph with S columns and random connections between adjacent columns (including wrap-around links).

The preprocessing algorithm can perform any preprocessing on a *start point* of the chain before executing the sub-algorithm on that point, and any postprocessing on the *end point* of the chain (for example, before storing it in long-term memory). The preprocessing algorithm can stop the sub-algorithm at any point, using any strategy that may or may not depend on the value of the hidden states and the results of the oracle accesses, and it can use unbounded amount of additional space during its execution. For example, in Hellman's original method, the chain is stopped after t applications of f. Therefore, the internal state of the preprocessing algorithm must contain a counter that counts the length of the chain. However, the length of the chain does not affect the way the next link is computed, and therefore this counter can be part of the internal state of the preprocessing algorithm rather than the hidden state of the sub-algorithm. As a result, only the table number has to be included in the hidden state of Hellman's scheme. In the Rainbow scheme, however, the current location in the chain determines the way the next link is computed, and thus the index of the link in the chain must be part of the hidden state.

The preprocessing algorithm can store in a *table* only the start points and end points of up to M chains, which are used by the *online algorithm*. Note that the requirement of passing information from the preprocessing phase to the online phase only in the form of chains does not restrict our model in any way, as the sub-algorithm that creates the chains can be designed to perform any computation. Moreover, the

preprocessing algorithm can encode any information as a collection of start points, which the online algorithm can decode to receive the information. Also note that this model of a single table can accommodate multiple tables (for example, Hellman's multiple tables) by including with each start point and end point the respective value of the hidden-state.

The input of the online algorithm is y that is to be inverted, and the table generated by the preprocessing algorithm. We require that the online algorithm performs all oracle accesses to f (including chain creation) through the same sub-algorithm used during the preprocessing. In the variant of time/memory/data tradeoffs, the input of the online algorithm consists of D values $y_1, y_2, \ldots, y_D$ and the table, and it suffices that the algorithm succeeds in inverting one image. This concludes the definition of our model.

In existing time/memory tradeoffs, the online algorithm assumes that the given $y = f(x)$ is covered by the chains in the table. Therefore, y appears with some hidden state s_i, which is unfortunately unknown. The algorithm sequentially tries all the values that s_i can assume, and for each one of them it initializes the sub-algorithm on (y, s_i). The sub-algorithm executed a certain number of steps (for example, until an end point condition has been reached). Once an end point that is stored in the table has been found, the start point is fetched, and the chain is reconstructed to reveal the x_i such that $y = f(x_i)$.[2] Existing time/memory/data tradeoffs work in a similar way, and the process is repeated for each one of the D given images.

5.2.1 Coverage Types and Collisions of Paths in the Stateful Random Graph

A Table with M rows induces a certain coverage of the stateful random graph. Each row in the table contains a start point and an end point. For each such pair, the matrix associated with the table contains the chain of points spanned between the start point and the end point in the stateful random graph. The set of all the points (y_i, s_i) on all these chains is called the *gross coverage* of the stateful random graph that is induced by the table.

The gross coverage of the M paths is strongly affected by collisions of paths. Two paths in a graph collide once they reach a common node in the graph, i.e., two links in two different chains have the same y_i value and the same hidden state s_i. From

[2]Note that the fact that an end point is found does not guarantee a successful inversion of y. Such a case is called a false alarm, and it can be caused, for example, when the chain that is recreated from y merges with a chain (whose end point is stored in the table) that does not contain y.

this point on, the evolution of the paths is identical (but, the end points can be different). As a result, the joint coverage of the two paths might be greatly reduced (compared to paths that do not collide). It is important to note that during the evolution of the paths, it is possible that the same value y_i repeats under different hidden states. However, such a repetition does not cause a collision of the paths.

To analyze the behavior of the online algorithm, we are interested in the *net coverage* (denoted by C), which is the number of different y_i values that appear during the evolution of the M paths, regardless of the hidden state they appear with, as this number represents the total number of images that can be inverted. Clearly, the gross coverage of the M paths is larger than or equal to the net coverage of the paths.

When we ask what is the maximum gross or net coverage that can be gained from a given start point, we can ignore the end point and allow the path to be of unbounded length, since eventually the path loops (as the graph is finite). Once the path loops, the coverage cannot grow further. An equivalent way of achieving the maximum coverage of M paths is by choosing the end point of each path to be the point (y_i, s_i) along the path whose successor is the first point seen for the second time along this path.

5.3 A Rigorous Upper Bound on the Maximum Possible Net Coverage of M Chains in a Stateful Random Graph

In this section we formally prove the following upper bound on the net coverage:

Theorem 4 *Let $A = \sqrt{SNM \ln(SN)}$, where $M = N^\alpha$, for any $0 < \alpha < 1$. For any U with S hidden states, with overwhelming probability over the choice of $f : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$, the maximum net coverage C of images ($y = f(x)$) values) on any collection of M paths of any length in the stateful random graph of U is bounded from above by $2A$.*

This theorem shows that even though stateful random graphs can have many possible shapes, the images of f they contain can only be significantly covered by using many paths or many hidden states (or both), as defined by the implied tradeoff formula above. Without loss of generality, we can assume that $S < N$, since otherwise the claimed bound is larger than N, and clearly, the net coverage can never exceed N.

	M_1	M_2	$\cdots$	$M_{\binom{NS}{M}}$
f_1	0	0		
f_2	1	0		
$\vdots$			$\ddots$	
f_{N^N}				

Figure 5.3: A Table W denoting for each function f_i whether the net coverage obtained from the set of start points M_j is larger (1) or smaller (0) than $2A$

טבלה W המציינת לכל פונקציה f_i האם הכיסוי נטו המתקבל על ידי קבוצה של M_j נקודות התחלה גדול (0) או קטן (1) מאשר $2A$

5.3.1 Reducing the Best Choice of Start Points to the Average Case

In the first phase of the proof, we reduce the problem of bounding the best coverage (gained by the best collection of M start points) to the problem of bounding the coverage defined by a random set of start points and a random f. We do it by constructing a huge table W (as shown in Figure 5.3) which contains a row for each possible function f, and a column for each possible set of M start points. In entry $W_{i,j}$ of the table we write 1 if the net coverage obtained by the set M_j of start points for the embedded function f_i (extended into paths of unbounded length) is larger than our bound ($2A$), and we write 0 otherwise. Therefore, a row with all zeros means that there is no set of start points for this embedded function that can achieve a net coverage larger than $2A$.

To prove the theorem, it suffices to show that the number of 1's in the table, which we denote by $\#1$, is much smaller than the number of rows, which we denote by $\#r$ (i.e., $\#1 \ll \#r$). From counting considerations, it follows that the vast majority of rows contain only zeros, and the correctness of the theorem follows.

We can express the number of 1's in the table by the number of entries multiplied by the probability that a random entry in the table contains 1, and require that the product is much smaller than $\#r$, i.e., $\#1 = \text{Prob}(W_{i,j} = 1) \cdot \#c \cdot \#r \ll \#r$, where $\#c$ is the number of columns in the table. Therefore, it suffices to show that for a random embedded function and random set of start points, $\text{Prob}(W_{i,j} = 1) \cdot \#c$ is very close to zero. We have thus reduced the problem of proving that the coverage in the best case is smaller than $2A$, to bounding the number of columns multiplied by the probability that the average case is larger than $2A$. This is proven in the next few subsections.

5.3.2 Bounding $\text{Prob}(W_{i,j} = 1)$

We bound $\text{Prob}(W_{i,j} = 1)$ by constructing an algorithm that counts the net coverage of a given function f and a given set of M start points, and analyzing the probability that the coverage is larger than $2A$. During this analysis, we would like to consider each output of f as a new and independent coin flip, as $\text{Prob}(W_{i,j} = 1)$ is taken over a uniform choice of the function f. However, this assumption is justified only when x_i does not appear as an input to f on any previously considered point. In this case we say that x_i is *fresh*, and this freshness is a sufficient condition for f's output to be random and independent of any previous event.

Denote by $\stackrel{x_i}{\rightarrow}(y_i, s_i)$ the event of reaching the point (y_i, s_i), where x_i is the input of f during the application of h, i.e., $y_i = f(x_i)$. When we view the points $(y_i = f(x_i), s_i)$ as nodes in the stateful random graph, the value x_i is a property of the edge that enters (y_i, s_i), rather than a property of the node itself, since the same (y_i, s_i) might be reached from several preimages. The freshness of x_i (at a certain point in time) depends on the order in which we evolve the paths (the x_i is fresh the first time it is seen, and later occurrences of x_i are not fresh), but it should be clear that the net coverage of a set of paths is independent of the order in which the paths are considered.

The algorithm is described in Figure 5.4. It refers to the ratio A/S, which for the sake of simplicity we treat in the rest of the analysis as an integer. Note that $A/S \geq 2\sqrt{M \ln(NS)}$ (as $S < N$), and $A/S \gg 1$ (as N grows to infinity) since $M = N^{\alpha}$. Thus, the rounding of A/S to the nearest integer causes only a negligible effect.

Lemma 1 *At the end of the algorithm $|NetCoverage|$ is the size of the net coverage.*

Proof We observe that the algorithm processes all the points (y_i, s_i) that are in the coverage of the chains originating from the M start points, since it only stops a path when it encounters a collision.

A necessary condition for a $y_i = f(x_i)$ to be counted in the net coverage is that y_i appears in an event $\stackrel{x_i}{\rightarrow}(y_i, s_i)$ that is not a collision and in which x_i is fresh. If this condition holds, the algorithm reaches Step 6a, and adds y_i to $NetCoverage$. ■

At the end of the algorithm

$$NetCoverage = \cup_{i=1}^{S}(LowerFreshBucket_i \cup UpperFreshBucket_i),$$

and thus

$$|NetCoverage| \leq \sum_{i=1}^{S}(|LowerFreshBucket_i| + |UpperFreshBucket_i|),$$

1. For $i \in \{1, \ldots, S\}$ $Bucket_i = LowerFreshBucket_i = UpperFreshBucket_i = \phi$.
2. $NetCoverage = SeenX = \phi$.
3. Apply h to the first start point to generate the first event $\stackrel{x_i}{\rightarrow}(y_i, s_i)$.
4. if y_i appears in $Bucket_{s_i}$ Jump to Step 7 (Collision is detected). Otherwise:
5. Add y_i to $Bucket_{s_i}$.
6. If x_i does not appear in $SeenX$ (i.e., x_i is fresh):
   (a) If y_i does not appear in $NetCoverage$, add it to $NetCoverage$.
   (b) If $|LowerFreshBucket_{s_i}| < A/S$, add y_i to $LowerFreshBucket_{s_i}$, otherwise, add y_i to $UpperFreshBucket_{s_i}$.
7. Move to the next event:
   - Add x_i to $SeenX$ (i.e., mark that x_i is no longer fresh)
   - If a collision was detected in Step 4, apply h to the next start point (stop if there are no unprocessed start points). Otherwise: generate the next event by applying h to (y_i, s_i).
8. Jump to Step 4.

Legend:

- $SeenX$ is used to determine freshness by storing all the values of x that have been seen by now. This is the only set that stores input values of f. All the other sets store output values of f.
- $Bucket_i$ stores the all the y's that have been seen along with hidden state i (used for collision detection).
- $NetCoverage$ stores all the y's that have been seen from all chains considered so far, but without repetitions caused by different hidden states.
- For fresh values of x, $LowerFreshBucket_i$ stores the first A/S values of $y = f(x)$ seen with hidden state i (note that the x is fresh, but the y could have already appeared in other Buckets).
- For fresh values of x, $UpperFreshBucket_i$ stores the values of y after the first A/S values were seen with hidden state i (again, such a y could have already appeared in other Buckets).

Figure 5.4: A Particular Algorithm for Counting the Net Coverage
אלגוריתם מסויים לספירת הכיסוי נטו

since each time a y_i value is added to $NetCoverage$ (in Step 6a) it is also added to either $LowerFreshBucket$ or $UpperFreshBucket$ in Step 6b. We use this inequality to upper bound $|NetCoverage|$.

Bounding $\sum_{i=1}^{S} |LowerFreshBucket_i|$ is easy, as the condition in Step 6b assures

that for each i, $|LowerFreshBucket_i| \leq A/S$, and thus their sum is at most A. Bounding $\sum_{i=1}^{S} |UpperFreshBucket_i|$ requires more effort, and we do it with a series of observations and lemmas.

Our main observation on the algorithm is that during the processing of an event $\stackrel{x_i}{\rightarrow}(y_i, s_i)$, the value y_i is added to $UpperFreshBucket_{s_i}$ if and only if:

1. x_i is fresh (Step 6); and
2. $LowerFreshBucket_{s_i}$ contains exactly A/S values (Step 6b); and
3. (y_i, s_i) does not collide with a previous point placed in the same bucket (Step 4).

Definition 4 *An event $\stackrel{x_i}{\rightarrow}(y_i, s_i)$ is called a coin toss if the first two conditions hold for the event.*

Therefore, a y_i is added to $UpperFreshBucket_{s_i}$ only if $\stackrel{x_i}{\rightarrow}(y_i, s_i)$ is a coin toss (but not vice versa), and thus the number of coin tosses serves as an upper bound on $\sum_{i=1}^{S} |UpperFreshBucket_i|$.

Our aim is to upper bound the net coverage (number of images in the coverage) by the number of different x values in the coverage (which is equal to the number of fresh x's), and to bound the number of fresh x's by A (for lower fresh buckets) plus the number of coin tosses (upper fresh buckets).

Definition 5 *A coin toss $\stackrel{x_i}{\rightarrow}(y_i, s_i)$ is called successful if before the coin toss $y_i \in LowerFreshBucket_{s_i}$.*

Observe that each successful coin toss causes a collision, as $LowerFreshBucket_{s_i} \subseteq Bucket_{s_i}$ at any point in time, i.e., a successful coin toss means that the node (y_i, s_i) in the graph was already visited at some previous time (the collision is detected at Step 4). Note that a collision can also be caused by events other than a successful coin toss (and these events are not interesting in the context of the proof): For example, a coin toss might cause a collision in case $y_i \in Bucket_{s_i}$ (but $y_i \notin UpperFreshBucket_{s_i} \bigcup LowerFreshBucket_{s_i}$) before the coin toss. Another example is when x_i is not fresh, and therefore, $\stackrel{x_i}{\rightarrow}(y_i, s_i)$ is not a coin toss, but $y_i \in Bucket_{s_i}$ before the event (x_i was marked as seen in an event of a hidden state different than s_i).

Since each chain ends with the first collision that is seen, the algorithm stops after encountering exactly M collisions, one per path. As a successful coin toss causes a collision, there can be at most M successful coin tosses in the coverage.

Note that the choice of some of the probabilistic events as coin tosses can depend on the outcome of previous events (for example, $LowerFreshBucket_s$ must contain A/S points before a coin toss can occur for hidden state s), but not on the current outcome. Therefore, once an event is designated as a coin toss we have:

Lemma 2 *A coin toss is successful with probability of exactly $A/(SN)$, and the success (or failure) is independent of any earlier probabilistic event.*

Proof As x_i is fresh, $y_i = f(x_i)$ is truly random (i.e., chosen with uniform distribution and independently of previous probabilistic events). $LowerFreshBucket_{s_i}$ contains exactly A/S different values, and thus the probability that y_i collides with one of them is exactly $\frac{A/S}{N} = \frac{A}{SN}$. As all the other coin tosses have an x_i value different from this one, the value of $f(x_i)$ is independent of their values. ■

It is important to note that the independence of the outcomes of the coin tosses is *crucial* to the correctness of the proof.

What is the probability that the number of coin tosses in the M paths is larger than A? It is smaller than or equal to the probability that among the first A coin tosses there were fewer than M successful tosses, i.e., it is bounded by

$$\text{Prob}\left(\text{B}\left(A, q\right) < M\right),$$

where $q = A/(SN)$ and $\text{B}(A, q)$ is a random variable distributed according to the binomial distribution, namely, the number of successful coin tosses out of A independent coin tosses with success probability q for each coin toss.

Note that choosing A too large would result in a looser bound. On the other hand, choosing A too small might increase our bound for $\text{Prob}\left(W_{i,j} = 1\right)$ too much. We choose A such that the expected number of successes Aq in A coin tosses with probability of success q satisfies $Aq = M \ln(SN)$. This explains our choice of $A = \sqrt{SNM \ln\left(SN\right)}$.

It follows that:

$$\begin{aligned}
&\text{Prob}\left(W_{i,j} = 1\right) = \text{Prob}\left(|NetCoverage| > 2A\right) \\
&\quad \leq \text{Prob}\left(\sum_{i=1}^{S}(|LowerFreshBucket_i| + |UpperFreshBucket_i|) > 2A\right) \\
&\quad \leq \text{Prob}\left(A + \sum_{i=1}^{S}(|UpperFreshBucket_i|) > 2A\right) \\
&\quad = \text{Prob}\left(\sum_{i=1}^{S}(|UpperFreshBucket_i|) > A\right) \leq \text{Prob}\left(\text{B}\left(A, q\right) < M\right).
\end{aligned}$$

The first inequality holds as $\sum_{i=1}^{S}(|LowerFreshBucket_i|+|UpperFreshBucket_i|) > |NetCoverage|$. The last inequality holds as the number of coin tosses upper bounds $\sum_{i=1}^{S}(|UpperFreshBucket_i|$.

We bound $\text{Prob}\left(\text{B}\left(A,q\right)<M\right)$ by $M\cdot\text{Prob}\left(\text{B}\left(A,q\right)=M\right)$ because the binomial distribution satisfies $\text{Prob}(\text{B}(A,q)=b)\geq\text{Prob}(\text{B}(A,q)=b-1)$ as long as $b<(A+1)q$, and in our case $b=M$ while $(A+1)q=Aq+q=M\ln(NS)+q>M$ (as $Aq=M\ln(NS)$). Therefore, we conclude that

$$\text{Prob}\left(W_{i,j}=1\right)\leq\text{Prob}\left(\text{B}\left(A,q\right)<M\right)\leq M\cdot\text{Prob}\left(\text{B}\left(A,q\right)=M\right).$$

5.3.3 Concluding the Proof

To complete the proof we show that $\text{Prob}(W_{i,j}=1)\cdot\#c$ is very close to zero by bounding $\#c\cdot M\cdot\text{Prob}\left(\text{B}\left(A,q\right)=M\right)$.

In the following equations, we use the bound $\binom{x}{y}\leq x^y/y!\leq(xe/y)^y$, since from Stirling's approximation $y!\geq(y/e)^y$. We bound $(1-q)^{-M}$ by estimating that $q=\frac{A}{SN}=\sqrt{\frac{M\ln(SN)}{SN}}=\sqrt{\frac{\ln(SN)}{SN^{1-\alpha}}}$ is very close to 0, certainly lower than 0.5 (recall that $M=N^\alpha$, and $\alpha<1$). Thus, $1-q$ is larger than 0.5, and $(1-q)^{-M}$ must be smaller than $(2)^M$. Moreover, as $q>0$ is very close to 0, we approximate $(1-q)^A$ as e^{-Aq}.

Since each column in W is defined by a subset of M out of the NS start points, $\#c=\binom{NS}{M}$, and thus

$$\begin{aligned}
&\#c\cdot M\cdot\text{Prob}\left(\text{B}\left(A,q\right)=M\right)\\
&=\binom{NS}{M}M\binom{A}{M}\left(q\right)^M\cdot\left(1-q\right)^{A-M}\leq Me^{-Aq}\left(\frac{2e^2AqNS}{M^2}\right)^M\\
&\text{and substitute } Aq=M\ln\left(SN\right)\\
&=Me^{-M\ln(NS)}\left(\frac{2e^2NSM\ln(NS)}{M^2}\right)^M\\
&=M(NS)^{-M}\left(\frac{2e^2NS\ln(NS)}{M}\right)^M=M\left(\frac{2e^2\ln(NS)}{M}\right)^M=N^\alpha\left(\frac{2e^2\ln(NS)}{N^\alpha}\right)^{N^\alpha}.
\end{aligned}$$

When N grows to infinity the expression converges to zero, which concludes the proof.

5.4 A Lower Bound for S

We now analyze the minimum S required by the scheme. By Section 5.3, the net coverage of even the best set of M chains contains at most $2\sqrt{SNM\ln(SN)}$ distinct y_i values. To make the success probability at least one half, we need a net coverage of at least $N/2$. Therefore (recalling that $S \leq N$),

$$N/2 \leq 2\sqrt{SNM\ln(SN)} \leq 2\sqrt{SNM\ln(N^2)}.$$

From this, we can derive the following rigorous lower bound on the number of hidden states in any time/memory tradeoff which covers at least half the space with high probability:

$$S \geq \frac{N}{32M\ln N}.$$

5.5 A Lower Bound on the Time Complexity

We lower bound the worst-case time complexity of the online phase under the following natural assumption on its behavior:

- Given y, the online algorithm works by sequentially trying the hidden states (in any order). For each hidden state s, it applies h on (y, s) at least t_s times in case (y, s) does not appear in a chain in the matrix, where t_s is the largest distance from any point with hidden state s in the matrix to its corresponding end point. Note that the t_s values can depend on the specific matrix that results from the precomputation (and thus depend on the function f).

A simplistic "proof" for the lower bound is to say that with overwhelming probability $S \geq \frac{N}{32M\ln N}$, and for each hidden state we should run on average half the width of the matrix (i.e., $\frac{N}{4M}$). Multiply the two figures to receive the "bound":

$$T \geq \frac{N^2}{128M^2\ln N}.$$

However, it should be clear that this proof is incorrect, as for example, there can be a correlation between the hidden state and the length of the path we have to explore. One example of such a correlation is the Rainbow scheme, in which some hidden states appear only near the end points. Moreover, there can be more hidden states close to the end points than hidden states far from the end points, which shifts the average run per hidden state towards the end points. In the rest of the section we

rigorously lower bound the running time in the worst case, based only on the above assumption.

Preparation: align the chains in the matrix such that their end points are aligned in a column. Consider the $l = \frac{N}{4M}$ columns which are adjacent to the end points. The sub-matrix which constitutes these l columns contains at most $N/4$ different images $f(x)$. We call this sub-matrix the *right sub-matrix*, and the rest of the matrix the *left sub-matrix*. As $M = N^\alpha$, l is large enough so we can round it to the nearest integer (with negligible effect).

The worst case (with regards to the time complexity) is when the input y to the algorithm is not an image under f, or y is an image under f but is not covered by the matrix. Then, the time complexity is at least the sum of all the lengths t_s. We divide the hidden states into two categories: *short hidden states* for which $t_s \leq l$, and *long hidden states* for which $t_s > l$.[3] We would like to show that the number of long hidden states S_L is large, and use the time complexity spent on long hidden states as a lower bound on the total time complexity.

The net coverage of $f(x)$ images in the left sub-matrix must be at least $N/4$ images which do not appear in the right sub-matrix (since the total net coverage is at least $N/2$). Note that all the $N/4$ images in the left sub-matrix must be covered only by the S_L long hidden states, as all the appearances of short hidden states are concentrated in the right sub-matrix. In other words, the left sub-matrix can be viewed as a particular coverage of at least $N/4$ images by M continuous paths that contain only the S_L long hidden states.

It is not difficult to adapt the coverage theorem to bound the coverage of the left sub-matrix (using only long hidden states). The combinatorial heart of the proof remains the same, but the definitions of the events are slightly changed. For more details see Appendix 5.12. The adapted coverage theorem implies that with an overwhelming probability, the number of long hidden states satisfies

$$S_L \geq \frac{N}{64M \ln((SN)^2)} \geq \frac{N}{256 \ln N}.$$

Since for each long hidden state $t_s \geq l$, the total time complexity in the worst case is at least

$$T \geq l \cdot S_L \geq \frac{N}{4M} \frac{N}{256M \ln N} \geq \frac{1}{1024 \ln N} \frac{N^2}{M^2}.$$

Note that we had to restrict the length of t_s such that it includes all occurrences of the hidden state s in the matrix, as otherwise (and using the unlimited preprocessing), each chain could start with a prefix consisting of all the values of $f(x)$, and

[3]Note that the distinction between short and long hidden states is unrelated to the number of images that appear with these hidden states.

thus any image in the rest of the chain (the suffix) cannot be a fresh occurrence. The algorithm can potentially encode in the hidden state information about the x_i and $f(x_i)$ values seen in the prefix, in such a way that it can change the probability of collision (and in particular, avoid collisions). Note that the preprocessed chains are very long, but the online phase can be very fast if it covers only the suffixes of each path. As a result, we cannot use the methods of our proof in such a case.

In Appendix 5.9, we present an algorithm that violates the assumption by spending less time on wrong guesses of the hidden state compared to the correct guesses of the hidden states. The resulting matrices are called *stretched* matrices, and allow the algorithm to achieves a time complexity which is better by a small factor compared to the known time/memory tradeoffs (but still far from the lower bound above), at the price of a lengthier preprocessing.

5.5.1 A Lower Bound on the Time Complexity of Cryptanalytic Time/Memory/Data Tradeoffs

The common approach to construct a time/memory/data tradeoff is to use an existing time/memory tradeoff, but reduce the coverage (as well as the preprocessing) of the tables by a factor of D. Thus, out of the D images, one is likely to be covered by the table. The decrease in coverage reduces the number of hidden states, and thus the time complexity per image is reduced by a factor of D^3. However, the tradeoff might need to be applied D times in the worst case (for the D images), which results in an overall decrease in the time complexity by a factor of D^2 (note that the D time/memory tradeoffs can be executed in parallel, which can reduce the average time complexity in some cases). Using similar arguments and assumptions to the ones in the case of time/memory tradeoff, it follows that the worst-case time complexity can be lower bounded by

$$T' \geq D\frac{1}{1024D^3 \ln N}\frac{N^2}{M^2} = \frac{1}{1024D^2 \ln N}\frac{N^2}{M^2}.$$

5.6 Notes on Rainbow-Like Schemes

5.6.1 A Note on the Rainbow Scheme

The worst-case time complexity of the original Rainbow scheme was claimed to be half that of Hellman's scheme. However, the reasoning behind the claim considers only the number of start points and end points, and completely disregards the actual number of bits that are needed to represent these points. What [67] ignores is that

the start points and end points in Hellman's scheme can be compressed twice as much as in the Rainbow scheme. If we double M in Hellman's scheme to get a fair comparison, we can reduce T by a factor of four via the time/memory tradeoff, which actually outweights the claimed improvement by a factor of two in the Rainbow scheme (ignoring possible complications such as false alarms). For more details, see Appendix 5.10.

5.6.2 Notes on Rainbow Time/Memory/Data Tradeoffs

The original Rainbow scheme does not provide a time/memory/data tradeoff, but only a time/memory tradeoff. The natural way to generalize the Rainbow scheme to a time/memory/data tradeoff is to reduce the number of colors, which can be reduced in several ways. The first method is to reduce the number of colors to S by repeating the series of colors t times:

$$f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} ... f_0 f_1 f_2 ... f_{S-1},$$

we call the resulting matrix a *thin-Rainbow* matrix. The stateful random graph can be described by $x_i = y_{i-1} + s_{i-1} \pmod N$ and $s_i = s_{i-1} + 1 \pmod S$. The resulting tradeoff[4] is $TM^2D^2 = N^2$, which is similar to the tradeoff in [20], i.e., we lose the claimed improvement (by a factor of 2) of the original Rainbow time/memory tradeoff. However, like the Rainbow scheme, this method still requires twice as many bits to represent its start points and end points, and thus it is far inferior to [20]. Additional details can be found in Appendix 5.11.

The second method is to group the colors together in groups of t, and a typical row looks like:

$$\underbrace{f_0 f_0 f_0 ... f_0}_{t \text{ times}} \underbrace{f_1 f_1 f_1 ... f_1}_{t \text{ times}} \underbrace{f_2 f_2 f_2 ... f_2}_{t \text{ times}} ... \underbrace{f_{S-1} f_{S-1} f_{S-1} ... f_{S-1}}_{t \text{ times}},$$

we call the resulting matrix a *thick-Rainbow* matrix. Note, however, that during the online phase the algorithm needs to guess not only the "flavor" i of f_i, but also the phase of f_i among the other f_i's (except for the last f_i). In fact, the hidden state is larger than S and includes the phase, as the phase affects the development of the chain. Therefore, the number of hidden states is $t(S-1)+1$ (which is almost identical to the number of hidden states in the original Rainbow scheme), and we get an inferior tradeoff of $TM^2D = N^2$. On the other hand, we retain the claimed savings of 2 in

[4]When we write a time/memory/data tradeoff curve, the relations between the parameters relate to the expected worst-case behavior when the algorithm fails to invert y, and neglecting false-alarms.

the time complexity. This example demonstrates the difference between "flavors" of f and the concept of a hidden state.

The new strategy we propose to implement a Rainbow-like time/memory/data tradeoff is to use the notion of distinguished points not only to determine the end of the chain, but also to determine the points in which we switch from one flavor of f to the next. In this case, the number of hidden states is equal to the number of flavors, and does not have to include any additional information. We can specify U as: $x_i = y_{i-1} + s_{i-1} \pmod N$, and *if* y_{i-1} is special, *then* $s_i = s_{i-1} + 1 \pmod S$ *else* $s_i = s_{i-1}$, where y_{i-1} is special if its $\log_2 t$ bits are zeros. We call the resulting matrix a *fuzzy-Rainbow* matrix, as each hidden state appears in slightly different locations in different rows of the matrix. The tradeoff curve is $2TM^2D^2 = N^2 + ND^2M$, with $T \geq D^2$. The factor two savings is gained when $N^2 \gg ND^2M \Rightarrow D^2M \ll N$ (which happens when $T \gg D^2$). The number of disk accesses is about $\sqrt{2T}$, when $D^2M \ll N$, but is never more than in thin-Rainbow scheme for the same memory complexity. Additional details are given in Appendix 5.11.

5.7 Summary

In this chapter we proved that in our very general model, and under the natural assumption on the structure of the online phase, there are no cryptanalytic time/memory tradeoffs which are better than existing time/memory tradeoffs, up to a logarithmic factor.

Acknowledgements

We would like to thank Joel Spencer for his contribution to the proof of the single table coverage bound in 1981, and Eran Tromer for his careful review and helpful comments on earlier versions of the chapter.

5.8 Appendix: A Time/Memory Tradeoff with Hidden State that Depends Only on the Previous Values in the Chain

Consider the following time/memory tradeoff scheme, in which we choose $x_i = y_{i-1} + s_{i-1} \pmod N$. We choose $s_i = s_{i-1} + y_{i-1} \pmod S$, where S is the number of hidden

states. The hidden state S is chosen to be equal to the chain length. The rest of the details are similar to Hellman's scheme.

Analysis similar to the other tradeoffs results in a $TM^2 = N^2$ tradeoff curve. We have simulated this tradeoff and verified that it gives similar performance compared to Hellman's original time/memory tradeoff.

We can convert the time/memory tradeoff to a time/memory/data tradeoff by reducing the hidden state from the chain length to the chain length divided by D, as well as reducing the number of memory rows by a factor of D. The resulting tradeoff is $TM^2D^2 = N^2$.

5.9 Appendix: Stretching Distinguished Points — A Time/Memory Tradeoff Scheme with a Deeper Preprocessing

The main observation behind this algorithm is that most of the time complexity of the algorithm is spent on wrong guesses of the hidden state. Therefore, there are two effective ways to reduce the time complexity: reduce the number of hidden states, and reduce the time that is spent on wrong guesses of the hidden state.

When distinguished points are used, there is variance in the length of the chains. Assuming the chain length is distributed according to the geometric distribution with success probability p (i.e., a point is distinguished with probability p), the expected chain length is $(1-p)/p \approx p^{-1}$. The standard deviation is $\sqrt{(1-p)/p^2} \approx p^{-1}$. Therefore, there is a large variation in the length of chains, and it is not surprising to find chains which are several times longer than their expected length.

Storing the longer chains in the matrices seems to accomplish both of the effective ways of reducing the time complexity: as the chains are longer, each matrix covers more, and less matrices are needed (i.e., the hidden state is reduced). Moreover, the time spent on wrong guesses is the average chain length, which is smaller than the average chain length in the table (as the table stores chains longer than the average). The suggested scheme is essentially Hellman's scheme with distinguished points, but we prefer to store longer chains in the matrices. The scheme performs a longer precomputation, in which many chains are created. Only the longer chains are stored in the matrices, and the shorter chains are discarded. We call the resulting matrices *stretched matrices*, as they contain longer (stretched) chains.

Another possible source of savings in the time complexity is having an idea choice of parameters for the scheme. Consider Hellman's time/memory tradeoff with distinguished points. Hellman suggests to fill a matrix until $mt^2 = N$, where

$f : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$ is a random function, m is the number of rows in the matrix, and t is the chain length. Adding rows beyond the $mt^2 = N$ matrix stop rule becomes increasingly difficult. However, from the point of view of tradeoff efficiency, it is worthwhile adding rows to the matrix only until the moment that where we gain more by adding a row in a new matrix (rather than adding the row to the existing matrix), i.e., the time savings using the time/memory tradeoff curve is better than adding a row. We the optimal point in the next few paragraphs.

Let S be the number of the hidden states, i.e., S the number of tables, let C be the number of distinct points that are covered by a single matrix, and let T be the time complexity as anticipated by the "regular" tradeoff. For a single matrix, let m be the number of rows, and let

$$\gamma = m/(Np^2) \tag{5.1}$$

i.e., γ is the fraction of the number of rows compared to a single Hellman table (for Hellman $\gamma = 1$). The total number memory rows is $M = Sm$. Therefore,

$$\gamma = M/(SNp^2). \tag{5.2}$$

Let

$$\beta(\gamma) = C(\gamma)/(Np), \tag{5.3}$$

i.e., $\beta(\gamma)$ is the fraction of the coverage gained by a single table with $\gamma(Np^2)$ rows compared to the maximum coverage that is gained from a single Hellman table.

In this paragraph we show that the worst-case time complexity $T = \frac{\gamma^2 N^2}{\beta^3 M^2}$. The number of required tables is $S = N/C(\gamma) = N/(\beta(\gamma)Np) = 1/(\beta(\gamma)p)$ (to reach a constant success probability). Substitute $p = 1/(\beta(\gamma)S)$ in $\gamma = M/(SNp^2)$ and express S as:

$$S = \frac{\gamma N}{\beta^2(\gamma)M}. \tag{5.4}$$

Substitute S back to

$$p = \frac{1}{\beta(\gamma)S} = \frac{\beta(\gamma)M}{\gamma N}. \tag{5.5}$$

The worst-case time complexity (ignoring false alarms) is

$$T = S/p, \tag{5.6}$$

as evaluating each matrix takes on average p^{-1} applications of f. In the equation for T, we substitute p and S with their respective expressions from above:

$$T = \frac{S}{p} \tag{5.7}$$

$$= \frac{\gamma^2 N^2}{\beta^3 M^2} \tag{5.8}$$

For Hellman method with distinguished points, the time complexity falls back to $T = \frac{N^2}{M^2}$. However, we are interested in the optimal value of γ that minimize the time complexity. We calculate the minimum (through the derivative of T), and reach the condition that:

$$\frac{d\beta}{d\gamma} = \frac{2\beta}{3\gamma}. \tag{5.9}$$

Suggested is the following *stretching algorithm* to construct a single *stretched* Hellman matrix with distinguished points:

1. Choose a work factor k.
2. Create kNp^2 rows, if two or more rows have the same end point, keep the longest row.
3. Sort the rows by their length.
4. Add rows to the final matrix, longest-row first.
5. Let L be the total length of the rows added until now, m be the number of the rows that have been added until now, and l be the length of the added row. Do not add the row and stop when $lp < 2Lp/(3m)$, i.e., $l < 2L/(3m)$ (alternatively, add new rows until β^3/γ^2 reaches a maximum).

The stop condition is equivalent to $\frac{d\beta}{d\gamma} = \frac{2\beta}{3\gamma}$.

As the matrix covers L distinct points using m rows, $\beta = L/(Np)$ and $\gamma = m/(Np^2)$. The time savings using this method is the ratio $\beta^3/\gamma^2 = L^3 p/(Nm^2)$, which we call the gain factor. The actual work factor, is the ratio between the time spent during preprocessing compared to the time that is spent during the construction of a regular Hellman matrix to achieve this coverage: $kNp/(L)$.

It is interesting to observe that the above method gains from the fact that the average time spent on wrong guesses of the hidden state is the average chain length p^{-1}. This figure is a several times smaller than the average chain length in the matrix.

Table 5.1: Experiments Results of the Stretching Algorithm
תוצאות ניסיוניות של אלגוריתם המתיחה

k	Gain Factor	Actual Work Factor
2^0	≈ 2.1	≈ 4.3
2^1	≈ 2.9	≈ 5.9
2^2	≈ 4	≈ 9
2^3	≈ 4.8	≈ 14.5
2^4	≈ 5.6	≈ 24.4
2^5	≈ 6.1	≈ 43
2^6	≈ 6.6	≈ 80.3

As most of the time complexity is spent on wrong guesses of the hidden state, the method gains the difference.

Experimental results of the stretching algorithm are shown in Table 5.1. It should be noted that this method can be adapted to other schemes that are based on distinguished points, such as in Appendix 5.11.

5.10 Appendix: Time Complexity of Hellman Versus Rainbow

It is surprising that the preprocessing and postprocessing that the algorithm can perform on start and end points is substantially different in the different schemes. For example, the start points in Hellman's scheme (using $M = N^{2/3}$) can be compressed to half of the size of what the start points in a Rainbow scheme can be compressed to (for $M = N^{2/3}$). This factor two increase in the memory complexity translates to a factor four degredation in the time complexity, which consumes the savings that are introduced by a Rainbow scheme (compared to Hellman's scheme). However, the real advantage of Rainbow over Hellman's scheme is more complicated as it involves other factors such as the false alarm rate. In Hellman's scheme, $(\log_2 N)/3$ bits are enough to store the start points: the y value can be constructed by setting the first $(\log_2 N)/3$ bits to zeros, the next $(\log_2 N)/3$ bits to the hidden state (table number, which can be globally stored), and the last $(\log_2 N)/3$ bits to be an index (only the index bits need to be stored). In a Rainbow scheme, however, the hidden state is identical to all the start points, and therefore, only $2(\log_2 N)/3$ bits per

start points are needed (these bits are used to store the index of the start point). We can overcome this disadvantage of the Rainbow scheme by dividing the single matrix to many smaller matrices each starting with another hidden state (and having $y_i = y_{i-1} + 1 \pmod{S}$), but this modification will also eliminate the factor 2 savings in the worst-case time complexity of the Rainbow scheme (and increase the number of disk accesses).

5.11 Appendix: Analysis of the New Cryptanalytic Time/Memory/Data Tradeoffs

5.11.1 Trivial Rainbow Time/Memory/Data Tradeoff: $TM^2D = N^2$

The memory is left the same — M, but each row is shortened to t/D elements. The new Rainbow matrix covers Mt/D points, which represent constant fraction of N/D of the space. This implies $Mt = N$, which when raised to the power of 2 is:

$$M^2t^2 = N^2.$$

The total running time is about

$$T = Dt^2/D^2 = t^2/D,$$

substitute t^2 in the equation $M^2t^2 = N^2$ and get:

$$TM^2D = N^2.$$

As $t/D \geq 1$ it follows that $t \geq D \Rightarrow t^2 \geq D^2 \Rightarrow TD \geq D^2$, and therefore,

$$T \geq D.$$

5.11.2 Thin-Rainbow Time/Memory/Data Tradeoff: $TM^2D^2 = N^2$

The matrix contain M rows of memory. Each row contains t sequences of S colors, i.e., it looks like:

$$f_0f_1f_2...f_{S-1}f_0f_1f_2...f_{S-1}f_0f_1f_2...f_{S-1}...f_0f_1f_2...f_{S-1}.$$

Therefore, the length of each row is St. The matrix stop rule in this case is $Mt^2S = N$ (as analyzed in Appendix 5.11.4). When raised to the power of two:

$$M^2t^4S^2 = N^2.$$

We require that the matrix covers N/D elements, i.e.,

$$MSt = N/D.$$

Therefore, $S = N/(DMt)$. Substituting S in $M^2t^4S^2 = N^2$ gives $t^2 = D^2$, i.e.,

$$t = D.$$

The total time is $T = DStS = DtS^2 = DtN^2/(DMt)^2 = N^2/(DtM^2)$, as for each data point we go over all the S colors, and continue the chain for a length of about St (the length is actually $St - s$, where s is the current hidden state, but we neglect s compared to St, which is accurate for $D \gg 1$.). Substitute t with D to achieve the tradeoff curve:

$$T = N^2/(D^2M^2).$$

As $S = N/(DMt)$ is at least 1, it follows that $N \geq DMt \Rightarrow N \geq D^2M \Rightarrow N^2/M^2 \geq D^4$. Substitute N^2/M^2 with TD^2 and get that:

$$T \geq D^2.$$

The number of disk accesses is $DtS = DtN/(DMt) = N/(M) = D\sqrt{T}$, as for each hidden state we need to have t disk accesses (whenever the chain reaches hidden state S), and we repeat the search D times. The number of disk accesses can be reduced to $\sqrt{T}$ by using distinguished points to mark the points of hidden state S that can end a chain (a point with hidden state S should be distinguished with probability t^{-1}).

5.11.3 Fuzzy-Rainbow Time/Memory/Data Tradeoff: $2TM^2D^2 = N^2 + ND^2M$

The matrix contain M rows of memory. Each row contains about t repetitions of S colors, i.e., it looks like:

$$\underbrace{f_0f_0f_0\cdots f_0}_{\text{about } t}\underbrace{f_1f_1f_1\cdots f_1}_{\text{about } t}\underbrace{f_2f_2f_2\cdots f_2}_{\text{about } t}\cdots\underbrace{f_{S-1}f_{S-1}f_{S-1}\cdots f_{S-1}}_{\text{about } t},$$

where the U function changes the value of the hidden state when a distinguished point occurs (with probability t^{-1}). The chain is terminated when a distinguished point is reached for hidden state S. Therefore, the expected length of each row is St. The matrix stop rule in this case is $Mt^2S = N$ (as analyzed in Appendix 5.11.4). When raised to the power of two:

$$M^2t^4S^2 = N^2.$$

We require that the matrix covers N/D elements, i.e.,

$$MSt = N/D.$$

Therefore, $S = N/(DMt)$. Substituting S in $M^2t^4S^2 = N^2$ gives $t^2 = D^2$, i.e.,

$$t = D.$$

The total time is $T = D(S+1)St/2 = D^2S(S+1)/2 = N^2/(2D^2M^2) + N/(2M)$. It follows that:

$$2TD^2M^2 = N^2 + ND^2M.$$

As $S = N/(DMt)$ is at least 1, it follows that $N \geq DMt \Rightarrow N \geq D^2M \Rightarrow N^2/M^2 \geq D^4$. It follows that $T = N^2/(2D^2M^2) + N/(2M) \geq D^4/(2D^2) + D^2/2 = D^2$, i.e.,

$$T \geq D^2.$$

Note that when $T \gg D^2$, $ND^2M \ll N^2$, and the factor two in time savings is gained.

There is one disk access per hidden state (once we reach the end of the chain), and the search is repeated D times. Therefore, the number of disk accesses is $SD = N/(D^2M)D = N/(DM) \approx \sqrt{2T}$ (this figure is not higher than in the thin-Rainbow scheme).

5.11.4 Analysis of the Matrix Stop Rule in the Modified Rainbow Scheme

The thin-matrix contains M rows and looks like:

$$f_0f_1f_2...f_{S-1}f_0f_1f_2...f_{S-1}f_0f_1f_2...f_{S-1}...f_0f_1f_2...f_{S-1}$$

$$f_0f_1f_2...f_{S-1}f_0f_1f_2...f_{S-1}f_0f_1f_2...f_{S-1}...f_0f_1f_2...f_{S-1}$$

$$f_0f_1f_2...f_{S-1}f_0f_1f_2...f_{S-1}f_0f_1f_2...f_{S-1}...f_0f_1f_2...f_{S-1}$$

$$f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} ... f_0 f_1 f_2 ... f_{S-1}$$

$$\vdots$$

$$f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} ... f_0 f_1 f_2 ... f_{S-1},$$

where S is the number of hidden states, and each hidden state appears t times in each row (the same analysis follows for the fuzzy-Rainbow scheme). Suppose that the matrix contains M rows and we are in the process of adding the $M+1$ row. Assuming that all the points in the first M rows are distinct, the new row which looks like:

$$f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} ... f_0 f_1 f_2 ... f_{S-1}$$

collides with the matrix if at least one of its points with hidden state k collides with another point in the matrix with the same hidden state k. The probability that it happens is:

$$1 - (\text{prob. no collision}) = 1 - ((N - Mt)/N)^{St} \approx 1 - e^{-(Mt^2 S)/N}.$$

In birthday paradox, and in the matrix stop rule, we stop when the probability is about 0.5 ($1 - e^{-1}$ to be exact), which implies a matrix stop rule of $Mt^2 S = N$.

5.11.5 Notes

Note that the trivial Rainbow time/memory/data tradeoff scheme is not better than the other tradeoffs at any point. Consider the most extreme point that the tradeoff allows, i.e., when $T = D$. When using the other tradeoffs, what should be the data D', such that the memory complexity and the time complexity is identical?

Substitute $T = D$ in the original tradeoff to obtain $M = N/D$. Substitute the expression for T and M in the other tradeoff curves ($TM^2 D'^2 = N^2$):

$$D(N/D)^2 D'^2 = N^2.$$

It follows that

$$D' = \sqrt{D},$$

which is within the limits of the other tradeoff curves. Moreover, fewer data points are needed to achieve the same memory and time complexity.

We have verified the above tradeoffs through computer simulations.

5.12 Appendix: Extended Coverage Theorem

We can extend the coverage theorem to bound the net coverage that can be obtained by M paths, where the paths contain only S' hidden states out of the $S \geq S'$ hidden states of U. We call the hidden states in the set of S' hidden states *insider hidden states* and call the rest of the hidden states *outsider hidden states*.[5] Therefore, we are interested in the coverage of the sub-chains that begin in the start points and end before the first occurrence of an outsider hidden state. We call this set of sub-chains the *insider matrix*.

The tricky point is that the specific choice of the insider hidden states (including the number S' of insider hidden states) can depend on the choice of f, which is not a priori known to the algorithm that counts the coverage in the main proof. The crucial observation that solves the tricky point is that the only affect of the specific choice of the insider hidden states is the location in which the chains are terminated. In particular, the choice of the insider hidden states cannot affect the development of the chains, as the development of the chains is part of the definitions of U.

We can model the specific choice of insider hidden states by letting the adversary choose not only a set of M starting points, but also a corresponding set of M *termination* points. Each path starts in its starting points, and continues until the termination point for the path is encountered. If no termination point is encountered, the path can continue indefinitely (but eventually it loops). The net coverage is uniquely defined by U, f_i and the set of M starting points and their termination points. Therefore, it suffices to prove that given any U, it holds that for the overwhelming majority of functions f, there is no set of M start points and M termination points such that the resulting coverage in the insider matrix using S' hidden states is larger than $2A'$, where $A' = \sqrt{S'NM}\ln(SN)^2$.

We have the following upper bound on the coverage of the insider matrix:

Theorem 5 *Let $A' = \sqrt{S'NM\ln(SN)^2}$, where $M = N^\alpha$, for any $0 < \alpha < 1$. Let U be any update function with $S \leq N$ hidden states. For any choice of f, and for any set of M start points, let the adversary choose $S' \leq S$ and a set of S' insider hidden states. Then, with overwhelming probability over the choice of $f : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$, there is no choice of start points, S', and S' insider hidden states such that the net coverage in the resulting insider matrix is larger than $2A'$.*

The proof begins with a reduction using a huge table W, similar to the one in the

[5] Insider hidden states correspond to long hidden states, and outsider hidden states correspond to short hidden states. We make the distinction in the names to avoid confusion.

main proof. The number of functions (rows) remains N^N, but the number of columns jumps to $\#c' = S\binom{(NS)^2}{M}$, as there are S possibilities to choose S', and we choose M starting points termination points pairs, out of the $(NS)^2$ such pairs. Denote the j^{th} specific choice of M start points and termination points, and the choice of S' by M_j (i.e., the columns of W are marked $M_1, M_2, \ldots, M_{S\binom{(NS)^2}{M}}$. In each entry of the table we write one if and only if: the coverage of f_i using the choice M_j of M start points and corresponding termination points contains at most S' hidden states, and the net coverage is larger than $2A'$, where $A' = \sqrt{S'NM\ln((SN)^2)}$. Otherwise (if the there are more than S' hidden states in the coverage, or the net coverage is smaller than $2A$), we write zero.

It suffices to prove that the number of ones in the table is considerably smaller than the number of rows, as from counting arguments it would follow that most of the rows are zeros (and a row of zeros for a function f_i means that there is no choice of S' and M start points and their termination points, such that the resulting coverage indeed contains only S' hidden states, and the net coverage is larger than $2A'$). Therefore, like in the main proof, it suffices to prove that the product of the number of columns and the probability that $W_{i,j}$ is 1 is very close to zero.

We now wish to upper bound the probability that $W_{i,j}$ is one. We use a similar method to the one in the main proof, i.e., an algorithm that counts the net coverage. However, this time, the algorithm receives not only the set of M start points as input, but also the set of M termination points and the specific choice of S', in order for the algorithm to count the resulting coverage. The exploration of each chain is stopped once either a collision occurs or the termination point is reached.[6] The only remaining differences in the algorithm compared to the original one is that the threshold of the lower fresh bucket is changed from A/S to A'/S', and once the algorithm encounters more than S' different hidden states, it sets the net coverage to zero and halts.

The analysis of the algorithm is similar. $W_{i,j}$ is one only if the coverage net coverage counted by the algorithm is larger than $2A'$. The algorithm can count a net coverage larger than $2A'$ only if it encounters more than $2A'$ fresh x's. The fresh x's are stored in the lower and upper fresh buckets. $W_{i,j}$ can be one only if the coverage contains at most S' hidden states, and in this case, only the buckets for the S' hidden states contain any elements. Therefore, the number of elements in the lower fresh buckets is at most $S'(A'/S') = A'$.

The net coverage is larger than $2A'$ only if the upper fresh buckets contain at

[6] Like in the main proof, we do not lose any coverage by stopping a chain once it collides with a previously explored chain.

least A' elements. That means that there are at least A' coin tosses. The probability of a coin toss of being successful is $q' = A'/(S'N)$ (the independence argument still holds, as a coin toss is performed only for a fresh value of x). We choose A' such that $A'q' = M \ln((SN)^2)$, i.e. $A' = \sqrt{S'NM \ln((SN)^2)}$. As each successful coin toss causes a collision that ends a chain, there can be no more than M successful coin tosses. Therefore, the probability that the net coverage is larger than $2A'$ is smaller than

$$\text{Prob}\left(\text{B}\left(A', q'\right) < M\right).$$

In the conclusion of the proof, the number of columns is $\#c' = S\binom{(NS)^2}{M}$. The increase in the number of columns compared to the original proof is eliminated by the increase of $\ln(NS)$ (in Aq) from the original proof to $\ln((NS)^2)$ in $A'q'$. This concludes the modified proof.

Note that there is no real reason to insist that $S \leq N$ in our model, but the model would not be fair if we allow S to be huge, as too much information on f can be encoded by every choice of the hidden state (and we do not count the memory complexity of representing U). For example, if $S = N^N$, than with $S' = 1$ we can encode all the information on f by the specific choice of single insider hidden state (note that a huge amount of $N \log_2 N$ bits are required to just represent that single insider hidden state). In other words, for each function f, the stateful random graph contains a path that goes through all the images of that f, and using only a single hidden state. It is possible to adapt the model (to a fair one), where $S \leq N^k$, for some constant k. We can adapt our lower bound to this model, and it would be k times lower.

Appendix A

Introduction to some of the Contemporary Methods of Cryptanalysis

We give an introduction to some of the modern methods of cryptanalysis on which most modern cryptanalysis methods are based. In Section A.1 we discuss the main idea behind differential cryptanalysis and in Section A.2 the main idea behind linear cryptanalysis. Section A.3 discusses time/memory tradeoff for block ciphers. Algebraic attacks are discussed in Section A.4. In Section A.5 we give an overview of stream ciphers and basic methods for their cryptanalysis.

A.1 Introduction to Differential Cryptanalysis

Differential cryptanalysis [16] was introduced by Biham and Shamir in 1990. The basic idea of differential cryptanalysis is to study the evolution of differences during encryption of two plaintexts (under the same encryption key). In particular, the XOR-difference during the different rounds of encryption is analyzed: starting with the difference of the two plaintexts, continuing through differences of intermediate values, and ending with the difference of the ciphertexts.

A.1.1 Simple Examples

We explain the idea behind differential cryptanalysis through some simple examples. Consider an affine section of a cipher (or a cipher which is affine). The affine section

can be modeled by

$$Y = A \cdot X \oplus G \cdot K \oplus D,$$

where A and G are constant binary matrices, D is a constant binary vector, K is the key, and X,Y are the input and output, respectively. For a given key the affine section can be modeled by

$$Y = A \cdot X \oplus B,$$

where the vector B is defined as $B \triangleq G \cdot K \oplus D$. Consider two inputs X_1, X_2 of the affine section, whose XOR difference is

$$X' \triangleq X_1 \oplus X_2.$$

The output difference is then

$$Y' \triangleq Y_1 \oplus Y_2 = (A \cdot X_1 \oplus B) \oplus (A \cdot X_2 \oplus B) = A \cdot X'.$$

Thus, for any affine section of a cipher, the output difference Y' is fixed given the input difference X', while the constant B has no affect on the difference.

The fact that XOR with fixed values (e.g., subkey mixing) vanish from the difference is useful for the cryptanalyst. While key mixing vanishes from the difference, it still affects the encryption process value. In differential cryptanalysis the evolution of differences is studied without taking specific key values into account (it is actually assumed that the specific key values XOR the encrypted data appear random). Then, the attacker looks for a pairs of plaintexts (and their corresponding ciphertexts), whose XOR differences satisfy the desired conditions, and thus gain insight to the differences during the encryption process of specific plaintexts, i.e., the attacker is able to distinguish the cipher from a random one.

In the previous paragraphs we discussed an affine section of a cipher. We now take into account the non-linear sections of a cipher, i.e., we consider the effect of an S box in the encryption process. Consider a simple cipher $T = S(P \oplus K_1) \oplus K_2$, where S is a known invertible non-linear function (an S box), e.g., with n-bit input and n-bit output, and $K = (K_1, K_2)$ is the secret key. Denote the input value of S by I_j, and denote the output value of S by O_j, i.e., $O_j = S(I_j)$. Unlike the case of the affine section of the cipher, since S is non-linear, it is well possible that its output difference is not a function of only its input difference, but that the output difference of S also depends on the actual value of the inputs. For a given S box a table, called the *Difference Distribution Table*, lists the output difference distribution as a function of the input difference as follows: for every input difference $I' \triangleq I_1 \oplus I_2$ and for every output difference $O' \triangleq O_1 \oplus O_2$ of S, the number of pairs (I_1, I_2) that

have an input difference of I' and an output difference of O' is counted. Assume that for this specific S box the following discovery is made (after consulting the difference distribution table): for a certain input difference $I' = \Delta$ of S a quarter of the pairs have an output difference $O' = \delta$. This discovery can be used to break the cipher: the attacker requests the encryption of many plaintext pairs P_1, P_2, whose difference is $P' = P_1 \oplus P_2 = \Delta$, and check if their corresponding ciphertexts T_1, T_2 have a difference $T' = T_1 \oplus T_2 = S(P_1 \oplus K_1) \oplus S(P_2 \oplus K_1) = S(P_1 \oplus K_1) \oplus S(P_1 \oplus K_1 \oplus \Delta)$ equal to δ, i.e., check that $T' = \delta$ (note that in this example $I' = P'$ and $O' = T'$). Due to the discovery on average one in every four pairs satisfies this output difference. Once a pair with the desired ciphertext difference is found, the input and output differences of S during the encryption of this pair is known to the attacker, but the actual input and output values of S are still unknown (these values depend on the value of K_1 and K_2). From the discovery, $2^n/4$ input pairs (I_1, I_2) of the S function (out of the 2^n possible input pairs) yield the desired output difference. Each such pair suggests two possibilities for K_1: $I_1 \oplus P_1 = I_2 \oplus P_2$ and $I_1 \oplus P_2 = I_2 \oplus P_1 = I_1 \oplus P_1 \oplus \Delta$. However, for each pair (I_1, I_2), the dual pair (I_2, I_1) also has the same differences, and suggests the same suggestions for K_1. Therefore, there are $2^n/4$ suggestions for K_1 in total (of which one is the correct K_1). Note that the set of suggestions for K_1 is a shift (i.e., XOR) by P_1 of the set of values $L_{\Delta \mapsto \delta} \triangleq \{I_j \mid S(I_j) \oplus S(I_j \oplus \Delta) = \delta\}$. Different P_i's result in different sets of suggestions for K_1, but the correct value of K_1 must be in all these sets (i.e., the correct value K_1 is in the intersection). Therefore, with additional plaintexts and their ciphertexts (about $n/2$ pairs if the set $L_{\Delta \mapsto \delta}$ is randomly distributed), the attacker can narrow the range of possibilities to two (every plaintext pair with the plaintext difference Δ that suggests K_1, also suggests $K_1 \oplus \Delta$).

A.2 Introduction to Linear Cryptanalysis

Linear Cryptanalysis [57] was introduced by Matsui in 1993. Linear cryptanalysis studies the evolution of parities of data bits during the encryption process. The goal is to approximate sections of the cipher by an affine function. Assume that the approximation of the cipher as affine correctly predicts the first bit of the ciphertext with probability p. We expect a "good" cipher to be indistinguishable from a random permutation, and we therefore expect that $p \approx \frac{1}{2}$. Attacks based on linear cryptanalysis usually succeed when $|p - \frac{1}{2}|$ is large enough, thus allowing an attacker to distinguish the cipher from a random one.

A.2.1 Simple Examples

In this section we discuss many of the basic ideas of linear cryptanalysis through simple examples.

We start by analyzing the case of an affine cipher (similar analysis holds for a linear section of a cipher):

$$T = A \cdot P \oplus B \cdot K \oplus D, \tag{A.1}$$

where A and B are constant matrices, D is a constant vector, K is the key, and P, T are the plaintext and ciphertext of the cipher. Given a parity of a subset of the bits of the plaintext, we can predict a parity of a subset of the bits of the ciphertext. In other words, given an equation $\Omega_P \cdot P$, where Ω_P is a row-vector that defines the subset of bits of P, we can predict a parity after encryption as:

$$\Omega_T(T \oplus D) = \Omega_P P \oplus \Omega_K K, \tag{A.2}$$

where Ω_T is a row-vector that defines the subset of the bits of the ciphertext, and Ω_K is a row-vector that defines a subset of the bits of the key K. Given a single plaintext/ciphertext pair we recover one parity bit of the key, which is $\Omega_K \cdot K$. We now calculate the subset of bits Ω_T of the ciphertext T, and the subset Ω_K of the bits of the key K given the subset Ω_P of the bits of the plaintext. The first step is to find Ω_T such that

$$\Omega_P = \Omega_T A, \tag{A.3}$$

if any such Ω_T exists (in the majority of the cases A is invertible, and thus, there is a single $\Omega_T = \Omega_P A^{-1}$). If a few such Ω_T exist, we gain several bits of information on the key by repeating the analysis with the different Ω_T^i values each contributing one parity bit $\Omega_K^i K$ of the key. We compute Ω_K as follows: multiply Equation A.1 on the left by Ω_T as follows:

$$\Omega_T T = \Omega_T(AP \oplus BK \oplus D) \Rightarrow \Omega_T(T \oplus D) = \Omega_T AP \oplus \Omega_T BK = \Omega_P P \oplus \Omega_T BK.$$

By substituting

$$\Omega_T(T \oplus D)$$

in Equation A.2 we get that

$$\Omega_K = \Omega_T \cdot B.$$

For an affine section of a cipher (such as Equation A.1), it is nice to note the relation between linear and differential cryptanalysis. The XOR difference after the affine section is an application of the linear section of Equation A.1 (i.e., A) on

the XOR difference before the linear section, i.e., $Y' = AX'$, where X' is the input difference of A, and Y' is the output difference. Linear subsets are somewhat more complex. It is well known that the subset of the input (when written as a binary vector) is the application of Q on the subset of the output (i.e., $(\Omega_P)^T = Q(\Omega_T)^T$, note that the transpose is needed since Ω_P and Ω_T are defined as row-vectors), where Q is the matrix A with XOR and duplication replaced, and applied from the end to the beginning (e.g., if the last operation in A is $c = a \oplus b$, then the first operation in Q is $a = c, b = c$). Note that when A is written as a matrix, then $Q = A^T$. The reason is that "1" in entry i, j of the A matrix implies that the input j is XORed into the i^{th} output; Replacing XOR with duplication means that now the output i is XORed into the input j, which is the value in entry j, i of Q — which is equivalent to transposing the matrix A. The property can be proven as follows: simply apply the matrix transpose operation to Equation A.3:

$$(\Omega_P)^T = A^T \cdot (\Omega_T).^T$$

To see Ω_T as a function of Ω_T multiply the equation on the left by $(A^T)^{-1}$ and get:

$$(\Omega_T)^T = (A^T)^{-1} \cdot (\Omega_P)^T.$$

When analyzing a non-linear cipher, we approximate the S boxes with a *linear approximation of the S box*, which is in the form $X'X = Y'Y$, where $Y = S(X)$, and X', Y' define a subset of bits of X, and a subset of bits of Y, respectively. This equation is correct with probability $1/2 + q$, $q \in [-0.5, 0.5]$. For every choice of X' and Y' we get a different equation with a (possibly) different probability. We then try and concatenate the equations with the affine sections of the cipher in order to get a *linear approximation of the cipher*:

$$\Omega_T T = \Omega_P P \oplus \Omega_K K, \tag{A.4}$$

A linear approximation of the cipher has a probability $1/2+p$ associated with it. p is often referred to as the bias from $1/2$. Out of all the possible linear approximations of the cipher, we are most interested in the one with the maximal bias from $1/2$.

In an affine cipher from a single plaintext/ciphertext pair we recover one bit of information on the key (derived from Equation A.2). However, in the case of the non-affine cipher, given only a single plaintext P and its corresponding ciphertext T, we cannot determine if Equation A.4 holds. Subsequently, we cannot determine if

$$\Omega_K \cdot K = \Omega_T T \oplus \Omega_P P. \tag{A.5}$$

holds. Therefore, a statistical approach should be employed. Clearly, the left hand side of the Equation A.5 is fixed for a fixed key K and a given subset Ω_K. Therefore, we try to deduce its value by considering many plaintext/ciphertext pairs, and substituting them in the right hand side of the equation. Taking a set of N plaintext/ciphertext pairs, we count the number of times that the right hand side of the equation is zero — we denote the counter by M. Assuming $p > 0$, If $M > \frac{N}{2}$, then the right hand side of Equation A.5 is zero for most pairs, and we therefore deduce that $\Omega_K K = 0$. Otherwise, the right hand side is one for most pairs, and we that $\Omega_K K = 1$. If $p < 0$ we deduce the complement value. The number of analyzed plaintext/ciphertext pairs affects the success rate: in general, N has to be chosen to be in the order of p^{-2}.

A.3 Time Memory Tradeoffs

In 1980, Hellman [48] found a generic scheme for cryptanalysis using a time/memory tradeoff. The basic idea is to choose a fixed plaintext P, and then to treat the function that computes the ciphertext as a function of the key and the fixed plaintext as a random function, e.g., $f(x) = DES_x(P)$. If an attacker inverts f, then with a high probability he recovers the secret key. Note, however, that Hellman's method is not restricted to block ciphers, and actually it applies to any random function f. However, it is probably best understood by considering a specific cipher such as DES. In Section A.7 we give an overview of the application of time/memory tradeoffs to stream ciphers.

The time/memory tradeoff attack is composed of two phases: a preprocessing phase and an inversion phase. In the preprocessing phase the attacker explores the structure of f, and summarizes his findings in a huge table. In the inversion phase given $f(x)$ the attacker uses the precomputed table to invert $f(x)$.

A degenerate case of the tradeoff is as follows: During the preprocessing phase the whole keyspace is explored, and for every key k_i the value $f(k_i)$ is computed. The pair $(k_i, f(k_i))$ is stored in memory, indexed by $f(k_i)$. In the inversion phase, the attacker receives $f(k_i)$, and retrieves all k_i's indexed by $f(k_i)$. The inversion phase requires one fetch operation, but the memory requirement is huge and unrealistic. The other extreme degenerate case of the tradeoff does not perform any preprocessing, but the inversion phase performs exhaustive search.

Hellman's tradeoff allows any tradeoff point on the curve $\sqrt{T}M = N$, where M is the number of table rows (each row consumes about $2\log_2(N)$ bits of memory), T is the time complexity of the inversion phase, and N is the size of the key space.

We now give an overview of Hellman's method. During preprocessing the attacker

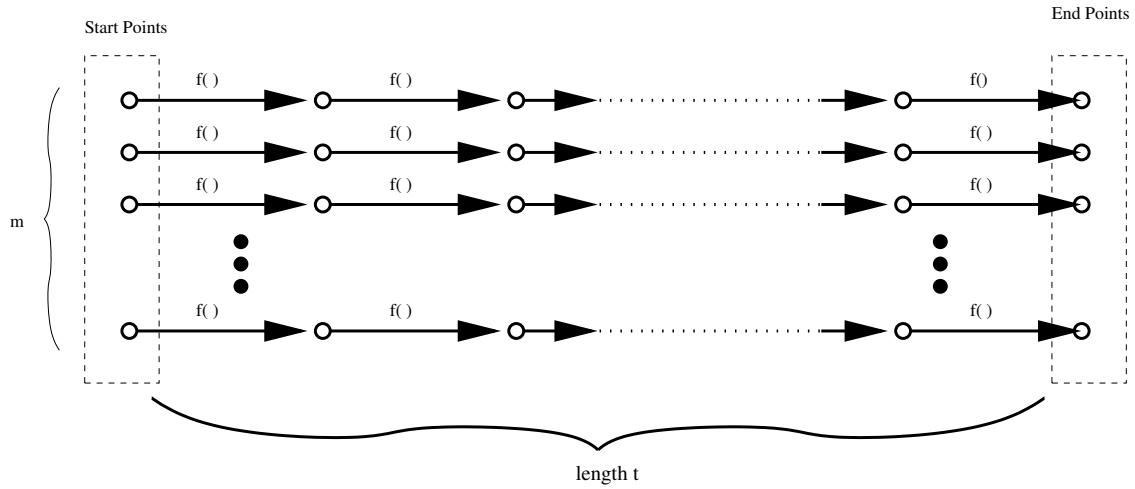


Figure A.1: Hellman's Matrix
המטריצה של הלמן

chooses m random starting points $k_1, \ldots, k_m$, where m is a parameter of the attack. For each starting point k_i the attacker computes $f^t(k_i)$, i.e., applies f iteratively t times

$$f^t(k_i) = \underbrace{f(f(f(\ldots f(k_i)\ldots)))}_{t \text{ times}},$$

where t is another parameter of the attack, and stores the pair $(k_i, f^t(k_i))$ in the table indexed by the $f^t(k_i)$ values (depicted in Figure A.1). To save space the attacker discards the intermediate values $f^1(k_i), \ldots, f^{t-1}(k_i)$. Note that this description is somewhat imprecise, as the input and output of f might not be of the same length. For example, in the case of DES, f has an input of 56 bits (key), and an output of 64 bits (ciphertext). Therefore, a simple reduction function R is used to reduce the output of f to the size of its input, e.g., by discarding the last eight bits.

In the inversion phase, the attacker receives $f(K)$. He searches in the table for the value $f(K)$. If it is found in the table (i.e., a pair $(k_i, f^t(k_i))$ for which $f^t(k_i) = f(K)$ is found), then either K is in the next to the last column in that row, or $f(K)$ has more than one preimage (the latter case is called a *false alarm*). Assume that $f(K)$ is an endpoint (and is therefore found in the search). In order to find K the attacker fetches k_i from the row of $f(K)$, and applies $f(\cdot)$ iteratively $t-1$ times to receive $K' = f^{t-1}(k_i)$. The attacker can test whether it is a false alarm with $K' \neq K$

by a trial encryption (of another plaintext). If $f(K)$ is not found in the table (or there is a false alarm), then the key K is not in the next column. In this case the attacker computes $f(f(K)) = f^2(K)$, and consults the table to see if $f^2(K)$ is an endpoint (if the predecessor of $f(K)$ is in the second next to the last column then $f^2(K)$ must be an endpoint). If $f^2(K)$ is not an endpoint, the attacker continues and iteratively applies f in a similar process to check if the predecessor of $f(K)$ appears in any column of the table. In the j^{th} application of f, he computes $f^j(f(K))$, and checks whether it is an endpoint. If it is, he computes the $(j+1)^{\text{th}}$ next to the last column by $K' = f^{t-j-1}(k_i)$. False alarms are then discarded by first checking that $f(K') = f(K)$, and by trial encryptions (of another plaintext).

The attack succeeds whenever the predecessor of $f(K)$ is covered by the precomputed table. The table covers at most mt points, thus if all points covered by the table are distinct, the probability of success is mt/N. However, Hellman noticed that due to the birthday paradox a single table cannot efficiently cover the whole space of N points. He calculated that a table for which $mt^2 > N$ is likely to have many collisions, and that if two points in two different rows collide, then all the points on their right side collide as well, and therefore that the table's coverage of the space becomes very poor.

Hellman's solution to this problem is to use t independent tables, each covers approximately N/t of the points. If the predecessor of $f(K)$ is not found in one table, then the next table is searched. He numerically calculated that if $mt^2 \approx N$ then the probability of success of a single table is about $0.80mt/N$. Hellman suggests to generate the different tables by using a slightly different reduction function, for example, by changing the reduction function R in a simple way (e.g., exchanging locations of bits). Note that although the change in the reduction function is mild, the cycle structure of the resulting functions is expected to be unrelated.[1] However, it should be noted that the structures of f_i and f_j are not independent, and actually they are dependent as both functions are based on the same underlying function f. This dependency could be problematic, and it was not taken into account in the analysis. A similar problem exists with the analysis of the Rainbow scheme.

The complexity of Hellman's tradeoff is as follows: In the preprocessing phase the function f is computed $O(N)$ times. Searching a single table takes t computations of f. In the worst case t tables are searched, resulting in a time complexity of $T = t^2$. Since $mt^2 = N$, and $M = mt$ is the size of the memory (t tables with m values each), it follows that $M\sqrt{T} = N$.

Hellman calculated the expected number of false alarms to be bounded by $\frac{mt(t+1)}{2N}$

[1] Also note, that if all the values of f reside in long cycles then one table can efficiently cover the whole space of N points.

per table. Since each false alarm causes at most t additional applications of f, and there are at most t tables to search in, the expected time complexity incurred by the false alarms is bounded by $\approx mt^4/(2N) = (NT)/(2N) = T/2$, therefore, false alarms add at most 50 percent to the attack's time complexity.

The inversion phase requires T table lookups, which in the case of DES is 2^{38} lookups in a huge memory of 2^{38} entries (by choosing $t = m = 2^{19}$). Assuming that a hard drive is used to store the tables, and the current technology's random seek time is about 5 milliseconds, there can be only about 200 table accesses in each second. The access time dominates the time it takes to evaluate f, e.g., on a 90MHz Pentium more than 200,000 f applications (based on DES encryptions with key setups) can be made every second.

An idea due to Rivest is to use *distinguished points*: A distinguished point is a point in the search space of N points whose first $\log_2 t$ bits have a fixed pattern, e.g., the first $\log_2 t$ bits are zero. In the preprocessing phase, for each a starting point k_i the function f is iterated until a distinguished point is reached. Thus, the endpoints are always distinguished points, and the resulting rows have an average length of about t. The benefit is that during the inversion phase, the attacker iteratively applies f over $f(K)$ until a distinguished point is reached and only then searches the table. Therefore, there is one database search for an average of t applications of f, reducing the number of database searches to $\sqrt{T}$. Note that some care is needed to break out of loops that contain no distinguished points.

Oechslin [67] recently suggested an improved time/memory tradeoff attack. Oechslin's idea is to use Hellman's original suggestion, but to use a different reduction function R_j with every iteration of f, i.e., every column in the matrix has a different reduction function associated with it (depicted in Figure A.2). Oechslin calls the resulting table a *rainbow table*. Rainbow tables induces a more efficient coverage of the search space by reducing the effect of collisions in the rows of the table. While in Hellman's suggestion if the same value appears in two different rows the rest of these rows is identical, in Oechslin's suggestions the same value must appear in two rows in the same column for a similar effect. A similar analysis to Hellman's suggestion shows that a collision in the same column in the matrix is likely to occur when $mt \approx N$, therefore, a single larger matrix can be used instead of t matrixes in Hellman's suggestion.

In the inversion phase of Oechslin's variant, the attacker is given $f(K)$, and needs to find the specific column in the matrix where the predecessor of $f(K)$ is located. Like in Hellman's suggestion, the attacker first assumes that the predecessor of $f(K)$ is in the column next to the last(thus $f(K)$ an endpoint), and searches the table for $f(K)$. If it is not found then the attacker assumes that the predecessor of $f(K)$ is in

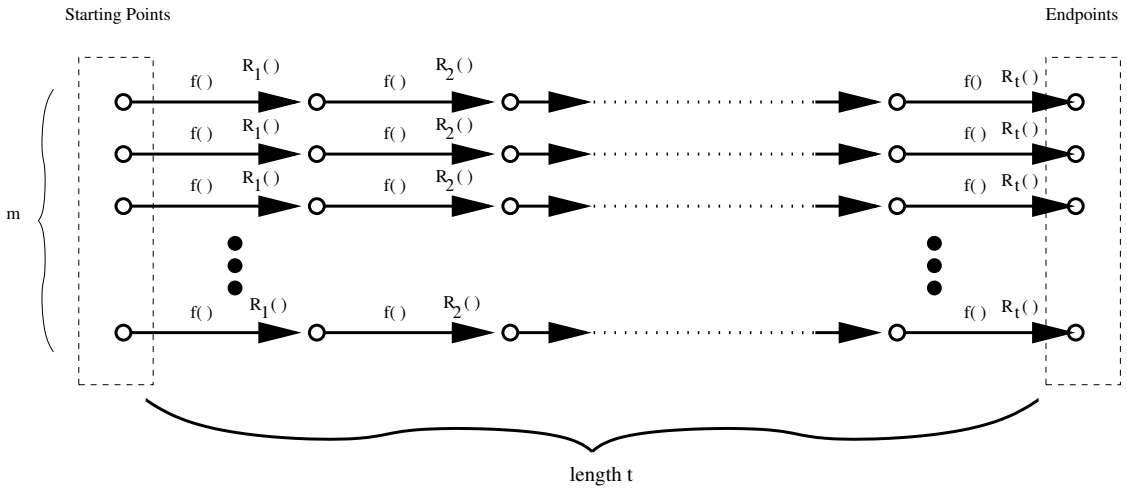


Figure A.2: Oechslin's Rainbow Matrix
מטריצת צבעי הקשת של אואשלין

the second column on the left of the last, like the case with Hellman's suggestion, the attacker applies f on $f(K)$ using the reduction function associated with this column. However, unlike the case of Hellman's suggestion the attacker cannot continue and apply f iteratively on the result of the previous trial, since the reduction functions are different. Therefore, for the j^{th} column on the left of the last the attacker must apply f using the relevant reduction functions $j-1$ times, and search the table once. The time complexity is therefore $\sum_{j=1}^{t} j-1 \approx t^2/2$ with t disk accesses. This result is faster by a factor of two compared to Hellman's tradeoff, and it requires only about $\sqrt{T}$ disk accesses.

A.4 Algebraic Attacks

Any block cipher can be described as a system of (complex) algebraic equation. For example, a bit of the ciphertext can be written in algebraic terms of the bits of the plaintexts and the bits of the key.

Every such system of equations can be written as a *quadratic system of equations* by introducing new variables, and using the new variables to break large terms to quadratic ones. For example, the equation system containing the single equation $x = yztw$ can be replaced by the three equations $x = yn_1$, $n_1 = zn_2$, and $n_2 = tw$.

Notice that the two systems of equations are equivalent, i.e., they have equivalent solutions, and the number of degrees of freedom is unchanged.

In algebraic attacks, the attacker writes the cipher (or sections of it) as algebraic equations, and then, given enough plaintext/ciphertext pairs, solves the set of equations to recover the secret key. In public-key cryptosystems the attacker might recover the plaintext given the ciphertext and the public key.

A basic tool in algebraic attacks is an algorithm that solves the system of the quadratic equations. The problem of solving a random system of quadratic equations is NP-complete. However, the systems of equations that arise in cryptology are not random. They are overdefined (or can be made overdefined given sufficiently many known plaintexts), usually very sparse, and they always have a solution. In contrast, a random overdefined quadratic system is not expected to have any solution.

In recent years there has been an increasing interest in algebraic attacks, both in developing algorithms to efficiently solve the kind of systems of equations that arise in cryptology, and also in attacks against specific ciphers. In 1999 Kipnis and Shamir developed the *relinearization* [52] method to solve overdefined systems of quadratic equations, and used it in an attempt to attack the HFE public key cryptosystem. Later, Courtois, Klimov, Patarin, and Shamir developed the *XL* [30] method that can be seen as an improvement of relinearization. *XSL* [31], which is focused at solving sparse systems, was developed by Courtois and Pieprzyk in an attempt to attack block ciphers in general, and Rijndael in particular. In 2003 Courtois and Meier presented an algebraic attack against the stream cipher Toyocrypt [28, 29]. In the following subsections we give a brief description of linearization, relinearization, and XL.

A.4.1 Linearization

A system of $n^2/2$ quadratic equations in n variables can be simplified by *linearization* to a linear system with about $n^2/2$ equations and about $n^2/2$ variables as follows. We rename every quadratic term to a name of a new variable (i.e., every instance of the term $x_i x_j$ is replaced by the new variable y_{ij}). Gauss elimination is then used to find the solution. Given the solution, checking for consistency (i.e., whether $y_{ij} = x_i x_j$) can be performed. The time complexity of this method is the time complexity of performing Gauss elimination on a $\frac{n^2}{2} \times \frac{n^2}{2}$ matrix. It succeeds whenever the system is sufficiently overdefined.

A.4.2 Relinearization

In [52] Shamir and Kipnis describe the relinearization technique, which is focused as solving overdefined systems of quadratic equations. The idea is to use linearization, but also to include additional higher-degree equations that describe the fact that multiplication is commutative, i.e., $y_{ij}y_{kl} = y_{ik}y_{jl}$. The resulting system is linearized again, and the process repeats itself until there is an equation that contains only one variable. Other methods are then used to solve this equation. In all the equations this variable is substituted with its value, and the process is repeated to find the value of the other variables. The entire relinearization algorithm can be described as follows:

1. Linearize the system of quadratic equations, name new variables as $y_{ij} = x_i x_j$.
2. Perform Gauss elimination. If a variable is found, stop and simplify the equations. If the system is not solved, it has $l < \frac{n\cdot(n+1)}{2}$ degrees of freedom. After the gauss elimination, it is easy to describe every variable as a linear combination of the l variables which are not eliminated. Therefore, express every variable y_{ij} as a linear combination of the l variables $t_1, \ldots, t_l$, where the $t_1, \ldots, t_l$, are the y_{ij}'s that are not eliminated.
3. Create a new system of quadratic equations that express the commutativity, e.g., $y_{ij}y_{kl} = y_{ik}y_{jl} = y_{il}y_{jk}$.
4. Express the new equations using the t_i's, i.e., substitute every appearance of y_{ij} in the quadratic equations with y_{ij}'s representation as a linear combination of $t_1, \ldots, t_l$. The resulting system is now quadratic in the t_i variables.
5. Solve the new system (by linearization, or perhaps recursively with relinearization).
6. Using $t_1, \ldots, t_l$, find the solution of the original system.

Assume that the original system has $m = \epsilon n^2$ equations. What is the minimum required ϵ for the system to be solvable by linearization in step 5? Assuming linear independence of the derived equations in step 3 (the equations that express the commutativity of multiplication) their number is as follows: there are about $n^4/4!$ ways to choose unsorted indices i, j, k, l, each one adds two new quadratic equations. These equations are translated in step 4 to the same number of equations ($2n^4/4! = n^4/12$) above the $t_1, \ldots, t_l$ variables. The number of degrees of freedom in step 2 is the number of variables minus the number of equations: $l = \frac{n\cdot(n+1)}{2} - \epsilon n^2 \approx (\frac{1}{2} - \epsilon)n^2$.

The new system can be linearized and solved successfully (in step 5) when the number of equations is larger or equal to the number of variables, i.e., $\frac{n^4}{12} \geq \frac{((\frac{1}{2}-\epsilon)n^2)^2}{2} \Rightarrow \epsilon \geq \frac{1}{2} - \frac{1}{\sqrt{6}} \approx 0.1$.

There are higher degree variants of relinearization. The above method is referred to as degree 4 relinearization, since the equations that we add are of degree 4 in the original variables. In higher degree variants, we include equations with higher degrees, i.e., 6 (in which we include equations of the form $y_{ij}y_{kl}y_{mn} = y_{ik}y_{jm}y_{ln}$), 8, 10, etc.

A.4.3 The XL Method

Here we describe the XL method, which can be seen as an improvement of relinearization. It is aimed at solving overdefined multivariate equation systems.

The input of the algorithm is a system of m multivariate quadratic equations with n variables. We denote the k^{th} equation by $l_k = 0$, where l_k is the multivariate quadratic polynomial $f_k(x_1, x_2, \ldots, x_n) - b_k$, where b_k is a constant. The equations are over the finite field K, therefore, from now on, we assume that the exponent is always reduced modulo $|K| - 1$ (as $a^{|K|} = a$). XL tries to find a solution $x = (x_1, \ldots, x_n)$ for which $l_k(x) = 0$ holds for any k.

An important component of the XL algorithm multiplies a certain l_t with all possible terms of degree k, i.e., to generate the set of equations of the form $(\prod_{j=1}^{k} x_{i_j}) \cdot l_t$. For each equation l_t the XL algorithm with degree $D \in \mathbb{N}$ creates a set of equations with degree D by multiplying l_t by all possible terms in $\{\prod_{j=1}^{k} x_{i_j}\}$ (for quadratic equations $k = D - 2$). The XL algorithm is as follows (assuming that all the input equations are quadratic):

1. **Multiply:** Generate all the products $(\prod_{j=1}^{k} x_{i_j}) \cdot l_t$ with $k \leq D - 2$.

2. **Linearize:** Consider each monomial $\prod_{j=1}^{k} x_{i_j}$ as a new variable $y_{i_1,i_2,\ldots,i_k}$ and perform Gauss elimination on the equations obtained in step 1. The ordering of the monomials must be such that terms containing one original variable (x_t) are eliminated last.

3. **Solve:** Assume that step 2 yields at least one univariate equation (in the powers of the original variable x_t, e.g., $y_{tt} + y_{tttt} + y_{tttttt} = \alpha$ which represents the equation $x_t^2 + x_t^4 + x_t^7 = \alpha$). Solve this equation over the finite field (e.g., by using Berlekamp's algorithm).

4. **Repeat:** Simplify the equations by substituting x_t with the found value, (e.g., assume that $x_t = \beta$, then we substitute $y_{\underbrace{tt...t}_{i \text{ times}}}$ by β^i), and repeat the algorithm from step 3 to find the values of the other variables.

Note that the XL algorithm does not guarantee that a solution is found. It is well possible that step 3 does not result with any univariate equation. In such a case we can try XL with a larger degree D.

In [30] it is proved that XL of degree D can solve all the equation systems that relinearization of degree D can solve. Moreover, XL uses less variables than relinearization and its complexity is lower. Also, XL might solve cases where relinearization fails.

A.5 Introduction to Stream Ciphers and Their Analysis

In this section we discuss basic stream ciphers, and review the main attacks against them.

Stream ciphers usually encrypt the plaintext, using a transformation that changes with time. The data is encrypted in small blocks, whose size is usually one bit, or one byte. By contrast, block ciphers usually encrypt the plaintext in large blocks (usually of size 64 or 128 bits) using a permutation that does not change with time. Stream ciphers usually have a simpler design than block ciphers, and are usually designed to be very fast, and to be efficiently implemented in hardware (or software). In many applications it is very natural to use stream ciphers, especially when the plaintext is given one bit (or one byte) at a time, or when encryption should be performed in a huge speed (e.g., fast telecommunications). While there are numerous block ciphers in the contemporary open literature that withstand public scrutiny (e.g., Triple-DES, AES, IDEA, etc.), there are almost no recently published stream ciphers with a similar status. A common solution is to use a block cipher under a *mode of operation* [64]. This solution enables the general functionality of stream ciphers, but only partially does it result in a fast implementation.

A well-known and a well-studied strategy for constructing stream ciphers is using *Linear Feedback Shift Registers* (LFSR). An LFSR is a shift register in which the feedback function is a linear combination of fixed locations of the register. Figure A.3 describes a typical LFSR. The fixed locations whose linear combination constitutes the feedback are called *taps*, e.g., in Figure A.3 the taps are in locations 19, 18, 17, and 14. The motivation for using LFSRs as components of stream ciphers are

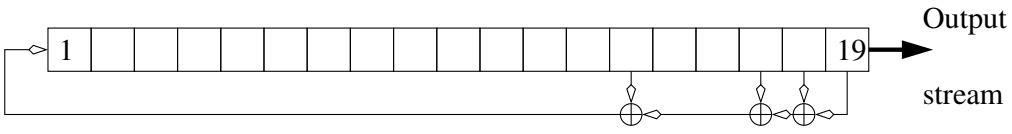


Figure A.3: A Typical LFSR
רגיסטר הזזה עם משוב לינארי

their simple structure, which can be efficiently implemented in hardware, together with a long period and "nice" statistical properties (of LFSRs with well selected parameters). With a careful choice of the taps an n-bit LFSR has a period of $2^n - 1$ (given a non-zero initial state), in which case it is called a *maximum-length LFSR*.

Let $c_1, \ldots, c_n$ be binary values, where c_i is 1 if there is a tap in location i of the LFSR and 0 otherwise. The *connection polynomial* of an LFSR is defined as the following polynomial above $GF(2)$: $C(X) = 1 \oplus \prod_{i=1}^{n} c_i X^i$. A necessary and sufficient condition for a LFSR to be a maximal-length LFSR is that its connection polynomial is primitive. Given a non-zero initial state, the output of a maximum-length LFSR has appealing statistical properties. In particular, in every window of length $2^n - 1 + k$ of the output, where $1 \leq k \leq n$, (or in every cycle) any non-zero string of length k appears exactly 2^{n-k} times, and the zero string of length k appears exactly $2^{n-k} - 1$ times. However, note that an LFSR cannot be used directly as a stream cipher, since every output bit is a linear combination of the initial internal state, and therefore after n output bits we can reconstruct the initial internal state. Note that not all stream ciphers use LFSRs as building blocks, although most fast hardware oriented stream ciphers do.

A.6 Correlation Attacks

A Correlation attack may be considered a "generic" attack on stream ciphers based on LFSRs. It takes advantage of a correlation between the output of a particular component of the stream cipher (e.g., LFSR) and the output of the stream cipher itself.

We give an example (taken from [60]) of a stream cipher with a strong correlation of the cipher's output and an output of an LFSR in the cipher: Geffe's generator contains three LFSRs: R1, R2, and R3 (see Figure A.4). In this generator, the output bit of R2 chooses the output either from the output of R1 or from the output of R3. Let $x_1(t), x_2(t), x_3(t), z(t)$ denote the t^{th} output bits of R1, R2, R3, and the generated keystream, respectively. The *correlation probability* of the output of R1

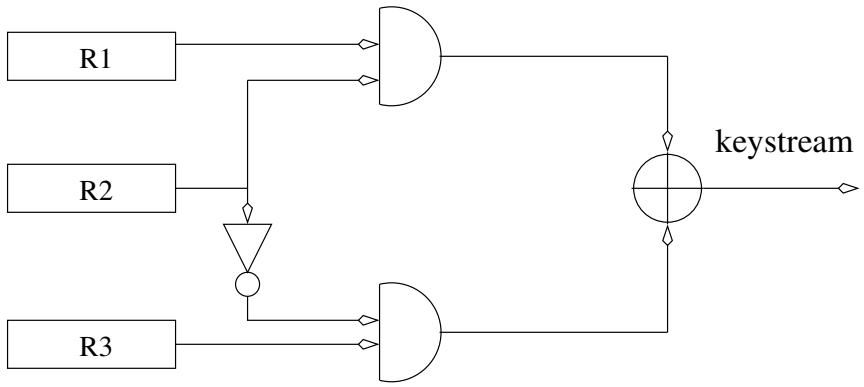


Figure A.4: Geffe's Generator
הגנרטור של גפה

$x_1(t)$ and the generated keystream $z(t)$ is:

$$\begin{aligned} Pr(x_1(t) = z(t)) &= Pr(x_2(t) = 1) + Pr(x_2(t) = 0) \cdot Pr(x_3(t) = x_1(t)) = \\ &= \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}. \end{aligned}$$

Similarly, $Pr(x_3(t) = z(t)) = \frac{3}{4}$.

Given an output of a stream cipher, we would like to find the initial internal state of LFSR Ri. Assume we know that the correct internal state satisfies $x_i(t) = z(t)$ with correlation probability $p \neq 1/2$, then we can try all the possible internal states, and count the number of coincidences between $z(1), z(1), \ldots, z(t)$ and $x_i(1), x_i(2), \ldots, x_i(t)$. For the correct internal state the number of coincidences is expected to be pt. When there are several LFSRs, $R_1, \ldots, R_k$ with lengths $n_1, \ldots, n_k$, respectively, whose correlation probabilities are different than $1/2$, we can repeat the attack for each such register independently. The number of possible keys of these registers (i.e., internal states of the registers) is about $\prod_{i=1}^{k} 2^{n_i}$, while the time complexity of this correlation attack is far smaller, i.e., about $\sum_{i=1}^{k} 2^{n_i}$.

A.7 Time/Memory/Data Tradeoff for Stream Ciphers

In section A.3 we discuss time/memory tradeoff for block ciphers. In his original paper about time/memory tradeoff [48] Hellman noted that his time/memory tradeoff

can be also applied to stream ciphers, by taking the first $|K|$ bits of the keystream as $f(K)$, where K is the key. Babbage [5] and Golic [46] independently discovered that a more efficient time/memory tradeoff exists for stream ciphers with a relatively small internal state. Later, Shamir and Biryukov [20] presented an improved time/memory/data tradeoff for stream ciphers.

We use the following notations for parameters of a time/memory/data tradeoff attack:

- N represents the size of the search space (usually the size of the internal state).
- P represents the time complexity of the preprocessing phase of the attack.
- M represents the amount of random access memory that is available to the attacker, usually the units are about $2\log_2(N)$ bits, depending on the specific details of the attack.
- T represents the time complexity of the inversion phase of the attacks.
- D represents the amount of data available to the attacker during the inversion phase.
- A represents the number of random memory accesses required by the inversion phase of the attack.

Babbage and Golic discovered that the following time/memory tradeoff curve applies to stream ciphers: $TM = N$ and $P = M$, for any $1 \leq T \leq D$. The attack associates with each internal state x out of the total N internal states of the cipher, the first $\log(N)$ bits y of the output stream from that state. The function $f(x) = y$ is considered a random mapping of N points into N points. f can be efficiently computed (by invocation of the stream cipher), but is expected to be hard to invert (i.e., it is difficult to find the internal state from an output prefix). The attacker wishes to invert f for a given an output, and thus, find the internal state. Assume the attacker has $\log(N) + D - 1$ bits of the output stream. This output stream can be seen as D different prefixes (of size $log(N)$) of an output of D different internal states. It suffices for the attacker to invert f for one of the D prefixes in order to recover an internal state. With the internal state the keystream can be predicted, and in many cases the initial key can be extracted.

In the preprocessing phase of Babbage and Golic's attack, the attacker chooses M random internal states x_i, and for each x_i calculates the corresponding $y_i = f(x_i)$. The pairs (x_i, y_i) are stored in memory, indexed by y_i. In the inversion phase, the attacker treats the data as D output prefixes $y_1, \ldots, y_D$, and for each y_i the attacker

searches the memory for a pair (x_i, y_i). If such a pair is found, then the internal state is x_i with a high probability. The attack is successful when there is a collision between the M precomputed prefixes and the D given prefixes. A collision is likely when $MD \approx N$. The attacker can overlook some of the given data, and therefore, $TM = N$ for any $1 \leq T \leq D$. The number of random disk accesses is therefore $A = T$.

Shamir and Biryukov discovered another tradeoff that applies to stream ciphers: $TM^2D^2 = N^2$ and $P = N/D$, for $D^2 \leq T \leq N$. They define a sampling resistance of a stream cipher to be $R = 2^{-k}$, where k is the maximum value for which it is possible to efficiently enumerate all internal states of the cipher that produce an output prefix of k zeros. They show that for a stream cipher with a sampling resistance R, a wider selection of T values $(RD)^2 \leq T \leq N$ is possible, and in addition the number of memory accesses is reduced to $A = R\sqrt{T}$.

Bibliography

[1] *The 3rd Generation Partnership Project (3GPP)*, `http://www.3gpp.org/`.

[2] Ross J. Anderson, *On Fibonacci Keystream Generators*, proceedings of Fast Software Encryption: Second International Workshop, Lecture Notes in Computer Science 1008, Springer-Verlag, pp. 346–352, 1995.

[3] Ross Anderson, Eli Biham, Lars Knudsen, *Serpent: A Proposal for the Advances Encryption Standard*, AES candidate, Available online on `http://www.cl.cam.ac.uk/~rja14/serpent.html`, 1998.

[4] Gildas Avoine, Pascal Junod, Philippe Oechslin, *Time-Memory Trade-Offs: False Alarm Detection Using Checkpoints (Extended Version)*, Available online on `http://lasecwww.epfl.ch/pub/lasec/doc/AJO05a.pdf`, 2005.

[5] Steve Babbage, *A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers*, European Convention on Security and Detection, IEE Conference Publication No. 408, 1995. Also presented at the rump session of Eurocrypt '96. Available online on `http://www.iacr.org/conferences/ec96/rump/`.

[6] Elad Barkan, Eli Biham, *Conditional Estimators: an Effective Attack on A5/1*, proceedings of SAC 2005, LNCS 3897, pp. 1–19, Springer-Verlag, 2006.

[7] Elad Barkan, Eli Biham, *In How Many Ways Can You Write Rijndael?*, Advances in Cryptology, proceedings of Asiacrypt 2002, Lecture Notes in Computer Science 2501, Springer-Verlag, pp. 160–175, 2002.

[8] Elad Barkan, Eli Biham, *On the Security of the GSM Cellular Network*, Security and Embedded Systems, NATO Security through Science Series,

D: Information and Communication Security – Vol. 2, IOS Press, pp. 188–195, 2006.

[9] Elad Barkan, Eli Biham, *The Book of Rijndaels*, IACR ePrint Report 2002/158, `http://eprint.iacr.org/2002/158/`, 2002.

[10] Elad Barkan, Eli Biham, Nathan Keller, *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communications*, Advances in Cryptology, proceedings of Crypto 2003, Lecture Notes in Computer Science 2729, Springer-Verlag, pp. 600–616, 2003.

[11] Paulo S.L.M. Barreto, Vincent Rijmen, *The Anubis Block Cipher*, submitted to NESSIE, 2000.

[12] Paulo S.L.M. Barreto, Vincent Rijmen, *The Khazad Legacy-Level Block Cipher*, submitted to NESSIE, 2000.

[13] Eli Biham, *How to decrypt or even substitute DES-encrypted messages in 2^{28} steps*, Information Processing Letters, Volume 84, Issue 3, pp. 117–124, 2002.

[14] Eli Biham, *New Type of Cryptanalytic Attacks Using Related Key*, Advances in Cryptology, proceedings of Eurocrypt'93, Lecture Notes in Computer Science 765, Tor Helleseth, Ed., pp. 229–246, Springer-Verlag, 1994.

[15] Eli Biham, Orr Dunkelman, *Cryptanalysis of the A5/1 GSM Stream Cipher*, Progress in Cryptology, proceedings of Indocrypt'00, Lecture Notes in Computer Science 1977, Springer-Verlag, pp. 43–51, 2000.

[16] Eli Biham, Adi Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.

[17] Eli Biham, Adi Shamir, *Differential Fault of Secret-Key Cryptosystems*, Advances in Cryptology, proceedings of Crypto'97, Lecture Notes in Computer Science 1294, Springer-Verlag, pp. 513–525, 1997.

[18] Alex Biryukov, *Some Thoughts on Time-Memory-Data Tradeoffs*, IACR ePrint Report 2005/207, `http://eprint.iacr.org/2005/207.pdf`, 2005.

[19] Alex Biryukov, Christophe De Cannière, An Braeken, Bart Preneel, *A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms*, Advances in Cryptology, proceedings of Eurocrypt 2003, Lecture Notes in Computer Science 2656, Eli Biham, Ed., Springer-Verlag, pp. 33–50, 2003.

[20] Alex Biryukov, Adi Shamir, *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers*, Advances in Cryptology, proceedings of Asiacrypt 2000, Lecture Notes in Computer Science 1976, Springer-Verlag, pp. 1–13, 2000.

[21] Alex Biryukov, Adi Shamir, David Wagner, *Real Time Cryptanalysis of A5/1 on a PC*, Advances in Cryptology, proceedings of Fast Software Encryption'00, Lecture Notes in Computer Science 1978, Springer-Verlag, pp. 1–18, 2001.

[22] Johan Borst, Bart Preneel, Joos Vandewalle, *On the Time-Memory Tradeoff Between Exhaustive Key Search and Table Precomputation*, Proceedings of 19th Symposium on Information Theory in the Benelux, Veldhoven (NL), pp. 111–118, 1998.

[23] Antoon Bosselaers, Joan Daemen, Erik De Win, Bart Preneel, Vincent Rijmen, *The Cipher Shark*, proceedings of Fast Software Encryption '96, Lecture Notes in Computer Science 1039, Dieter Gollmann, Ed., Springer-Verlag, pp. 99–112, 1996.

[24] Marc Briceno, Ian Goldberg, David Wagner, *A pedagogical implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms*, `http://cryptome.org/gsm-a512.htm` (originally on www.scard.org), 1999.

[25] Marc Briceno, Ian Goldberg, David Wagner, *An implementation of the GSM A3A8 algorithm*, `http://www.iol.ie/~kooltek/a3a8.txt`, 1998.

[26] Marc Briceno, Ian Goldberg, David Wagner, *GSM Cloning*, `http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html`, 1998.

[27] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr., Luke O'Connor, Mohammad Peyravian, David Safford, Nevenko Zunic, *MARS — a candidate cipher for*

AES, AES candidate, IBM Corporation, Available online on `http://www.research.ibm.com/security/mars.html`, 1998, revised 1999.

[28] Nicolas Courtois, *Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt*, proceedings of ICISC 2002, Lecture Notes in Computer Science 2587, Springer-Verlag, pp. 182–199, 2003.

[29] Nicolas Courtois, Willi Meier, *Algebraic Attacks on Stream Ciphers with Liner Feedback*, Advances in Cryptology, proceedings of Eurocrypt 2003, Lecture Notes in Computer Science 2656, Springer-Verlag, pp. 345–359, 2003.

[30] Nicolas Courtois, Alexander Klimov, Jacques Patarin, Adi Shamir, *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*, Advances in Cryptology, proceedings of Eurocrypt 2000, Lecture Notes in Computer Science 1807, Springer-Verlag, pp. 392–407, 2000.

[31] Nicolas Courtois, Josef Pieprzyk, *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*, Advances in Cryptology, proceedings of Asiacrypt 2002, Lecture Notes in Computer Science 2501, Springer-Verlag, pp. 267–287, 2002.

[32] Joan Daemen, Lars R. Knudsen, Vincent Rijmen, *The Block Cipher Square*, proceedings of Fast Software Encryption '97, Lecture Notes in Computer Science 1267, Eli Biham, Ed., Springer-Verlag, pp. 149–165, 1997.

[33] Joan Daemen, Vincent Rijmen, *The Design of Rijndael, AES - The Advanced Encryption Standard*, Springer-Verlag, 2002.

[34] Edsger W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, Vol. 1, pp. 269–271, 1959.

[35] Patrik Ekdahl, Thomas Johansson, *Another Attack on A5/1*, IEEE Transactions on Information Theory 49(1), pp. 284-289, 2003.

[36] European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Channel Coding*, TS 100 909 (GSM 05.03), `http://www.etsi.org`.

[37] European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Mobile radio interface; Layer 3 specification*, TS 100 940 (GSM 04.08), `http://www.etsi.org`.

[38] European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Mobile Station - Base Stations System (MS - BSS) Interface Data Link (DL) Layer Specification*, TS 100 938 (GSM 04.06), `http://www.etsi.org`.

[39] European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Multiplexing and multiple access on the radio path*, TS 100 908 (GSM 05.02), `http://www.etsi.org`.

[40] European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Physical layer on the radio path; General description*, TS 100 573 (GSM 05.01), `http://www.etsi.org`.

[41] European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Security related network functions*, TS 100 929 (GSM 03.20), `http://www.etsi.org`.

[42] Niels Ferguson, Richard Schroeppel, Doug Whiting, *A Simple Algebraic Representation of Rijndael*, proceedings of Selected Areas in Cryptography, Lecture Notes in Computer Science 2259, Serge Vaudenay and Amr Youssef, Eds., Springer-Verlag, pp. 103–111, 2001.

[43] Amos Fiat, Moni Naor, *Rigorous Time/Space Tradeoffs for Inverting Functions*, STOC 1991, ACM Press, pp. 534–541, 1991.

[44] Amos Fiat, Moni Naor, *Rigorous Time/Space Tradeoffs for Inverting Functions*, SIAM Journal on Computing, 29(3): pp. 790–803, 1999.

[45] Ian Goldberg, David Wagner, Lucky Green, *The (Real-Time) Cryptanalysis of A5/2*, presented at the Rump Session of Crypto'99, 1999.

[46] Jovan Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, Advances in Cryptology, proceedings of Eurocrypt '97, Lecture Notes in Computer Science 1233, pp. 239–255, Springer-Verlag,1997.

[47] Shai Halevi, Don Coppersmith, Charanjit Jutla, *Scream: a Software-Efficient Stream Cipher*, preproceedings of Fast Software Encryption 2002, pp. 190–204, 2002.

[48] Martin E. Hellman, *A Cryptanalytic Time-Memory Trade-Off*, IEEE Transactions on Information Theory, Vol. IT-26, No. 4, pp. 401–406, 1980.

[49] Walter Hoffman, Richard Pavley, *A Method for the Solution of the Nth Best Path Problem*, Journal of the ACM (JACM), Volume 6, Issue 4, pp. 506–514, 1959.

[50] Thomas Jakobsen, Lars R. Knudsen, *The Interpolation Attack on Block Ciphers*, proceedings of Fast Software Encryption '97, Lecture Notes in Computer Science 1267, Eli Biham, Ed., Springer-Verlag, pp. 28–40, 1997.

[51] Il-Jun Kim, Tsutomu Matsumoto, *Achieving Higher Success Probability in Time-Memory Trade-Off Cryptanalysis without Increasing Memory Size*, IEICE Transactions on Fundamentals, Vol. E82-A, No. 1, pp. 123–129, 1999.

[52] Aviad Kipnis, Adi Shamir, *Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization*, Advances in Cryptology, proceedings of Crypto '99, Lecture Notes in Computer Science 1666, Springer-Verlag, pp. 19–30, 1999.

[53] Lars R. Knudsen, *Practically Secure Feistel Ciphers*, proceedings of Fast Software Encryption '93, Lecture Notes in Computer Science 809, Ross Anderson, Ed., Springer-Verlag, pp. 211–221, 1994.

[54] Paul Kocher, Joshua Jaffe, Benjamin Jun, *Differential Power Analysis*, Advances in Cryptology, proceedings of Crypto'99, Lecture Notes in Computer Science 1666, Springer-Verlag, pp. 388–397, 1999.

[55] Koji Kusuda, Tsutomu Matsumoto, *Optimization of Time-Memory Trade-Off Cryptanalysis and Its Application to DES, FEAL-32, and Skipjack*, IEICE Transactions on Fundamentals, Vol. E79-A, No. 1, pp. 35–48, 1996.

[56] Chae Hoon Lim, *Crypton: a new 128-bit Block Cipher - Specifications and Analysis*, submitted to the Advanced Encryption Standard (AES) contest, 1998.

[57] Mitsuru Matsui, *Linear cryptanalysis method for DES cipher*, Advances in Cryptology, proceedings of Eurocrypt '93, Lecture Notes in Computer Science 765, T. Helleseth, Ed., Springer-Verlag, pp. 386–397, 1994.

[58] Alexander Maximov, Thomas Johansson, Steve Babbage, *An improved correlation attack on A5/1*, proceedings of SAC 2004, LNCS 3357, pp. 1–18, Springer-Verlag, 2005.

[59] Willi Meier, Othmar Staffelbach, *Fast Correlation Attacks on Certain Stream Ciphers*, Journal of Cryptology, Volume 1, Issue 3, pp. 159–176, Springer-Verlag, 1989.

[60] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography* , CRC Press, October 1996.

[61] Sean Murphy, Matthew J.B. Robshaw, *Essential Algebraic Structure within the AES*, Advances in Cryptology, proceedings of Crypto'02, Lecture Notes in Computer Science 2442, M.Yung, Ed., Springer-Verlag, pp. 1–16, 2002.

[62] National Bureau of Standards, *Data Encryption Standard*, Federal Information Processing Standard, FIPS-46, U.S. Department of Commerce, 1977.

[63] National Institute of Standards and Technology, *Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard*, United States Federal Register, September 12, 1997.

[64] National Institute of Standards and Technology, *Announcing the Standard for DES Modes of operation*, Federal Information Processing Standard, FIPS-81, U.S. Department of Commerce, 1980.

[65] National Institute of Standards and Technology, *Advanced Encryption Standard*, Federal Information Processing Standard, FIPS-197, U.S. Department of Commerce, 2001.

[66] NESSIE (New European Schemes for Signature, Integrity, and Encryption), Available online on `http://www.cryptonessie.org`, 2000.

[67] Philippe Oechslin, *Making a Faster Cryptanalytic Time-Memory Trade-Off*, Advances in Cryptology, proceedings of Crypto 2003, Lecture Notes in Computer Science 2729, Springer-Verlag, pp. 617–630, 2003.

[68] Francois-Xavier Standaert, Gael Rouvroy, Jean-Jacques Quisquater, Jean-Didier Legat, *A Time-Memory Tradeoff Using Distinguished Points: New Analysis & FPGA Results*, proceedings of CHESS 2002, Lecture Notes in Computer Science 2523, Springer-Verlag, pp. 593–609, 2003.

[69] Ronald Rivest, Matthew Robshaw, Ray Sidney, Yiqun Lisa Yin, *The RC6^{TM} Block Cipher*, AES candidate, Available online on `http://www.rsasecurity.com/rsalabs/rc6/`, 1998.

[70] Ronald L. Rivest, Adi Shamir, Leonard Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of the ACM, 21(2):120–126, 1978.

[71] Kenneth H. Rosen (Ed. in chief), *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, 2000.

[72] Atri Rudra, Pradeep K. Dubey, Charanjit S. Jutla, Vijay Kumar, Josyula R. Rao, Pankaj Rohatgi, *Efficient Rijndael Encryption Implementation with Composite Field Arithmetic*, proceedings of CHES 2001, Lecture Notes in Computer Science 2162, David Naccache, Çetin K. Koç and Christophe Paar, Eds., pp. 171–184, Springer-Verlag, 2001.

[73] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, *Twofish: A 128-Bit Block Cipher*, AES candidate, Available online on `http://www.schneier.com/twofish.html`, 1998.

[74] Security Algorithms Group of Experts (SAGE), *Report on the specification and evaluation of the GSM cipher algorithm A5/2*, `http://cryptome.org/espy/ETR278e01p.pdf`, 1996.

[75] Thomas Siegenthaler, *Decrypting a Class of Stream Ciphers Using Ciphertext Only*, IEEE Transactions on Computers, Volume 49, Issue 1, pp. 81–85, 1985.

[76] Slobodan Petrović, Amparo Fúster-Sabater, *Cryptanalysis of the A5/2 Algorithm*, IACR ePrint Report 2000/052, `http://eprint.iacr.org`, 2000.

[77] Serge Vaudenay, *Alert on Non-Linearity: Linearities in RIJNDAEL, KASUMI,...*, presented in the rump session of Crypto'01.

[78] Shee-Yau Wu, Shih-Chuan Lu, Chi-Sung Laih, *Design of AES Based on Dual Cipher and Composite Field*, proceedings of CT-RSA 2004, Lecture Notes in Computer Science 2964, Tatsuaki Okamoto, Ed., pp. 25–38, Springer-Verlag, 2004.

[79] Andrew Chi-Chih Yao, *Coherent Functions and Program Checkers (Extended Abstract)*, STOC 1990, ACM Press, pp. 84–94, 1990.

קריפטאנליזה של צפנים ופרוטוקולים

אלעד פנחס ברקן

Technion - Computer Science Department - Ph.D. Thesis PHD-2006-04 - 2006

קריפטאנליזה של צפנים ופרוטוקולים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת תואר
דוקטור לפילוסופיה

אלעד פנחס ברקן

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
אדר ה'תשס"ו חיפה מרץ 2006

המחקר נעשה בהנחיית פרופ' אלי ביהם בפקולטה למדעי המחשב.

זו זכות גדולה להודות לאלי ביהם על תמיכתו רבת התובנות, שאפשרה את העבודה הזאת, וכן על כך שהכשיר אותי כמדען וחוקר. אני מוקיר את אלי במיוחד על הכבוד והאמון שנתן בי, ועל כך שאפשר לי דרגת חופש גבוהה מאוד במחקר. אלי מצא את שביל הזהב בין חינוך, נוקשות, ודאגה. יכולתו המיוחדת לתקשר במהירות באופן אישי (ולפעמים בשובבות) תמיד משאירה אותי עם חיוך.

אני מלא תודה לעדי שמיר על שיתוף הפעולה הפורה, על כך שהיה זמין מסביב לשעון (ומסביב לגלובוס), ועל סבלנותו ותבונתו. אני מוקיר את נתן קלר על סקרנותו הנפלאה והמועילה, ועל כך שהוא brainmaker מדהים. אני שמח להודות לעמיתיי בטכניון, אור דונקלמן ורפי חן, על דיונים פוריים, ועל הזמן הנהדר שהעברנו יחד.

אני מרגיש שאין מילים שיכולות לבטא את הכרת התודה העמוקה שלי למשפחתי האוהבת, שנתנה לי את אהבתה הבלתי מסוייגת, את תמיכתה, והבנתה בזמנים הטובים יותר והטובים פחות של מחקר זה. תודות מיוחדות שלוחות לאשתי לעתיד תמר קשתי על הקשבתה לכל רעיונות המחקר בזמן ההליכות הספורטיביות שלנו, על אהבתה, על עצותיה המוצלחות ועידודה, ועל היותה חברה נהדרת.

אני מודה לבית הספר ללימודי מוסמכים בטכניון על התמיכה הכספית הנדיבה בהשתל-מותי.

תוכן עניינים

רשימת טבלאות

רשימת איורים

תקציר

קריפטוגרפיה מספקת סט חשוב של כלים המאפשרים את דרך החיים המודרנית. קריפטוגרפיה מספקת כלים למסחר מקוון בטוח, חתימות דיגיטליות, פרוטוקולים בטוחים, ממירי טלביזיה לווינית בטוחים, אבטחת לשיחות טלפון, בחירות אלקטרוניות, ועוד. קריפטאנליזה מוודאת את נכונות הבטחות הבטיחות של הקריפטוגרפיה. לדוגמה, אנו מעוניינים לוודא שאלגוריתמי הצפנה והפרוטוקולים שמשתמשים בהם אכן מספקים בטיחות. בהרבה מקרים, עדיפה מערכת ללא הבטחות בטיחות על פני צופן חלש במערכת עם תכנון בטיחות לקוי, כיוון שהמשתמשים במערכת הראשונה מודעים לכך שאיננה בטוחה, אולם המשתמשים במערכת השניה עלולים לסמוך על המערכת לגבי סודותיהם הכמוסים.

בעבודה זו ארבע תרומות נפרדות בתחום הקריפטאנליזה של צפנים ושל פרוטוקולים. התרומה הראשונה עוסקת בצפנים בכלל, ופרט בתקן ההצפנה שנבחר לאחרונה (- AES Advanced Encryption Standard, המבוסס על הצופן ריינדל). שאלנו מה יקרה אם נחליף את כל הקבועים בצופן, כולל את הפולינום האי-פריק, את המקדמים הכפליים בפעולות הצופן, וכן את הטרנספורמציה האפינית המהווה חלק מהטרנספורמציה הלא לינארית (ה-S box). אנו מראים שהחלפות אלו יכולות ליצור צפנים דואלים חדשים, אשר איזומורפיים לצופן המקורי. אנו מציגים מספר צפנים דואלים, כולל ריינדל בריבוע וצפנים דואלים שבהם הפולינום האי-פריק מוחלף בפולינום פרימיטיווי. בנוסף אנו מתארים משפחה נוספת של צפנים דואליים המורכבת מהלוגריתם של ריינדל. אחר כך, אנו דנים בצפנים שהם דואלים לעצמם, ומראים שצפנים אלו ניתנים להתקפה בזמן מהיר יותר מחיפוש ממצה. לבסוף, אנו דנים בשימושים לצפנים דואלים. לצפנים דואלים שימושים רבים: מצד אחד

הם עשויים להעניק תובנות לפיתוח התקפות, ומצד שני הם יכולים לשמש להגנה בפני התקפות ערוץ צדדי (Side-channel attacks). בפן השימושי, צפנים דואלים יכולים לשמש למציאת מימושים יעילים יותר של צפנים קיימים. לאחר פרסום העבודה הראשונית שלנו, מצאו חוקרים אחרים מימוש יעיל יותר לריינדל על בסיס הצפנים הדואליים שגילינו.

התרומה השניה עוסקת בבטיחות מערכת התקשורת הסלולרית העולמית - GSM. אנו מציגים התקפות מאוד מעשיות (במודל כתב סתר בלבד, שהוא המודל היותר קשה לתקיפה). ההתקפות יכולות אפילו לפרוץ לרשתות GSM שמשתמשות בצפנים "לא-שבירים". ראשית אנו מתארים התקפת כתב גלוי ידוע על הצופן החלש של GSM (צופן בשם A5/2). אנו משפרים את ההתקפה הזו להתקפת כתב סתר בלבד, הדורשת מספר עשרות מילישניות של שיחה מוצפנת ומאחזרת את מפתח ההצפנה בתוך פחות משניה על מחשב אישי. אז, אנו מרחיבים את ההתקפה להתקפה מורכבת יותר על הצופן החזק יותר — A5/1. אנו מתארים התקפות אקטיביות חדשות על הפרוטוקולים של רשתות GSM המשתמשות בהצפנה החזקה A5/1 או בהצפנה היותר חזקה וחדשה — A5/3. ההתקפות על הפרוטוקולים מצליחות גם כנגד רשת הנתונים (דור שתים וחצי) — GPRS. ההתקפות מנצלות פגמים בפרוטוקולים של GSM, והן מצליחות כאשר הטלפון תומך באלגוריתם A5/2, גם אם האלגוריתם איננו בשימוש ברשת. יש להדגיש שההתקפות האקטיביות הן על הפרוטוקולים, ולכן ניתן להפעיל אותם כאשר הטלפון תומך באלגוריתם חלש על מנת לתקוף רשתות המשתמשות באלגוריתם חזק יותר. למשל, ההתקפות מתאימות לתקיפת רשתות המשתמשות ב-A5/3 בעזרת התקפה על A5/1. בניגוד להתקפות קודמות על GSM שדרשו כמויות מידע גדולות הקשות מאוד להשגה בפועל (כמו למשל רצפים ארוכים של כתב גלוי ידוע), ההתקפות שלנו מאוד פרקטיות ולא דורשות את ידיעת תוכן השיחה. מעבר לכך, אנו מתארים כיצד ניתן לחזק חלק מההתקפות כך שיהיו עמידות לשגיאות קליטה. אנו מראים מספר תרחישים שבהם ניתן לבצע את ההתקפה, כגון ציתות לשיחות, חטיפת שיחות, גניבת שיחות ושינוי תוכן של הודעות מידע (SMS).

התרומה השלישית עוסקת בצפני שטף המבוססים על רגיסטרי הזזה עם משוב לינארי (Linear-Feedback Shift Registers), אשר מוזזים באופן בלתי-סדיר על מנת להסתיר את

המצב בו הם נמצאים. אנו רותמים את כוחם של משערכים מותנים לצורך יצירת התקפות מתאם (קורלציה) על צפנים המבוססים על רגיסטרים כאלו. בכוחם של משערכים מותנים לפצות על חלק מאפקט ההסתרה של ההזזה הלא סדירה של הרגיסטרים. כתוצאה מכך ניתן לקבל מתאם חזק יותר בין ביטי הפלט של הצופן לבין הערכים הפנימיים של הרגיסטרים. אנו מדגימים את ההתקפה המשתמשת במשערכים מותנים על הצופן היותר חזק של GSM — A5/1, ומשיגים שיפור משמעותי של פי שתיים במתאם. בנוסף, אנו מבחינים בשלוש חולשות שלא דווחו לפני כן ברגיסטר ההזזה של A5/1 הידוע כ-$R2$. החולשות החדשות שחשפנו רומזות על קריטריון חדש שמתכנני צפנים צריכים לקחת בחשבון. ההתקפה החדשה שאנו מציגים על A5/1 משיגה ביצועים שלא היו ניתנים להשגה עד כה: בהנתן כ-1500 עד 2000 מסגרות מידע (כ-6.9 עד 9.2 שניות של כתב גלוי ידוע של שיחה), ההתקפה לוקחת עשרות שניות עד מספר דקות על מחשב אישי, עם שיעור הצלחה של בערך 91%. לבסוף, אנו חושפים מקור לכתב גלוי ידוע ב-GSM שיכול לספק את הכמות הנדרשת של כתב גלוי ידוע מתוך כתב הסתר של כ-3 עד 4 דקות שיחה, וכך ההתקפה שלנו הופכת להתקפת כתב סתר בלבד.

התרומה הרביעית מתייחסת להתקפות גנריות על צפנים. בפרט, אנו מוכיחים חסמים על החלפת תמורות זמן/זכרון קריפטאנליטית. בהתקפות גנריות, מתייחסים לצופן כאל פונקציה שהיא "קופסאה שחורה" הנבחרת באופן אקראי ממרחב הפונקציות. מטרת התוקף היא להפוך פונקציה זו על ערך נתון. דוגמה פשוטה ביותר (וקיצונית) להתקפה גנרית היא חיפוש ממצה. בהתקפה זו עובר התוקף על כל הערכי המקור האפשריים לפונקציה, ומפעיל את הפונקציה על כל אחד מערכים אלו על מנת לבדוק הם הוא מוביל לערך הנתון. דוגמה קיצונית נוספת להתקפה גנרית היא לשמור בטבלה את כל התוצאות האפשריות של הפונקציה, וליד כל תוצאה לשמור ערך מקור המוביל לתוצאה (ניתן לבנות את הטבלה בעזרת חיפוש ממצה). כעת ניתן למצוא מקור לערך הנתון על ידי מציאתו בטבלה. החלפת תמורות זמן/זכרון עוסקת במציאת פשרה בין סכמות קיצוניות אלו, כלומר מציאת סכמות בהן נדרש זכרון קטן משמעותית משמירת כל תוצאות הפונקציה בטבלה, אבל גם זמן החישוב הנדרש נמוך משמעותית מחיפוש ממצה (וכמובן ארוך יותר מאשר חיפוש

בטבלה). בשנת 1980, הלמן הציג את השיטה הידועה ביותר להחלפת תמורות זמן/זכרון קריפטאנליטית. שיטתו מבוססת על הרבה טבלאות קטנות, כך שכל טבלאה מכסה חלק קטן מערכי הפונקציה. מאז, הוצגו שיפורים רבים, ובהם בשנת 2003 הוצעה שיטת צבעי-הקשת, המסוגלת להשתמש בטבלה אחת גדולה במקום המוני הטבלאות של הלמן, וזמן ריצתה נטען להיות קטן פי שתים לעומת שיטתו של הלמן (עבור אותה כמות זכרון). בעבודתנו, אנו מגדירים מודל כללי להחלפת תמורות זמן/זכרון קריפטאנליטית, הכולל את כל הסכמות הקיימות כמקרים פרטיים. המודל מבוסס על מושג חדש: גרף אקראי בעל מצב, שבו התפתחות של מסלולים בגרף תלויה במצב חבוי. בעזרת ניתוח קומבינטורי מדוייק, אנו מוכיחים חסם עליון על מספר הערכים עבורם ניתן להפוך פונקציה בעזרת סכמת החלפת תמורות זמן/זכרון. מחסם עליון זה אנו מוכיחים חסם תחתון על מספר המצבים החבויים הדרושים בגרף. תחת תוספת של מספר הנחות טבעיות על התנהגות האלגוריתם, אנו מוכיחים חסם תחתון על זמן הריצה שלו. יש להדגיש שחסמים אלו נכונים לרוב המכריע של הפונקציות האקראיות. בנוסף, אנו מתארים גרסאות חדשות של סכמות קיימות, הכוללות, בין היתר, שיטה לשיפור זמן הריצה (בפקטור קטן) תמורת זמן עיבוד-מקדים ארוך יותר. כמו כן אנו מתארים התאמות של סכמת צבעי הקשת להחלפת תמורות זמן/זכרון/מידע.